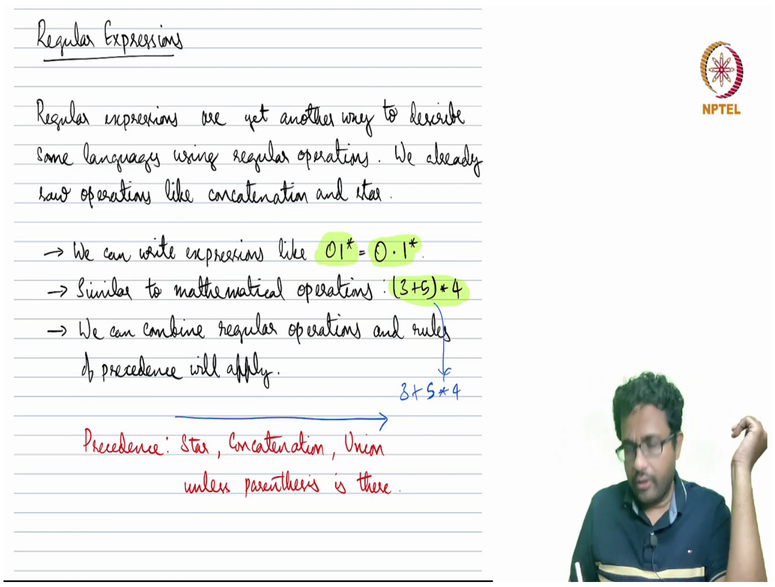
Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 10
Regular expressions – Part 01

(Refer Slide Time: 00:17)



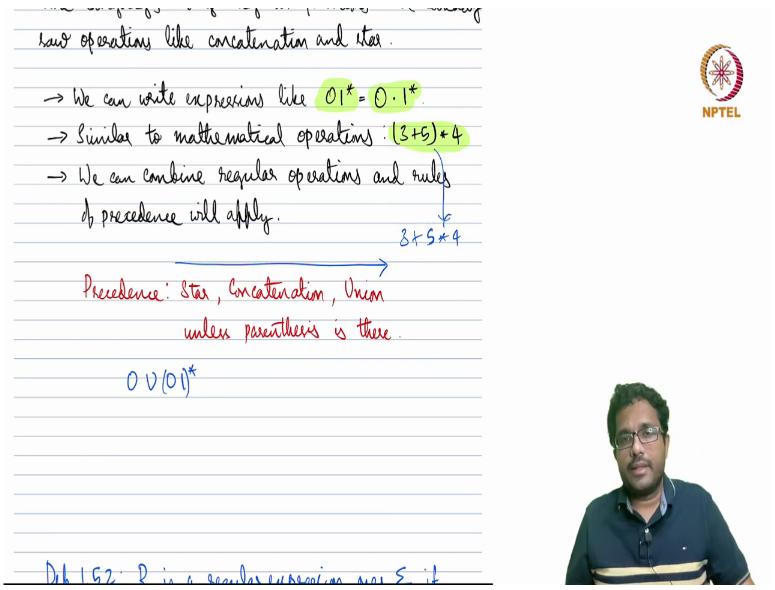
Regular Expressions

Regular expressions are yet another way to describe some languages using regular operations. We already saw operations like concatenation and star.

→ We can write expressions like $01^* = 0 \cdot 1^*$.
→ Similar to mathematical operations : $(3+5)*4$
→ We can combine regular operations and rules of precedence will apply.

$3+5*4$

Precedence: Star, Concatenation, Union
unless parenthesis is there.



saw operations like concatenation and star.

→ We can write expressions like $01^* = 0 \cdot 1^*$.
→ Similar to mathematical operations : $(3+5)*4$
→ We can combine regular operations and rules of precedence will apply.

$3+5*4$

Precedence: Star, Concatenation, Union
unless parenthesis is there.

