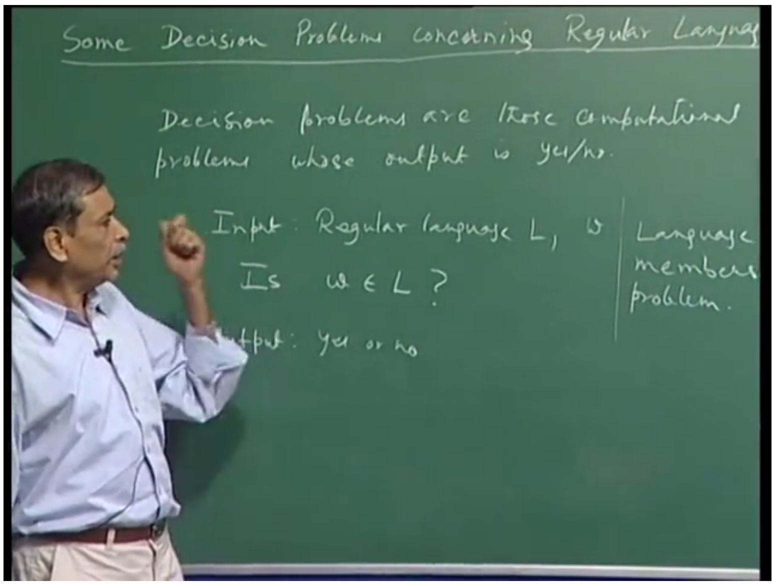
Theory of Computation.

Professor somenath Biswas.
Department of Computer Science & Engineering.
Indian Institute of Technology, Kanpur.
Lecture-15.
Decision Problems for Regular Languages.

(Refer Slide Time: 0:19)



Some Decision Problems concerning Regular Languag

Decision problems are those computational problems whose output is yes/no.

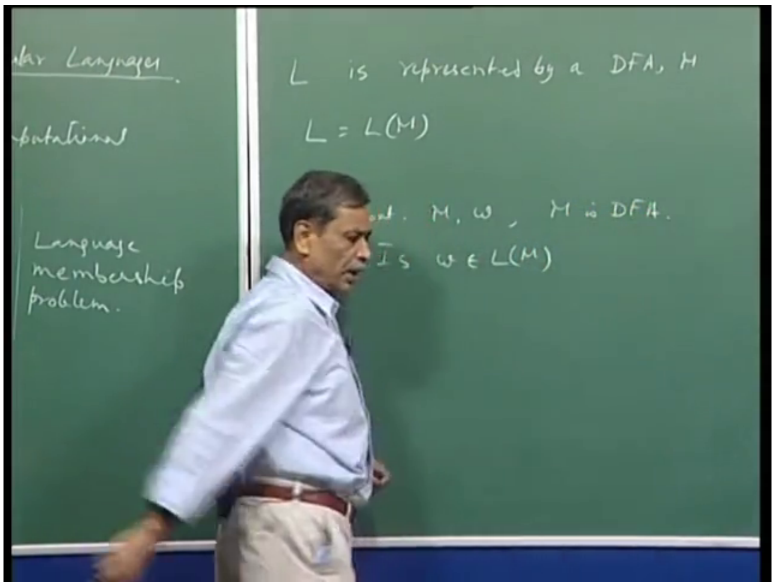
Input: Regular language L, w
Is $w \in L$?
tput: yes or no

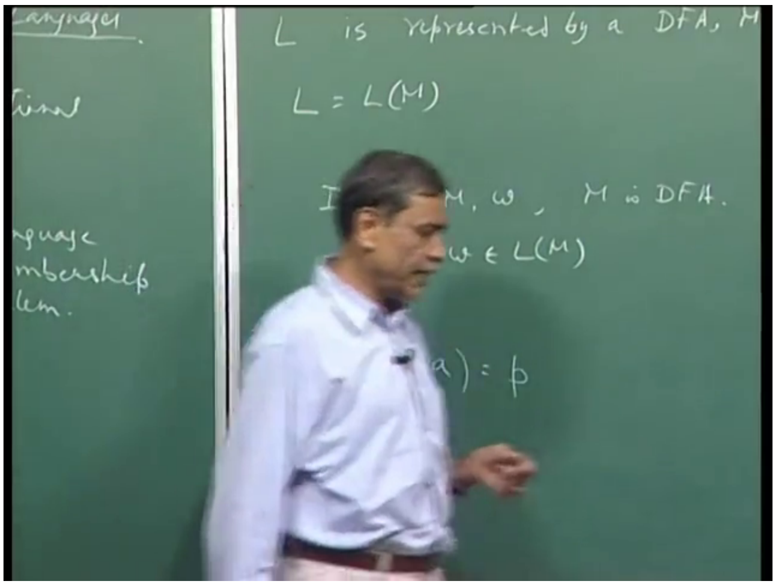
Language membe problem.

Let us discuss some decision problems concerning regular language. What are decision problems? Decision problems are those computational problems whose output is yes or no. So you know many decision problems, for example given a graph, if the draft corrected. So in decision problems there will be some input and output will be either yes or no. That is, in that sense it is a decision problem, we need to decide something. And in these problems today which we will look at, the decision problems, there is the input is going to be a regular language.

And we may ask, for example given a regular language does it satisfy certain properties, then the answer is yes or no, that these are the kind of questions that we will discuss. In fact the most important decision problem concerning regular languages is that, you are given 2 things a regular language L and a string W and you are supposed to decide this question, is the string W in the language, the answer is either yes or more. So this is called language membership problem. We are asking whether the string W is a member of the language here. And that is what we need to decide.

