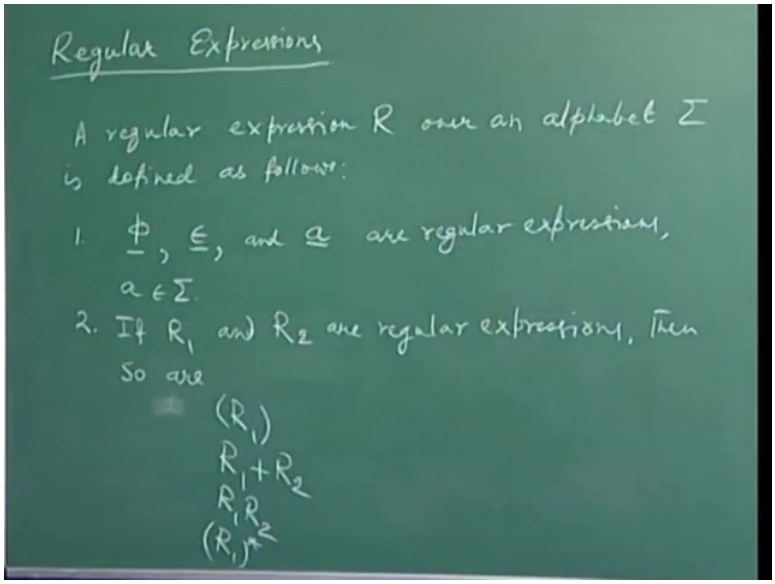
Theory of Computation.
Professor Somenath Biswas.
Department of computer Science & Engineering.
Indian Institute of Technology, Kanpur.
Lecture-11.
Regular Expressions, they denote Regular Languages.

We have seen 3 distinct ways of describing Regular languages finitely. And these are either we can use a DFA to describe a regular language or they can use an NFA to describe the same language as well as we could have used an NFA with epsilon transitions in general to describe the same language. And all these 3 are finite ways of describing in general what are infinite sets.

(Refer Slide Time: 1:01)



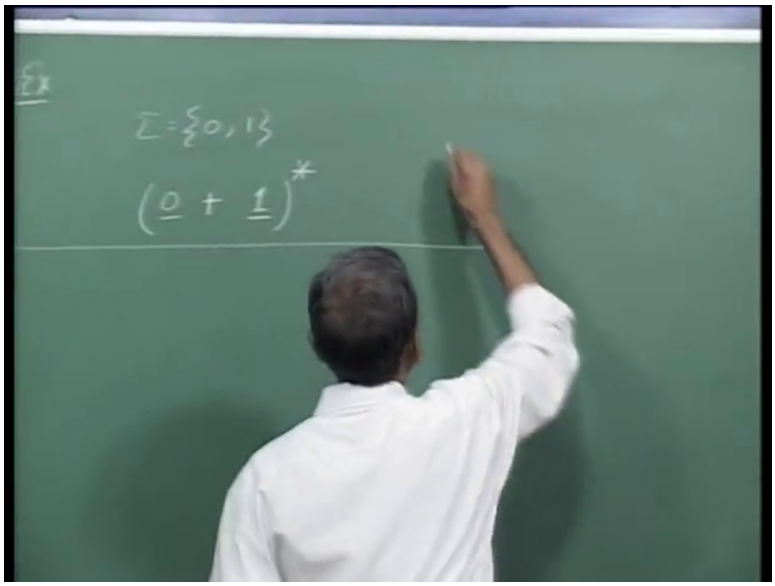
Now today we will find yet another way of describing the same class of languages, that is regular languages. And this will be done through something which are known as the regular expressions. So 1st of all I will define what are regular expressions. Now we say a regular expression R and I will tell you what this R is inductively. But 1st of all we fix an alphabet to let say this is Sigma over and alphabet Sigma. So we are now describing regular expressions over an alphabet Sigma, is defined as follows. 1st of all we have a base case, we say phi, epsilon and a are regular expressions, where a is a symbol over Sigma, a is a symbol in the alphabet Sigma.

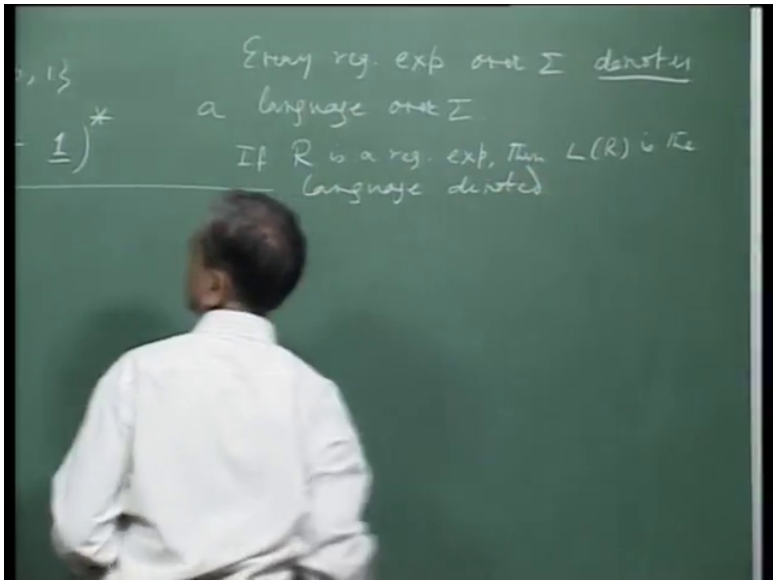
Now certain notational remarks, as we will see every regular expression denotes a set of strings, so this as a regular expression, although we are writing the same symbol a and here

also we are writing a, this is a symbol in the alphabet, whereas this is a regular expression over the alphabet Sigma. And to distinguish these 2 uses of the same symbol a we put an underline here and typically in textbooks this will be written in bold. So you see this underline bold Phi, bold epsilon and bold a are regular expressions, what they mean, what they denote, that I will explain a little letter.

Then let me, 1st let me tell you what are these expressions syntactically. 2nd, our 2nd rule is if R1 and R2 are regular expressions, then so are, we can put R1 regular expression in a bracket, this is the 1st let me write it as a without anything to that we do not get confused, the other will be, so are this R1 + R2, then R1 R2 and finally R1 star. Right. And we have, you know we say a regular expression over R is a expression which is obtained by using finitely many times these rules. So basically by saying this what we are ruling out, that infinite string as a regular expression. So one point is all regular expressions are finite strings and they will look like, they can be constructed in this manner using these rules.

(Refer Slide Time: 6:35)

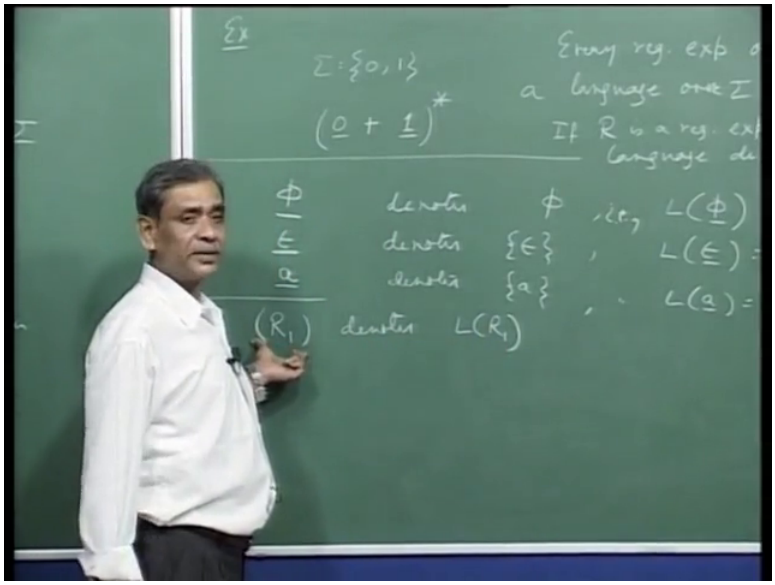




So let me just give you an example 1st. We can say, suppose our alphabet is 0, 1, right, then this clearly is the regular expression because 0 is the symbol in Sigma and therefore this is a regular expression. And another regular expression is this and now what I could have done is for example I could have put + in between, this is again a regular expression by this rule because 2 regular expressions, in between if I put a + sign, that means it is a regular expression. And then, since this whole thing is a regular expression, therefore this is also a regular expression.

This is one simple example of making use of these rules to obtain a regular expression. Now, syntactically these are the kinds of things, these are the expressions which we call regular expressions, now one point which we have to keep in mind, so this was an example and point is every regular expression over Sigma denotes a language over Sigma. And let us use this notation, that if R is a regular expression, then LR is the language denoted.