$0 \cup (01)^*$

Defn 1.52: R is a regular expression over Σ if

Hello, and welcome to Lecture 10 of the course, Theory of Computation. In the previous lectures, we saw DFAs and NFAs, which are two ways to characterize regular languages. Then we also saw the closure properties using DFAs and NFAs, that is regular languages are closed using union, concatenation, star, etc. In this lecture, we will see regular expressions, which are yet another way to capture the regular languages.

So, we will first describe what regular expressions are and later we will come to the proof that the languages that can be described using regular expressions are indeed the same class of regular languages that we have been seeing.

So, what are regular expressions? So, we have already seen regular operations like union, concatenation, star, etc. So, we can write expressions like this like 01^*. The way to interpret this is 0 concatenation with 1^*. So, you have a 1^* which is of the form $\{\varepsilon, 1, 11, 111, \ldots\}$. So, the class of languages looks like $\{0, 01, 011, 0111, \ldots\}$.

So, this is similar to general math expressions like arithmetic expressions. So, we write things like (3+5)*4. Here you distribute the multiplication over addition. But another way to think about it is to add 3 and 5 and then multiply by 4. And you know that this is actually different from 3+5*4 because we have precedence rules. So, it would be 5 times 4 and then you add the 3 to it.

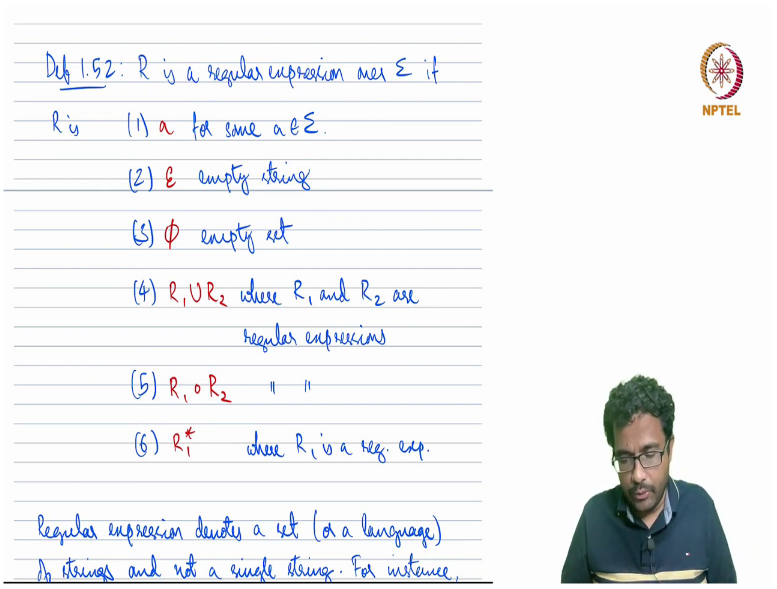
Similarly, regular expressions are expressions that are formed using regular operations like union, concatenation, star, and there is an order of precedence. The order is as follows: Star, Concatenation, Union unless parentheses is there.

Consider the example: $0 \cup 01^*$

So, here the star takes, star is first then the concatenation of 0 and 1^* and then the union. If brackets were put as follows: $0 \cup (01)^*$ then the 01 will be concatenated.

So, basically it is using the symbols of the language and then combining them using regular operations, like union, concatenation, and star, and they are in this order of precedence order.

(Refer Slide Time: 04:25)



Def 1.52: R is a regular expression over Σ if

R is (1) a for some $a \in \Sigma$.

