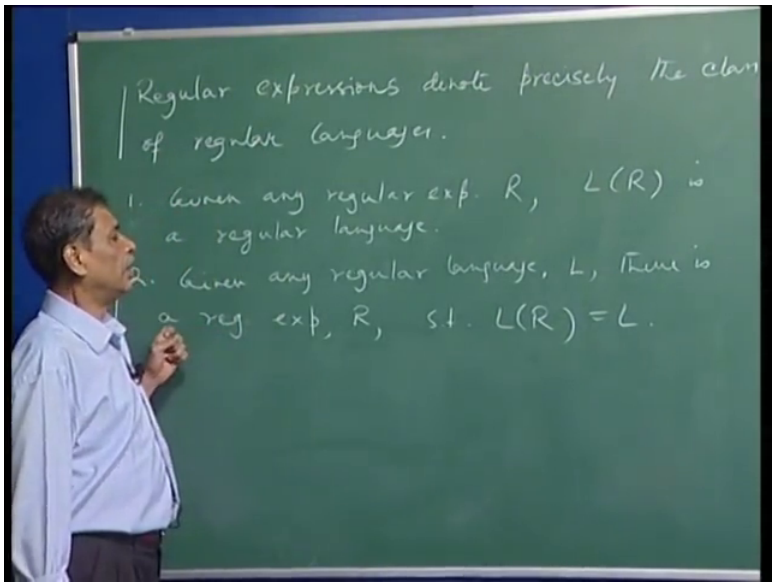
Theory of Computation.
Professor Somenath Biswas.
Department of Computer Science & Engineering.
Indian Institute of Technology, kanpur.
Lecture-12.
Construction of a regular expression for a language given a DFA accepting it.
Algebraic closure properties of regular languages.

(Refer Slide Time: 1:06)

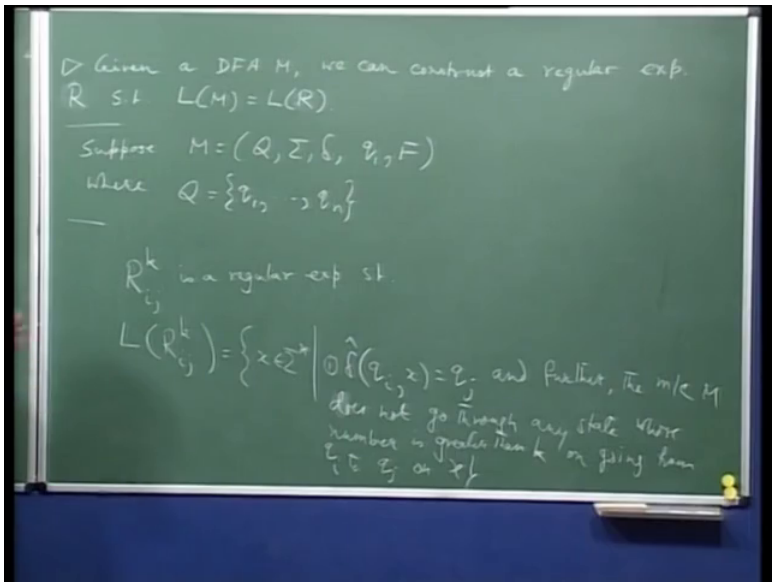


In the last lecture we started proving this important fact that regular expressions denote precisely the class of regular languages. By that what we mean is that the set of regular languages, if you take all the languages that they denote, that class of all languages denoted by regular expression is precisely the class of regular languages. And for this there are 2 parts in proving this in a straightforward manner. What we 1st showed in fact last time we completed this that given any regular expression r, we can construct an nfa which will accept the language denoted by the regular expression r.

Lr is a regular language, this improved by providing an nfa with epsilon transitions for the language lr. Now we need to prove the other way and that other way is, given any regular language, say l, there is a regular expression r such that the language denoted by that regular expression is exactly the language l which was given to us, which is of course regular to begin with. So one point is that we are saying given any regular language l, how does one give a language, regular language because in general regular languages infinite, so one cannot list out all the strings in the language.

And that is where these machines come handy. So given some regular language, it is indeed regular because there is an nfa or nfa with epsilon transitions or a dfa, in fact all 3 of them, there will be such one of the all 3 machines which will accept that regular language.

(Refer Slide Time: 2:52)



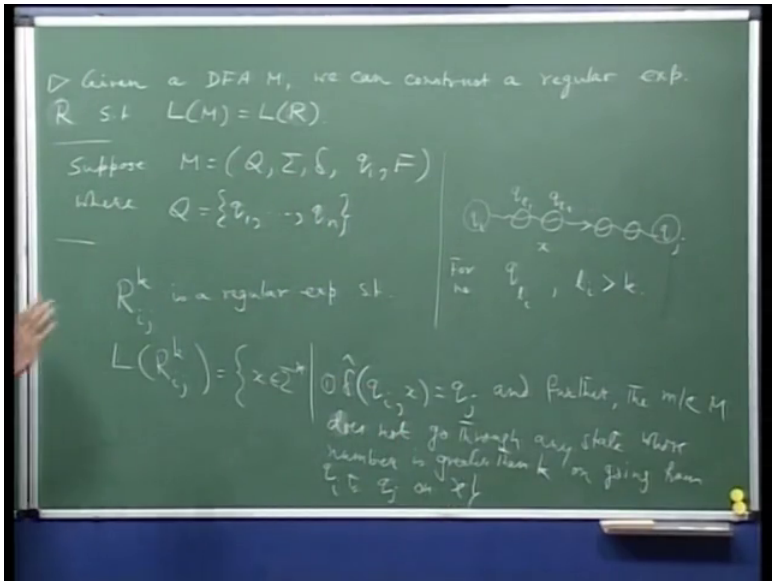
So what in effect is there is that what we can say is, like what we are going to prove the following that given a dfa m, we can construct a regular expression r such that language accepted by dfa is the language denoted by the regular expression r. Okay. So we start with the given dfa and we show how for any given dfa we can construct a regular expression which is equivalent in the sense as shown here. So let us give the details of the construction that we have in mind. So suppose m which is the dfa is q, sigma, delta, q1 and f, where the set of states is q 1 up to qn. There are n states and they are numbered as 1 through n. We are not using q0 for a different reason, for convenience of notation as we shall see now.

Now what we are going to do is we are going to construct regular expressions for this certain number of regular expressions in fact and then we will put them together and should, i will get the regular expression r. So now let me define this r ij k, right. This r ij k is a regular expression such that the language denoted by r ij k, this language, is the set of all strings which 2 things, one delta hat of q i x is qj. So that means in other words that such a string x we take the machine from state qi to state qj and further x , let us say the machine m does not go to any state whose number is greater than k on going from qi to qj on x.