(Refer Slide Time: 4:32)



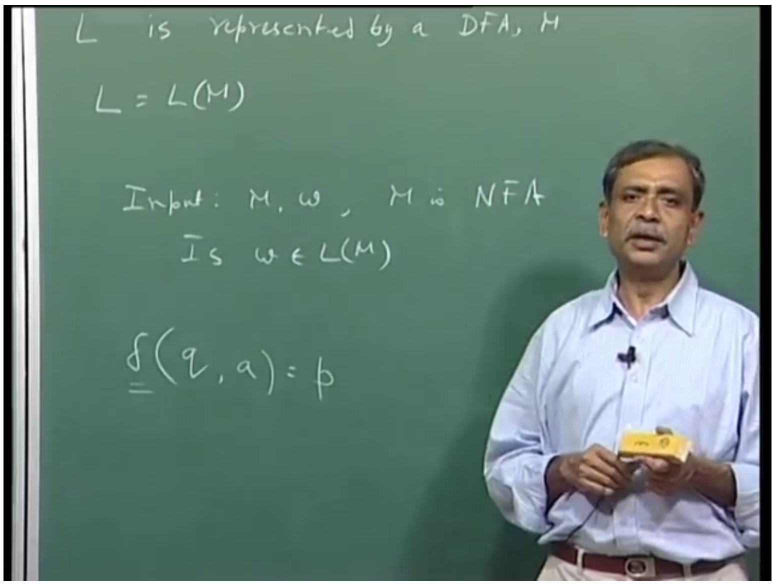


Now our algorithm of course will depend on how we represent the regular language. And now you know regular languages can be represented, can be given in several ways, either in terms of a DFA, NFA or an NFA with Epsilon transitions or through a regular expressions. So therefore we will have different algorithms depending on the way we would present the regular language. So the 1st case is the simplest case, let us take the simplest case that L is represented by a DFA say M. In other words what we are saying is that this language L is the language accepted by the DFA M. And now the problem boils down to your input is M and W and you would like to know is W accepted by the DFA M, let me take M is the DFA.

This algorithm is rather straightforward, you can see what we can do, our algorithm can just simulate the actions of DFA and what is that, it looks at the string W symbol by symbol starting from the leftmost symbol and it keeps track of the state in which the DFA is after scanning some prefix of the input W. And supposing at some point the DFA is in M at some state Q and the symbol, next symbol in the input is small a, so at some point of time the state is Q, the symbol is a and I know the next state is going to be whatever is the p according to the transition function of a.

So this part of the input is consumed, the symbol of the input is consumed, next state is P and this way we carry out the simulation till the string is over and at that time if the state is another final states of the machine M, then the string is accept it, we give the answer yes, it is in the language, otherwise we say the string is not in the language and therefore the answer is no. It is therefore quite straightforward the algorithm for membership decision problem when the language is given as a DFA. What about if it is an NFA?

(Refer Slide Time: 6:28)



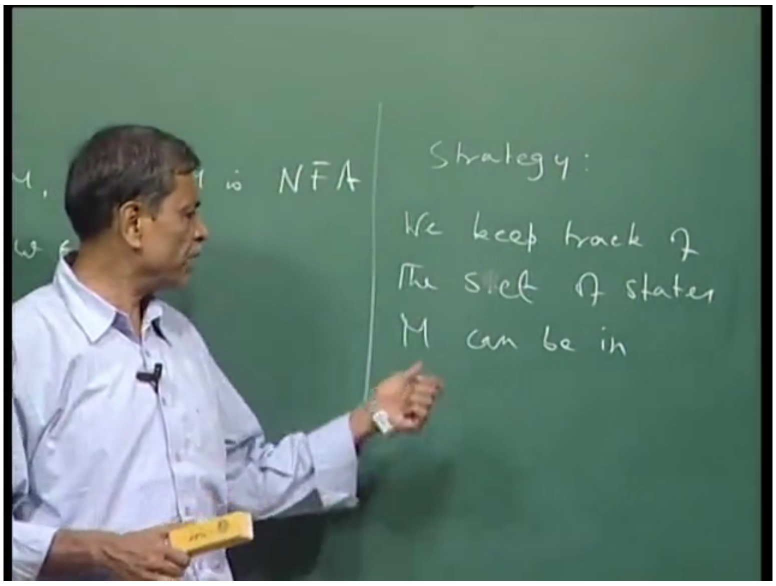
L is represented by a DFA, M

$L = L(M)$

Input: M, ω, M is NFA

Is $\omega \in L(M)$

$\underline{\delta}(q, a) = p$

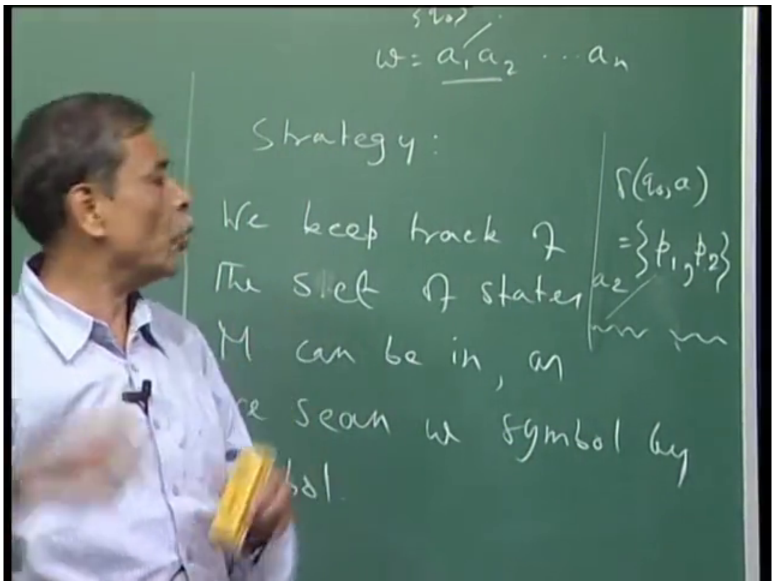


And here when I say NFA, I do not let say the NFA M does not have Epsilon transitions. What you could do is 1st of all convert that NFA to an equivalent DFA and then use our own algorithm. But the problem with this is we know that if this NFA has n states, the description of the NFA, it is part of the input and this n is a parameter, is a size parameter, is one of the parameters which determines the input size, then the DFA can have 2 to the power n states, right. If, we never that it is possible for some nfas that the corresponding DFA, equivalent DFA will have exponentially many states.