(2) ε empty string

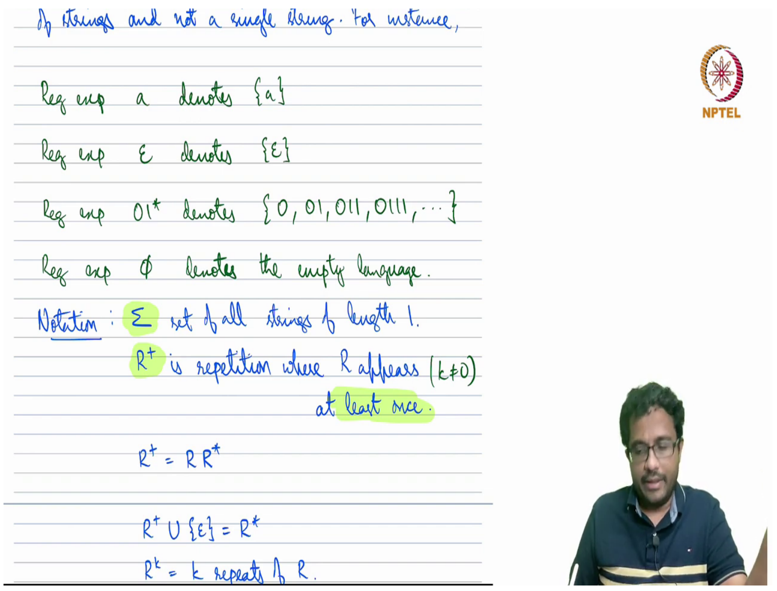
(3) ϕ empty set

(4) $R_1 \cup R_2$ where R_1 and R_2 are regular expressions

(5) $R_1 \circ R_2$ " "

(6) R_1^* where R_1 is a reg. exp.

Regular expression denotes a set (or a language) of strings and not a single string. For instance,



of strings and not a single string. For instance,

Reg exp a denotes $\{a\}$

Reg exp ε denotes $\{\varepsilon\}$

Reg exp 01^* denotes $\{0, 01, 011, 0111, \dots\}$

Reg exp ϕ denotes the empty language.

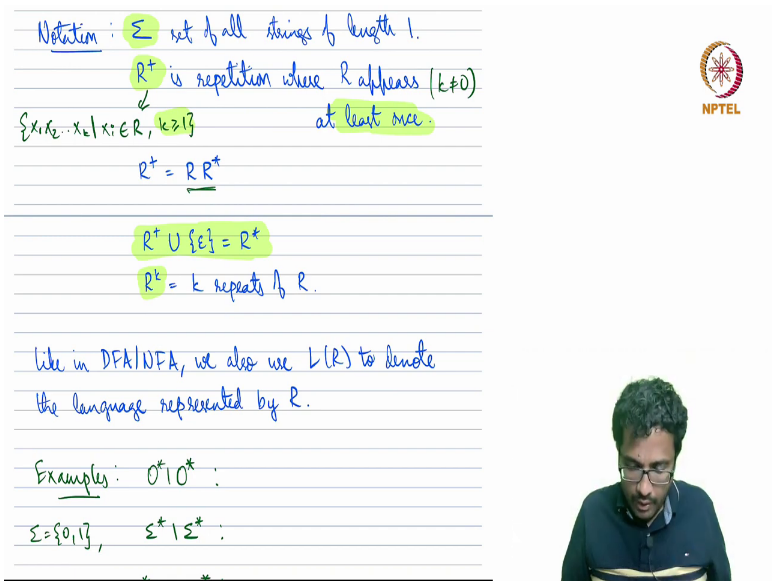
Notation : Σ set of all strings of length 1.

R^+ is repetition where R appears $(k \neq 0)$ at least once.

$R^+ = R R^*$

$R^+ \cup \{\varepsilon\} = R^*$

R^k = k repeats of R.



Notation : Σ set of all strings of length 1.

R^+ is repetition where R appears $(k \neq 0)$ at least once.

$\{x_1 x_2 \dots x_k \mid x_i \in R, k \geq 1\}$

$R^+ = R R^*$

$R^+ \cup \{\varepsilon\} = R^*$

R^k = k repeats of R.

Like in DFA/NFA, we also use L(R) to denote the language represented by R.

Examples: $0^* 1 0^*$:

$\Sigma = \{0, 1\}$, $\Sigma^* 1 \Sigma^*$:

Formally regular expressions are the following: for any symbol in the alphabet a, a is a regular expression and epsilon is the empty string. The empty set is a regular expression, which denotes the empty language.

