Theory of Computation

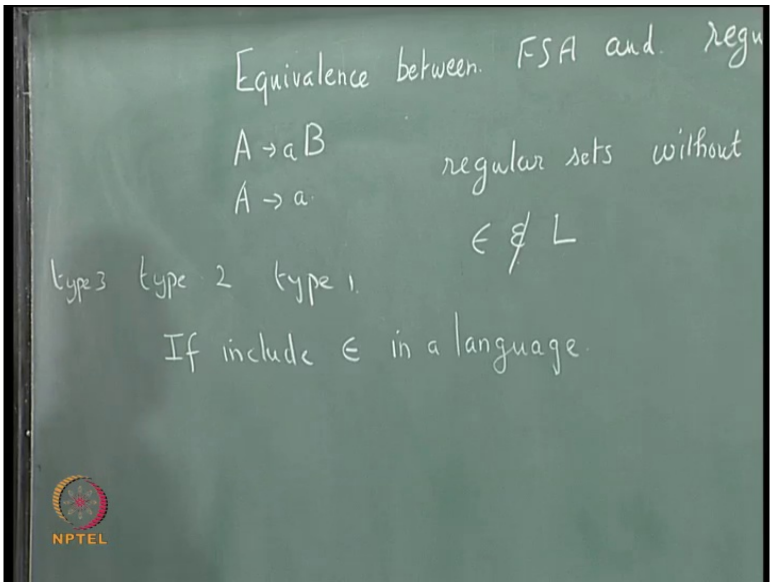
Prof. Kamala Krithivasan
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture No # 12

Equivalence Between FSA And Type 3 Grammars

So today, we shall see the equivalence between regular grammars and finite state automata. We have seen the definition of a deterministic F S A and non deterministic F S A and a non deterministic F S A with epsilon moves. All are equivalent, as far as acceptance power is concerned, given a non deterministic automata with epsilon moves you can remove the epsilon transitions and get an equivalent non deterministic F S A and given a non deterministic F S A by subset construction, we can find the deterministic F S A.

(Refer Slide Time: 00:22)



Equivalence between FSA and regu

$A \to aB$

$A \to a$

regular sets without

$\epsilon \notin L$

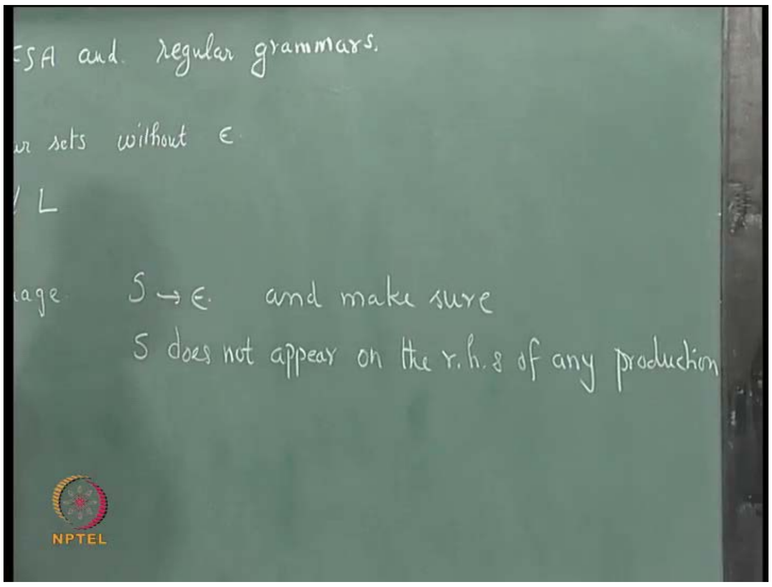
type 3 type 2 type 1

If include ϵ in a language

Now, the grammar, what is a regular grammar? A regular grammar has rules of the form A goes to a B or A goes to a. On the left hand side, you have a non terminal, on the right hand side, you have a single terminal or a single terminal followed by a non terminal. First, I shall consider regular sets without epsilon that is consider languages of the form L where epsilon does not belong to L, because for epsilon, we have to do some slight modification that I will consider later, but we have to remember that any grammar type

2, type 1 or anything, type 1, type 3, whatever it is to include epsilon in a language you can do this, to include epsilon in a language that is, you have the rule S goes to epsilon which can be applied in the first step only, start symbol going into epsilon. And make sure and make sure S does not appear on the r h s right hand side of any production. See in some cases, even if it appears, it may not make much of a difference and in some examples, it may make a difference, we want to be sure, we want to be careful that is why we make this restriction.

(Refer Slide Time: 02:44)



FSA and regular grammars.

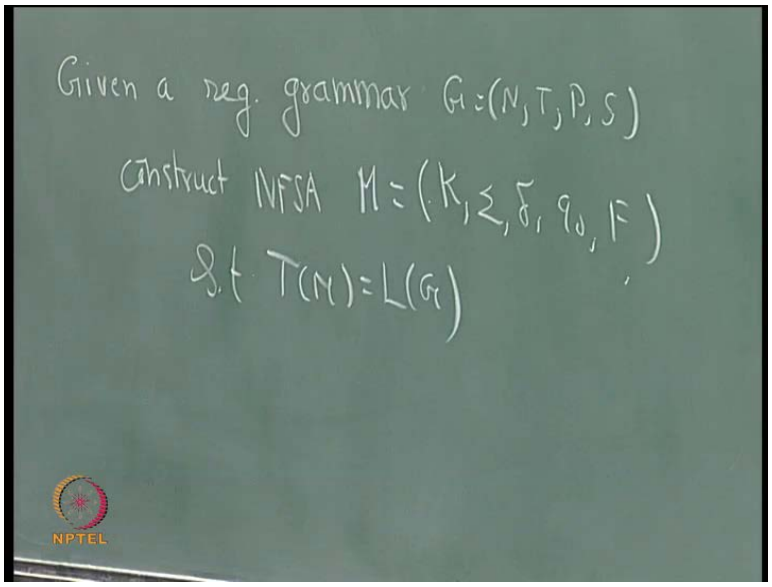
r sets without ϵ

/ L

age $S \to \epsilon$ and make sure

S does not appear on the r.h.s of any production

(Refer Slide Time: 03:54)



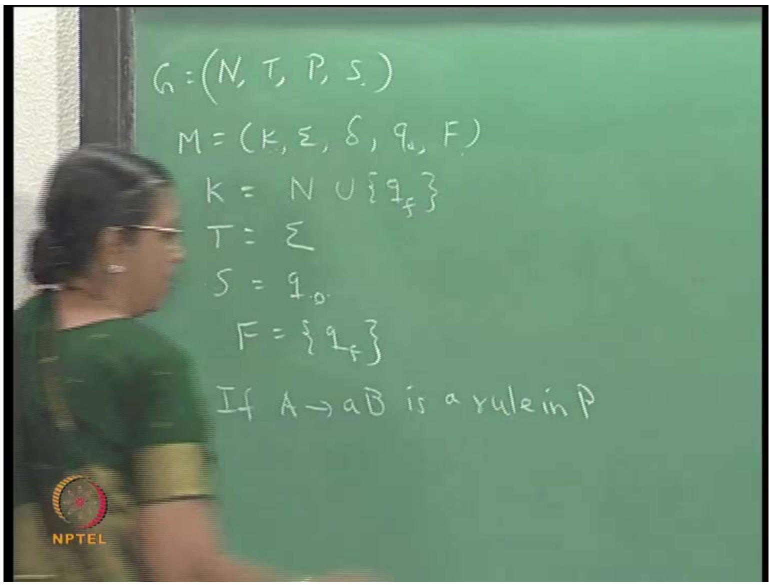
Given a reg. grammar $G = (N, T, P, S)$

Construct NFSA $M = (K, \Sigma, \delta, q_0, F)$

s.t. $T(M) = L(G)$

So, first I will consider regular sets without epsilon, I will start with examples and illustrate the construction with examples. Given a regular grammar G is equal to N, T, P, S construct a non deterministic F S A without epsilon moves, M is equal to K sigma delta q naught F such that T of M is equal to L of G. You given a grammar how will you construct an equivalent automata? The automaton which you are going to construct is a non deterministic one. So, the construction is like this G is equal to N T P S, S is the start symbol, N is the set of non terminals and so on. M is defined as K sigma delta q naught F.

(Refer Slide Time: 05:05)



$G = (N, T, P, S)$

$M = (K, \Sigma, \delta, q_0, F)$

$K = N \cup \{q_f\}$

$T = \Sigma$

$S = q_0$

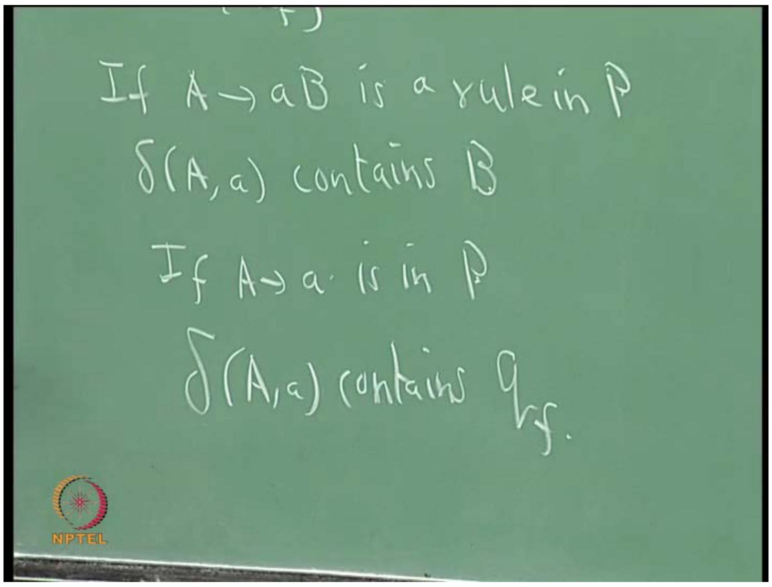
$F = \{q_f\}$

If $A \to aB$ is a rule in P

Now, here K is the same the set of states will be the set of non terminals plus one final state. Each non terminal will correspond to a state in K, apart from that you have a one more final state. And terminal symbols are the input symbols for the automaton and S corresponds to the initial state and initial state q 0 corresponds to the initial symbol, F consists of just q f and delta, you have to define. How do you define delta? If A goes to a B is a rule in P. You have delta of A, a contains B. In automaton, you have delta of A, a contains B and if it is a terminal rule, if A goes to a is in P, it is a terminal rule then delta of A, a contains q f. This is the construction.