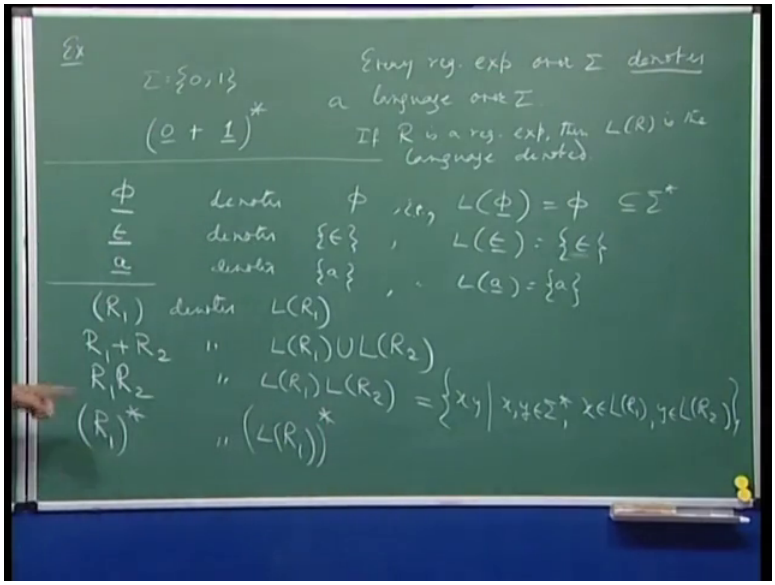
(Refer Slide Time: 8:48)

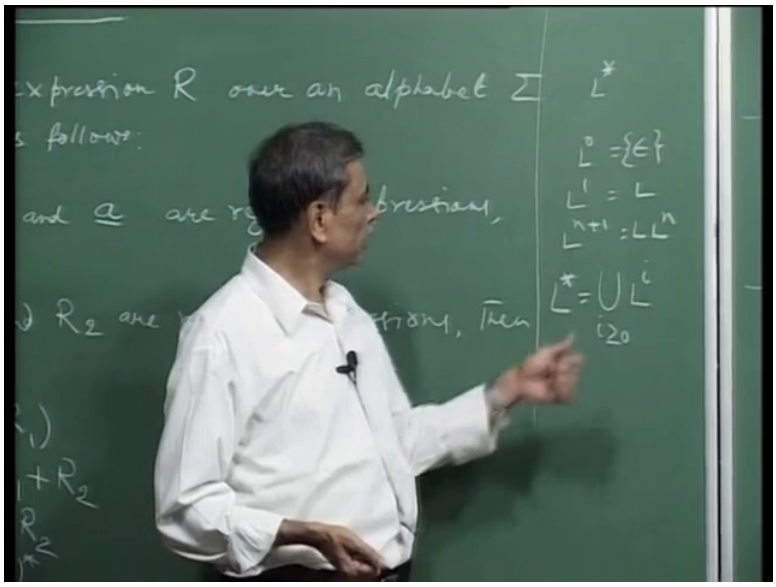


Now let me describe for the regular expressions what are the languages denoted by regular expressions and we will again do it inductively. So let me write for the 1st this is the base case, so we write denotes the empty language phi. Now what does it mean? See, we say, we can, if we use this notation, in other words I can write that is, our Sigma was the alphabet, so what we are saying is L of the regular expression, the language denoted by this is, now clearly phi is a subset of Sigma star and therefore it is a language over Sigma and this regular expression denotes that language, which is the empty language.

Similarly denotes the language, similarly in other words. So what is this, this regular expression denotes the language which has 1 string and that is the empty string let us have it. A denotes, again this is a language which has this one element and that element is the symbol which was used to obtain the regular expression. So like this, that is, similarly L of a is. This completes the base case rule 1 and now for the other cases, of course this one just leaves, brackets are used for convenience, so we can simply say R1 denotes the same language as, so R1 is a regular expression, if you put you know brackets around it, the language denoted does not change.

(Refer Slide Time: 11:23)





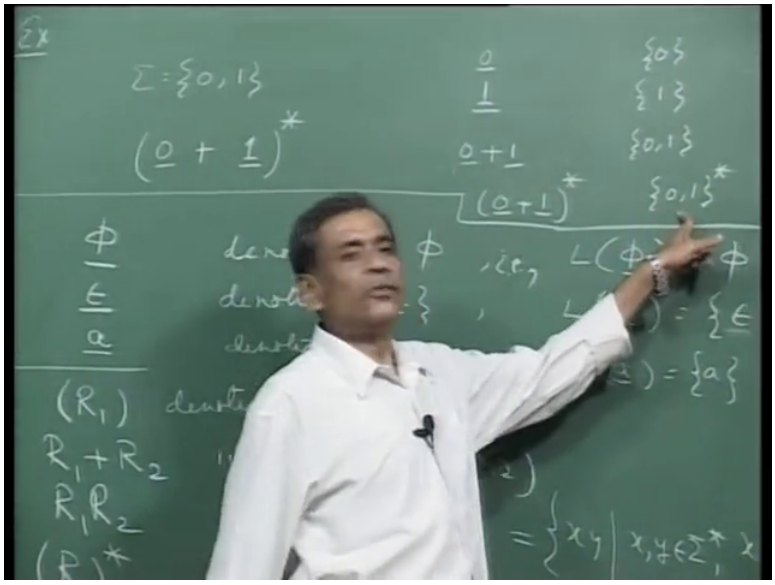
Now I can say R1 + R2 denotes, this is a nontrivial case, what we are saying is that if I take 2 regular expressions R1 and R2, put a + in between, then this expression denotes the union of the 2 languages denoted by the 2 arguments here. Then R1 R2 denotes the concatenation of these 2 languages. Okay. Which of course you know, this language is seen as all those xy such that x and y are in Sigma star and x is in L R1 and y is in L R2, right. This was our old concatenation of 2 languages, all I am saying the regular expression R1 R2 denotes the concatenation of the 2 languages L R1 and LR 2.

That means everything else, any string in this language denoted by this expression, there is a way of breaking it up so that the 1st part is in the language of R1, 2nd one is in the language of R2. And then finally this, this again denotes the language LR 1 star. It should remind

ourselves what the star was, the star was the Kleene closure or simply closure and just to remember what, if I have a language L, then L Star what, this is defined in this manner, 1st of all I use the notion L0 is epsilon, right, language which has only the string epsilon.

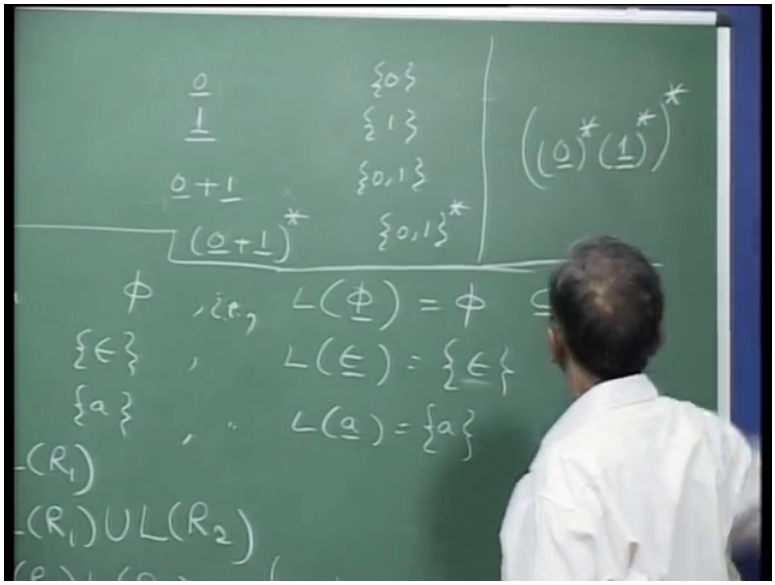
L1 is of course the language L itself, L N +1 is L concatenated with LN. Now I am giving it recursively what it is and then L Star is simply the union of all these Li. And intuitively, when do I have a string in L Star, if I can break up that string in a number of, if we can see a string as a concatenation of several strings, each of which is in language L, then such a thing is, such string will be in L Star, right. So if this is our definition, this is our understanding of the sets denoted by these expressions, for the simple case can we figure out what is the language denoted.

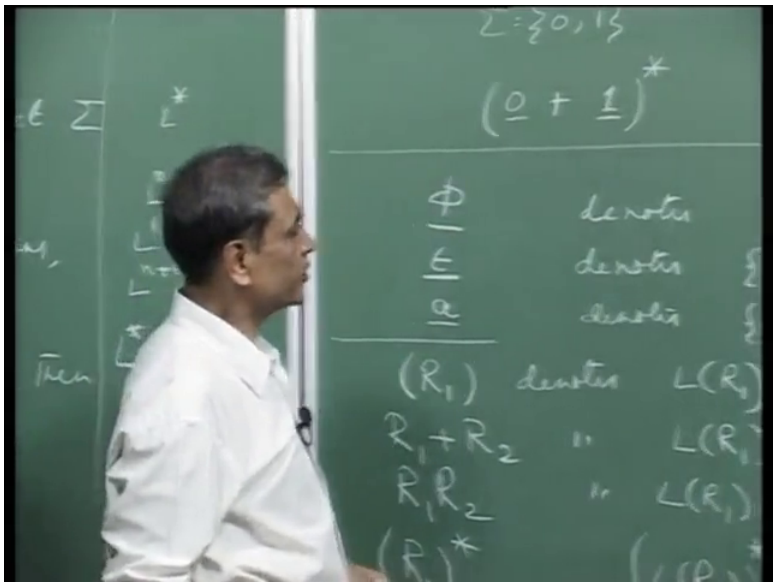
(Refer Slide Time: 15:47)



So let us use our definition here, we will do it the same way as we did here, we will go from inside to out. So of course 0, this denotes the language with just the string 0, 1 denotes the language 1, we are just applying these base cases. Now 0 + 1, what it will denotes, it will denote the union of the 2 denoted languages. So what is the union of these 2, clearly, it is the set which is 2 elements, one is 0 and the other is 1. And what is therefore 0 + 1 star, this is simply, this is our old familiar set of all binary strings, set of all finite binary strings and this is the regular expression. Right.