Then if R_1 and R_2 are two regular expressions, then you can combine them using a regular operation such as union or concatenation. If R_1 is a regular expression, you can also use the regular operation star. So, these are the three regular operations that we can use to connect already created regular expressions. And then in addition we have the three building blocks which are some symbols, then empty string and empty set.

So, typically regular denotes a set of strings, not a single string, it could be a set of strings. So, for instance the regular expression a denotes the language that contains only the string a. Regular expression empty string indicates the language that contains only the empty string. Regular expression 01^* indicates the language that contains the following

$$01^* = \{0, 01, 011, 0111, \dots\}$$

And the ϕ of course denotes the empty set which is the empty language. So, this is how we can describe some languages using regular expressions. And some simple things like some more notation, the regular expression Σ is just used to denote the set of all strings of length 1, having the same symbols as in the alphabet. So, Σ, if it is the binary alphabet it denotes the set $\{0, 1\}$.

And in addition to this, some more additional notation. So R^+ is defined by the sixth rule here as follows-

$$R^+ = RR^*$$

So R^+ is similar to R^* but just that R is repeated at least once! This basically just eliminates the empty string. In other words-

$$R^+ \cup \{\varepsilon\} = R^*$$

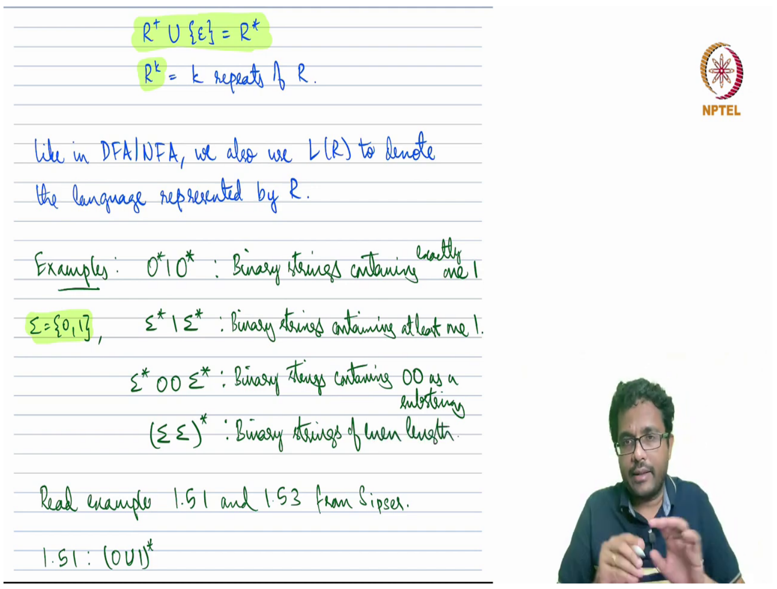
So the formal definition is-

$$R^+ = \{x_1x_2\ldots x_k \mid x_i \in R, k \geq 1\}$$

And another notation is-

$$R^k = k \text{ repetitions of } R$$

(Refer Slide Time: 10:31)



$R^+ \cup \{\varepsilon\} = R^*$

R^k = k repeats of R.

Like in DFA/NFA, we also use L(R) to denote the language represented by R.