So this sentence is a big mouthful but the idea is very simple. What we are saying that this r ij k, now notice there are 3 of these ij and k and how they are occurring, firstly r ij k is a regular

expression which will have strings such that each such string x will take the machine from state qi to state qj.

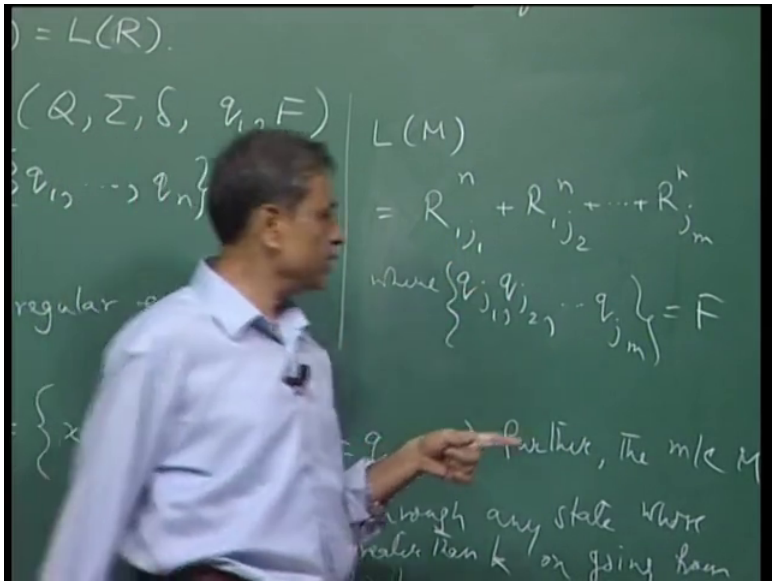
(Refer Slide Time: 8:26)



But there is a condition that the, as the machine, in fact the picture will look okay. So this is the state qi, this is the state qj and this is, this is let say the string x, so there are these states which are intermediate states as a machine is going from qi to qj. And this is let us ql 1, this state is ql 2, what we are saying, none of these states is such that for no ql i, li is greater than k. So the machine is on x, the machine is going from q while to qj, what we mean by that is not go through any state is that that on the string x, sometimes let say this state is reached and then the state is left.

All these states must have its number, every state has a number from 1 to n, all these states the machine is going through on this x from qi to qj, these numbers are strictly less than or equal to k. But notice either i or j or both can be greater than k because we are not going through the initial or the final state region. By going through one means one passing through this, you reach and then you leave.

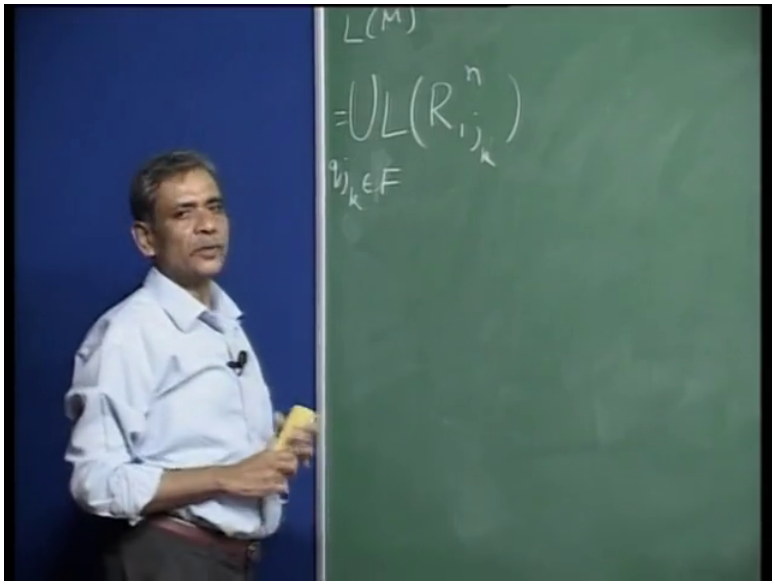
(Refer Slide Time: 10:49)



Suppose i manage to build this regular expressions and k can vary, you can see, either the machine takes a direct edge from i to j or machine goes from i to j going through the state q1 or q1 q2 or q1 q2 q3 and so on. What should be clear that the language accepted by this machine m is r1 j1 n + r1 j2n… r where j1, j2, q j1 qj2, qjm, these are the only final states, right. Let us understand this but in the beginning before we see how these can be defined. What is it, see, firstly remember according to this, q j1 is the final state.

So all the strings which take the machine from the initial state which is q1, so this is where the index 1 coming, to that state j1 which is a final state, all these strings must be in the language accepted by m because they all take the machine from the initial state to one of the final states which is j1. Then let us say i have another set of strings which take the machine from 1 to another final state which is q j2. Now that, what about that passing through? You see here what i have here as a superscript, which is n, now there are only q1 through qm, only the states are there.

(Refer Slide Time: 13:55)



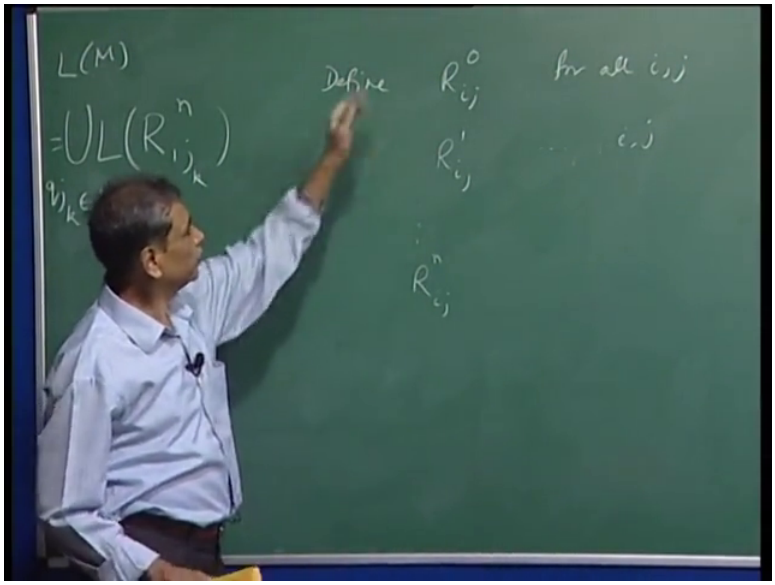
So in a way when i have n as superscript, this means that the machine is read to visit any state whatsoever in passing through, in its path to 1 to j1, 1 to j2 and so on. So in that sense, this quantity r1 j1 n denotes maybe i should write that, clearly what we are trying to say the language denoted by this regular expression is the set of all strings x and of course sigma star that is the alphabet, sigma is the alphabet, all those strings in sigma star such that delta hat of q1, q1 on the input x is q j1. Point i am trying to say is although there is a qualification which is in terms of the 1st step, that means in general what it is saying is this string does not pass through any state whose number higher than n.