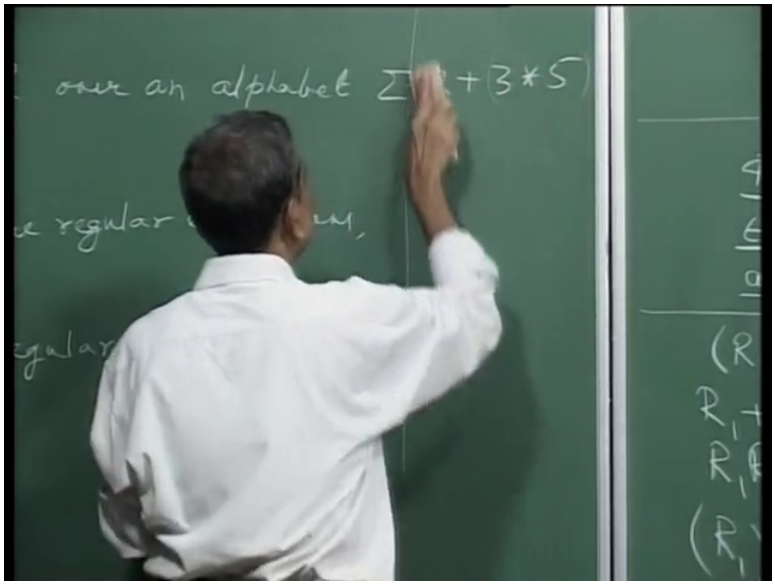
(Refer Slide Time: 17:34)

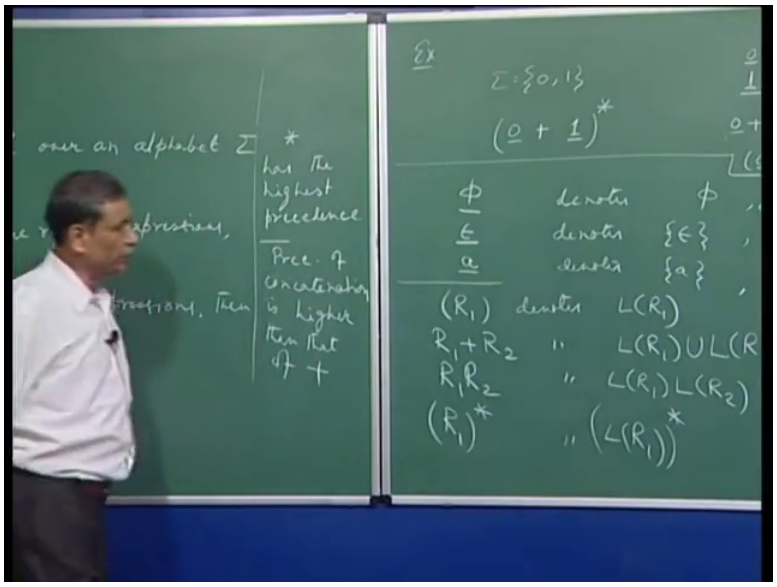


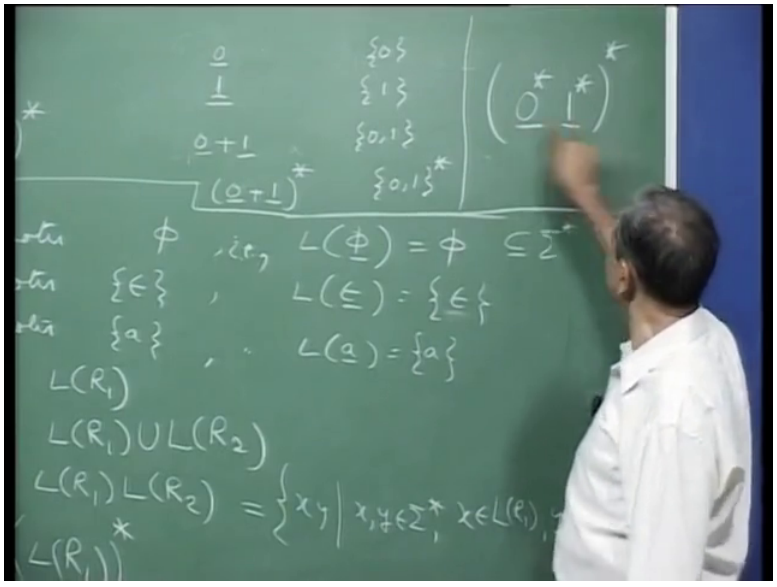


Before we proceed, we should realise one thing that it is possible to have several different regular expressions denoting the same language. Okay. For example, if you think a little bit, you will see that this regular expression, also denotes this same language, that is a set of all finite binary strings. Though this looks very, as an expression it looks very different from this one but it is also denoting the same set of strings. Let me now give you a number of examples so that our idea about the regular expressions and what they denote gets clear.

(Refer Slide Time: 19:08)



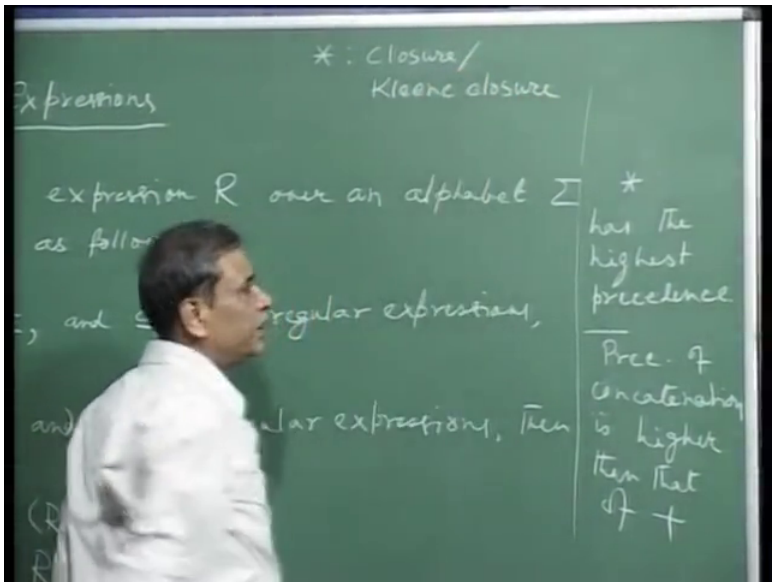


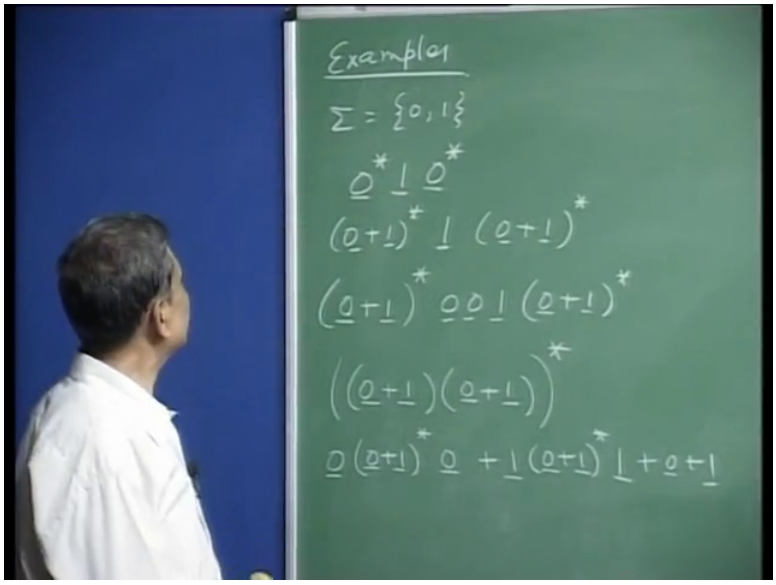


Before I give you this some examples, more examples, we had already seen 2 with simple examples, there is a way of writing these things, writing regular expressions so that we do not have to put too many brackets. And that is as we use a normal arithmetic, for example if you say 2 + 3 star 5, everybody will say that this means 17 and not 2 + 3, 5, into 5, 25, why, what is our, actually we see this, if we put brackets, then we could have put like this and then another bracket but that is not necessary because we use the convention that this operator, this binary operation has higher precedence over the other one which is +.

So let me just say this therefore that this has the highest precedence and concatenation, precedence of concatenation, namely this one is higher than that of + which is of course union. Now using that notion of precedence, this I could have written as, you see it is clear, this star, it immediately bind these 2, the one which is immediate, namely this, similarly this star immediately binds to this 1 and not this whole thing. And therefore it is 0 star concatenated with 1 star and then we are taking the star.