Examples: 0^*10^* : Binary strings containing exactly one 1

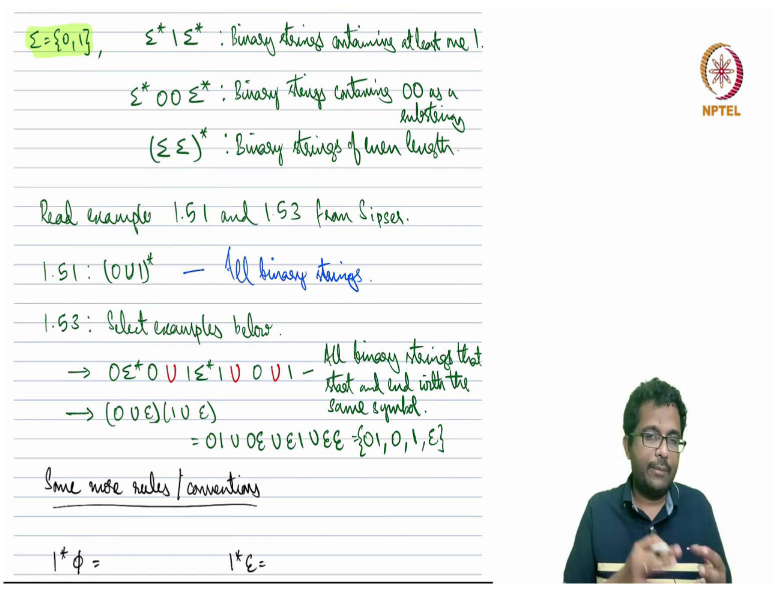
$\Sigma = \{0,1\}$, $\Sigma^* 1 \Sigma^*$: Binary strings containing atleast one 1.

$\Sigma^* 00 \Sigma^*$: Binary strings containing 00 as a substring

$(\Sigma\Sigma)^*$: Binary strings of even length.

Read example 1.51 and 1.53 from Sipser.

1.51 : $(0 \cup 1)^*$



$\Sigma = \{0,1\}$, $\Sigma^* 1 \Sigma^*$: Binary strings containing atleast one 1.

$\Sigma^* 00 \Sigma^*$: Binary strings containing 00 as a substring

$(\Sigma\Sigma)^*$: Binary strings of even length.

Read example 1.51 and 1.53 from Sipser.

1.51 : $(0 \cup 1)^*$ — All binary strings.

1.53 : Select examples below.

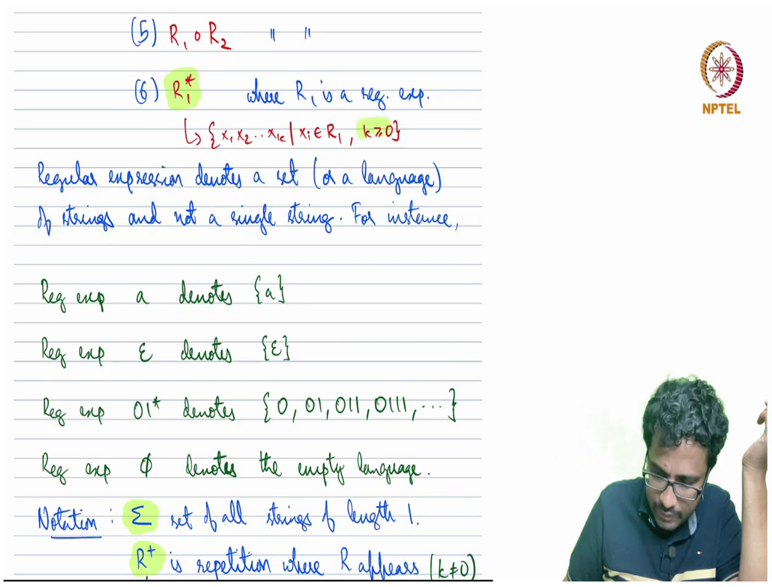
$\rightarrow 0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$ – All binary strings that start and end with the same symbol.

$\rightarrow (0 \cup \varepsilon)(1 \cup \varepsilon)$

$= 01 \cup 0\varepsilon \cup \varepsilon 1 \cup \varepsilon\varepsilon = \{01, 0, 1, \varepsilon\}$

Some more rules / conventions

$1^*\phi =$ $1^*\varepsilon =$



(5) $R_1 \circ R_2$ " "

(6) R_1^* where R_1 is a reg. exp.