But since there are no strings whose number is more than n, this really is no restriction at all, therefore this is true. Now as we have said here q j1 is an element of the set of final states, recall, according to that q j1, q j2, qjn, these are the only set of, i mean these are the only set of final states of the machine m. Therefore a string which is, a string is accepted if and only if it takes the machine to one of the final states. So clearly we can see that the language therefore, language accepted by m is nothing but the union of, so let me just write this here, union of the language denoted by the set of strings or the regular expressions, this regular expression where this union is over q jk is one of the elements of the final states.

So that is not difficult to see. If my definition, if i am, if i can indeed define r i jk, ultimately r ij n, then i can, this union, of course i can do in terms of regular expression if i just take this each individual regular expression, put + in between, that is the union of the languages, languages denoted by each one of them individually. So that i would be able to write down the regular expression for language accepted by l. So now how can we build r1 jn, in

particular of course we are interested more with r1 to final state, that kind of regular expression but with the superscript n.

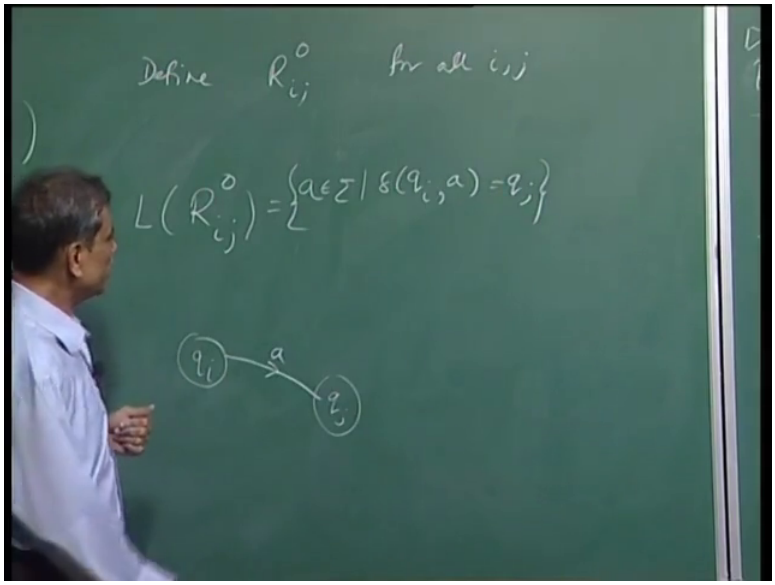
(Refer Slide Time: 17:37)

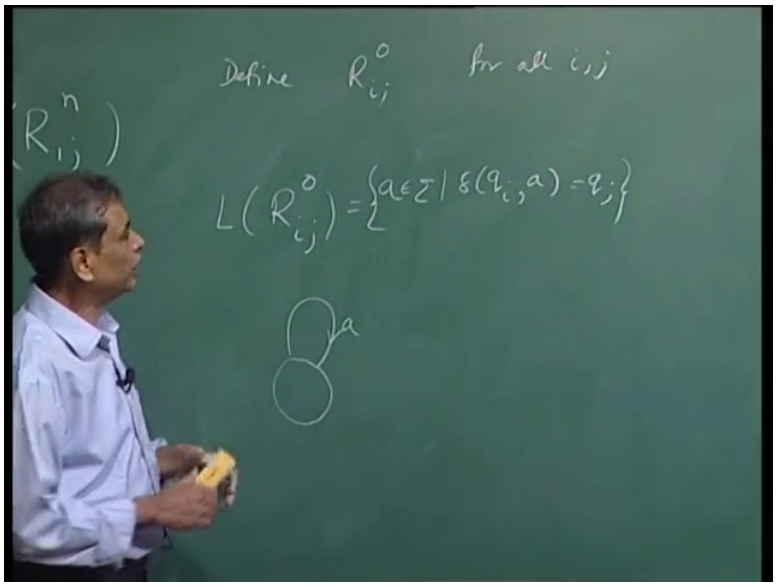


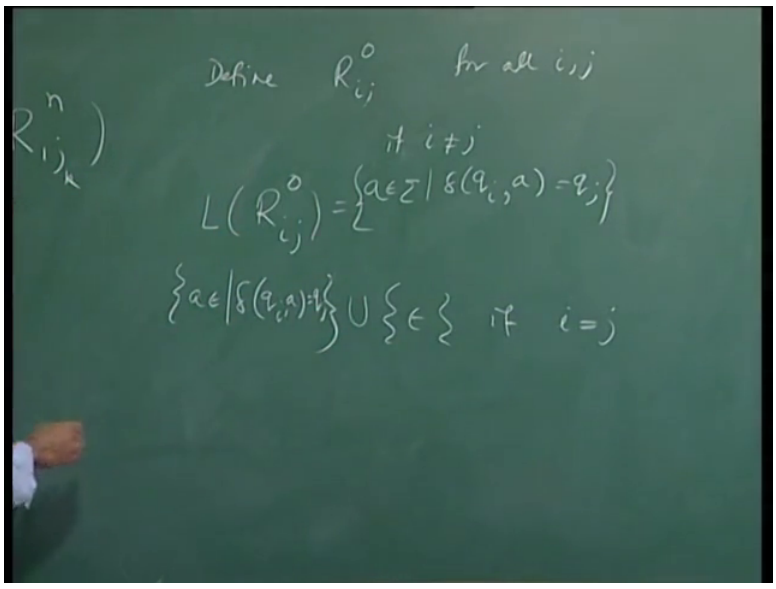
And this is where a very nice idea which will remind you of your dynamic programming algorithm that you have seen in algorithms courses comes into play. So basically what we are going to do is 1st of all define r ij 0 for all ij, then using these definitions we will build r ij 1 for all ij and so on. And then we will be able to see that will be able to build r i j with the superscript n, again for all ij during the previous one. So 1st of all let us see what does it mean to say this regular expression, what r ij 0 should be? According to us that this is a set of all strings which will take the machine from i to j without passing through any state whose number is greater than 0.