So in that case the algorithm, just in converting the NFA to DFA is going to take exponential amount of time and the algorithm is not going to be efficient. But there is a way out, you can have actually a (())(7:54) algorithm but we do not come in that case we do not convert the NFA to an equivalent DFA. Instead what we do is, strategy that we use that we keep track of the set of states the NFA can be in after reading some portion of the input. So in other words we keep track of the set of states M can be in, that means M is in some computational path of the other M is in this state another computational path of the other, M within that state, we collect all those, we keep track of the set of states as we scan the input string W symbol by symbol.

Now let us see what it means we understand because we have seen many nfas, the idea of acceptance and all that, what are the states the machine can be in. So initially of course if the machine is in its initial state, so this is the set of states the machine can be initially and suppose W is of the form A1, A2… An, so W is in, W is a string with n symbols, so right now the machine is in the state, the set of states it is in is Q0 and then when A comes, we know which states it can be in after A1, it is precisely the set of states given by the transition function Delta Q0 a that will provide me with the set of states, these are the set of states the machine can be, M can be here. Right.

(Refer Slide Time: 11:26)



$\{q_0\}$

$w = a_1 a_2 \cdots a_n$

Strategy:

We keep track of the set of states M can be in, as [illegible]e scan w symbol by [illegible]bol

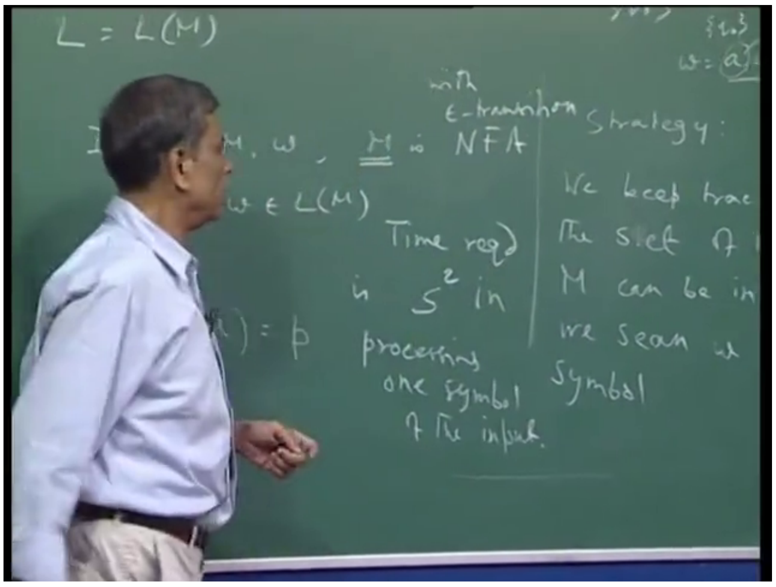
$\delta(q_0, a) = \{p_1, p_2\}$

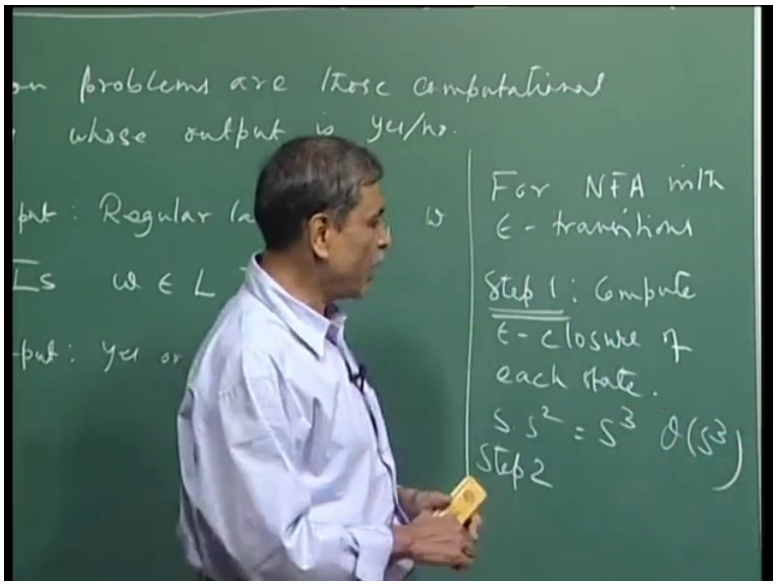
a_2

Then A2 will come and suppose this set of states is, let me just given an example here P1, P2, supposing these are the 2 states that result. So I see on A2 from P1 which all states the machine can be in, again that is just consulting the transition function and similarly from P2 on A2, which all states the machine can be, we take the union and that is the set of states the machine can be after seeing A1 and A2 and this way you go up. Only thing is that this union taking can be done fairly efficiently, what you can in this simple datacentre idea that we can keep a bit vector, right to indicate which all states are there in that set.

So now this one processing of 1 symbol therefore, how much time would it take? In general, let us say this is A I, the set of states the machine can be is bounded by in size by the number of states in M, the NFA M. So supposing that number is s, right and from, so the machine is in set of states, that size can be at most s and we have to look at each state in that set and look at on the next input symbol which all states the machine can be. So s into s, so we require time s square, let me write time required is s square in processing 1 symbol of the input.

