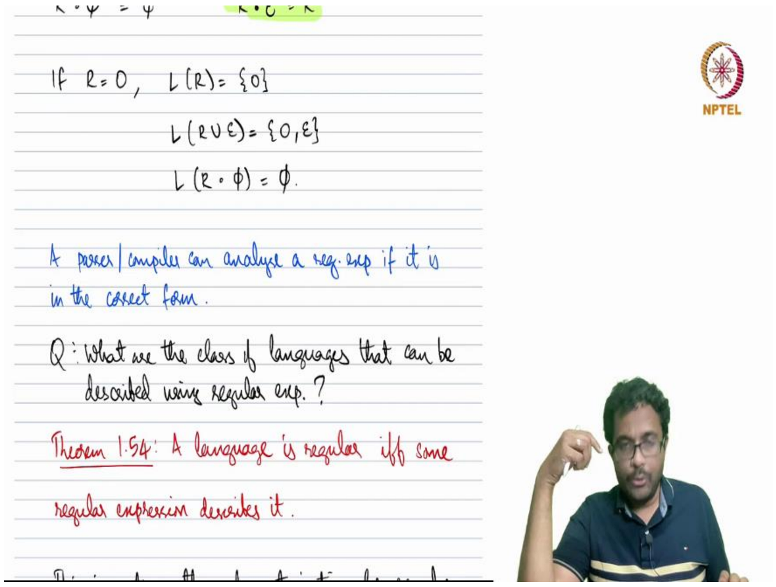
Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 11
Regular expressions – Part 02

(Refer Slide Time: 00:16)



$R \cdot \phi = \phi$ $\quad R \cdot \varepsilon = R$

If $R = 0$, $L(R) = \{0\}$

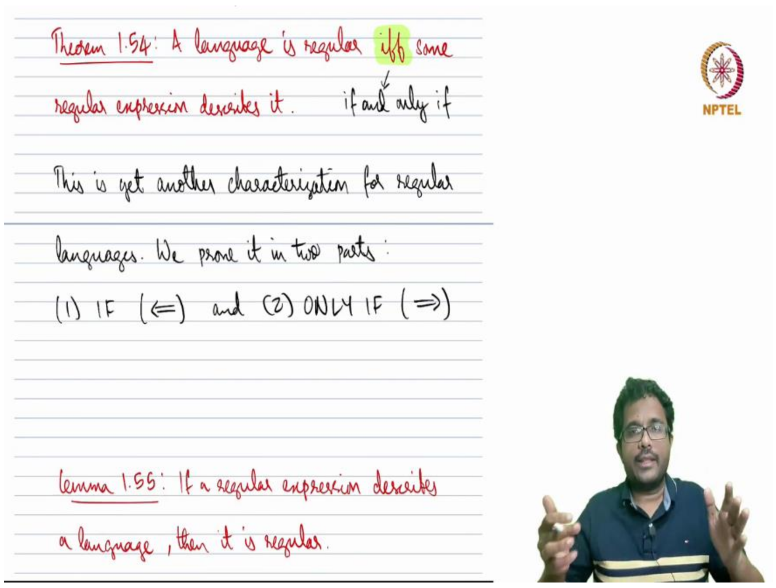
$L(R \cup \varepsilon) = \{0, \varepsilon\}$

$L(R \cdot \phi) = \phi$.

A parser/compiler can analyse a reg. exp if it is in the correct form.

Q: What are the class of languages that can be described using regular exp.?

Theorem 1.54: A language is regular iff some regular expression describes it.



Theorem 1.54: A language is regular iff some regular expression describes it. $\quad$ if and only if

This is yet another characterization for regular languages. We prove it in two parts:

(1) IF $(\Leftarrow)$ and (2) ONLY IF $(\Rightarrow)$

Lemma 1.55: If a regular expression describes a language, then it is regular.

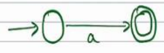
(1) IF ($\Leftarrow$) and (2) ONLY IF ($\Rightarrow$)

Lemma 1.55: If a regular expression describes a language, then it is regular.

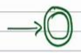
Proof: Given a regular expression R, we will construct an NFA that recognizes L(R).

(1) $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$

(1) $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$



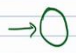
(2) $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$.



10

(3) $R = \phi$. Then $L(R) = \phi$.

(3) $R = \phi$. Then $L(R) = \phi$.



(4) $R = R_1 \cup R_2$
(5) $R = R_1 \circ R_2$
(6) $R = R_1^*$

} Follows using closure properties of regular languages.

Example: $01^* \cup 1$

So, a regular expression actually describes a language. One natural question that comes up is, what is the class of languages that can be described using regular expressions. Can we describe any language using this or is there some kind of restriction? So, in regular expressions, basically we have some building blocks and then you combine them using regular operations.