(Refer Slide Time: 21:23)

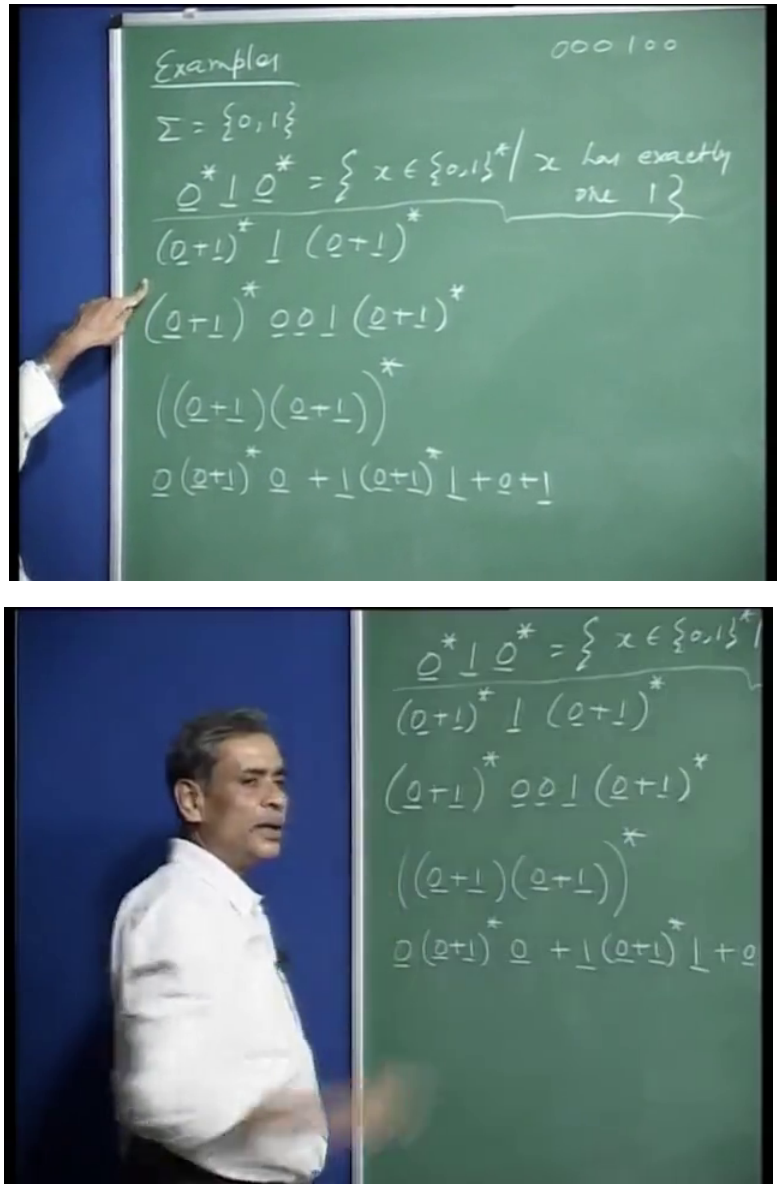




I should mention this, this operator is called either closure or Kleene closure, Kleene is one of the founders in this area who was a logician and done a lot of work to base the foundations of theory of computation. Now we will give examples. I have written a number of examples and you can see each one of them there, each one of them is well formed using those rules of construction of regular expressions, that is clear. And also I am using the precedence which I explained, that closure has the highest precedence followed by concatenation, followed by +, union.

So let us understand each of these expressions and what they are denoting, the sets they are denoting. The alphabet here is binary 0 and 1 and this is saying, if you see what it is saying, that you can have any number of zeros followed by 1 followed by any number of zeros. Right, this is what it is saying, 0 star means 0 or more number of zeros. Clearly this means what, this means that the set of all binary strings which has exactly one occurrence of 1, one occurrence, the symbol 1 occurrence in the string exactly once, no more, no less.

(Refer Slide Time: 23:59)



Such a thing, such a string may have only 1 and nothing else and that is allowed because you see this goes to epsilon, you know you take that epsilon is a set of all zeros, 0 or more number of zeros, in particular epsilon is there in this language, here also epsilon is there, concatenated epsilon 1 epsilon get 1. So just a single 1 will do, on the other hand if I take something like 0001000, clearly this can be, this will be in the language 0 star and this also will be in the language 0 star and therefore it is an expression, not too difficult to see that this is precisely all those binary strings x has exactly one 1.

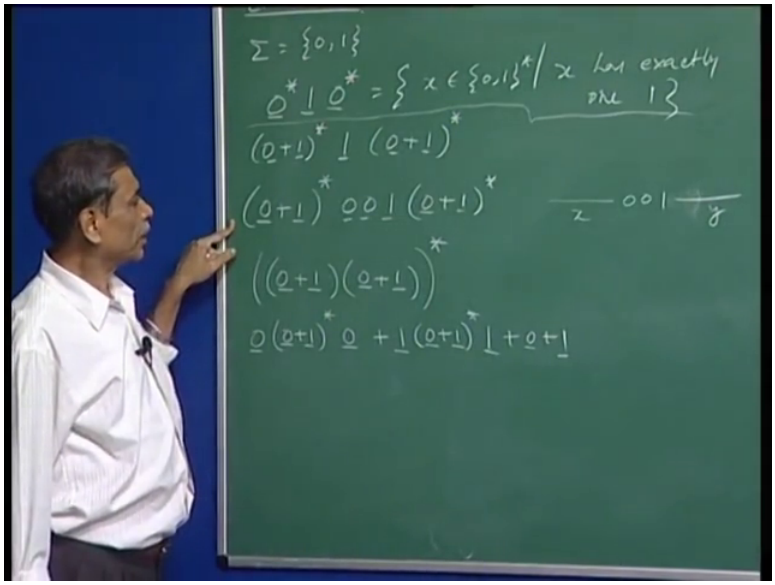
What about this, now again, you see, remember this, this is our old friend which denoted the set of all binary strings, finite binary strings, any string over alphabet 0, 1. So you are saying take any string over alphabet 0, 1 concatenated with 1 and follow it by another string. So

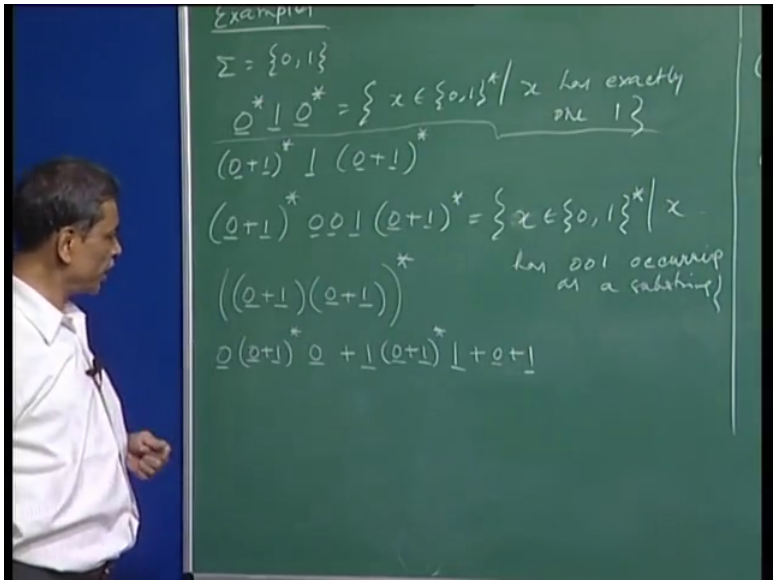
what you are guaranteeing, you are guaranteeing that the strings denoted by this set will have at least one 1, right. It cannot have only zeros, why, because of course you can produce only zeros through this through this but then we are contributing this and this and this, so one occurrence of 1 has to be there.

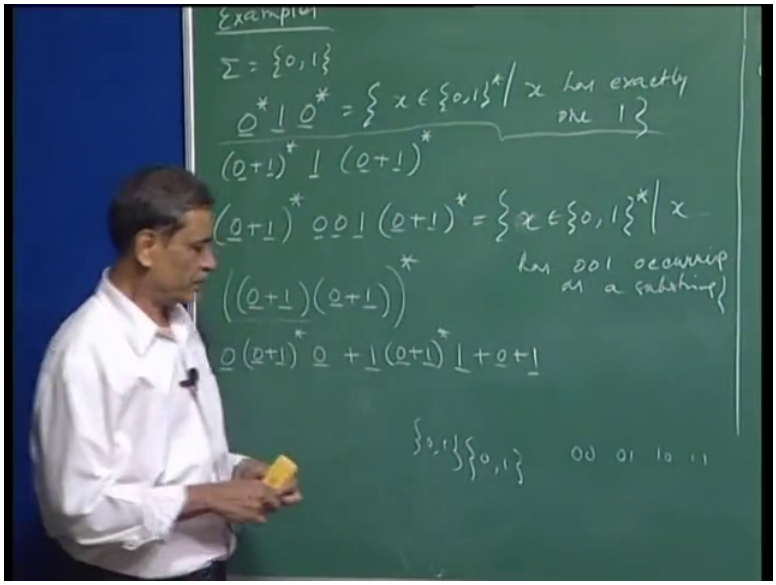
And why at least one 1, you see there can be other 1s also coming out of this and coming out of this. So therefore this regular expression denotes the set of all finite binary strings in which there is at least one 1. What about this, now actually regular expressions are very easy ways of describing that something happens in the string. So let us look at this, it is a slight generalisation of the old thing, what we are saying is that any string in this, in the language denoted by this regular expression, it is clear that it must have the substring 001.