(Refer Slide Time: 13:18)



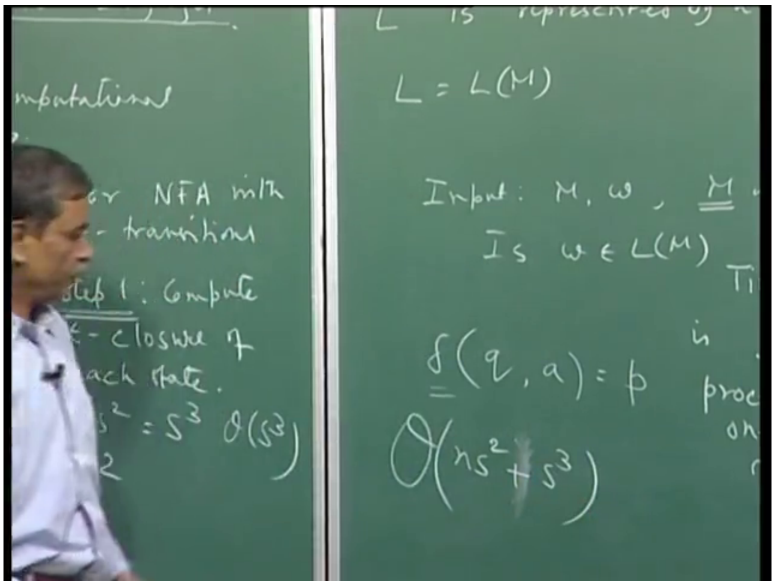

problems are those computational
whose output is yes/no.
put : Regular la
Is w ∈ L
put : yes or
w
For NFA with ε - transitions
Step 1 : Compute ε - closure of each state.
$S \cdot S^2 = S^3$ $O(S^3)$
Step 2

So and there are n symbols in the input W, so that our algorithm, what I just outlined is going to be order ns square. Remember are n is part of the input, we know number of states, so this is polynomial in the size of the input. So this is also a polynomial time algorithm. What if M is NFA with Epsilon transitions? So here we will do exactly the same thing, per symbol we have to keep track of which all states the machine can be but remember now after we get the set of states the machine can be on scanning a particular symbol, then we need to take the Epsilon closure of each such state and then take their unions. So strategy better for this, when I have Epsilon transitions is right in the beginning to compute the Epsilon closure of each state for NFA with Epsilon transitions.

Step 1 compute Epsilon closure of each state and then you do exactly same thing, at any given time after processing a number of symbols in the input, I have a set of states the machine can be, then the next symbol will take care of, take the union of the whole idea that we take the union of the set of states for where the machine can be for each one of these and then Q, because of it is an NFA with Epsilon transitions, then I take the epsilon closure of that set of states. And that is the set of states the machine can be after scanning the next symbol and this way we will go on.

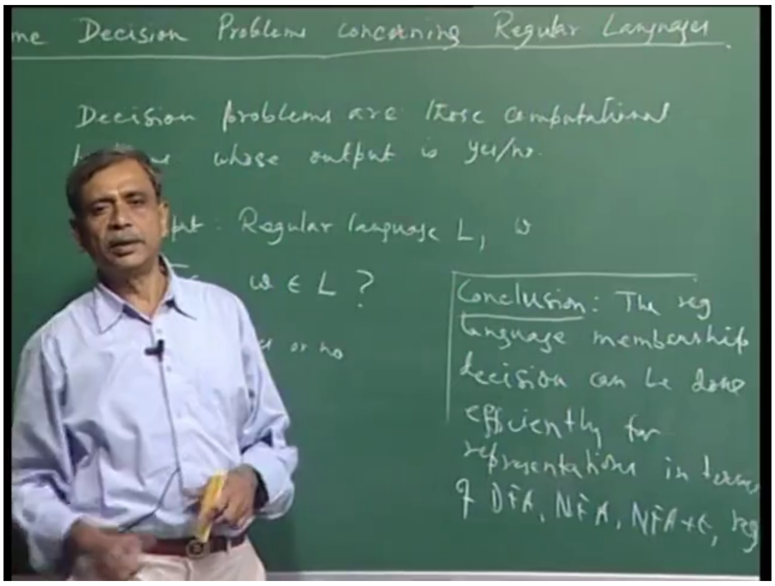
Right, so what is the time involved? For step 1, we have again s states and to compute the epsilon closure which is basically, recall which all state the machine, which are reachable from that state, so that means we only look at the transition diagram where the arcs or the edges are labelled with Epsilon, so it is really a reachability problem. So at most per state will require the size of the transition diagram which is s square. So we need to do this for each state, so s cube, this is the order. But this is step one which is done only once. Step 2, you recall that we collect, at any given time we have a set of states, right and from each that that set can be at most, can have at most s members, for each I need to consult the transition diagram and then we take the union.

(Refer Slide Time: 18:33)



So you know, so that is again s square and then again it is same thing, that is the old thing, this part is same, s square per symbol and then we need to take the epsilon closure of these but then now I have already, done the epsilon closure of each state here, so it will be basically boiling down to taking the union of certain states and that also can be done efficiently. So you can see overall it will be order s cube + order ns square and so n is usually length of the input is going to be larger than or put it this way, this, the amount of time you going to take is ns square + s cube whatever is that, whichever is bigger it does not matter.

(Refer Slide Time: 19:08)



me Decision Problems concerning Regular Languages

Decision problems are those computational [illegible] whose output is yes/no.

[illegible]put: Regular language L, w

[illegible] w ∈ L ?

[illegible] or no

Conclusion: The reg language membership decision can be done efficiently for representations in terms of DFA, NFA, NFA+ε, reg

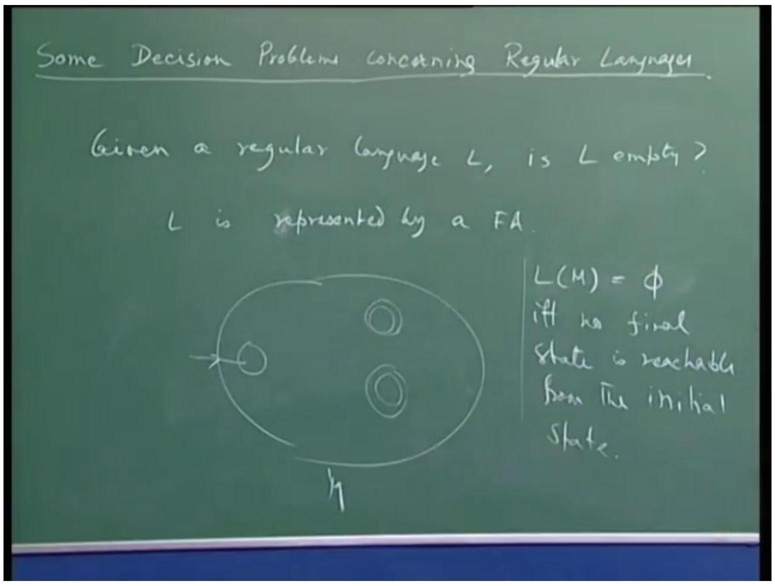
So this is roughly again, this is therefore a polynomial time algorithm. Alright. So there is yet another representation for regular languages and that is regular expression. We were discussing the case when L is represented by regular expressions say R. And here what we ought to do, the simplest straightforward method would be 1st obtain an equivalent NFA M, by equivalent of course we mean that the language represented by the regular expression R is the language accepted by the NFA M. But remember the standard conversion that we gave, this NFA will be with Epsilon transitions. And in the 2nd step, decide if W, the input string, is in language accepted by the NFA M, we know how to do it.