$\hookrightarrow \{x_1 x_2 \ldots x_k \mid x_i \in R_1, k \geq 0\}$

Regular expression denotes a set (or a language) of strings and not a single string. For instance,

Reg exp a denotes $\{a\}$

Reg exp ε denotes $\{\varepsilon\}$

Reg exp 01^* denotes $\{0, 01, 011, 0111, \ldots\}$

Reg exp ϕ denotes the empty language.

<u>Notation</u> : Σ set of all strings of length 1.

R^+ is repetition where R appears $(k \neq 0)$

So we'll just see some examples and then it will be clearer.

0^*10^*: Binary strings containing exactly one 1

$\Sigma = \{0, 1\}$ and

$\Sigma^*1\Sigma^*$: <Any string from the binary alphabet>1<Any string from the binary alphabet>

$\Sigma^*00\Sigma^*$: <Any string from the binary alphabet>00<Any string from the binary alphabet>

(this means that it has at least one time 00 appearing, i.e. 00 as a substring)

$(\Sigma\Sigma)^*$:

$\Sigma\Sigma$ is a binary string of length 2

$(\Sigma\Sigma)^*$ is a binary string of even length

There are some more examples that are there in the Sipser book.

For instance, example 1.51 is $(0 \cup 1)^*$. Basically this is Σ^* with Σ being the binary alphabet as described earlier.

Some other examples such as 1.53 is $0 \Sigma^* 0 \cup 1 \Sigma^* 1 \cup 0 \cup 1$.

Basically it is the set of all binary strings that start and end with the same symbol.

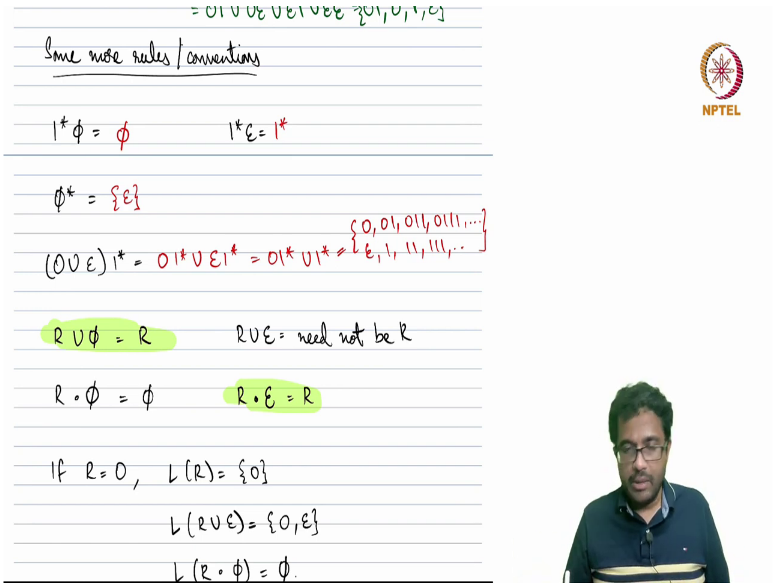
So, you may recall, in Lecture 2 or 3, we considered a DFA that recognizes the language of all the strings that start and end with the same symbol. The same thing is happening here.

And lastly we have (0 U ε)(1 U ε) = 01 U 0ε U ε1 U εε = {01, 0, 1, ε}.

So, if you just work it out, you get, you can distribute the union over the concatenation.

So, hopefully now it is clear what and how we deal with regular expressions. So, regular expressions are expressions like this where we have 0, symbols and then things like empty string, empty set and then connected with regular operations.