We will see one or two examples then, it will be very clear why we have constructed this way. Then, the formal proof is by induction on the length of the string accepted and the string on the length of a induction on the string length of the string accepted by the automata and also one way it is like, this the other way it will be induction on the length of the string generated by the grammar.

(Refer Slide Time: 06:38)



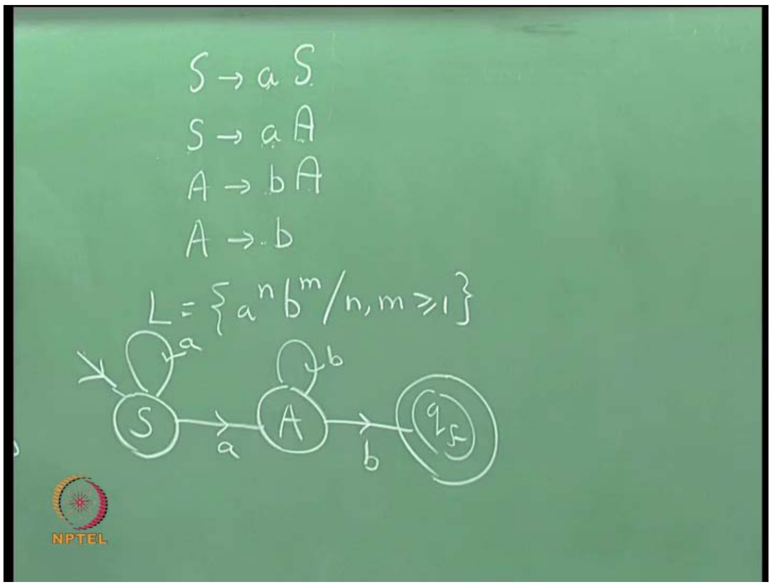
If $A \to aB$ is a rule in P

$\delta(A, a)$ contains B

If $A \to a$ is in P

$\delta(A, a)$ contains q_f.

(Refer Slide Time: 08:00)



$S \to aS$

$S \to aA$

$A \to bA$

$A \to b$

$L = \{a^n b^m / n, m \geq 1\}$

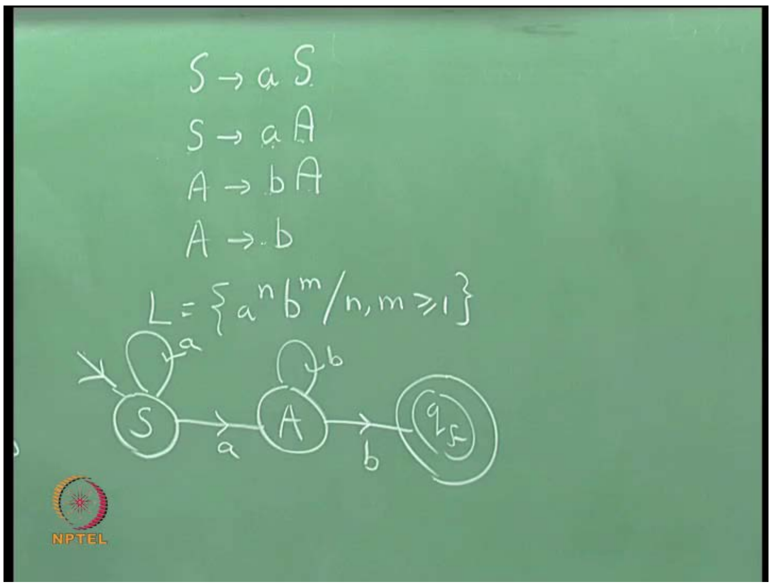
$$\begin{tikzcd}
\arrow[r] & S \arrow[loop above, "a"] \arrow[r, "a"'] & A \arrow[loop above, "b"] \arrow[r, "b"'] & \boxed{q_f}
\end{tikzcd}$$

So, that formal proof is as usual. So, I will omit the formal proof, but informally we will see the equivalence by a seeing the examples. Look at this example, S goes to a S, S goes to a A, A goes to b A, A goes to b. What is a language generated by this grammar? This is a regular grammar. Start with the start symbol S and the non terminal A and what is the language generated. You can have any number of a’s at least one a then followed by any number of b’s at least one b. So, the language is a power n b power m n, m greater than or equal to. Now, use this construction and construct the N S F A. If you use the construction it will be like this, for each non terminal you have a state, S is a non terminal you have a state, A is a non terminal you have a state, each non terminal you have a state and you have now final state q f.

Now, how do you write? If S goes to a A is a rule, delta of S a contains a isn’t it? By the construction, how we have if A goes to a b is a rule in P, delta of A, a contains B, not equal to b, because non deterministic. We write it as delta of A, a contains B. So, S goes to a S, is a rule. So, delta of S a will contain S that is from S to S there is an arc with label a then, S goes to A a. So, buy or construction delta of S a will contain A. So, there will be an arc with labeled a going from S to A and you have A goes to b b that is delta of A b will contain A. So, there is an arc with label b going from a to a then you have a rule a goes to b. It is a terminal rule.

For the terminal rule what is the mapping delta of a b contains a final state, q f this is non deterministic in nature, because two arcs with labeled a leaving S isn’t it? From S if you read, A can go to S or a possible non deterministic F S A and you can very easily see that it also accept the same language a power n, b power n, to the construction is very clear. If we have the rule, you draw the form. If you have a rule of the form A goes to a B draw an arc with label A from A to B. A will be represented as a state b, capital B will be represented as a state then there I will be an arc with label small a from A to B, that is all and if it is like this. There will be an arc with label small a from A to a final state that is only one final state q f that is all.

(Refer Slide Time: 11:44)


$S \rightarrow aS$

$S \rightarrow aA$

$A \rightarrow bA$

$A \rightarrow b$

$L = \{a^n b^m / n, m \geq 1\}$

$$\begin{tikzcd} \arrow[r] & S \arrow[loop above, "a"] \arrow[r, "a"'] & A \arrow[loop above, "b"] \arrow[r, "b"'] & \boxed{q_f} \end{tikzcd}$$

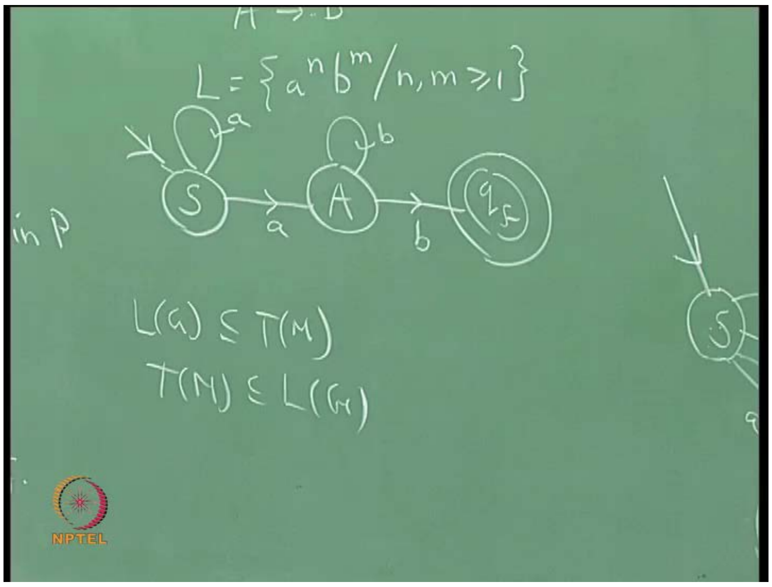
Let us consider example, ((no audio 11:42 to 12:25)) you have three non terminals S a and b, S is the start symbol the rules are of the form S goes to a A, S goes to b A and so on. What is the language accepted? The alphabet is a b, what is the language accepted or what is the language generated, what is the language generated?

Strings of (( )).

String of a's and b's but, the length is a multiple of three, the length of the string is a multiple of three. Write an example, constructed deterministic F S A over a and b satisfying two conditions, the string should begin with a and end with a b and a a a is not a (( )) isn't it. You consider an example like that and add one more condition, the length of string is divisible by three. Now, you construct the deterministic F S A, try it as a home work. Three conditions are there now, the strings begins with a and ends with a b, a a a is not a (( )), the length of the string is divisible by three. Now, let us construct the automaton by our construction, there are three non terminals. So, corresponding to each non terminal, we have a state. So, S is a state, A is a state, B is a state and there is a final state q f, S will be the initial state.