And so it all depends on what is the size of the M in terms of the regular expression R. And if we look at that construction once more, it becomes clear that the size of M is that most twice the size of the regular expression R. So therefore this construction, of course we do in polynomial time, we know how to do that and the machine M is not too large, it says at most in the size of the regular expression, the number of states the machine M would have is twice the number of symbols in the regular expression.

And that is why we said in a way that, so let us say in fact this, more correct statement would be the number of states of M is at most twice the size of R. So in other words this step on can be done in polynomial time and step 2 of course can be done in the polynomial time, polynomial in the size of W at the size of M but since the size of M is also polynomial with respect to the size of R, this entire process also would be efficient. So what we have discussed so far is that that the language, the membership problem. So we can we can, conclusion is the language, the regular language membership decision can be done efficiently because we have polynomial time algorithm in every case for every representation for the or let me say representation, for representations in terms of DFA, NFA, NFA + Epsilon transitions all regular expressions.

Thus this problem is at least for regular language is can be done efficiently, later on in this course we will see not only we will not be able to do in general, for certain classes of automaton that this problem which now appeared so simple in case of regular languages, that not only we may not have efficient algorithm, we may not have any algorithm at all, for example when we talk of Turing machine as an automaton which we will see later on. Our next problem for decision, our next decision problem will be, we can ask this question for example, given a regular language in terms of course some representation either a or NFA or NFA with Epsilon or a regular expression about the size of the language.

(Refer Slide Time: 24:54)


Some Decision Problems concerning Regular Languages

Given a regular language L, is L empty?

L is represented by a FA.

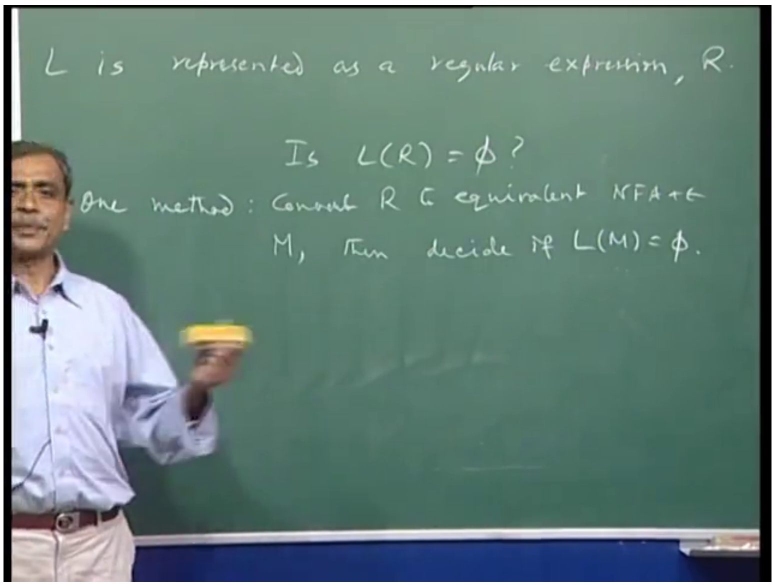
$L(M) = \phi$ iff no final state is reachable from the initial state.

For example we can see is the language empty, is the language infinite and so on, at least these 2 questions we can ask. So let us consider this problem, given a regular language L, is L empty? Now as we understand the language will be given in terms of either one of the automaton DFA, NFA or NFA with Epsilon transitions or as a regular expression. So 1st consider the case when the regular language L is given in terms of one of the one automaton. So L is represented by a finite automaton. Now what will be that finite automaton, whether it is deterministic, nondeterministic with Epsilon transitions.

When can you say such an automaton, think of this that I have this automaton and there is some initial, I mean there is this initial state and the number of final states. Clearly this automaton accepts some strings if and only if there is some part from the initial state to one of these any one of these, one or more of these final states. Right. So that is clear, right because, you see remember that through the transition diagram, think of the transition diagram as a graph by this is a particular vertex, let us say the start vertex and this transition diagram graph is a directed graph but we can always use any of the graph search algorithm DFA (())(27:10) search to find out all the vertices in the graph which of course correspond to states reachable from this initial state, this particular vertex.

Now it is very to see that M accepts or let me write it this way the language accepted by M is empty if and only if no final state is reachable from the initial state. So basically this decision whether a language is empty, if you give me that language in terms of finite automaton, the problem is really doing a reachability analysis of a graph and that we know can be done quite efficiently, right. Essentially it means at most I need to traverse each edge of the graph once and since there is, if there are n states, there can be you know quadratic many edges. Therefore, this problem can be done efficiently in polynomial time because we are solving a reachability problem.