But every state has number greater than 0 which means what, such a string x cannot pass through any state at all, it can just go directly to from the state qi to state qj, that means what, that means these are really single symbols which takes the machine directly from state i to state j state qi to state qj and such a direct translation of course does not pass through any state. This is one case and there is another case that we will talk of right away. So let me just say this is the set of, in fact let me write it this way, it is the set of all a such that delta qi a is q. This is symbol in sigma. I am giving you the set rotation, now this will be a bunch of symbols, right.

(Refer Slide Time: 20:31)



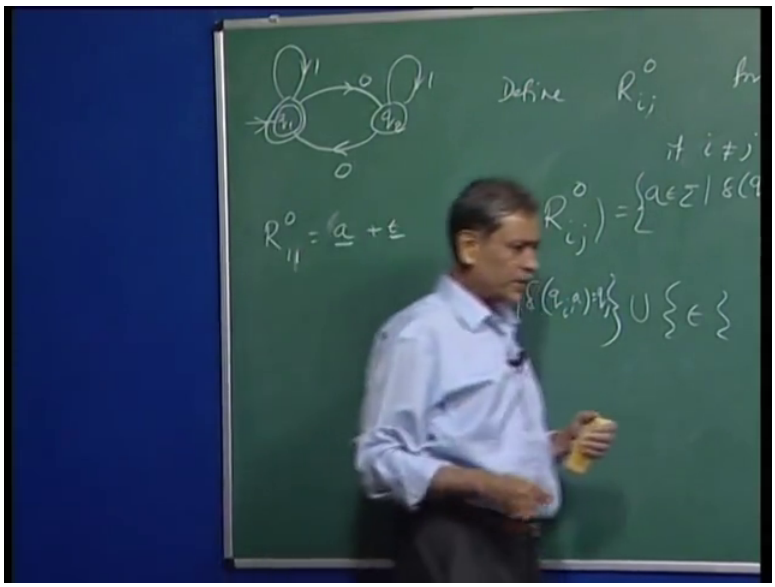


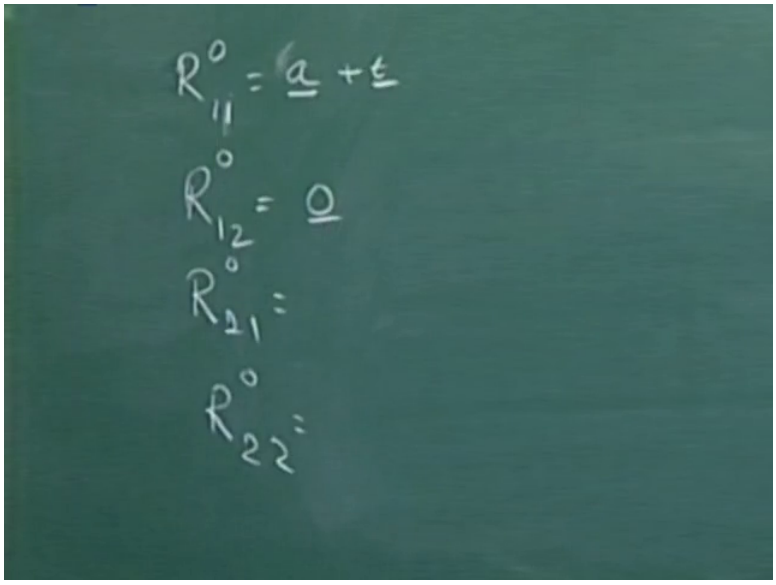


In fact if it is a deterministic machine, i to j, they can be at most from state, you know this is the situation qi come qj and this we are talking of all the symbols which take, which will take the machine from qi to qj. At this, noticed this transition is not passing through any intermediate state, so this is, okay this gets qualified to be such strings. Now think of a as a string, that string should be in the language denoted by r ij superscript 0 by definition. Now there is an extra thing that can happen when i and j is the same state, right. So of course this is this is there, that when i equal to j, in that case you see the string epsilon takes the machine from the same state to same state.

So union, i should say union epsilon and this union will be effective is qi or let me say i is equal to j. So we can actually, more correctly should say or a in sigma such that delta qi come a is qj, this set union, so we will add the epsilon is i is equal to j. There are 2 cases, if i not equal to j, this is the case, in that case it is only this, that epsilon will not come, otherwise epsilon also of course takes the machine from qi to qi, therefore epsilon should be there. And let me take a simple example just to illustrate this what we are saying.

(Refer Slide Time: 22:51)