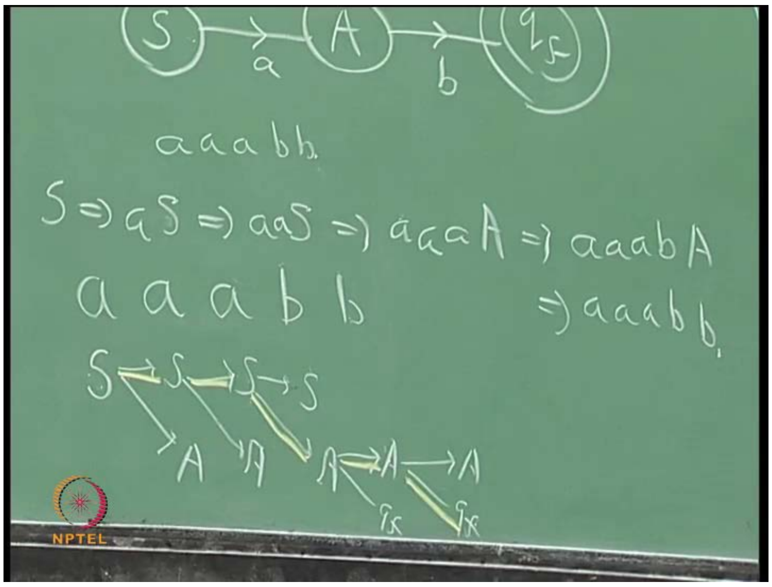
Now the mappings, you have to write S goes to a A, means there should be an arc with label a here, S goes to b A means that should be an arc with label b, A goes to a B there is an arc with label a, like this, there is an arc with label b, from b if you get a can go to S. So, a, b, the last two rules from b after getting a A can terminate. So, there will be an arc with label a, there will be an arc with label b. Note that this is a non deterministic F S A, but it will accept the same set of strings like the language generated by this grammar. In a or b, you can read by going from S to A, another a or b you can read, another a or b can take. So, going from S to S you will read a string of length three. Then again another one, two, three, you have to end up here. So, the string of length three will be accepted, we have assumed that epsilon is not there in the language, that is why this. If epsilon is in the language, initial state will be a final state.

(Refer Slide Time: 16:42)



So, grammar also you can see that it does not generate epsilon. So, this construction itself should be very, make it very clear that the language generated by the grammar and the language accepted by the N F S A are the same, isn't it. But a formal proof, if you are writing it as a theorem, you must give a formal proof. The formal proof will be you have to show that L G is contained in T of M and T of M is contained in L G and for this portion, you will use induction on the length of the string generated in the grammar, for this, you will use induction on the length of the three accepted by the measure. So, general induction proof. So, I will omit that, I will not go into that formal proof. Just for your, take this example, let us consider a a a b b. How is this generated? This is generated like S goes to a S, a S, a a S, a a a A, a a a b A, a a a b b, the generation is like this.

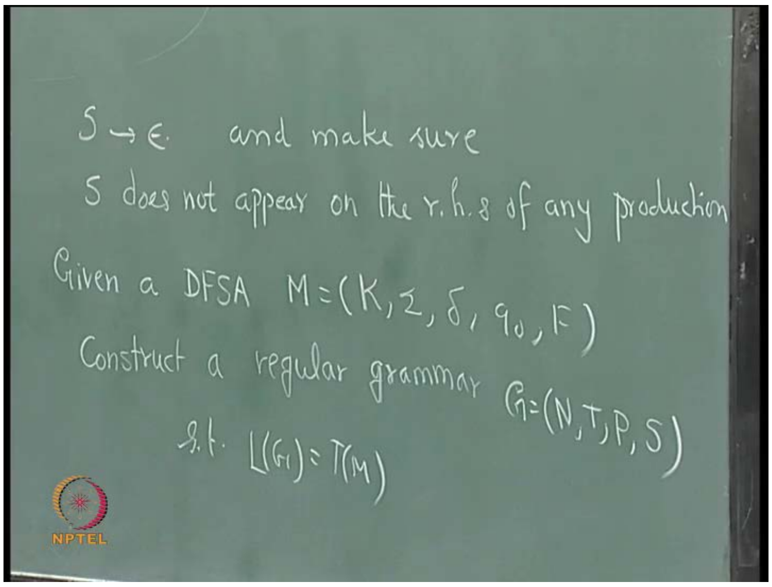
(Refer Slide Time: 17:24)



S
a
A
b
q_f
aaabb
$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaA \Rightarrow aaabA$
$\Rightarrow aaabb$
a a a b b

Now, if you look at the string and the acceptance you are in q naught, the initial state that is S, S then, S A can give you S r A, again S reading a you can get S r A, from S after reading a again you can get S r A, from A S you cannot read a b from A if you read a b you will go to A or q f, from A again if you read a b you can go to A or q f. Now, the sequence of stage which leads you to acceptance is this, this is the sequence which leads you to acceptance, look at a non terminals S, S, S, A, A, of course, the state is q f you can see S,S, S, A, A, and it is q f it is not that that is all final. So, see the correspondence the way it is generated it is accepted in the sequence. That is the idea that that portion formally you have to prove by induction. So, we will leave that, but examples make it very clear. Now, the other way around, we have to prove the other way around, what is the other way around? Given, now, we take a D F S A, given a deterministic F S A, M is equal to K sigma delta q naught F, construct a regular grammar G is equal to N T P S, such that L of G is equal to T of M.

(Refer Slide Time: 20:07)



$S \rightarrow \epsilon$ and make sure

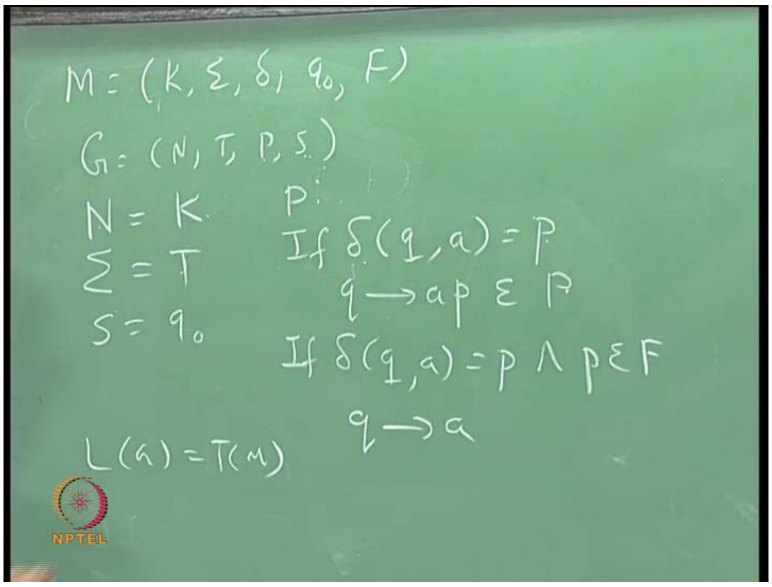
S does not appear on the r.h.s of any production

Given a DFSA $M = (K, \Sigma, \delta, q_0, F)$

Construct a regular grammar $G = (N, T, P, S)$

s.t. $L(G) = T(M)$

(Refer Slide Time: 21:41)



$M = (K, \Sigma, \delta, q_0, F)$

$G = (N, T, P, S)$

$N = K$

$\Sigma = T$

$S = q_0$

P:

If $\delta(q, a) = p$

$q \rightarrow ap \in P$

If $\delta(q, a) = p \wedge p \in F$

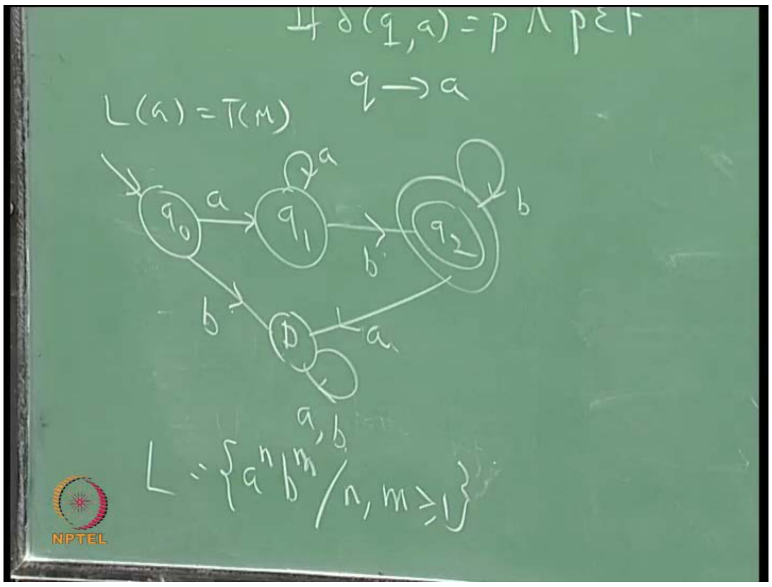
$q \rightarrow a$

$L(G) = T(M)$

Now, we start with a deterministic automaton and construct a grammar. Again, we assume that epsilon is not accepted, I will come to the case, it is just you have to make a some slight adjustment, that is all. We will take more example with epsilon. So, you are given M is equal to K sigma delta q naught F, you are constructing the grammar, G is equal to N T P S. How do you? N is the same as K, for each state, you have a non terminal each state. For each state, you have a non terminal and set of terminal symbols are the same as the set of input symbols, isn't it?