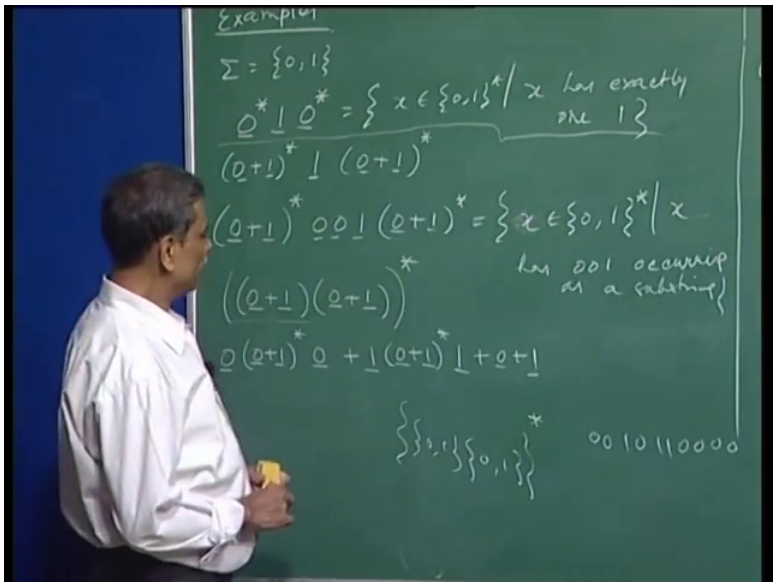
Why, because how do you get a string in this language denoted by the regular expression, by taking a string from this language, taking a string from this language and putting 001 in between. So 001 must occur and if we have any string, the other way what I have trying to say is that any binary string which has an occurrence, one or more crisis, but at least one occurrence of 001, such a string will be in the language denoted by the regular expression because such a string is of the form x 001 followed by y, right so this part is y. X get, we can imagine, we can think of x coming out of this part, y coming out of this part and there will be a concatenating these 3.

(Refer Slide Time: 27:25)




Example
$\Sigma = \{0,1\}$
$0^*10^* = \{x \in \{0,1\}^* \mid x$ has exactly one $1\}$
$(0+1)^*1(0+1)^*$
$(0+1)^*00(0+1)^* = \{x \in \{0,1\}^* \mid x$
$((0+1)(0+1))^*$ has 001 occurring as a substring$\}$
$0(0+1)^*0 + 1(0+1)^*1 + 0 + 1$

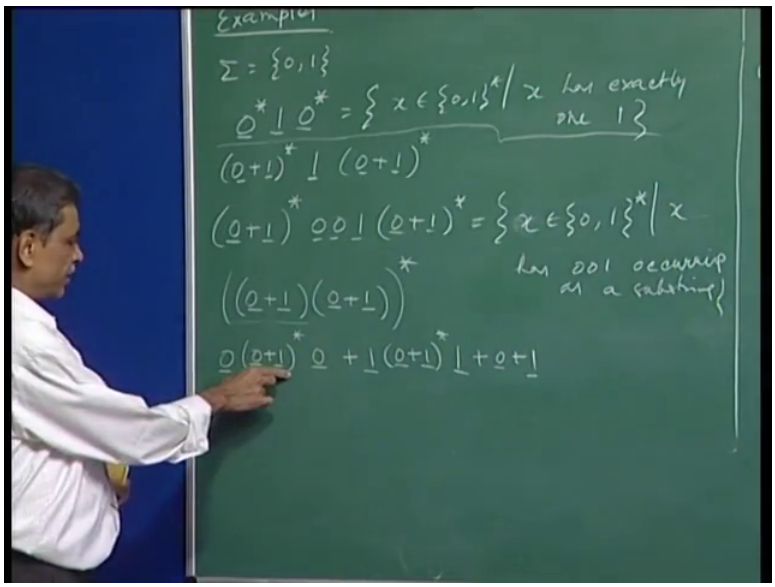

Example
$\Sigma = \{0,1\}$
$0^*10^* = \{x \in \{0,1\}^* \mid x$ has exactly one $1\}$
$(0+1)^*1(0+1)^*$
$(0+1)^*00(0+1)^* = \{x \in \{0,1\}^* \mid x$
$((0+1)(0+1))^*$ has 001 occurring as a substring$\}$
$0(0+1)^*0 + 1(0+1)^*1 + 0 + 1$
$\{0,1\}\{0,1\}$ 00 01 10 11

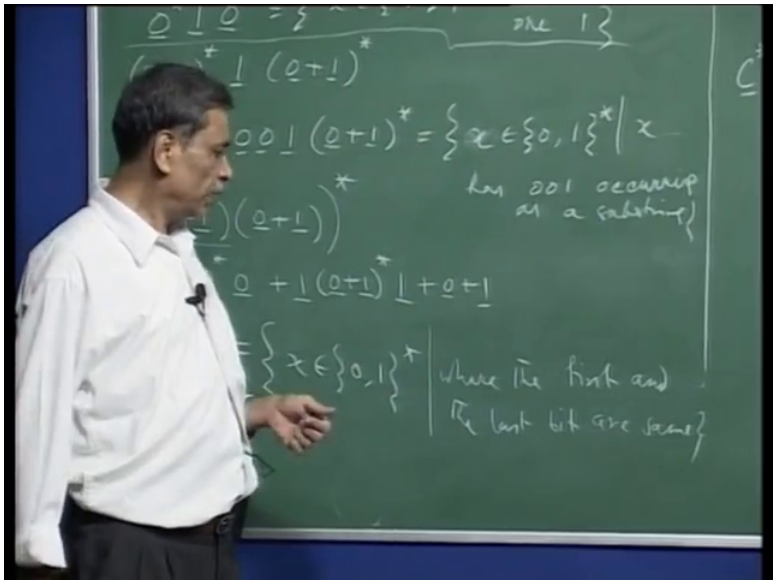

Example
$\Sigma = \{0,1\}$
$0^*10^* = \{x \in \{0,1\}^* \mid x$ has exactly one $1\}$
$(0+1)^*1(0+1)^*$
$(0+1)^*00(0+1)^* = \{x \in \{0,1\}^* \mid x$
$((0+1)(0+1))^*$ has 001 occurring as a substring$\}$
$0(0+1)^*0 + 1(0+1)^*1 + 0 + 1$
$\{\{0,1\}\{0,1\}\}^*$ 00 10 1100 00

So both ways it is true that the language generated by, denoted by this regular expression, every string in that language will have 001 and any string in which 001 is the substring will be in the language denoted by this. So I can write in this way, is the set of such that x has 001 occurring as a substring. Alright. Now this is also not too difficult to see what the language denoted. You see what is this, this is saying, I take, this particular part of the expression will denote a language which is precisely 0 or 1. So basically either 0 or 1, this just denotes a language with 2 strings which are the 2 bits and you are concatenating with this.

So when you concatenate what you are going to get, you are going to get 00, 01, 10, 11, these 4 strings you will get and now you are taking the closure. What would it mean? That any string in which, which can be broken up in this, this, this or this, anyways, that is you know something like I can say 00, 10 11, 00, 00, right, this would come the spring surely would come from the closure of this language which is the concatenation of just these simple 1 bits. And in effect, what is this string doing therefore, it is denoting all strings, all binary strings, which has given number of symbols. Right. Because you know you can see that happening.

(Refer Slide Time: 30:10)