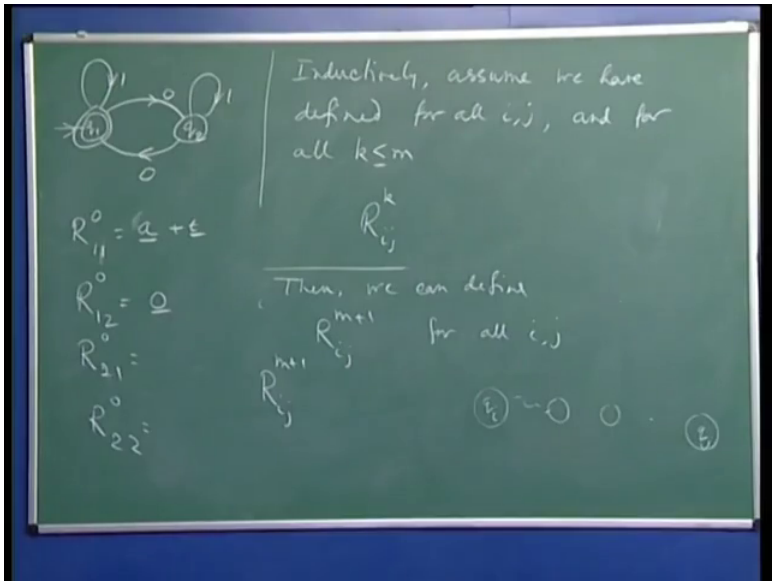


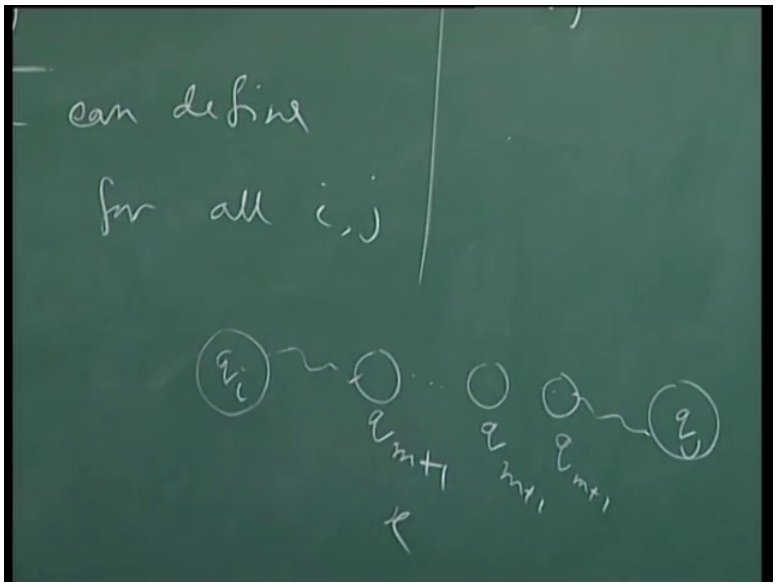


So take a very simple machine, in fact the 1st dfa that we have seen and recall that this dfa accepts the set of all binary strings, finite binary strings which each of which has even number of zeros. So let me number these states as one, q1 and q2, so what is, let me say what should have should be r 11, let us say r11 0. According to this, since both i and j is same, firstly is there a direct relation, indeed it is there and that is denoted by the regular expression a and epsilon of course takes the machine from here to there itself. So we will agree that this regular expression which stands for the set, a and epsilon, just the string a and the epsilon has 2 elements, this is a set of all strings which will take this machine from 1 to 1 without visiting any other state.

In fact here is, cannot visit 1 or 2 because their numbers are greater than 0. Right. So let us take another example if we wish. So what is r 12 0, it is basically all those direct transitions and that is exactly one. So this is given by the regular expression, so 0 which means the set of all strings which will take the machine from q1 to q2 without going through any step whatsoever, then that means there is a direct, we can only take the direct transition and there is only one symbol which can take the machine from q1 to q2. So therefore this way you can write, this is clear i can complete, so how many will be there for with this, such regular expressions will be there with superscript 0.

(Refer Slide Time: 26:11)





Of course r 11, r 12, r 21, r 22, so these are the at the base level these are the regular expressions that i build. And now let me inductively assume that i have built all regular expressions. So now i am using an inductive argument, inductively assume we have defined for all ij and for all ij and for all k, let us say less than equal to m, what i have defined, defined already r ij k. We assume inductively that these destinations are there with me, r ij with superscript k where k is less than equal to m. And in fact what i was trying to show you a little ago that you can start the induction with the base case where m is 0, right.

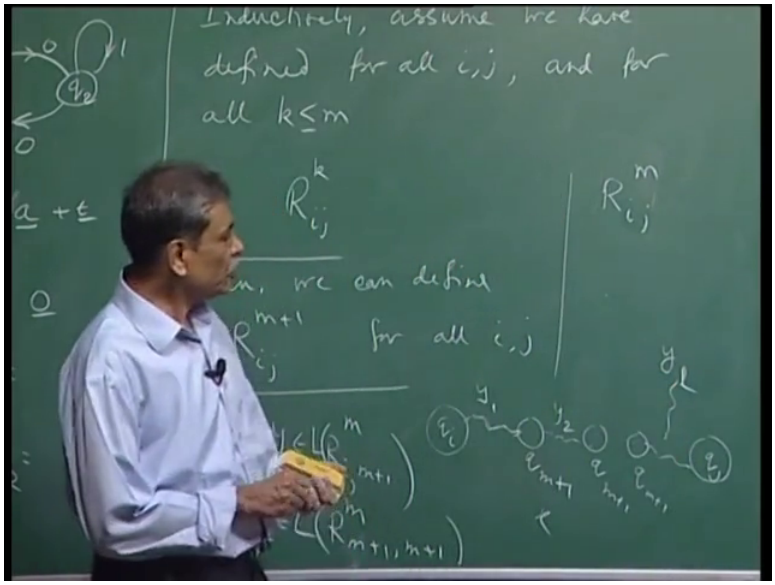
With this, then we can define r ij m +1 for all i come ij, right. So you had the definition of rij m and now you would like to define r ij m +1, okay. So you see, look at this, what is r ij m +1? Suppose i have a string which is in the language denoted by this regular expression, by

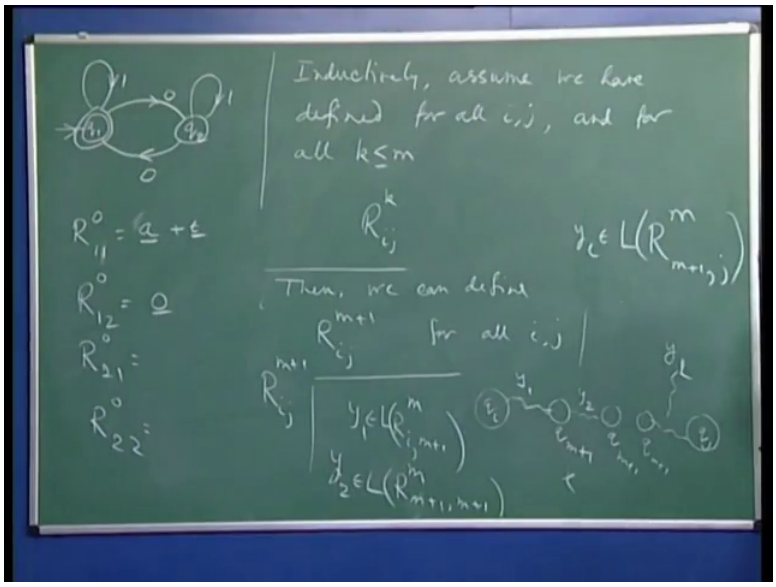
that what i mean, that that string take the machine from qi to qj and of course it can visit some states in between, okay. None of these states can be, they are indexed, they are super scripted this one can be higher than m +1. So what they are saying is assume x is a string in this which take the machine from qi to qj without passing through any state whose number is higher than m +1.

Now such a string can of course go from q, i mean take the machine from qi to qj without passing through m + 1 at all. So what are those things which take the machine from qi qj passing through all states passing through posted whose number is m + 1 or higher. That by definition is r ij m and now consider the more interesting case. Here is a string x which takes the machine from qi to qj and in this, the state q m +1 occurs number of times, one or more times. If it is occurs 0 number of times, then of course it is here and we are only talking of strings which take the machine from i to j without passing through any state higher than m + 1.

And now, so it is interesting case, let us take qm was here, so this is 1st m, it came to qm + 1 and so on and finally it came to again qm + 1 in general and then there is this part of the string does not pass through m + 1 at all. So in other words what we are doing is, we are marking out in the sequence of states the machine goes through in going from qi to qj on the string x, those states whose numbers are precisely q, i mean those states which are qm + 1. So such a string x can be what, is a string like this and must, you must realise, this is clear that since this entire string is in the language generated by r ij m + 1 and in this part, this is the 1st occurrence of the state qm + 1, so this string has to be from the regular expression which is ri m + 1 m, right because m + 1 does not occur and all other states are, their numbers are less than this m + m, m or less.