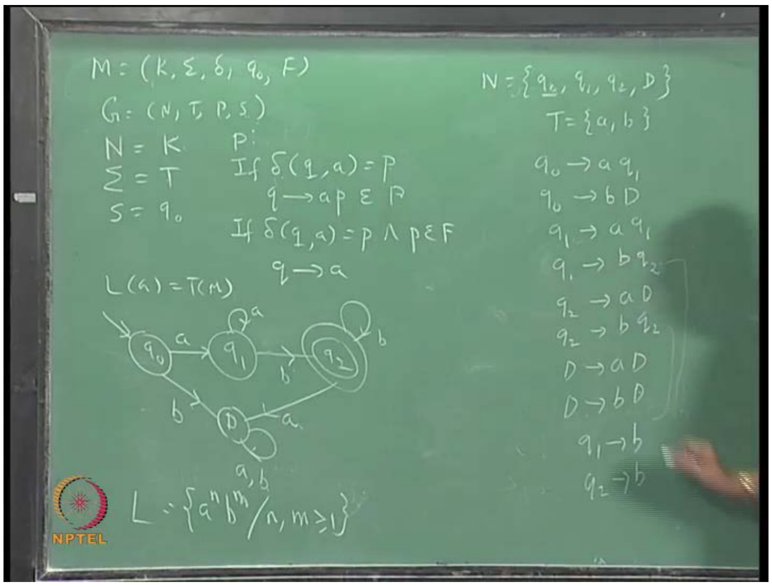
Now, what is q naught, you have to write P and S, S corresponds to the initial state q naught and P we have to write, how do you write P? If delta of q a is equal to P, I am writing q a is equal to P not contains by, because we are starting with the deterministic machine, q goes to a P is a rule, q goes to a P will be a rule. Also, So, we are able to write the non terminal rules, there will be some terminal rules is it not? How do you write them? Apart from that apart from that if delta of q a, is equal to P and P belongs to F, P is also a final state, then you write the rulers q goes to a.

(Refer Slide time: 24:12)



So, let us this is you have written N T S P we have written and in this case L G will be equal to T of M. So, here L G will be equal to T of M. How do you? Let us consider examples again. The proof will be, formal proof will be by induction. Take this one, we had it, for the same language L this is the deterministic machine isn't it? What is L? Language is a power n, b power m, n m. Now, we will write the grammar for this q naught corresponding the set of non terminals will be q naught, q 1, q 2, D, this will be the set of non terminals. Terminal symbols will be a, right. Now, we have to write the rules. Of course, q naught is the start symbol, this is the start symbol. How do you write the rules? Now, q naught a goes to q 1. So, you have a rule q naught goes to a q 1, q naught goes to b D. q 1 a, q 1, isn't it or delta of q 1, A is q 1. So, q 1 goes to a q 1, q 1, goes to b q 2, q 2 goes to a D, q 2 goes to b q 2, D goes to a D, D goes to b D, these are the non terminal rules.

(Refer Slide Time: 25:23)



$M = (K, \Sigma, \delta, q_0, F)$

$G = (N, T, P, S)$

$N = K$ P:

$\Sigma = T$ If $\delta(q, a) = p$

$q \rightarrow ap \in P$

$S = q_0$ If $\delta(q, a) = p \wedge p \in F$

$q \rightarrow a$

$L(G) = T(M)$

$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, "a"] \arrow[dr, "b"'] & q_1 \arrow[loop above, "a"] \arrow[r, "b"] & q_2 \arrow[loop above, "b"] \arrow[dl, "a"] \\
& & D \arrow[loop below, "a,b"] &
\end{tikzcd}$$

$L = \{a^n b^m / n, m \geq 1\}$

$N = \{q_0, q_1, q_2, D\}$

$T = \{a, b\}$

$q_0 \rightarrow a q_1$

$q_0 \rightarrow b D$

$q_1 \rightarrow a q_1$

$q_1 \rightarrow b q_2$

$q_2 \rightarrow a D$

$q_2 \rightarrow b q_2$

$D \rightarrow a D$

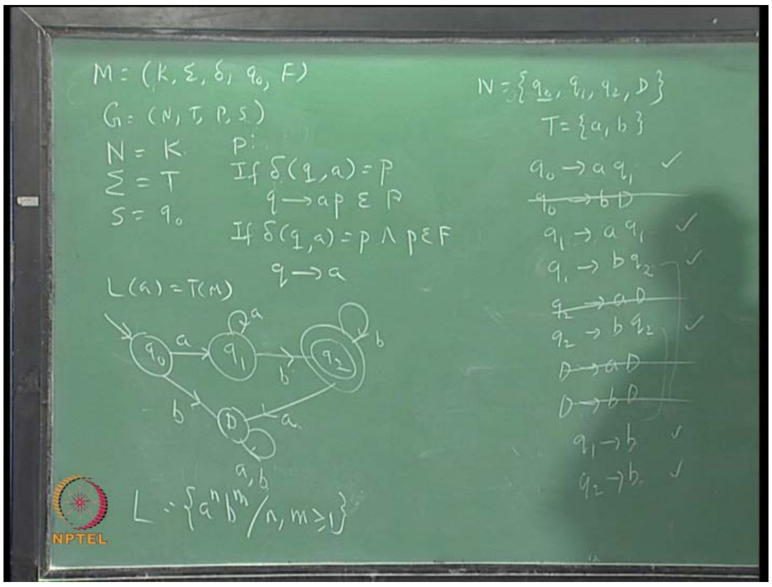
$D \rightarrow b D$

$q_1 \rightarrow b$

$q_2 \rightarrow b$

Now, terminal rules wherever, you have q 2 here, q 2 is a final state here. Wherever, you have q 2 here and here corresponding to these two rules, you will have terminal rules that will be q 1 goes to b, q 2 goes to b, but you will realize that we have consider removal of useless symbol, it is the useless symbol here, D is the useless symbol. So, any rule involving D, D is useless so, anything involving D you can remove. So, you will end up with these six rules. So, you can generate one a and any number of a’s we can generate with this rule, one b you will generate and any number of these we can generate. We can terminate with this or this, if we terminate with this there will be only one b then after generating many b’s if you want to generate you want to terminate, you will use this.

(Refer Slide Time: 27:26)



$M = (K, \Sigma, \delta, q_0, F)$

$G = (N, T, P, S)$

$N = K$ $\quad$ P:

$\Sigma = T$ $\quad$ If $\delta(q, a) = p$

$S = q_0$ $\quad$ $q \rightarrow ap \in P$

If $\delta(q, a) = p \wedge p \in F$

$q \rightarrow a$

$L(G) = T(M)$

$L = \{a^n b^m / n, m \geq 1\}$

$N = \{q_0, q_1, q_2, D\}$

$T = \{a, b\}$

$q_0 \rightarrow a q_1$ ✓

~~$q_0 \rightarrow bD$~~

$q_1 \rightarrow a q_1$ ✓

$q_1 \rightarrow b q_2$ ✓

~~$q_2 \rightarrow aD$~~

$q_2 \rightarrow b q_2$ ✓

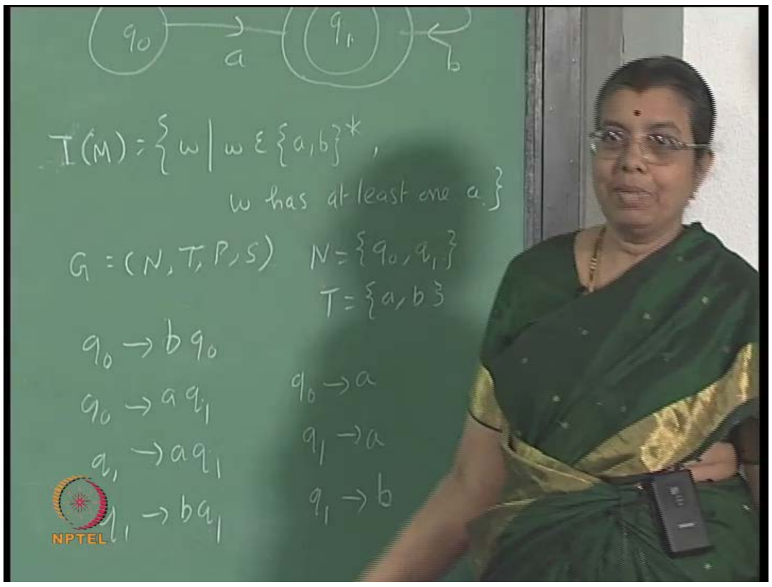
~~$D \rightarrow aD$~~

~~$D \rightarrow bD$~~

$q_1 \rightarrow b$ ✓

$q_2 \rightarrow b$ ✓

(Refer Slide Time: 28:44)



$T(M) = \{w \mid w \in \{a, b\}^*,$ w has at least one a$\}$

$G = (N, T, P, S)$ $\quad$ $N = \{q_0, q_1\}$

$T = \{a, b\}$

$q_0 \rightarrow b q_0$ $\quad$ $q_0 \rightarrow a$

$q_0 \rightarrow a q_1$ $\quad$ $q_1 \rightarrow a$

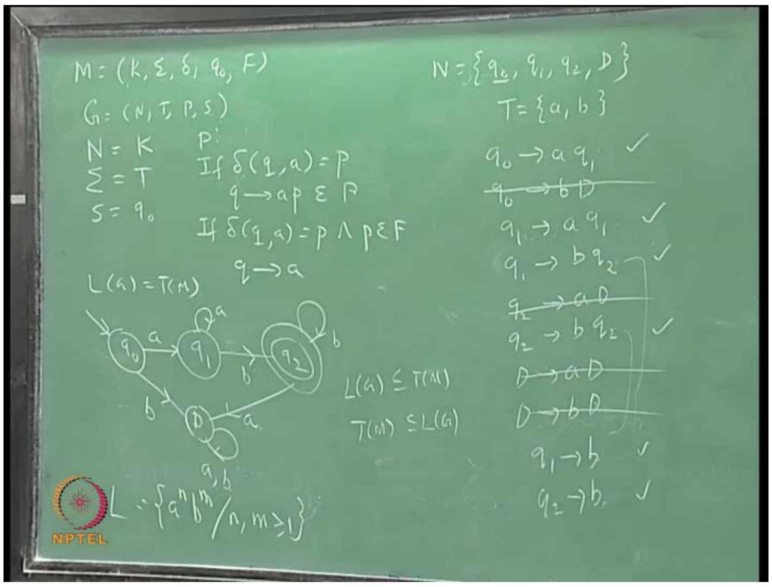
$q_1 \rightarrow a q_1$ $\quad$ $q_1 \rightarrow b$

