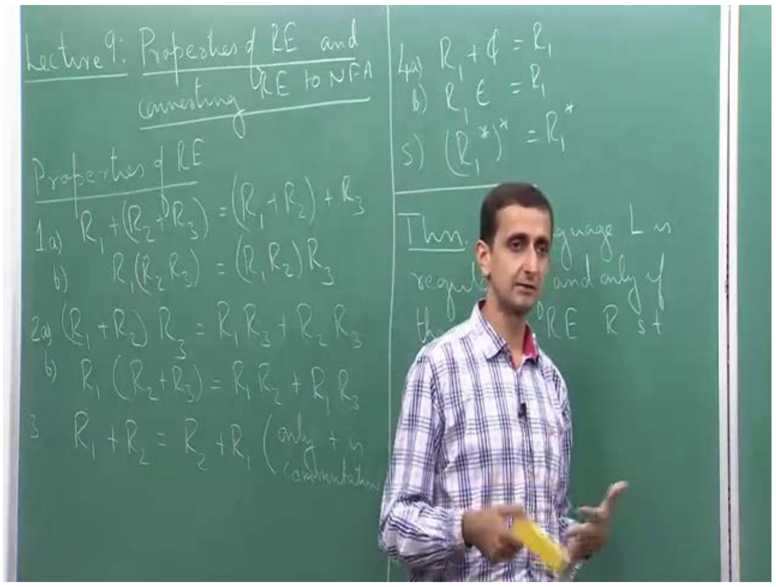
Theory of Computation

Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 09
Algebraic properties, RE to NFA conversion

Welcome to the 9th lecture of this course. So, today we will look at some properties of regular expression, and then we will see how to convert a regular expression to an NFA. So, this will be part of the main question that we raise last time that what is the power of regular expression, and what we said is that regular expressions are equivalent in power to finite automaton. So, the first part of showing that would be to show how we can convert a regular expression to an NFA, so that is what we are going to see today.

(Refer Slide Time: 01:02)



Lecture 9: Properties of RE and converting RE to NFA

Properties of RE

1a) $R_1 + (R_2 + R_3) = (R_1 + R_2) + R_3$

b) $R_1(R_2 R_3) = (R_1 R_2) R_3$

2a) $(R_1 + R_2) R_3 = R_1 R_3 + R_2 R_3$

b) $R_1 (R_2 + R_3) = R_1 R_2 + R_1 R_3$

3 $R_1 + R_2 = R_2 + R_1$ (only + is commutative)

4a) $R_1 + \phi = R_1$

b) $R_1 \epsilon = R_1$

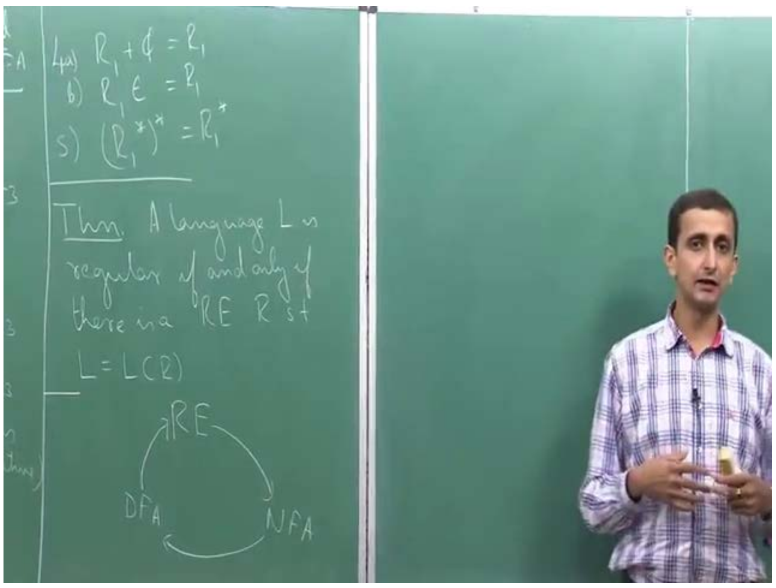
5) $(R_1^*)^* = R_1^*$

Thm. [illegible] guage L is regul[illegible] and only if th[illegible] RE R s.t

So, let us start by looking at some properties. So, let R 1, R 2 and R 3 be three regular expressions. Then what we have is firstly, the associativity law, so R 1 plus R 2 plus R 3 is equal to R 1 plus R 2 whole plus R 3, so I will right this as 1 a. And 1 b is R 1 R 2 R 3 is equal to R 1 R 2 concatenated with R 3. The second is the distributive law that is R 1 plus R 2 concatenated with R 3 gives R 1 R 3 plus R 2 R 3, and similarly, from the other side if we have R 1 concatenated with R 2 plus R 3 that gives us R 1 R 2 plus R 1 R 3.