(Refer Slide Time: 32:13)



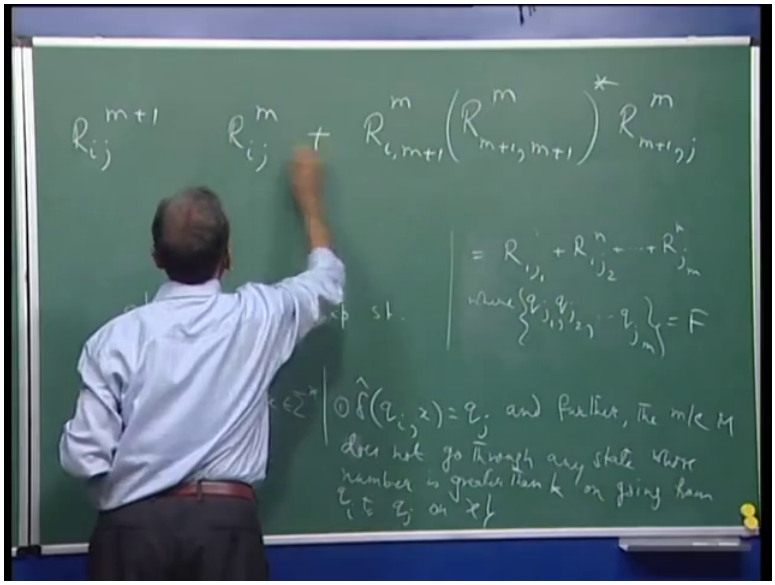


So this is a typical string, so let me say y1, y1 is to be in the language generated by this. And take a string like this, let me call it y2, what is y2, y2 takes the machine from state qm + 1 back to state qm + 1 without passing through any state whose number is higher than m. Right. So therefore can i say y2 is to be in the language r ij, sorry rm + 1 to m + 1, y2 takes the machine from m + 1 to m + 1, so r m + 1 m + 1 and superscript is m because it is not passing through any state higher than this. So then there are many such strings because the machine may go from qm + 1, qm + 1, qm + 1 and so on and then finally there is this last bit of these drinks, let me call it yl.

Now what does yl do? Yl takes the machine from qm + 1 to qj, again without passing through any state whose number is higher than m. So clearly yl will be in the language of rm + 1 j m.

It is not written correctly, i should say yl is a member of the language generally denoted by this regular expression.

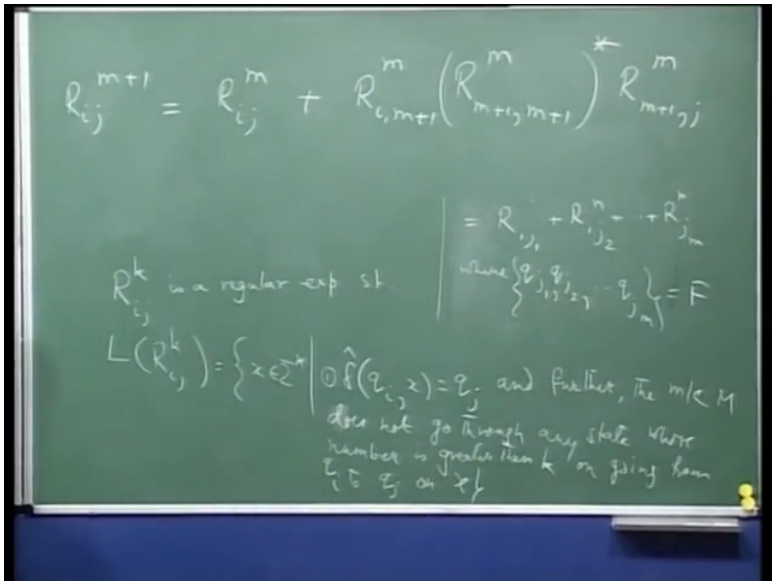
(Refer Slide Time: 34:30)

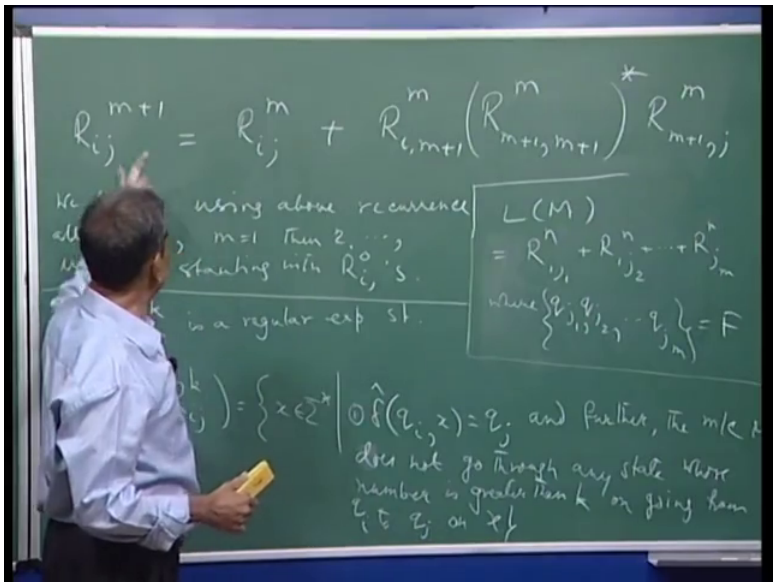


Now let me see now what is our r i j with superscript m + 1 should be. This regular expression can denote a language in which there are string which takes the machine from qi to qj without passing through any state higher than m. So those strings, that set of strings is denoted by r ij m or it can pass through, there we now need to talk of the 2nd case that we talked of, strings which take the machine from qi to qj but in doing so it does pass through this qm + 1. And we have seen such strings can be written as r i, 1st the machine goes from i to m + 1 without passing through any state higher than m, then you know it goes to this, goes to this, those will be the strings which takes the machine from m + 1 back to m + 1 without passing through any state whose number is higher than m.

And how many such pieces can be there? 0 or any number, so therefore i will put a star here, all such strings are okay for me and then finally after the last qm + 1 occurrence, there has to be a string which will take the machine from m + 1 to j, again without passing through any state higher than m. So these 2 regular expressions, their language is denoted, if you take their union, so this precisely the definition of r ij m + 1. Now notice what we have done, using the definition of various r ijs with superscript m, notice the superscript here is m, using that i have defined r ij m + 1. Right.