$q_1 \rightarrow b q_1$

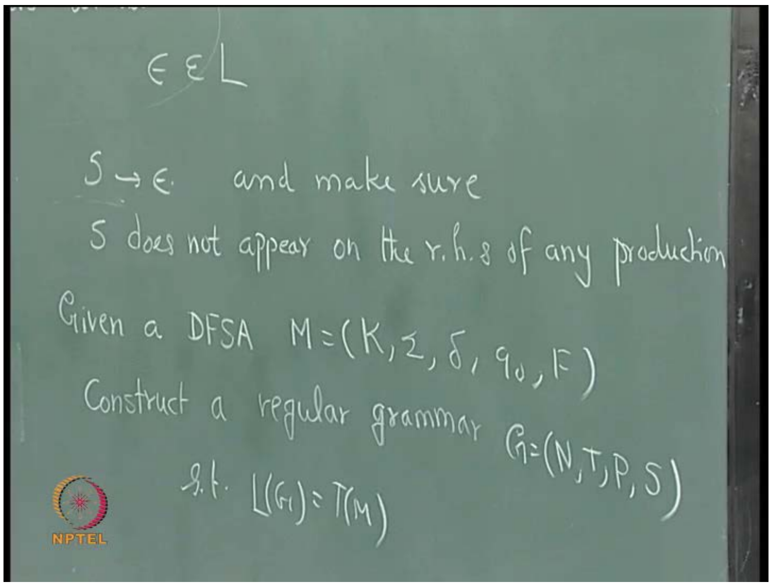
So, the language accepted is a language generated by the grammar is a same as the language accepted by the machine. One more example, we will consider, this is the deterministic machine. What is the language accepted? The language accepted should have at least one a, any number of b’s you can have and one a you should have, afterwards, we can have anything. So, the language accepted should have a minimum one a. So, L G, T of M is w, w belongs to a, b star, w has at least one a. Now, let us write the grammar for this G is equal to N T P S, where the set of non terminals is the states q naught, q 1, T of course is a b, q naught will be the start symbol and the rules are like this q naught goes to b q naught, q naught goes to a q 1, q 1 goes to a q 1, q 1 goes a q 1, q 1 goes to b q 1, q 1 goes to b q 1, q 1 is the final state.

So, corresponding to these three rules, you must also have a terminal rule. So, we will have q naught goes to a, here you will have q 1 goes to a, you will also have q 1 goes to b. We can very easily check this, here the language generated and the language accepted are the same, there are no useless symbol. Roughly you can see that the death state will correspond to a death state or state from which you cannot go to a final state will become a useless symbol in the grammar, useless non terminal. Again the formal proof you have to show that L G is contained in T of M and T of M is contained in L of G which is done by using induction on the length of the strings generated and accepted.

(Refer Slide Time: 32:02)



(Refer Slide Time: 32:36)



$\epsilon \in L$

$S \to \epsilon$ and make sure

S does not appear on the r.h.s of any production

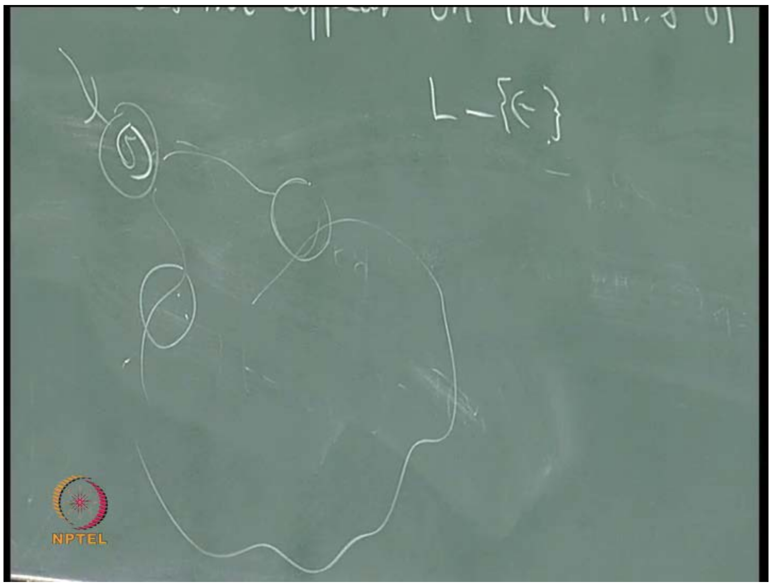
Given a DFSA $M = (K, \Sigma, \delta, q_0, F)$

Construct a regular grammar $G = (N, T, P, S)$

s.t. $L(G) = T(M)$

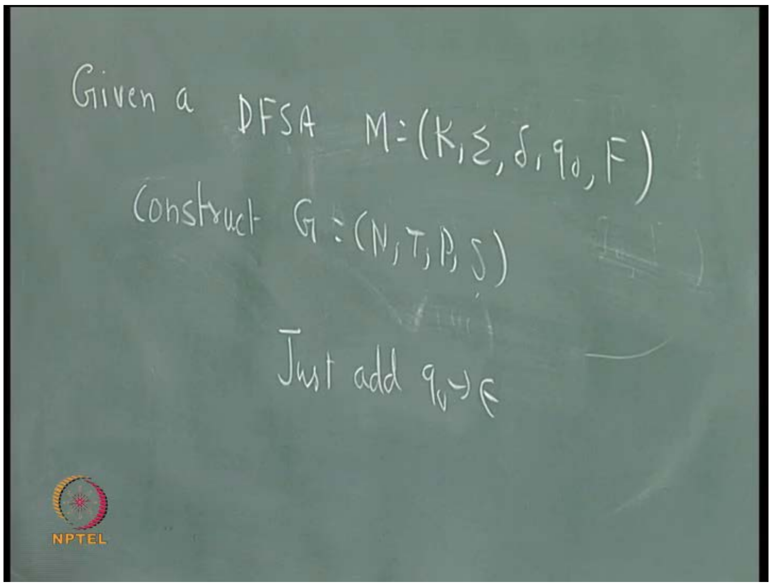
Now, the problem with epsilon. So, the next thing, what happens if you have epsilon belong to, when the empty string is also accepted by the machine or generated by the language. How you have to change? Now, first step will be given a regular grammar construct an N F S A such that T M S equal to L G, this is the way we constructed isn't it?

(Refer Slide Time: 32:25)



Now in this case the grammar, we take has this property. If you want to include epsilon, only S goes to epsilon is the rule and S does not appear on the right hand side of any production. So, the diagram which you construct will be something like this. In this case, if you want to accept a, from here you will go to some other state, but you will not come back to S isn't it? You will not back to S, the reason is S does not occur on the right hand side of any production. So, any string in L minus epsilon, if you take that will be accepted, the usual way you construct and it will be accepted, for epsilon make this a final state, make the initial state a final state. So, epsilon will be accepted. This does not affect the other portion, because you do not come back to epsilon I am sorry you do not come back to S. Now, the other way around, given D F S A, M is equal to K sigma delta q naught F, construct G is equal to N T P S. Here the q naught is S, you have to just add the rule q naught goes to epsilon. If you add the rule q naught goes to epsilon, the other construction will not be affected, only just you have to add this, but you can also make sure that in this case q naught may occur on the right hand side of a production, you know also make sure that that does not happen.

(Refer Slide Time: 34:33)

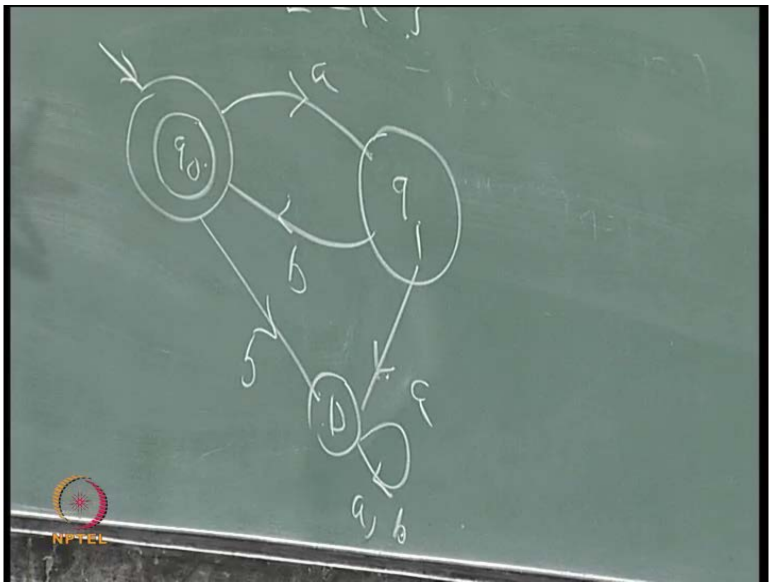


Given a DFSA $M:(K, \Sigma, \delta, q_0, F)$

Construct $G:(N, T, P, S)$

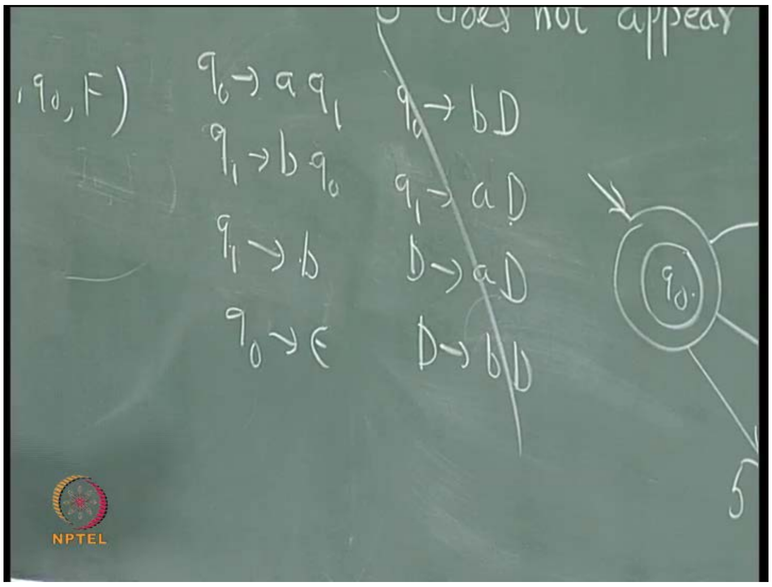
Just add $q_0 \to \epsilon$

(Refer Slide Time: 36:22)