We have commutative of the plus operator only. So, this is if we have R 1 plus R 2, so this is the same as R 2 plus R 1. And what is important here is that only the plus operator is commutative the concatenation operator is not commutative in the case of regular expressions. So, then we have some other identities R 1 plus phi is R 1 and R 1 dot epsilon is again just R 1. R 1 star whole star is R 1 star. And so these are some of the basic identities involving regular expressions that we frequently use.

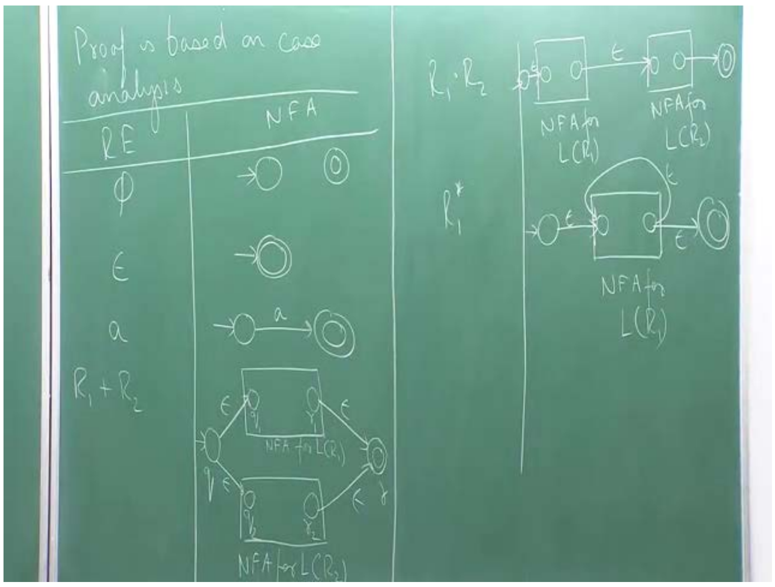
(Refer Slide Time: 04:55)



So, now let us come to the question about the power of regular expression. So, I will state it as a theorem. A language L is regular if and only if there is a regular expression R such that L equals L of R. So, basically if I take a regular language then there is a regular expression for it; and every language for which we have a regular expression is a regular language, so both directions. So, what we will show today is the reverse direction that if we have a language for which we have a regular expression then we will show how to construct an NFA for that language, which immediately implies that the last the language is regular. Because that is what we have seen last in previous lectures that if we have an NFA for a language then we can construct a DFA for it and by definition language is regular if there is a DFA for that language. So, this is the path that we are going to take.

In fact, let me just give a overview, I mean although this we are not going to look at the complete proof of this theorem today, but nevertheless let me give an overview as to how the theorem will work and how do we prove the theorem. So, what we show today is that from regular expression, we can construct an NFA. So, this is the agenda for today. We have seen earlier that from an NFA, we can get a DFA. So, this is using the subset construction. And what we will show in the next lecture is that from a DFA, we can construct an R E. So, if we have a DFA for a language, we can construct an R E which accepts same the language that the DFA accepts, so this we will see next time. So, this is basically our agenda to prove this, and this will show that the expressive power of regular expressions NFA and DFA are equivalent.

(Refer Slide Time: 08:39)



So, let us look at the proof. So, the proof of this is quite simple and it is just follows a similar pattern as the proof of closer property of union, concatenation and the star operators. So, last time, we had seen how to prove that the regular operators union, concatenation and star are closed, so that is exactly what the same idea will be used in this proof. So, we prove this by on a case-by-case basis. So, the proof is based on case analysis. So, we will look at each different way I mean each way forming a regular expression. So, if the regular expression is so let us say we have regular expression over a here, and what we will do is we will construct the corresponding NFA on the right hand side. So, if our regular expression is phi, what is the language for this regular expression, the language for this regular expression is the empty language, which does not accept any string. So, the NFA can be any NFA that does not accept any string, for example, it can just be a single state it does not have any accept state, hence it does not accept any string. So, this is the first case.