(Refer Slide Time: 20:26)



$= 01 \cup 0\varepsilon \cup \varepsilon 1 \cup \varepsilon\varepsilon = \{01, 0, 1, \varepsilon\}$

Some more rules / conventions

$1^*\phi = \phi$ $\quad\quad 1^*\varepsilon = 1^*$

$\phi^* = \{\varepsilon\}$

$(0 \cup \varepsilon)1^* = 01^* \cup \varepsilon 1^* = 01^* \cup 1^* = \begin{Bmatrix} 0, 01, 011, 0111, \ldots \\ \varepsilon, 1, 11, 111, \ldots \end{Bmatrix}$

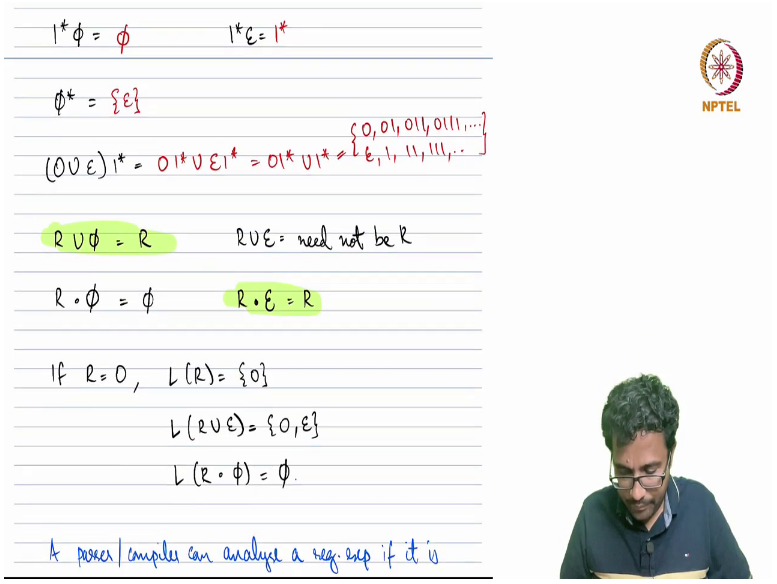
$R \cup \phi = R$ $\quad\quad R \cup \varepsilon =$ need not be R

$R \circ \phi = \phi$ $\quad\quad R \circ \varepsilon = R$

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

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

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



$1^*\phi = \phi$ $\quad\quad 1^*\varepsilon = 1^*$

$\phi^* = \{\varepsilon\}$

$(0 \cup \varepsilon)1^* = 01^* \cup \varepsilon 1^* = 01^* \cup 1^* = \begin{Bmatrix} 0, 01, 011, 0111, \ldots \\ \varepsilon, 1, 11, 111, \ldots \end{Bmatrix}$

$R \cup \phi = R$ $\quad\quad R \cup \varepsilon =$ need not be R

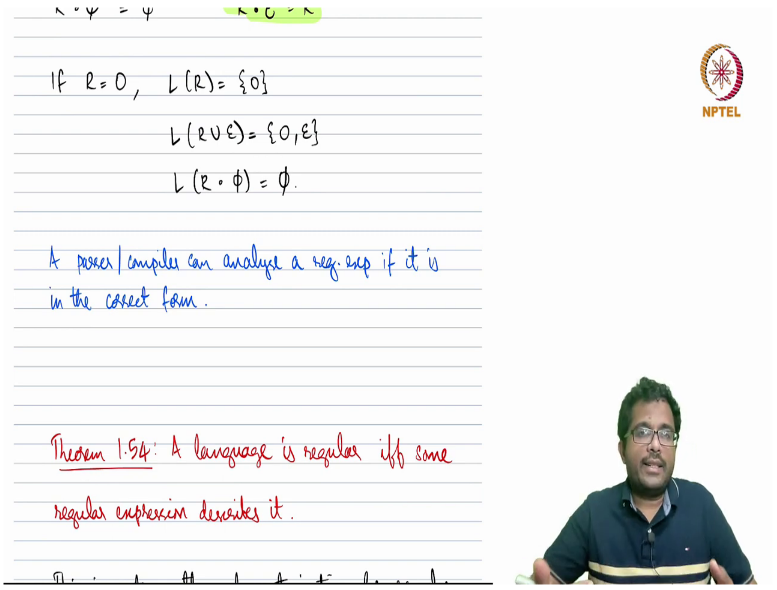
$R \circ \phi = \phi$ $\quad\quad R \circ \varepsilon = R$