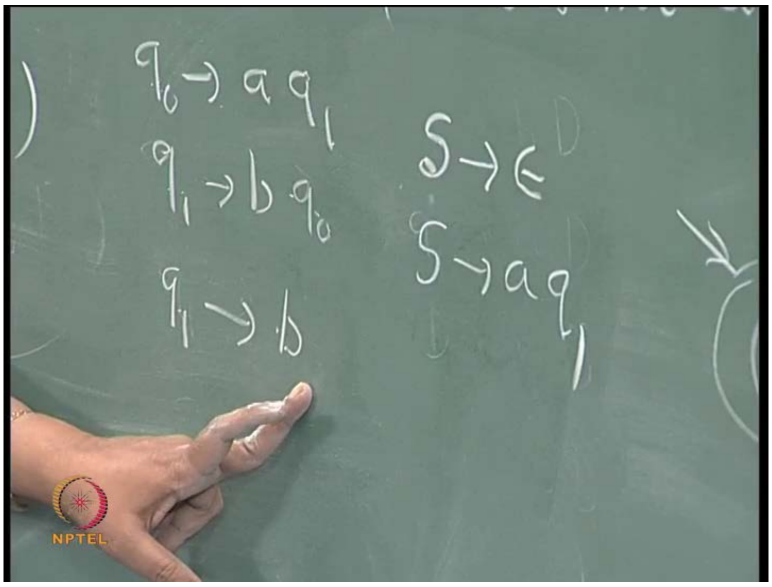


So, just I will put see if the D F A, D F S A accepts epsilon, initial state will be a final state and in that case just add, just add q naught goes to epsilon, nothing else will be affected, but the it will not satisfy this property, isn't it. If you want that property also to be satisfied you have to do something else again I will illustrate with an example. Consider this automaton a b, a b, a b, strings of the form a b, a b, a b, will be accepted, epsilon will also be accepted, other strings will be rejected by going to the dead state. If you want to write I will not tell the non terminals and all just the rules alone, if you write q naught goes to a q 1, q naught goes to b D, q 1 goes to, I will write the D rules in this side. So, the later on, you can remove b D, q 1 goes to b q naught, q 1 goes to a D, D goes to a D, D goes to b D, then q naught is the final state. So, you will also have q 1 goes to b, this you will have. Now, your initial state is the final state. So, what do you have q naught goes to epsilon you can add.

(Refer Slide Time: 37:07)

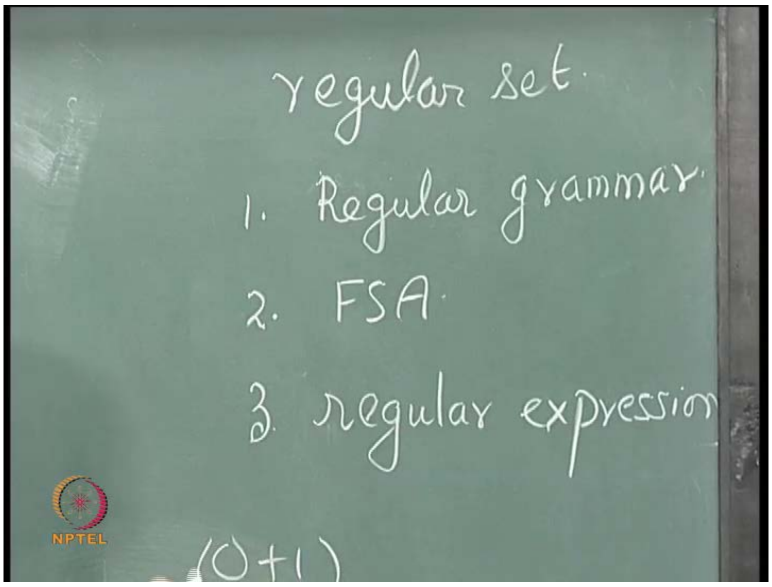


(Refer Slide Time: 38:40)



Now, once you go to D you cannot derive a terminal string. From D you cannot derive a terminal string. So, all these D is a useless non terminal, all these rules can be removed. So, I will rub this off. Now, the start symbol q naught goes to epsilon, but it is occurring on the right hand side. In order to avoid that what do you do? You can add a new start symbol S and have S goes to epsilon and this you will remove, then whatever is there with q naught you also have with S that is, this is a rule with q naught on the left hand side, you also have S goes to a q 1, this S will be the new start symbol epsilon is derived, any other string a b will be derived by S goes to a q 1, q 1 goes to b, a b, a b, will be derived using this, this, this, this. You have to just make a slight adjustment that is all. So, given regular grammar, you can construct an equivalent automaton and given an equivalent even a F S A can construct a grammar.

(Refer Slide Time: 40:17)

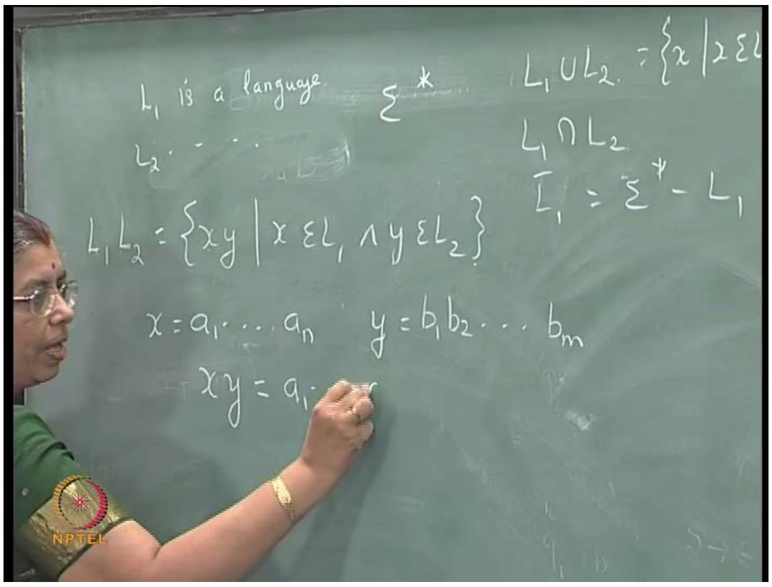

regular set.

1. Regular grammar.
2. FSA.
3. regular expression

(0+1)

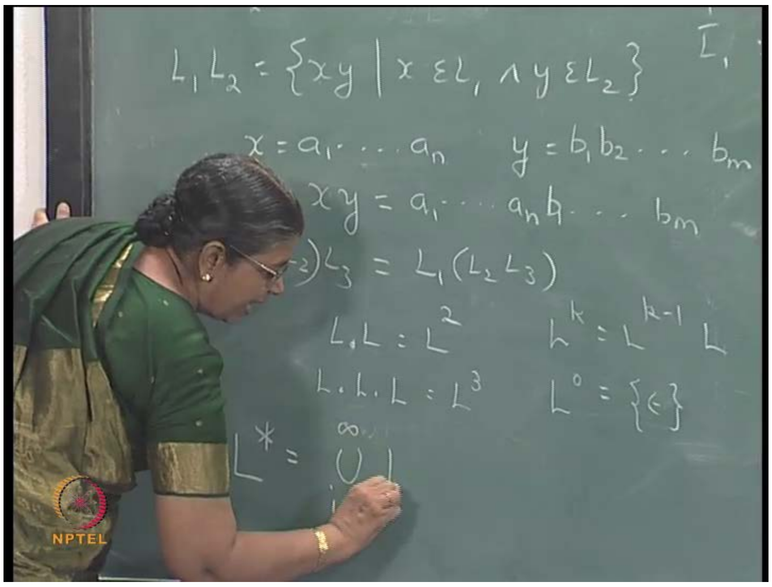
Now, the way we went about this given a grammar we constructed a N F S A, but we know that given an N F S A you can construct an equivalent D F S A by subset construction and in other way around given a D F S A we constructed the grammar. So, a regular set there are two ways, if you consider a regular set, one way is a regular grammar. You can use a regular grammar to generate it, another is you can use a F S A to accept it. This is a generative device, this is an acceptance device. F S A you can have with epsilon moves, without epsilon n, we can have a deterministic F S A, you can have non deterministic F S A without epsilon moves. A non deterministic F S A with epsilon moves, all three are possible. They are all equivalent.