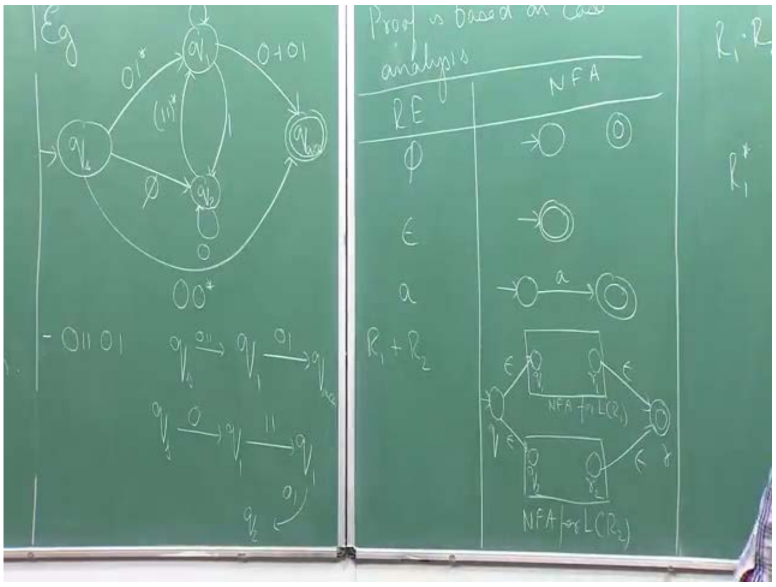
Then if we have regular expression which accepts only the string epsilon, what would be the NFA for it. So, the NFA for this will be an NFA, whose start state is the same as its accept state. So, this is a NFA which has a single state having start state and the accept state. So, this string epsilon is accepted, but no other string is accepted by this NFA. If we have the regular expression a, for a, some symbol in the alphabet, then our regular or the NFA for it, we have a start state and from that we go to an accept state with the label a on it. So, if we see the symbol a, it gets accepted, but no other symbol is accepted by this NFA. So, these are the base cases.

Now what about the inductive cases, if we have a regular expression R 1 plus R 2 by induction we assume that we have an NFA for R 1. So, let us say that this is my NFA for L of R 1. So, this NFA accepts the same language as L of R 1, and we will assume that this NFA has a unique start state and a unique accept state. So, let us call this q 1 and let us call this r 1. If we look at the NFAs that we had constructed earlier each one of them had a unique start state and a unique accept state I mean here we can actually add an accept state also if you want basically with no transition from the start state to the accept state. So, it is just an empty. So, we have an NFA for L of R 1.

Then we have similarly an NFA for L of R 2. Now using these two NFAs, I construct another NFA, which has a unique start state; let us say I will call it q, with epsilon transition to q 1 and q 2. And it has a unique accept state; I will call it r, with epsilon transition coming from r 1 and from r 2. So, if we look at this NFA, every string that is there in either L of R 1 or L of R 2 is accepted by this NFA. And to preserve our assumption, so to maintain our assumption this entire NFA has a unique start state and a unique accept state. So, we can keep building it recursively. So, starting from these three base cases, I can build an NFA for a regular expression which has the plus operator in it.

Now, you can very well guess what would be the NFA for R 1 concatenated with R 2. So, we have let us say an NFA for L of R 1, and we have an NFA for L of R 2. These are the start and accept states. I add a global start state and I have for global accept state and I just connect using epsilon transition. So, from this to this, from this accept state to this start state, and then from this accept state to our new accept state. So, this is the NFA for L of R 1 concatenated with L of R. And finally, for R 1 star just write it as R 1 star, if we have an, so let us say this is our NFA for L of R 1, I add an epsilon transition from the accept state to the start state of that NFA. I add a new start state, and the new accept state, and I add epsilon transition from here to here, and from here to here, also this is a epsilon transition. So, in both these cases, we have an NFA, which has a unique start state, a unique accept state and it accepts the required language. So, this proves that given any regular expression, I can look at each part of that regular expression; and in an inductive manner, I can contrast an NFA for it which accepts the same language.