So, the name itself indicates, why did we call them regular expressions? It will be quite confusing if the class of languages that can be described using regular expressions is something and we have a separate class of regular languages. Thankfully, we do not have this confusion because the class of languages that can be described using regular expressions is the same, which is the class of regular languages.

That is what we have next to show. The class of languages that can be described using regular expressions is exactly the class of regular languages. So, now we already saw two different ways to characterize or define regular languages. So, these are the languages that can be recognized by a deterministic finite automaton, DFA. We also later saw that the class of languages that can be recognized using NFAs is the same. Now, we have the third characterization, the class of languages that can be described using a regular expression. So, three different characterizations.

So, now let us try to see how to show this. The theorem is that a language is regular if and only if it can be described using some regular expression. So, iff denotes if and only if. Sometimes it is a commonly used abbreviation, meaning a language is regular if some regular expression describes it. So, something that is described using a regular expression has to be regular. And then if a language is regular then there has to be a regular expression that describes it. So, both directions of the proof.

So, we prove it using two parts. One is the if part and one is the only if part. So, if part meaning, if there is a regular expression, the language described by that is regular. In this particular lecture we will first show the if part. If there is a regular expression, the language described by this regular expression is regular. Meaning, as of now, we have not proved the equivalence.

We may not prove that. We are still in the process of proving that the class of languages described with a regular expression is regular. So, now we have to show that there is a DFA for this or there is an NFA for this. So, what we will do is, let us say there is a regular expression. It describes some language. Let us say R is a regular expression, it describes some language $L(R)$. We will show how to construct an NFA for this.

So, what we do here is we describe a procedure to construct an NFA. We will not give the NFA itself because it depends on the expression. So, depending on what the expression contains, we will show how to build the corresponding NFA.

If you remember regular expressions, the definition had six things. It could be a symbol, it could be an empty string ϵ, it could be empty set ϕ or it could be a combination of these using regular operations, union, concatenation or star. For each of these, we explain how to build an NFA.

So, suppose the regular expression is just consisting of a symbol, let us say a. We already know how to construct an NFA for this. We can have a start state and we can have an accept state. And that is it. This simple NFA accepts only the string a. There is nothing else that it accepts. You can verify this.

Suppose R is the empty string ϵ, which means that we need an NFA that accepts only the empty string and nothing else. That is also easy to construct. We can construct an NFA like this which contains only one state and the start state is an accept state. So, the empty string will be accepted because the start set is an accept state.

If there is any more symbol, this machine cannot process it because there are no arrows outgoing. So, if the string 1 or 10 is input, we cannot process the string because there are no outgoing arrows from here. So, this accepts only the empty string. This is a valid NFA that only accepts the empty string.

Now, if you want to consider the regular expression empty set ϕ, that is also easy. We can have a one state NFA again, which just has a starting state and with no accept states. There is no accept state so nothing is going to get accepted. In other words, it accepts only the empty set because nothing is going to get accepted.

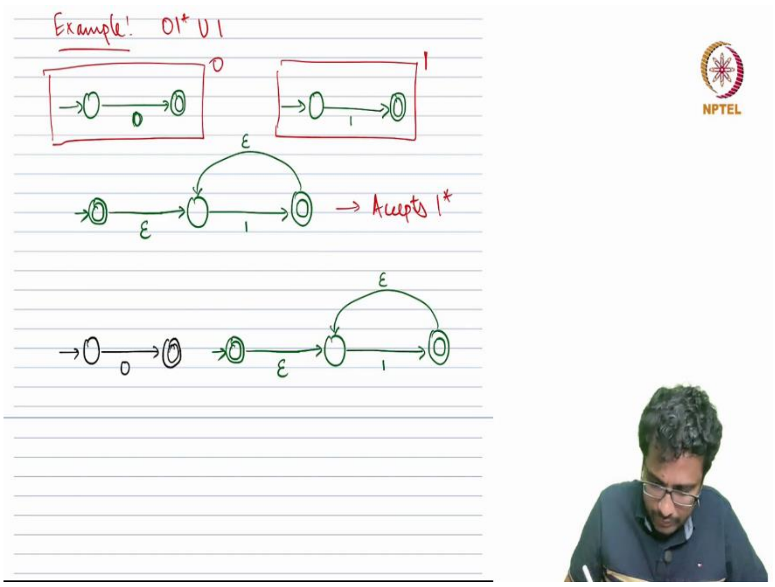
So, there you go three simple NFAs that accept the language, just the symbol a, the, just empty string and nothing. And then we have given R_1 and R_2 being regular expressions, we have the union, concatenation and star. So, what do we do for this? The solution is the same for all of this.