(Refer Slide Time: 42:01)


L_1 is a language
L_2 . . . .
Σ^*
$L_1 \cup L_2 = \{x \mid x \in L$
$L_1 \cap L_2$
$\bar{L}_1 = \Sigma^* - L_1$
$L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$
$x = a_1 \ldots a_n$ $y = b_1 b_2 \ldots b_m$
$xy = a_1$
NPTEL

There is one more way of representing a regular set that is by means of what is known as a regular expression. Informally, we have used expressions of the form 0 plus 1 star, a a star like that, they are all relay regular expressions. How do we formally define a regular expression. Before that let us just recall some operations on sigma star, we have studied earlier the previous course L 1 is a language, L 2 is also a language over sigma star. Sigma star, sigma is alphabet, sigma star denote the set of all strings over sigma including the empty string. And L 1 and L 2 are languages and we just mean set of strings, then the following operations you can define L 1 union L 2, because they are sets, sets operations can be defined, L 1 union L 2 will be x, x belongs to L 1 or x belongs L 2. The usual set theorem operation, you can also define L 1 intersection L 2, L 1 bar is sigma star minus L 1. The set theoretic operations you can define on this. Apart from that if you remember we also defined something else earlier L 1, L 2 the concatenation of L 1 and L 2. What is this? This is a set of strings of the form x y, x concatenated with y, x belongs to L 1 and y belongs to L 2. What is a concatenation? If x is equal to a 1, a 2, a n and y is equal to b 1, b 2, b m, these are two strings, then the concatenation of x and y is denoted as x y and that is a 1, a 2, a n, b 1, b 2, b n, this is a concatenation of two strings. The languages, for languages it is defined like this, what can you say about the operator concatenation? Is it commutative? Is it commutative? No. It cannot be commutative, x y cannot be equal to y x, is it associative? Concatenation is associative, right?

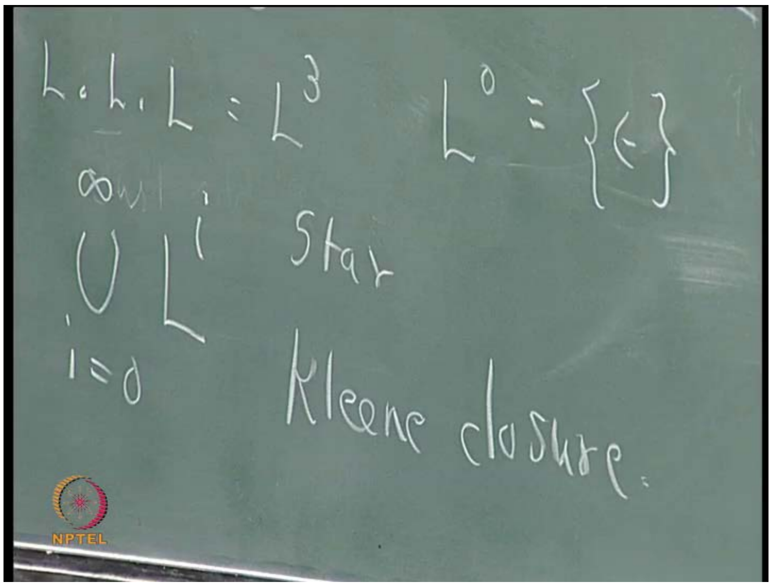
(Refer Slide Time: 44:23)



$L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$
$x = a_1 \cdots a_n$ $y = b_1 b_2 \cdots b_m$
$xy = a_1 \cdots a_n b_1 \cdots b_m$
$)L_3 = L_1(L_2 L_3)$
$L.L = L^2$ $L^k = L^{k-1} L$
$L.L.L = L^3$ $L^0 = \{\epsilon\}$
$L^* = \bigcup^{\infty}$

So, without any difficulty, you can L 1, L 2, L 3 will be the same as L 1, L 2, L 3 concatenation is not commutative, but it is associative. So, what do you mean by L n t L usually denoted as L squared and you can, without any problem, you can talk about this, either you group this way or you group this way, it does not matter, because of associative. So, this is L cubed. In general L power k is L power k minus 1 concatenated with L. What is L 0? This also we studied if you remember in the last semester, it just contains the empty string. So, L star, how do you define L star? This is union of i is equal to 0 to infinity L. The kleene closure star operation or kleene closure or sometimes just denoted as closure, L star is union of i is equal to 0 to infinity L power i. What is L plus union of i is equal to 1 to infinity, these things we have studied earlier for regular expression we use this.

(Refer Slide Time: 45:29)

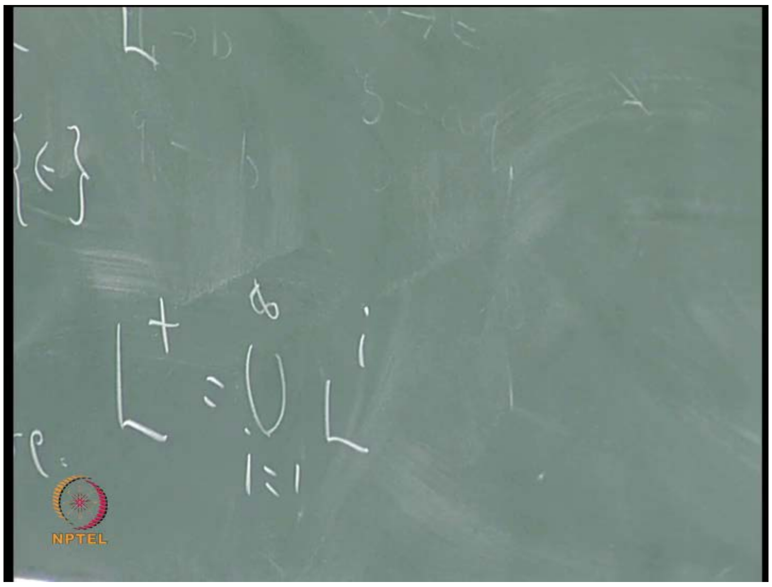


$L.L.L = L^3$ $L^0 = \{\epsilon\}$

$\bigcup_{i=0}^{\infty} L^i$ Star

Kleene closure.

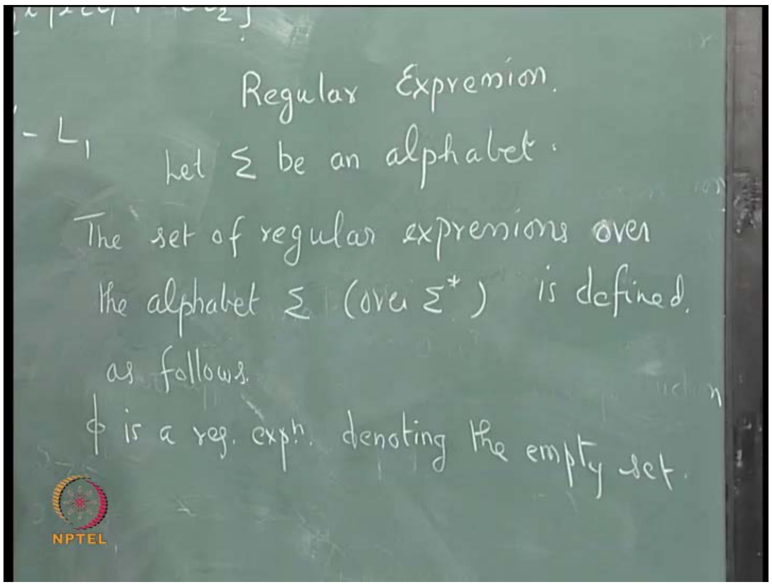
(Refer Slide Time: 46:01)



$\{\epsilon\}$

$L^+ = \bigcup_{i=1}^{\infty} L^i$

(Refer Slide Time: 46:25)



Regular Expression.

$-L_1$

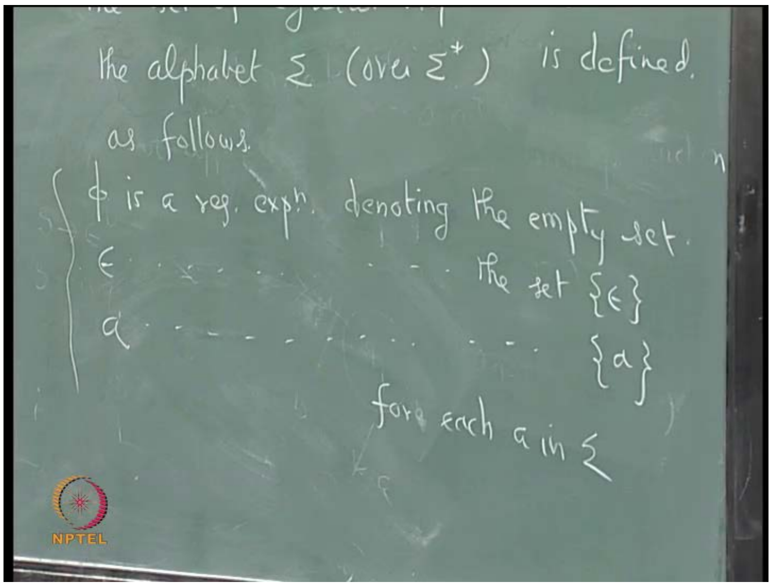
Let Σ be an alphabet.

The set of regular expressions over the alphabet Σ (over Σ^*) is defined as follows.

ϕ is a reg. exp^n. denoting the empty set.

Consider the alphabet, let sigma be an alphabet, sigma is an alphabet. phi the set of regular expression over sigma star is defined. The set of regular expressions on over the alphabet sigma over the alphabet sigma or you can say over sigma star is defined as follows. phi is a regular expression denoting the empty set.

(Refer Slide Time: 48:16)



the alphabet Σ (over Σ^*) is defined as follows.

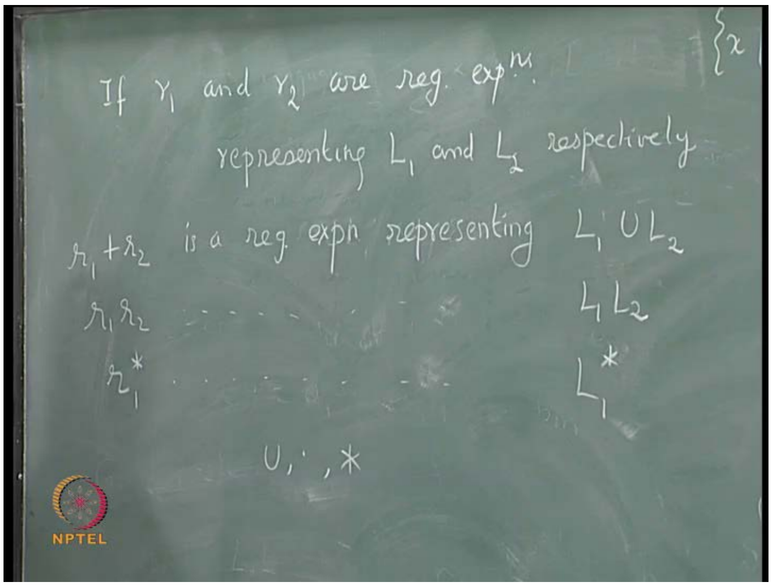
ϕ is a reg. exp^n. denoting the empty set.