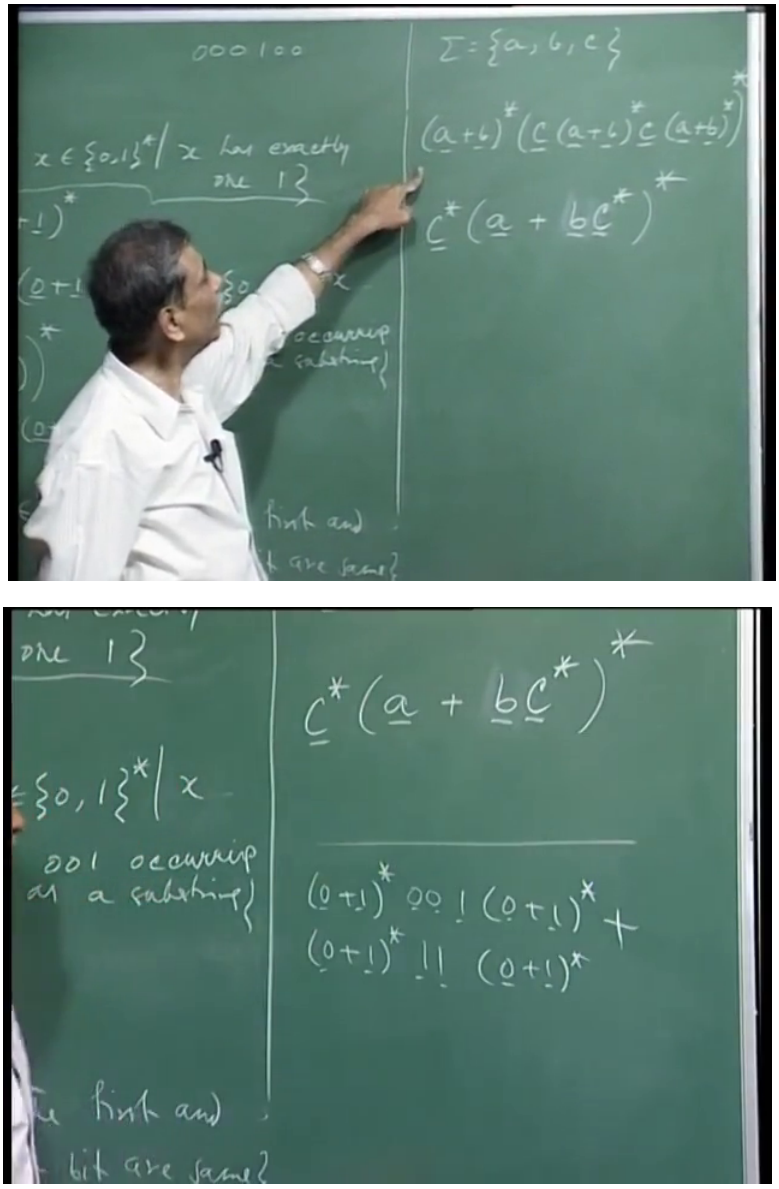




So this is all strings with even number of symbols, all binary strings with even number of symbols and final example here is what, this is any string, this denotes any binary string, similarly this denotes any binary string, now what we are saying, this part is saying what, string which comes out, which is denoted by this regular expression, is in the language denoted by this regular expression must begin with 0 and end with 0. And similarly this one must begin with 1 and end with 1. And then we are saying or it can be a single 0 or it can be a single 1.

So in effect what we are saying is that all that I can compress in a simple sentence that it is the set of all binary strings where the 1st and the last bit is same. If the 1st bit is 0, the last bit also has to be 0, if the 1st bit is 1, the last bit also has to be 1 and of course the single bit string which is 0 or 1, that satisfies the same property because that bit is both the 1st and the last. So I can say this way that this denotes the language where the 1st and the last bits are same. Can we generate Epsilon, can Epsilon come out of this, not really because the smallest thing that this can denote is 00, the smallest thing this can denote is 1 1, this is of course 0 and 1.

(Refer Slide Time: 32:50)



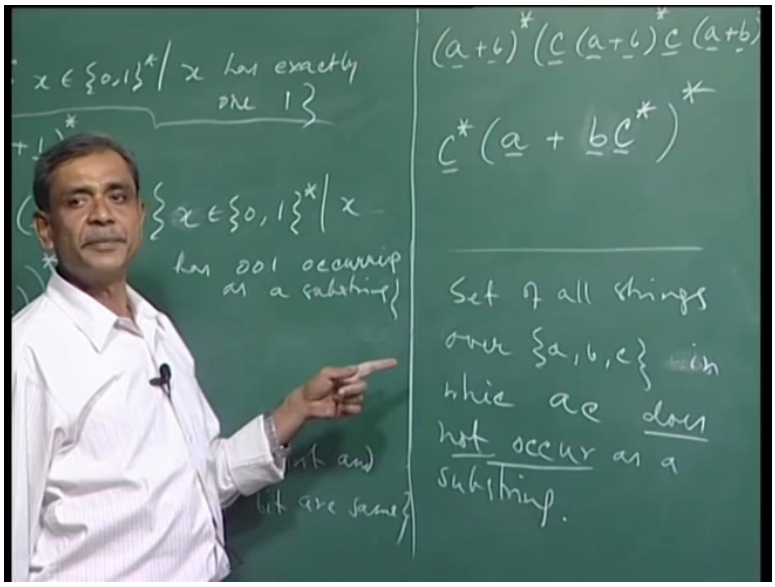
Now it makes sense because when you talk of the 1st and the last bit, they should exist and therefore this is correct way of denoting this language through this regular expression. And our next example is from an alphabet which is a, b, c. So let us take this, the 1st one, what is it saying, by now you know this just means any string over a and b, similarly I have one more such same component, so any string over a and b concatenated with a c followed by any stream over a and b concatenated with c, this and then it can repeat any number of times, 0 or more times.

So do you see that language denoted by this is the set of all strings over a, b, c where the number of c is even, the total number of cs in the string will be even. And by the way 0 is an even number and that is allowed because the string does not have any c in it can also be in

this language because you are taking the star of this whole thing, so in particular it can generate, it can, it does have Epsilon and Epsilon you add with this, will give you, you will get the spring with no c at all. Our final example here is possibly a little more advanced.

Can you see whereas it is very easy through regular languages to say up of strings which has some, this substring or that substring, right, for example if I wanted set of all binary strings with either have 001 or 11 as a sub string, what I would have done, I would have simply done 0 + 1 Star +1 star, this will generate a string with 001 as a substring. Of course I should also take care of 11 substring, so that is this and I put a + in between and of course I should do all this so that we are not confused. And this expression again clearly see denotes the set of all binary strings where each of which either has 10, either has 001 occurring as a substring or 11 occurring as a substring.

(Refer Slide Time: 36:00)

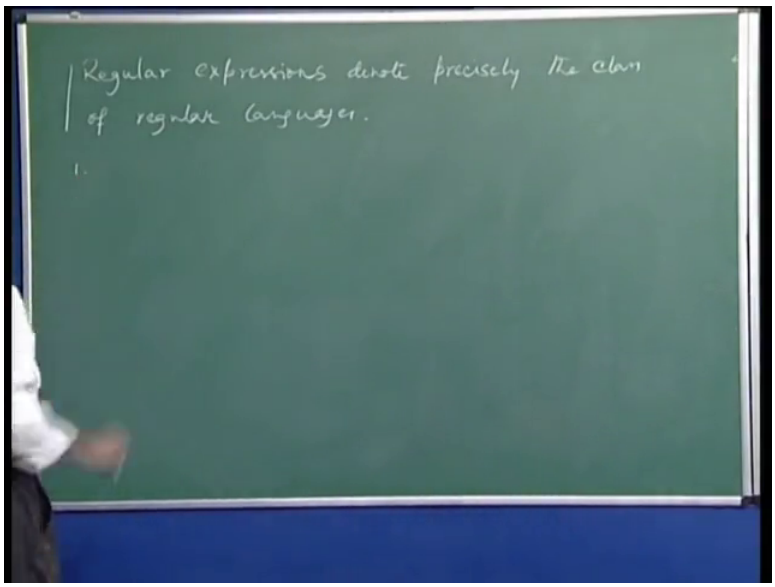


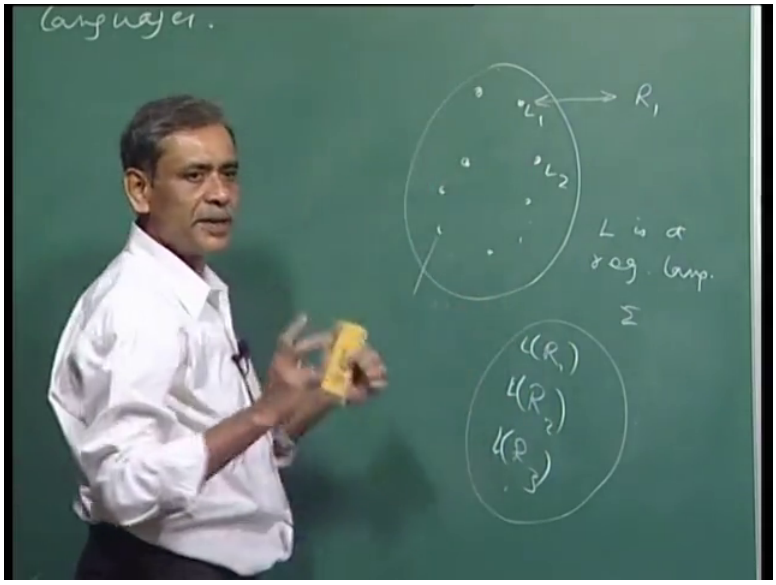
So these kinds of things are very very easy to do it. Right. But how do I specify through my device of regular language, regular expressions, the set of all strings in over a, b, c, let me write it, set of all strings over a, b, c in which the substring ac does not occur, does not occur. What I said little while back was that when I want stop strings in which some substrings or number of substrings occur, that is easy, but what about this? And contrast this with the DFA equation. In case of DFA, if I had a DFA which accepts all strings in which ac occurs as a substring, in that DFA, all I needed to do was to switch the final states, accepting states with non-accepting nonfinal states and that would give me this language, set of all strings in which ac does not occur as a substring.

I can actually write a regular expression for this and in fact this is that regular expression. Can we understand this? Of course such a string in which ac does not occur can begin with any number of cs, no problem. And then what we have, what we are seeing is that I will have 0 or more strings from this language. Now you see in this language what is happening? If you have to get a c, you know that you can get out of here, and number of cs including 0 but then that has to be preceded by b, all right and so therefore b followed by 0, 0 or more cs, again b followed by 0 or more cs, in between anywhere you put a s, but when you put a s, a can come, only a single a can come only after one of these, of course you could put another a because in closure you can take any copies of, any number of copies of this.

But notice it is not very difficult to see that the symbol c, if it occurs, it has to have b in front here in this part. And therefore the set of string generated or the language denoted by this regular expression does not have ac occurring as a substring.