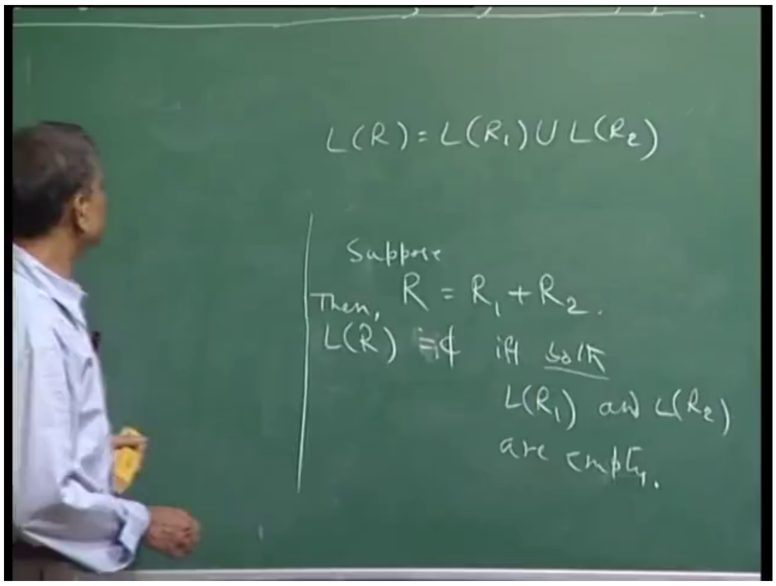
(Refer Slide Time: 29:13)



L is represented as a regular expression, R.

Is $L(R) = \phi$?

One method : Convert R to equivalent NFA+ε M, then decide if $L(M) = \phi$.



$L(R) = L(R_1) \cup L(R_2)$

Suppose $R = R_1 + R_2$.

Then, $L(R) = \phi$ iff both $L(R_1)$ and $L(R_2)$ are empty.

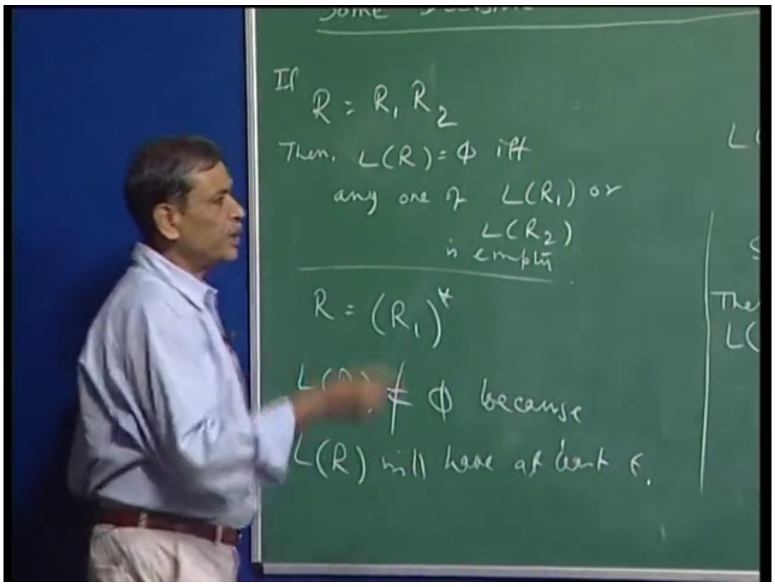
What about L is given as a regular language, regular expression? L is represented as a regular expression say R. So the question we are asking here, the language denoted by R, does it contain at least some string that is LR, is LR empty, this is the question we are asking. So we can use our old strategy, given this regular expression, we can efficiently convert it to an equivalent NFA with Epsilon transitions and then we can use the idea that we had just mentioned to see whether one of the final states of the NFA, in fact NFA with Epsilon transitions. Our construction, in our construction there will be likely on final accepting state, that is how construction went.

We have to just see whether this final state is reachable from the initial state of the ultimate NFA with Epsilon transitions that we defined for the regular exhibition. However it can be done in another way, so let us say one method was convert R to equivalent NFA with Epsilon transitions M, then decide if L M is empty. That is another matter where we do not convert the regular expression R to an equal and NFA with Epsilon transitions and the reason for discussing that is just it gives us another intuition about regular expressions, the algorithm is not any more efficient because anyways this algorithm is efficient enough.

And the 2nd method goes like this, see, we can recursively define or let me use it we can inductively define the set of all regular expressions denoting empty language phi and by now we know that this kind of interactive definition will be in terms of the inductive definition of regular expressions. So that inductive definition is this, there are base cases, there are 3 base cases Epsilon, Phi and symbol a, right. Epsilon base case, so let me write it as base case. If your regular expression R is either Phi or Epsilon or a, then we know that this one denotes empty language and other 2 are not empty. Right, this is by definition.

And now we can see this that suppose our regular expression is R1 + R2, then it is quite clear that R denotes an empty language, so let me say write this way that LR is, LR is empty if and only if both LR 1 and L R2 are empty. So this is fairly easy to see. What is L of the language denoted by R when R is R1 + R2, clearly in that case LR is union of 2 languages denoted and if either of these is nonempty, then of course LR is not going to be empty. So LR is going to be empty if and only if both of these are empty, so that is clear.

(Refer Slide Time: 35:03)



If $R = R_1 R_2$

Then, $L(R) = \phi$ iff any one of $L(R_1)$ or $L(R_2)$ is empty

$R = (R_1)^*$

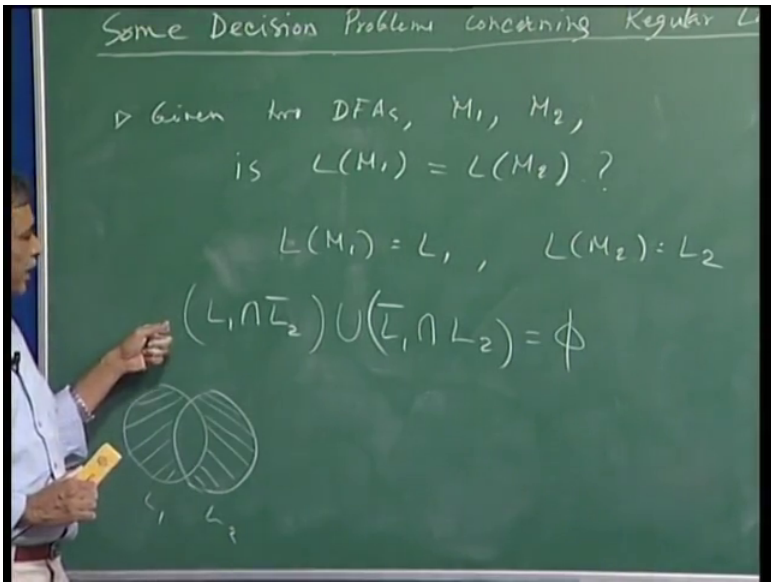
$L(R) \neq \phi$ because $L(R)$ will have at least ϵ.