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

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

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

A parser / compiler can analyse a reg. exp if it is



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

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

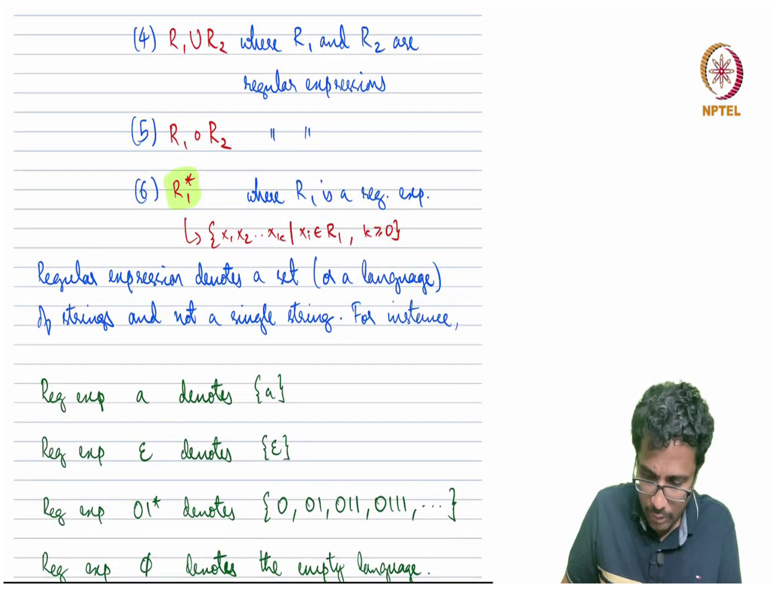
$L(R \cup \varepsilon) = \{0, \varepsilon\}$

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

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

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

[illegible]



(4) $R_1 \cup R_2$ where R_1 and R_2 are regular expressions

(5) $R_1 \circ R_2$ " "

(6) R_1^* where R_1 is a reg. exp.

$\hookrightarrow \{x_1 x_2 \ldots x_k \mid x_i \in R_1, k \geq 0\}$

Regular expression denotes a set (or a language) of strings and not a single string. For instance,

Reg exp a denotes $\{a\}$

Reg exp ε denotes $\{\varepsilon\}$

Reg exp 01^* denotes $\{0, 01, 011, 0111, \ldots\}$

Reg exp ϕ denotes the empty language.

Some more conventions and rules which should be clear but maybe I will go over it anyway.

$1^*\phi = \phi$ (here ϕ is an empty set and not a string)

$1^*\varepsilon = 1^*$ (here ε is the empty string)

$\phi^* = \{\varepsilon\}$

$(0 \cup \varepsilon)\, 1^* = 01^* \cup \varepsilon 1^* = 01^* \cup 1^* = \{0, 01, 011, 0111\ldots, \varepsilon, 1, 11, 111, \ldots\}$

$R \cup \phi = R$

$R \cup \varepsilon =$ need not be R

$R \circ \phi = \phi$

These are some basic points. And one thing that I have skipped here is that, like for DFAs, NFAs etc, we said M is a machine or m is an automata and L(M) is a language described by machine.

Similarly for R, the regular expression R, we sometimes use L(R) to denote the language described by the regular expression.

So we have the following-

$$\text{If } R = 0,\ L(R) = \{0\},\ L(R \cup \varepsilon) = \{0, \varepsilon\},\ L(R \circ \phi) = \phi$$

These are some kind of degenerate situations where you have deal with empty sets and empty strings.

And one point is that if I give a regular expression to a machine, we can have it parsed the regular expression and checks whether it is in the correct form and then tries to interpret it. So, I can give a regular expression to a machine and ask it to, instead of actually writing the entire language in a set or something try to interpret it.