(Refer Slide Time: 39:42)

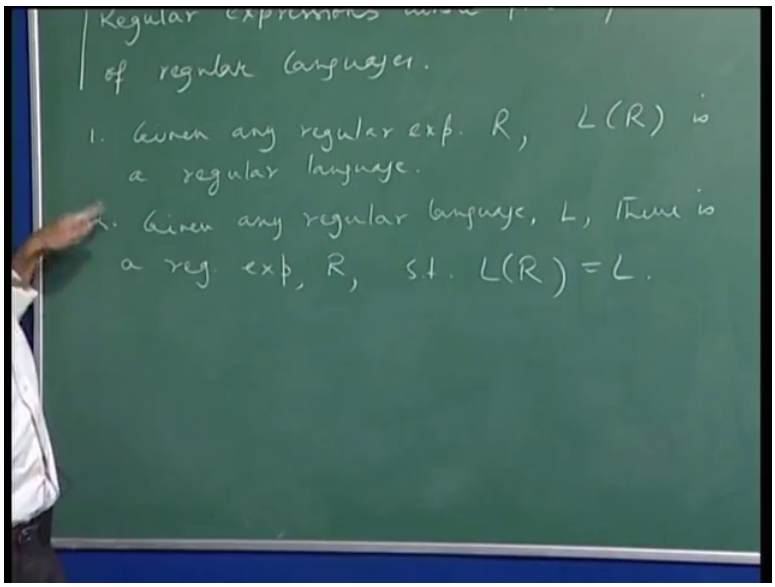




Now we are going to prove something very important, which says that regular expressions, you take this infinite regular expressions that is possible over an alphabet Sigma with all these, each one of these regular expressions, of course we know each one of these denotes a language. So if you take the sum total of all the languages which can be denoted through these regular expressions over an alphabet, the class of languages that you get is precisely the class of regular languages. In other words what I am saying is that this is the class of regular languages, just a picture to keep our ideas clear.

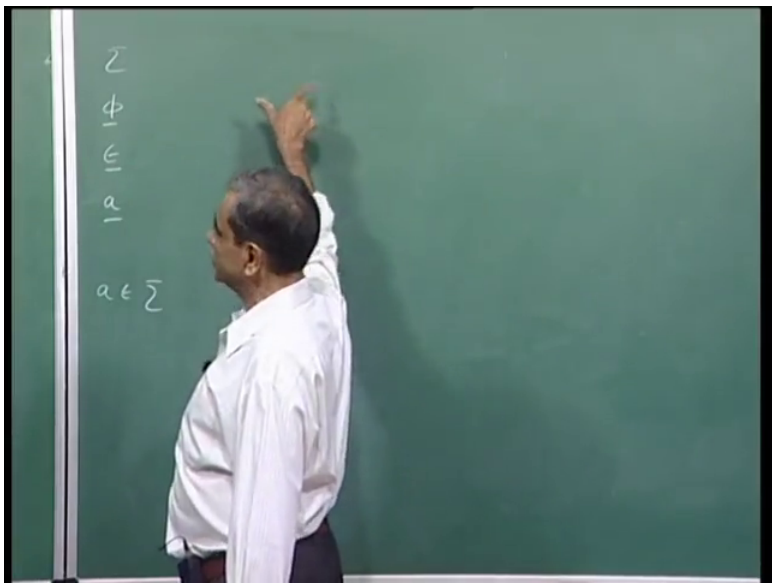
So you know each one here, each element here is a regular language L1, L2, right each L is a regular language. By this what I mean is for each one there is a regular expression which denotes this language, that is one part and the 2nd part is that if you take the set of all regular expressions over the alphabet Sigma, then these are now, I am thinking the class of all regular expressions, each one again denoting a language, each one of them is also regular. In other words the language denoted by the set of all regular expressions and the regular languages, the class of all regular languages, these 2 things are precisely one and the same.

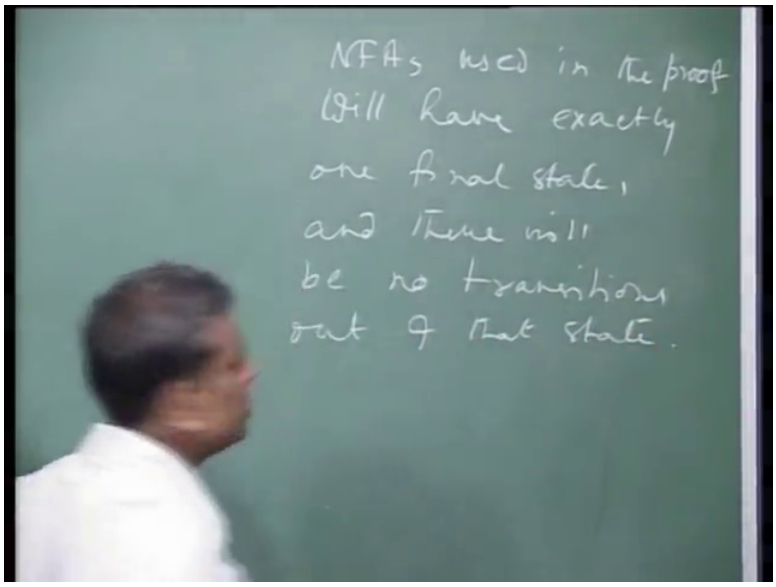
(Refer Slide Time: 41:49)



And I will prove this, this as you can see requires 2 proofs, direct way of showing the fact and the 1st one I need to show is that given any regular expression let us say R, LR is a regular language. And the 2nd part will be the other way round, given any regular language say L, there is a regular expression say R such that the language denoted by the regular expression R is same as the regular language that you have given, L. Today let me show you this part and we can take this part next lecture. How do I show this, the given any regular expression, the language denoted is a regular language, we follow the same inductive rules that we had for farming regular expressions.

(Refer Slide Time: 43:45)



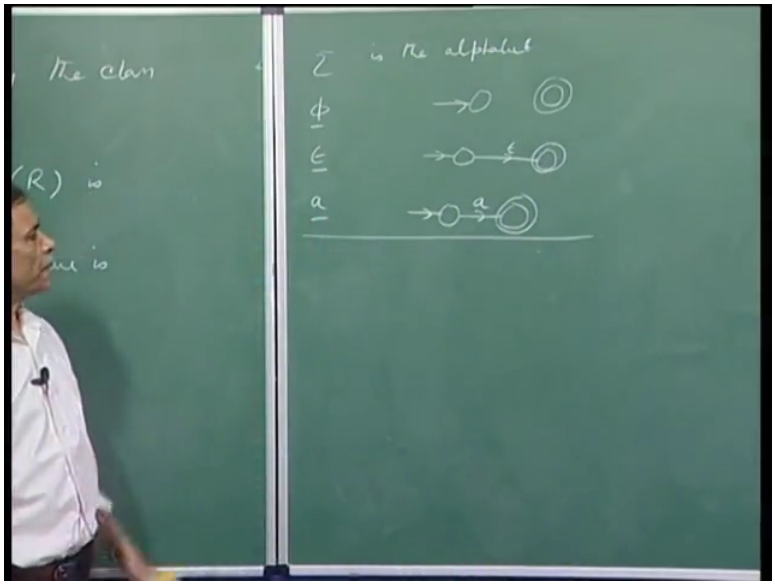


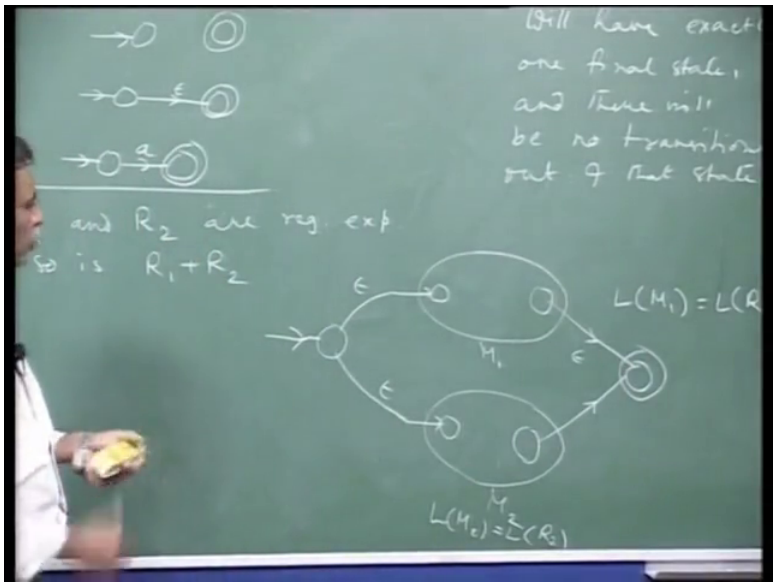
So as you will see, you see I have the base cases where is Sigma is the alphabet, phi, Epsilon and a, where a is in Sigma, these are regular expressions. So it is trivial to give for each one of these regular expressions some DFA or NFA to accept this. But for our purpose it will be simpler, the proof will become simpler when I use NFAs with epsilon transitions and what more, in my construction what we are going to do is that as we use the inductive rules, we will keep forming NFAs but these NFAs will and these NFAs will be NFAs with epsilon transitions which is okay because every NFA with epsilon transitions, we had proved last time that every NFA with epsilon transitions also accepts only regular languages.