(Refer Slide Time: 37:32)





So in this example if we wish we can, before taking example let me complete the proof, at least say what the proof is now. So i see that inductively i can create, i can define all r ij starting from 0, then 1, then 2, then 3, then when i have r ij n for all r ijs, then of course i take those regular expressions whose strings take the machine from the initial state to one of the final states and i add them up, i put + in between and therefore i get the regular expression for the language accepted by the entire machine m. Clear. So what we are saying is, once more.

We define using above recurrence, all r ij m starting m equal to1, then 2, so on up to n, starting with the base case r ij 0s and i have told you r ij zeros are reasons to define, then use this recurrence to define r ij 1s, then r ij 2s, then r ij ns and once you have all the rij ns, then the language accepted by m, this machine is of course, we had explained little while back is

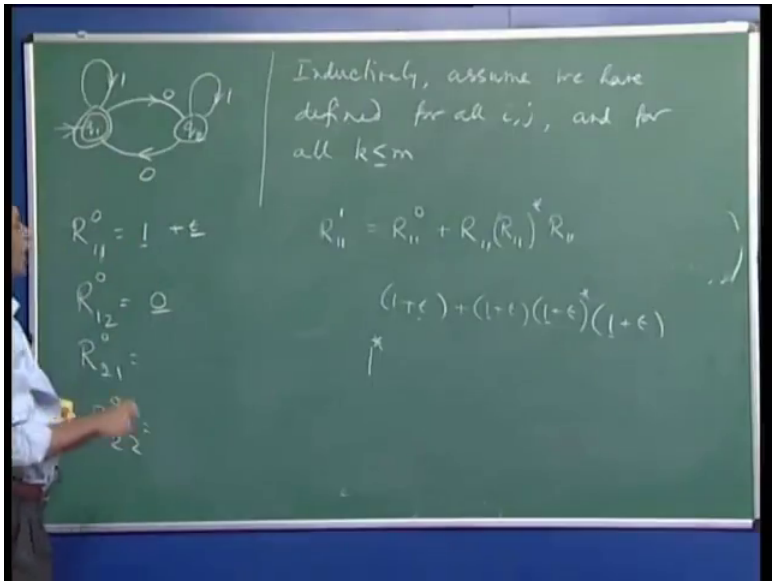
going to be the regular expression r ij 1 + r ij 2 + r ij m with superscript n, whereas these are the qj 1, qj 2, qj n constitute the set of final states. Therefore it is now clear to start with r ij zeros, then we use the recurrence to define r ij 1, r ij 2s, r ij 3s or when i get r ij ns, i will be able to write out what the regular expressions is which denotes the set of all strings which are accepted by the machine m.

So in other words what i have shown is that using this process, this algorithm in fact, what we have outlined is an algorithm, you can indeed build, you can indeed construct the regular expression given any dfa such that that regular expression is going to denote the language accepted by the machine. Now, 2 things i should tell you, one is that and the 2 things are, this can give you horrendously long bigger regular expressions. And you can see why, you see that in general as you go from you know r ij m to r ij m + 1, look at the size, so this is you are writing one regular expression here, another regular expression, another regular expression, of course then you are putting a star and then this.

So a regular expression with suffix m + 1 can be as large as 4 times the length of a regular expression with suffix m. So as you are going from lower subscript to higher subscript, the size of the regular expression that you are building can quadruple, right. And is so basically in the beginning of course this is some constant, each r ij 0 is some constant in length, but then next time it becomes 4 times of that constant, next time it becomes 4 times of the previous constant, so 16 times of the base case and so on. So therefore you can see that the expression that you are going to build in general can become exponentially long in terms of exponential in the number of states. This suffix is of course goes up to the number of states.

So that is point number 1 that regular expressions built through this process can be exponentially in length, exponential in length where the, where in terms of the states of the machine. The other thing is that these huge regular expressions can be brought down somewhat in size using some simple manipulations.

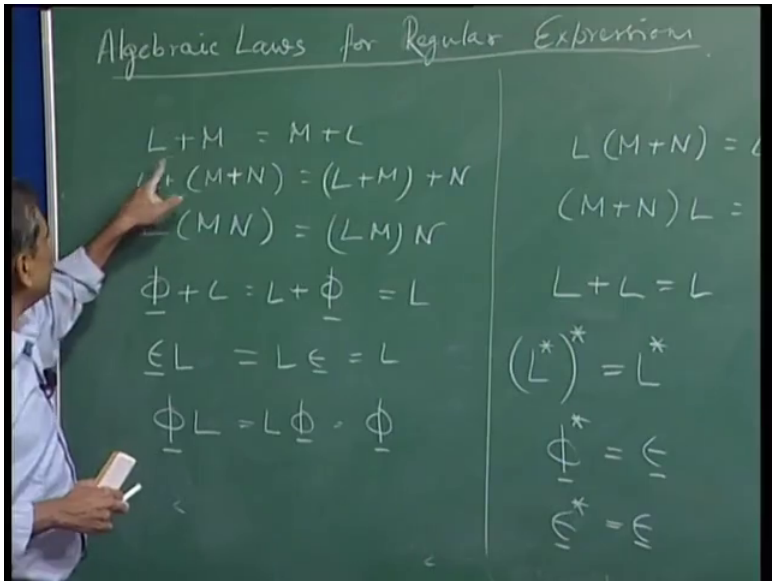
(Refer Slide Time: 43:50)