(Refer Slide Time: 16:27)

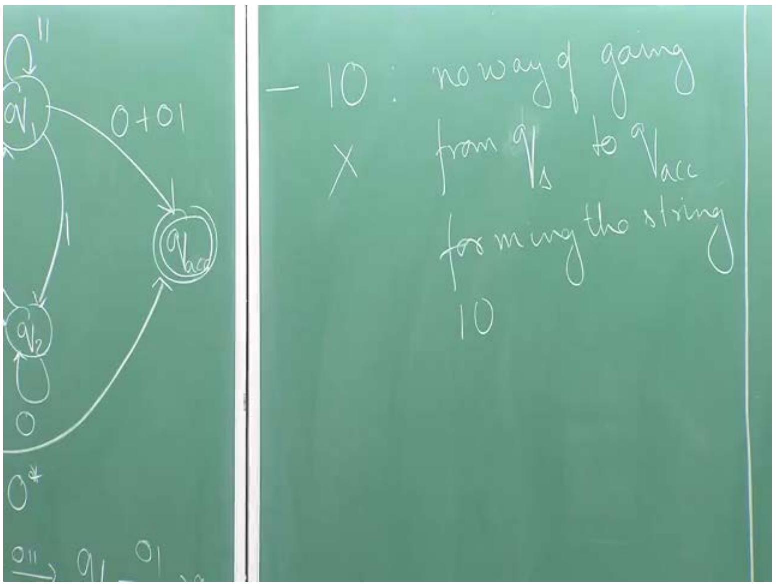


So, finally, what we will see today is what is called a generalized nondeterministic finite automata, I will just give the definition of it will see more about it in the next lecture. So, a generalized non-deterministic finite automaton, so in short, this is also known as a G NFA. So, a generalized non-deterministic finite automaton is a non deterministic automaton which has regular expressions labeling its transitions. So, in a standard non-deterministic finite automaton, the transition either have symbols from the alphabet the input alphabet or it has epsilon and but it cannot have anything else. But suppose if I generalize this idea, and I allow a non-deterministic finite automaton to also have regular expressions on its transitions that is what gives a generalize non-deterministic finite automaton. So, G NFA is a NFA that has regular expressions labeling its transitions.

So, just let me give an example. So, here is a example of a G NFA. So, we have let us say a start state here, I will call it q s. We have a state q 1, q 2, and let us say q accept. And suppose we have transitions from q start to q accept with the regular expression 0 0 star, from q star to q 2 with the regular expression phi, q start to q 1 with the regular expression 0 1 star 1 1 0, q 1 to q 2 with the regular expression 1, q 2 to q 1 with 1 1 star and let us say q 1 to q accept with the regular expression 0 plus 0 1. So, this is an example of a generalize non-deterministic finite automata it has transitions who labels have regular expressions on them.

So, let us look at few strings which are may be accepted and some strings which are not accepted. So, if we look at the string 0 1 1 0 1, is this accepted or not, how can we get this string. So, basically the way a string is accepted is by so can we partition this string into pieces in two parts such that each part is basically in the language of the regular expression of a transition and concatenating them I get a path from the start state to the accept state. So, one way of getting q 0 1 1 0 1 is, if I take 0 1 1 from here because I have 0 1 star. So, form q s to q 1 I take 0 1 1, and then from q 1 to q accept I take 0 1. So, one way of going is I go from q start to q 1 by matching with 0 1 1; and then from q 1 to q accept, I take the string 0 1, so this is one way. May be there are other ways also for example, what I can do is I can go from q s to q 1 on the string 0, because this has 0 with 1 star. So, I can take 1 star as epsilon then on q 1 I have basically a self look with a 1 1 on a. So, from q 1 I stay at q 1 on 1 1, so this gives me 0 1 1. And then from q 1 I go to q 2 on a 0 1, so I can form a 0 1 1 0 in these two ways. And if you look I can actually form via other ways also; so there are other ways by which I can for the string 0 1 1 0 1.

(Refer Slide Time: 23:03)


0+01
1
0
0*
– 10 : no way of going from q_s to q_{acc} forming the string 10
X

Now, what about another string? Let us say so hence this string is accepted; so what about a string 1 0. So, if I look at this regular expression, you realize that there is no way I can go from q star to q accept via sequence of transitions, which will give me the, whose concatenation will give me the string 1 0, there is no way I can do that. So, no way of going from q start to q accept forming the string 1 0, hence this string is not accepted. So, these are two examples and this is basically what this is the way a G NFA accept or does not accepts a string. So, the reason why we are looking at G NFA is because next time we will see that when we convert a regular expression, when we convert a DFA to a regular expression that is where we will use the power of a G NFA. We will use G NFA to allow us or to enable us to construct a regular expression from a DFA. So, I will stop here today.

Thank you.