So the NFAs that will define will have, NFAs used in the proof will have exactly one final state and there will be no transitions out of that state. It is just easy for our purpose because

you know this will be able to compose, put together different NFAs of this kind and it will be very clear what they do. So let me and Sigma is the alphabet and what is such an NFA for this, for the set denoted by this regular expression, that is of course trivial, okay. Now this is an NFA which accepts the language denoted by this regular expression, it is clear to see but the point I want to make is as we are, as we will do this, please make sure that we keep this in mind that every such NFA will have one final state and there will be no transition out of that final state.

(Refer Slide Time: 47:03)





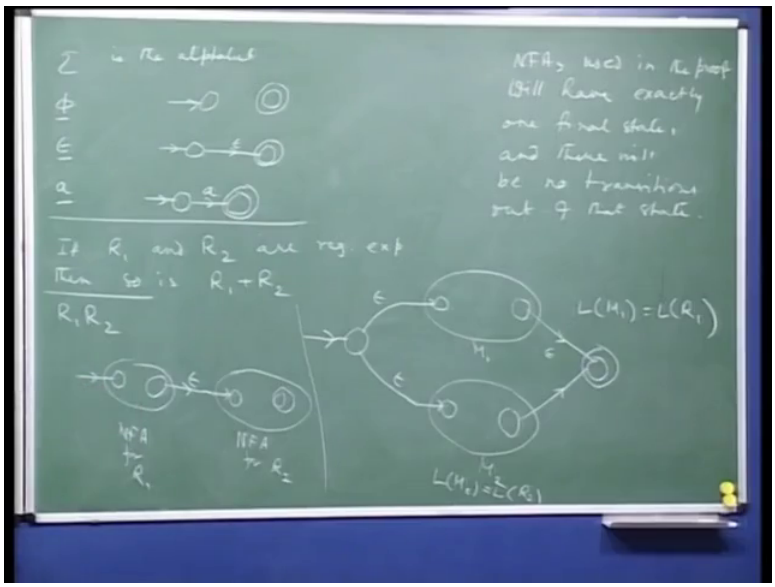
And what about this one, well, that is also easy. Okay, it is easy to see that this NFA accepts only this language, right and there is no transition out of this final state. And a is a symbol and that is only one transition and that is on a, okay. Now let us use our recursive rules. So

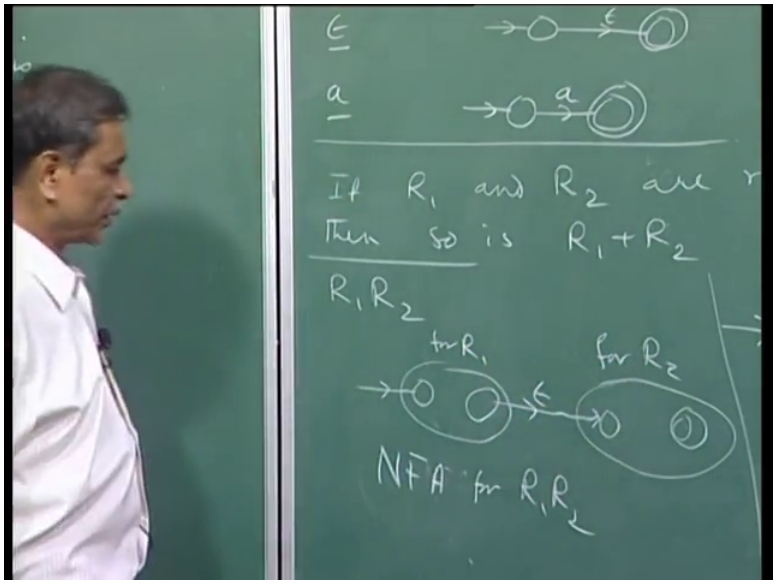
what did the recursive rules say, that if for example, 1st point says that R1 and R2 are regular expressions, then so is R1 + R2. Okay. Now, R1 and R2 are 2 regular expressions, inductively I assumed that for each R1 and for R2, for each of these R1 and R2 I have an NFA which accepts the same language as the language denoted by R1.

So let me draw a diagram, so this is an NFA M1, and there is, and remember that there is exactly one final state with no transition going out of it, that is inductively I have assumed. So what is the point I am saying, that this NFA accepts the same language which is denoted by the regular expression R1 and similarly I have an M2, inductively I can assume such that L M2 is L of R2. And I want to, this is easy, this I want to get that NFA which will accept the union of these 2 languages and that is fairly easy to see how we can do it.

I will put a new initial state, have epsilon transitions to the 2 old initial states and these will no longer be marked as final, these 2 final states because I need to have only one final state and no transition going out of it and this will be it. So this is the new machine I obtained using these 2 and similarly you can see what is going to be for R1 R2.

(Refer Slide Time: 50:14)

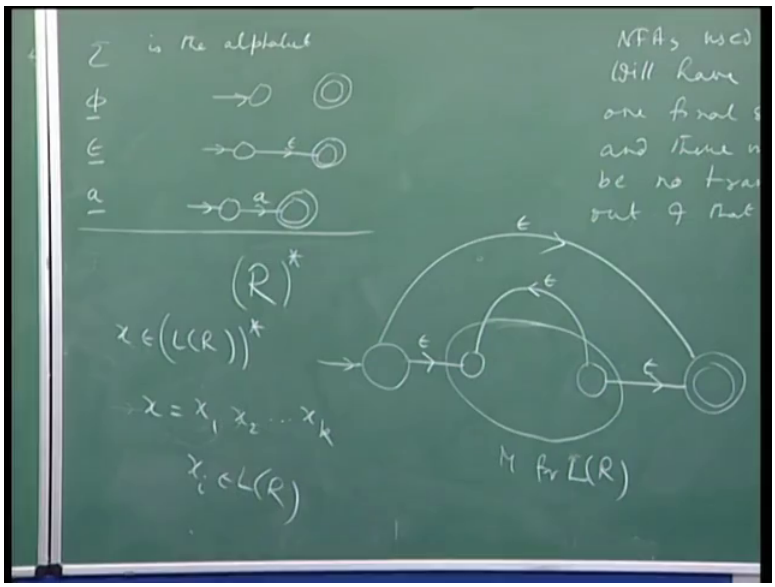




For the R1 R2 case, I had the 2, this is the NFA which you would accept, the language denoted by R1 and let say this is the NFA case 2 for, NFA for R1, NFA for R2 and clearly if I just do this, do not mark it as final state but an epsilon transition from here and this is it. So why, look at this, I mean the argument for that will be similar, so consider this combined, this new NFA which is of course one NFA. Now for any string to go from here to here, it has to 1st go from here to here and that means that string must be 1 in the language of R1, the language denoted by R1.

And then it has to go, the 2nd part of the string will take, just combine NFA from this state to the state and that can be done only through. So this particular NFA, just this part was accepting all strings which are in the language LR 1, this LR 2 and the combined machine will accept the string if it takes the string from here to the only final accepting state and that means it has to use LR 1 string and then and LR 2 string. So therefore this NFA is the NFA for R1 R2, provided this is for R1 and this is for R2. And then we put this epsilon transition, we change this final state to nonfinal state and again you notice our invariant that this machine also has exactly one final state and there is no transition out of it because R2 machine also did not have any transition going out of it.

(Refer Slide Time: 53:23)



More interesting is the case of closure, so let me write that here. Suppose I have a regular expression R and the, this is the corresponding NFA for R, the language denoted by R and from this I would like to create or define an NFA for R star. And I claim that this will be the NFA, like, put a new state which is my initial state, I put a new state which is the final state, new final state, so this original NFA final state is now marked as nonfinal. However I put an epsilon transition from this state to the old initial state and this transition is epsilon, here I have epsilon and there is one transition from this state to.

I claim this is a machine which will accept this language and it is not too difficult to see why because what is a string x in LR star, x, such a string x can be thought of as x1, x2, xk where each xi is in the language denoted by R. So you see on such a string our machine what it is going to do, it will 1st of all, 1st of all it will make this transition to come here and then on x1 it can reach this state and now it will use an epsilon to come back here, then on using x2 it can reach again this state, on epsilon is welcome here and so on. And finally after xk it will, it can take this transition.

So therefore x1 through xk, this stream can take this NFA from its initial state to one of the final, I mean the only final states that it had, that it has, also notice this language contains epsilon and this machine of course has a epsilon because there is this transition which is there from the initial state to final state. and you can also prove the other way round that suppose this is a string which can take the machine from this initial state to the final state, that is again not difficult by the similar argument that that string has to be in this language.

Now one thing I should point out that you may be you might have been tempted not to put this state, instead have an epsilon going from here to here. This epsilon has to be there because you know concatenation of strings in the language, so that means from the final state there should be transition back to the initial state, but if you put an epsilon transition from here to here, then that will not be correct and you can check that out.

So what I am saying is that it will, it would have been wrong not to have added this initial state, new initial state and instead working with the old initial state, putting this epsilon transition and to take care of the epsilon that can be there or that has to be there in LR star, if you attempted to add an epsilon from here to here, that construction would have been wrong, you can try to figure out why. So in the next lecture will do the 2^{nd} part.

What I am try to say that using this part what we have done, inductively have shown that for each of the base case I have an NFA which accepts the same language and inductively if I have 2expressions R1, R2 and the corresponding NFAs for them, then I can combine these NFAs to do for example R1 R2 or R1 + R2 or for the NFA for R, R star. So this part of the proof is taken care of, now in the next lecture we will take care of this.