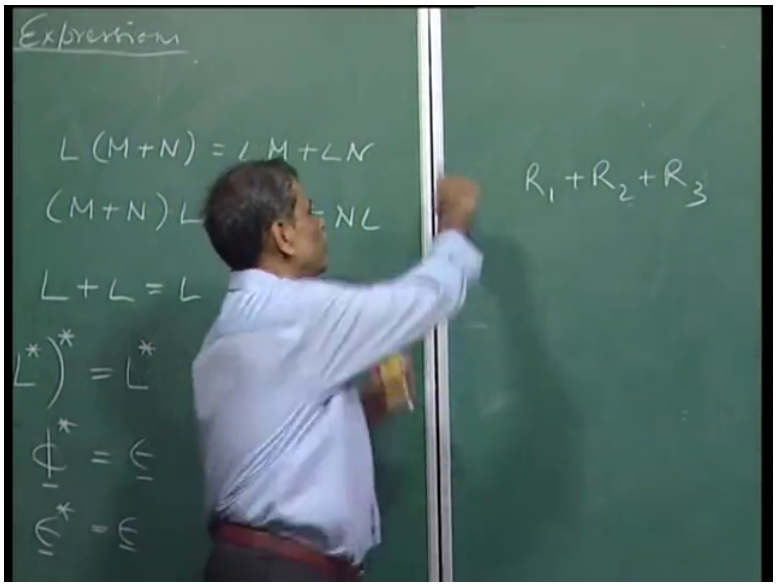


For example here, take this case we are talking of, so of course r 11 0 was, this is the direct 1 + epsilon. Now if i want to write out r 11 1, according to this, what do i need to do, i will write r 11 0, then r, now what is m + 1, this is 1, so again r 11 1 then r, remember m + 1 is 1, so r 11 whole star and then r 11, m + 1 is 1 and your j is also 1, right, so this is, this is what we will write. But if i, so in terms of this what is r 11 0, is of course 1 + epsilon, right and then 1 + epsilon, then 1 + epsilon star, then 1 + epsilon, okay. Now it is easy to see what we are saying that this is of course a long expression but do you see that this is actually denotes, you know much simply could have been written as simply 1 star, is not it.

R 11 which can go through only 1 and not this state q2 is basically any number of loops here, either 0 or more time. So this whole thing could be simplified, you know at least here using ad hoc arguments or using some algebraic laws of regular expressions to a very simple expression. So this whole huge thing is basically boils down to this simple regular expression. So how do we simplify regular expression, that maybe one question. Like this is a this is a case in point, you know you start using this, using our recurrence blindly i got this. But this can be massaged to this very simple thing. So for that one uses, one can use some laws which are very general. And i will talk about them now.

(Refer Slide Time: 46:59)





Here i have listed out some standard algebraic laws for regular expressions. What we mean by algebraic laws, you see for example when you say x + y whole square equal to x square + 2 xy + y square, that certain identity, algebraic coming out of from arithmetic. So here in the same manner what x and y were in that example, where they stood for any number, here this l and m in these for example, they stand for any regular expression. So what we are saying here. For example if you take the 1st one, we are saying, take any regular expression and another regular expression, if we put a + in between, in the regular expression that we get, that denotes the same language as the language denoted by taking, you know putting it the other way round as you can see.

So you have to understand this equality in the sense that the languages denoted by the left-hand side regular expression is same as the language denoted by the right-hand side regular expression and sometimes you may call this l and m, they are variables for regular expressions in the sense they, you can plug-in any regular expression for l, any regular expression for m, then you will get some equality. What you are saying is that the left-hand side regular expression denotes the same language as the right-hand side regular expression on plugging in any regular expression into l or m.

You know, for example the 1st one, it is coming out of the fact, what is +, + this basically union, right. + is union of the 2 languages denoted and we are saying, as you know union operation is commutative, right. And this is associativity of union, now therefore we do not have to put brackets, right, the way we do not have to put brackets to say do i 1st add 3 and 5 and then add come to the result i add 5 to we know that this is same as 5 + 3 + 5, by this what we mean is the order of these operations, these are binary operations, we could have done only 2 at a time, that order is not important. 1st you guys do this, then do this or evaluate this and then do this, that does not matter.

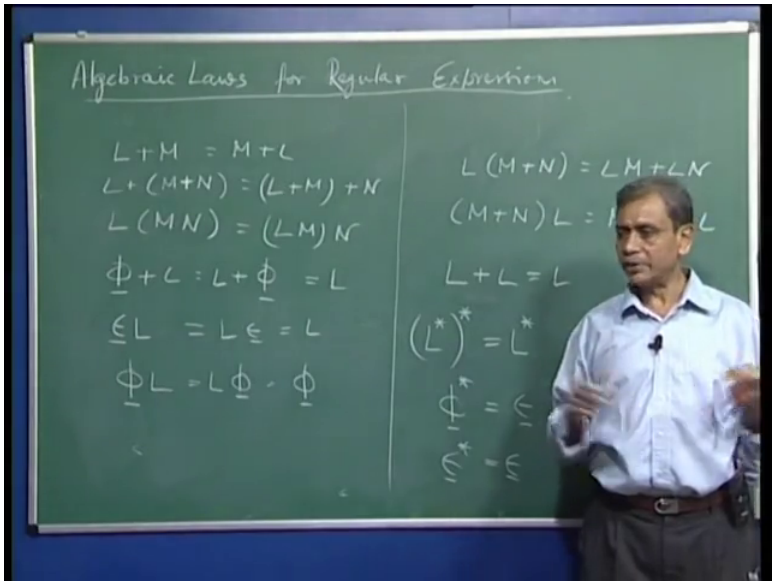
So that is why we just write for regular expression we may just right, you know r1 + r2 + r3, in fact these kinds of things will be very common, right. You do not need to put brackets, this is precisely saying that same thing that concatenation, the dot is actually, is also associative, right. Order of which manner i did the concatenation which 2 of the regular expression languages i 1st concatenated, that is unimportant. Now this is saying and this says another point here. In an algebraic law you can also have constant.

So for regular expression we have 2 constants phi and epsilon, you might have said what about symbols, you see symbols would have come, would have needed, for that i needed to assume some alphabets. See all these things are independent of alphabets, whatever be the alphabets, these laws will be true. So what we are saying, this is saying something very trivial that if you take the union of empty set with any set, you get back that set, right. This is saying that if you take the union of the language, language consisting of the empty string, not union concatenation of that language with this, you know get back the same thing, that is clear.

And this is an interesting thing, this is saying that if you concatenate the empty language with any language, remember empty language is a language which has no string at all as opposed to this language, this is a language one thing and that is the empty string, you will get back

the empty language. And that is not too difficult to see if you just use the definition of concatenation.

(Refer Slide Time: 52:13)



This is saying that concatenation distributes over addition or union if you like, these 2 are same things. And this is again coming out of, this is the law, this is a law, it is showing that union is, or using the fact that union is idempotent, l union l is of course l, so we write l + l is l. And this is also something interesting that if you take l star, if you try closure one more and more and more on the same thing, you do not get back anything new. So l star and whole of that star is l star. And what is this phi star, the empty language, if you take its closure, then you get the language with the empty string.

That is because closure necessarily, closure of any language has to have epsilon, it cannot have anything else and therefore you get this, epsilon star is epsilon. By using some of these algebraic laws you may be able to simplify a regular expression. How do you prove this, well you can prove, you know argue from the 1st principles, for example in this case using the fact that we are talking of the union of 2 languages and in this is one of the ways of doing it and as we said that these algebraic laws used, they are used sometimes simplifies a regular expression. So this completes our discussion on the regular expressions and we have a very important, several couple of topics left which are on regular languages which we will cover in subsequent lectures.