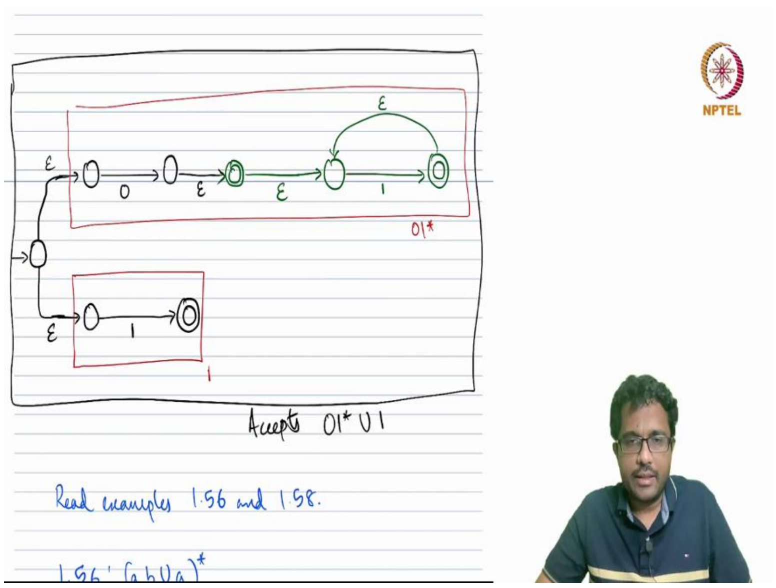
We have already seen that regular languages are closed under union, concatenation and star. So, we can rely on the same thing here. Suppose I am giving you a regular expression R, which is of the form $R = R_1 \cup R_2$. Now, by induction we can assume that R_1 is a regular language and R_2 is a regular language.

Now, since we know that regular languages are closed under union, it follows that $R_1 \cup R_2$ is also regular. And similarly for the concatenation, if R_1 is regular and R_2 is regular, then the concatenation also is regular. We have already seen that in the previous lectures. So, by induction, it follows that $R_1 \cdot R_2$ is regular. And similarly, it follows it $R_1 *$ is also regular.

Therefore, whichever way we describe the regular expression, it could be a, ϵ or empty set, and if it is built using the regular operations, in whatever may be the case, we can say that the language is regular. So, this is a very constructive proof. But just to get a feel of this, perhaps we can do an example.

(Refer Slide Time: 09:03)

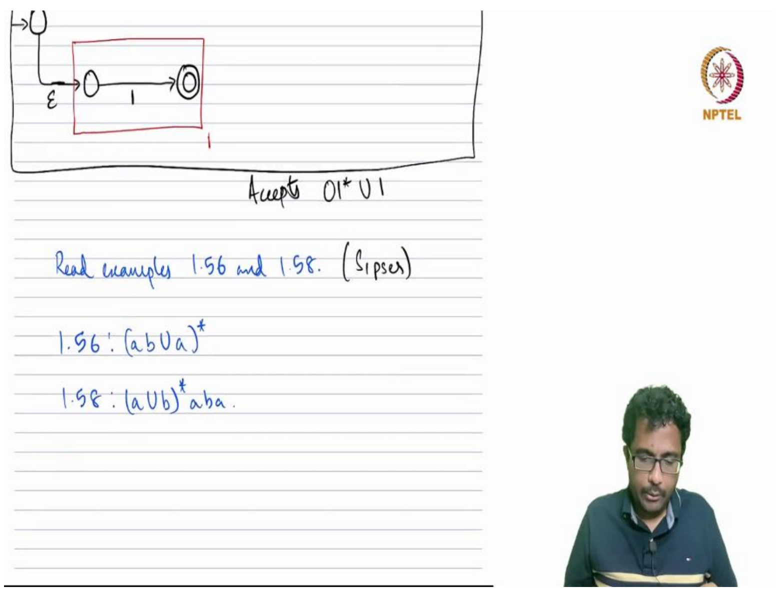




Accepts $01^* \cup 1$

Read examples 1.56 and 1.58.