So other 2 ways, a bigger regular expression can be formed out of smaller regular expressions, where the other way was concatenation, R is R1 R2 and the last one is R is some R1 star. Now in this case so if R is R1 R2, so we are talking of concatenation, then again it is fairly easy to see LR is empty if and only if any one of L R1 or L R2 is empty. Right. Because if you concatenated the empty language with any language whatsoever, the result is going to be empty language, that we know. So for this entire thing to be empty, if either of this is empty, then of course the regular expression will denote the empty language. On the other hand if none of them is empty, then clearly if L R1 and L R2, both are not empty, then clearly when we concatenate, I will get some strings, so LR is not going to be empty.

And finally in this case we know that LR is not empty because LR will have at least Epsilon. Therefore you see all these 3 things together up to the base case, what we have NFA is a recursive algorithm to decide whether a regular expression is empty or not. So the input is a regular expression, then you see whether it is of the form, one of the forms or it is one of the base cases and then accordingly whichever is the form that it is, if it is the regular expression given is one of the base case regular expressions, then if it is given, if the regular expression is this, then answer yes it is empty, these 2 cases, answer no.

Then depending on whatever is your regular expression, otherwise if it is of the form R1 + R2, then recursively decide LR, the emptiness of R1 and emptiness of R2 and we can answer and similarly for the other. In this case of goes the moment you see the regular expression is this form, you can immediately say the regular expression denotes the nonempty language because you can get at least Epsilon. Now that we have seen efficient algorithm for deciding whether a regular language is empty or not, we can use this to decide many other questions, especially when the representation is in terms of DFA.

(Refer Slide Time: 39:03)



Some Decision Problems Concerning Regular L

▷ Given two DFAs, M_1, M_2,

is $L(M_1) = L(M_2)$?

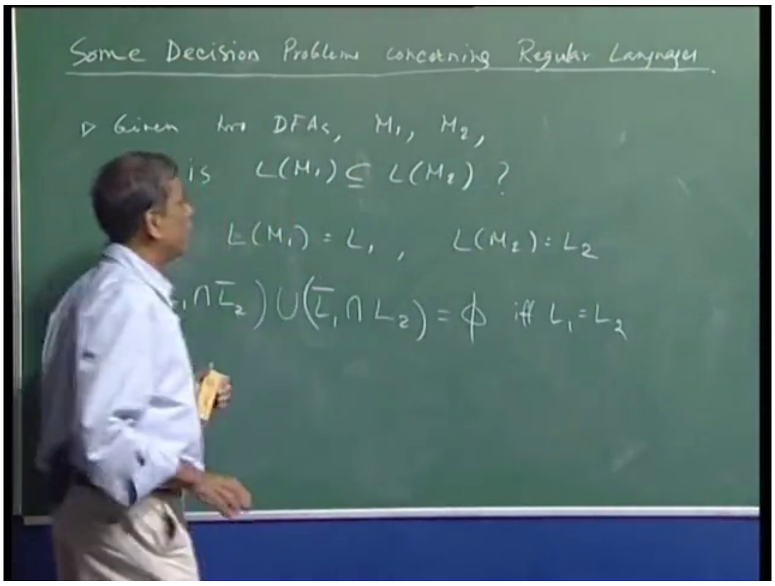
$L(M_1) = L_1$, $L(M_2) = L_2$

$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \phi$

So let me give an example that given 2 dfas M1, M2 is language accepted by M1 same as the language accepted by M2? See the base, one way we can do this is very simply that suppose, it is clearly like this that you take the language, the machine M1, okay and so let me say this is L1, the language is L1 for notational convenience. What will this language be? What I think it is simpler to see in terms of, supposing this is L1, this is L2, what we are looking for, L1 intersection with all those which are not in L2, so this is apart. And L1 complement L1 complement intersection L2, so all those things of the L2 which are not in L1, right.

Now if see if both of these are empty, then that means what, then that means L1 is equal to L2. So you see the point is that because L1 and L2 are given to me by DFA, I can very easily construct a DFA, we have seen this. For this, do accept this regular language as well as the DFA for this regular language and then a DFA for this entire language and then we can check whether that regular language denotes or accept the empty language. So we can say that this is the case if and only if L1 is equal to L2. So given 2 dfas M1 and M2, then I can construct this language and then LM1 is equal to L M2 if and only if, basically we are saying there is no string which is accepted by M1 and not accepted by M2 as well as there is no string which is accepted by M2 and not accepted by M1.

(Refer Slide Time: 42:58)



Some Decision Problems concerning Regular Languages

▷ Given two DFAs, M_1, M_2,

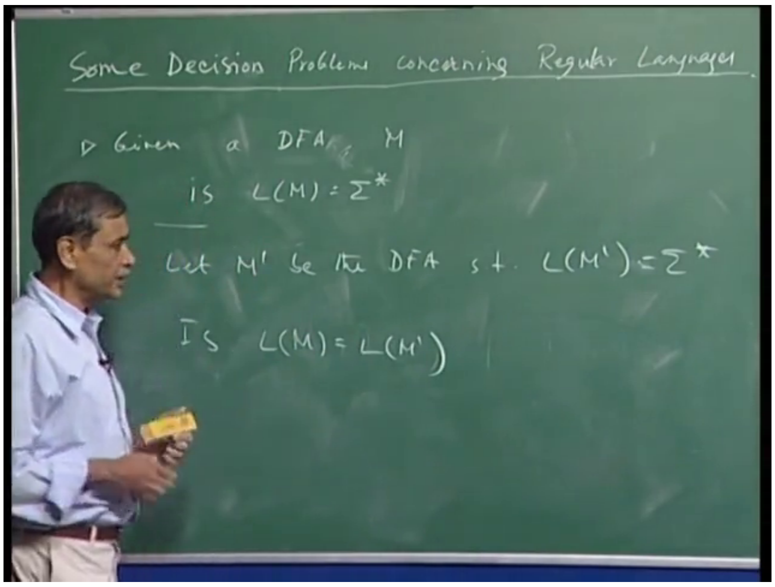
is $L(M_1) \subseteq L(M_2)$?