ϵ . . . . the set $\{\epsilon\}$

a . . . . $\{a\}$

for each a in Σ

(Refer Slide Time: 49:19)


If r_1 and r_2 are reg. exp^{ns}.

representing L_1 and L_2 respectively

$r_1 + r_2$ is a reg expn representing $L_1 \cup L_2$

$r_1 r_2$ . . . . $L_1 L_2$

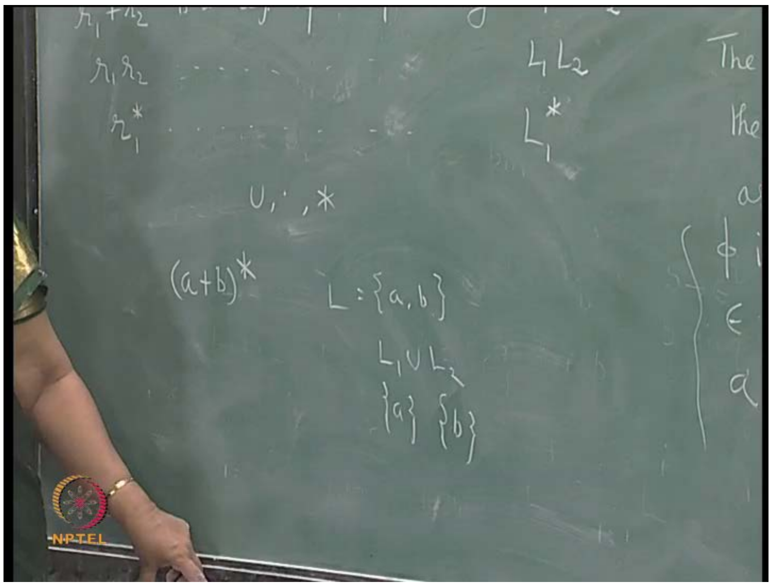
r_1^* . . . . L_1^*

$\cup, \cdot, *$

$\{x$

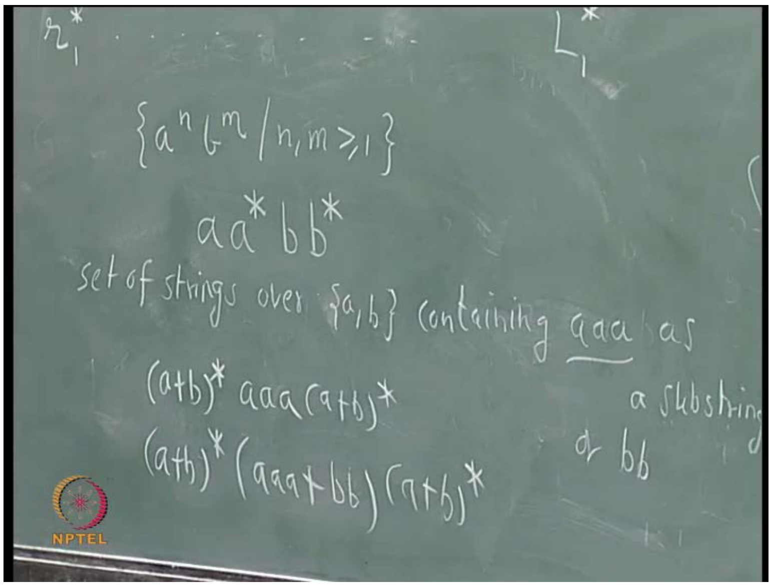
We have defined or we have considered inductive definition of sets. So, here it is, we are making use of inductive definitions. These are bases class, phi is a regular expression denoting the empty set, epsilon is a regular expression denoting the set having the empty string (( )) epsilon is a regular expression and it denotes the empty string (( )), a is a regular expression denoting the set a, having the single string a for each a in sigma, for each a in sigma, this is the basis portion, the inductive class is if r 1 and r 2 are regular expressions representing L 1 language L 1 regular sets L 1 and L 2 respectively, r 1 plus r 2 is a regular expression representing L 1 union L 2, r 1 plus usually is r union is defined like that L 1 union L 2 is x belongs to L 1 or a x belongs to L 2, r 1 r 2 concatenated with this is a regular expression representing the concatenation of L 1 and L 2, r 1 star is a regular expression representing L 1 star. We make use of these three operators, union concatenation and stars usually this dot you do not write explicitly. It is the implicitly understood, then the (( )) will come. All regular expressions are obtained in this manner and so on, and nothing else will be obtained.

(Refer Slide Time: 51:23)



So, this is another example of a inductive definition of a sets. For example, you consider a plus b star. What does that mean? Set of all strings over a and b including epsilon. This is of the form L star, L itself has two strings a and b. L has, suppose the L has a, b L 0 is epsilon, L is this, L squared will contain all strings of length two, L cubed will contain all strings of length three and so on. So, union over i is equal to 0 to infinity contains all strings are over a n.

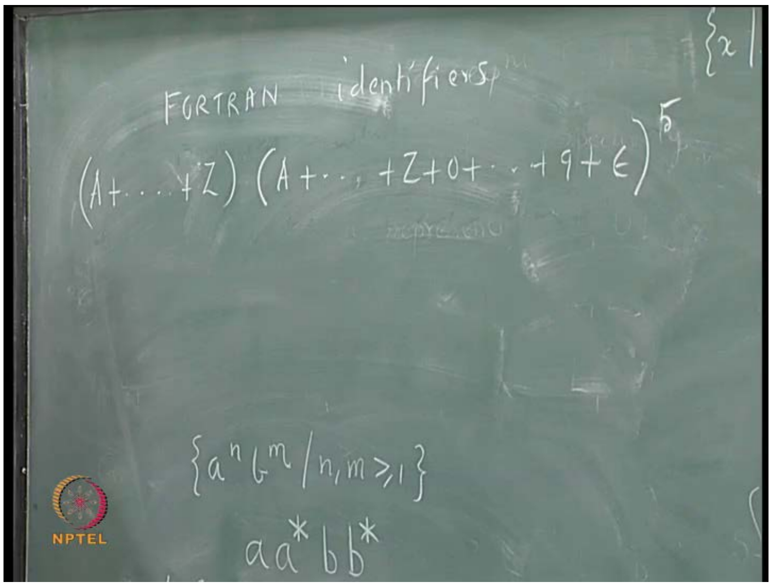
(Refer Slide Time: 52:53)


r_1^*
L_1^*
$\{a^n b^m / n, m \geq 1\}$
$aa^* bb^*$
Set of strings over $\{a, b\}$ containing $\underline{aaa}$ as a substring
$(a+b)^* aaa (a+b)^*$
$(a+b)^* (aaa + bb) (a+b)^*$
or bb

Now, this L itself we can look at it as L 1 union L 2 isn't it? This is just a, this is just b, that is why this is represented as a plus b. So, from the base things by having one operation like this, one operation like this you are able to get the set. Some more examples, we have considered this a power n, b power m, n m several times. We have considered this example, what will be the regular expression for this? What will be the regular expression there should be at least one a, any number of a's you can have, one b, any number b's you can have. So, it will be a a star b b star, the regular expression corresponding to this is, the set of strings over a and b where a a a should occur as a substrate.

Where a a a set of strings over a b containing a a a as the substrate. What will be the regular expression? You must, you can have any string proceeding the three a's three a should occur somewhere in the middle. So, in the middle you must have three a's, it can be followed by anything. The regular expression for that will be like this, change this a little bit, set of strings over a b containing three a's or two b's. It should contain three a's or two b's in the middle. What will be the expression? It will be a anything you have in the beginning, but in the middle you will have three a's or two b's. Then it can be followed by input. How will you represent the fortran identifiers? Fortran identifiers, it is not now used, but in any way for example, sake fortran identifiers. The length will be six, it will begin with a letter and after words you can have letter or a digit.

(Refer Slide Time: 55:34)


FORTRAN identifiers
$(A+\cdots+Z)(A+\cdots+Z+0+\cdots+9+\epsilon)^5$
$\{a^n b^m / n,m \geq 1\}$
aa^*bb^*

So, the expression for will be A plus B up to Z, first has to be a letter then followed by a letter or a digit or nothing else, you can just have A B alone also isn’t it. So, this will be A plus Z plus 0 plus up to 9 plus epsilon and you can go up to length six. So, this power so, we have now defined? What is a regular expression? Regular expression represents a regular set that we have to prove. So, given a regular expression we shall see how to construct a non deterministic F S A with epsilon moves next class. So, from a regular expression, we can go to a non deterministic F S A with epsilon moves then we know how to remove the epsilon transitions and get a N F S A, from N F S A how to go to D F S A by subset construction. Then, the other way around given a deterministic finite state automaton, how to find the equivalent regular expressions for this?

Meanwhile, I want to think on two points, there are three set theoretic operations, we considered union, intersection and compliment. So, try to prove that if L 1 and L 2 are regular sets, L 1 union L 2 is a regular set, L 1 intersection L 2 is a regular set, L 1 bar is a compliment of L 1 is a regular set. How will you prove this? Think over this, we will consider it later.