1.56: $(ab \cup a)^*$

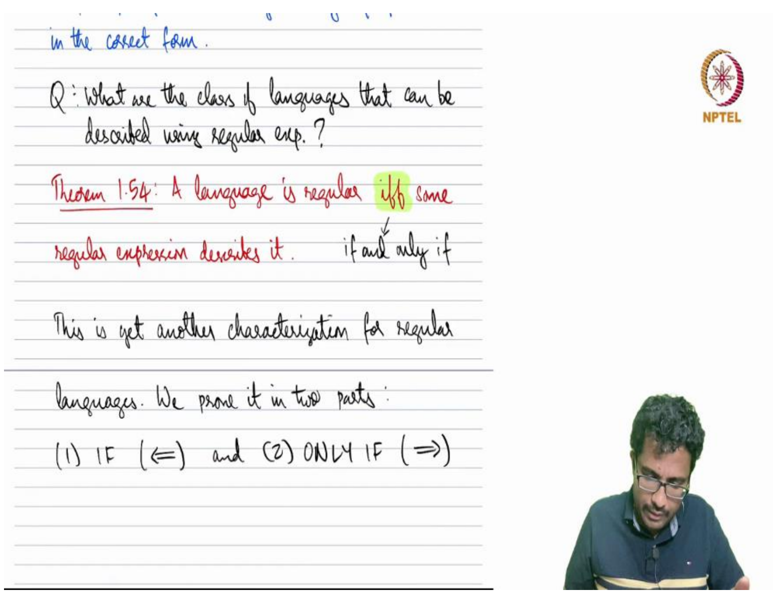


Accepts $01^* \cup 1$

Read examples 1.56 and 1.58. (Sipser)

1.56: $(ab \cup a)^*$

1.58: $(a \cup b)^* aba$.



in the correct form.

Q: What are the class of languages that can be described using regular exp.?

Theorem 1.54: A language is regular iff some regular expression describes it. if and only if

This is yet another characterization for regular languages. We prove it in two parts:

(1) IF $(\Leftarrow)$ and (2) ONLY IF $(\Rightarrow)$

So, the example is let us say, $01*\ \cup 1$. Let us see how to build this. Let us first consider 0. So, this is an NFA that accepts only 0. Similarly, there is an NFA that accepts only 1. From this we need $01*$. So, if you recall, the proof for showing that regular languages are closed under star operation involved something like this.

So, we want to build the NFA for $1*$. So, this is NFA for 1. Now for $1*$, we need to make an ϵ transition from the accept state to the start state. But this does not accept the empty string. So, for that, we need a new start state which is also an accept state, and then have an epsilon transition to the older start state. Maybe I will just draw this. This accepts 0, this accepts 1, and this accepts $1*$.

Now, how do we get a machine that accepts $01*$. So, let me just copy paste this. So, this is basically 0 concatenated with $1*$. So, this is $1*$. And basically, we need to combine them, we need to have an ϵ translation from the 0 machine to the $1*$ machine. So, maybe what I will do is I will redraw the 0 machine.

This is the 0 machine. The second state which is this state would have been an accepting state for the 0. But now that we are concatenating, that is no longer the case. So, it will be like this. So, this accepts $01*$. And I need union with 1. So, maybe I will just replicate the 1 machine here. So, now the machine on top accepts $01*$ and this on the bottom accepts 1. We need the union of this. Again, we can construct this using the closure properties that we saw earlier.

So, the way to construct the union is you have a new start state and then you have ϵ transitions to the two machines that accept the respective languages. So, now this combined machine, with all the listed things, accepts $01*\ \cup 1$. Accepts or recognizes this regular expression.

This is how, if you are given a regular expression, how to construct an NFA for it. So, we saw the proof which is abstract, and then we saw an example where we build an actual NFA for a specific regular expression. There are other examples that are worked out in the book. When I say book, I mean of Sipser. I am just restating the examples in case you do not have the Sipser book but you want to work out. So, one is $(ab \cup a)*$, the other one is $(a \cup b)*aba$. You may want to work that out.

And that is all that I have in this particular lecture. But I will just summarize everything. So, we saw what regular expressions are. This is another way to describe languages. So, basically it is like a, ϵ and empty set. And then combined using regular operations. Then we saw examples and then we said that the class of languages represented by regular expressions is the same as regular languages.

And then we said that the proof has two directions. Anything that can be expressed using a regular expression is regular and we gave a constructive proof for that. We explained how to construct an NFA, given a regular expression, that recognizes the same language. So, we also worked out an example for a simple regular expression, $01* \cup 1$.

And what we have not done is the other direction of the proof, the only if direction of the proof. So, this will require us to show that if I give you a regular language meaning if I give you a language that has a DFA or an NFA, so something that we know is regular as per our previous definitions, then we want to build a regular expression that describes it.

So, given a DFA or given an NFA that recognizes some language, we have to build a regular expression that describes it. Because all we have shown is that anything that can be described in a regular expression is regular, meaning for all you know at this point, regular expressions describe only a subset of regular languages. But we want to show that it actually describes everything, then we have to show the opposite also.

In it, you take a regular language, perhaps the DFA or NFA, then construct a regular expression for that language. So, that is what is pending. This will be shown in the next lecture.