$L(M_1) = L_1$, $L(M_2) = L_2$

$\cap \overline{L}_2) \cup (\overline{L}_1 \cap L_2) = \phi$ iff $L_1 = L_2$

Therefore in that case the 2 languages are equal, therefore the 2 machines accept precisely the same set of strings. So similarly in a very similar manner you can, actually you can see I can decide like this, right, this question, is LM1 subset of L M2, is LM1 a strict subset of L M2? All these questions once given 2 dfas, there are very easy to decide because we can you know in this kind of way we can look at the equivalent a DFA whose emptiness is going to decide such a thing.

(Refer Slide Time: 43:38)



Some Decision Problems concerning Regular Languages

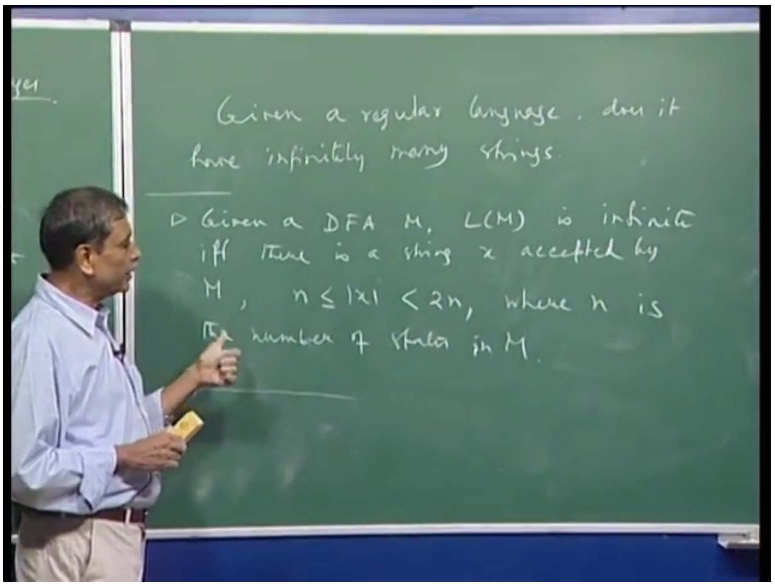
▷ Given a DFA, M
is $L(M) = \Sigma^*$

Let M' be the DFA s.t. $L(M') = \Sigma^*$

Is $L(M) = L(M')$

So now let me just take another example, there is another example to illustrate this point. So here I am given just one DFA M, given a DFA M is LM sigma star, that is it contains everything, every string in the language. Well, what will be, what we can do? That what we can do is to, you know let M dash be the DFA, such that language accepted by M dash is sigma star and we just ask the old question. Is LM equal to LM dash? This is very easy to define a DFA with, which will accept all strings and then we are just checking whether these 2 dfas accept the precisely the same set of strings which is where we will use the previous idea, algorithm and therefore you can even decide this.

(Refer Slide Time: 45:18)



Given a regular language, does it have infinitely many strings.

▷ Given a DFA M, $L(M)$ is infinite iff there is a string x accepted by M, $n \leq |x| < 2n$, where n is the number of states in M.

There is another class of problems people are interested, which is, given a regular language does it have, I mean is it infinite, does it have infinitely many strings. In other words what we are asking is a language infinite? And if we can decide that then of course we can decide, given a language L, whether it has finite, that is it has only finitely many strings. Now it can be done at least a 2 ways and one way actually is from our idea of pumping lemma. When this is, although algorithmically it is not going to be very efficient, but let me write this out. Given a DFA M, LM, this language accepted by DFA is infinite if and only if there is a string x in or x accepted by M such that the length of x is greater than or equal to n less than equal to 2 n, where n is the number of states in M. So this remember M is a DFA and let say it has n strings.

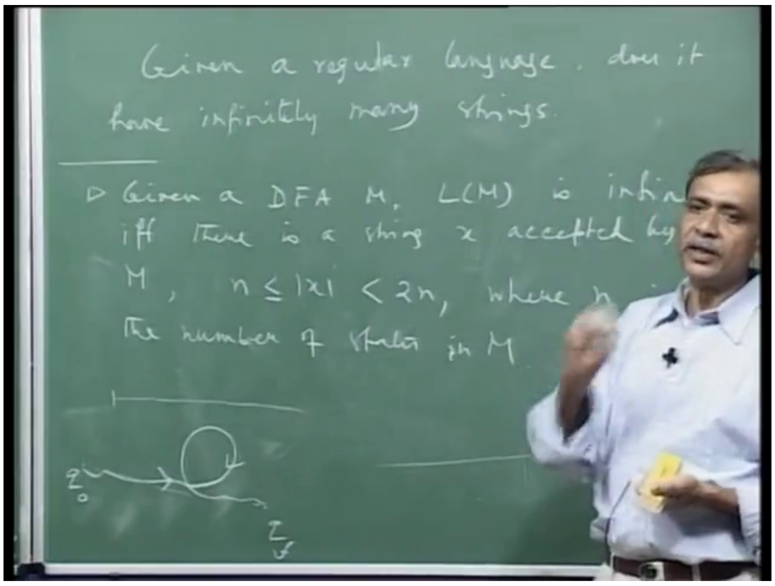
And now let us look at it, at least this, that suppose you take a string, we know that that any string whose length is greater than equal to M, the pumping lemma proof shows that and which is accepted by n, such a string can be pumped infinite amount of times, right. And therefore each time I pump I will get a new string, different from the previous strings. So therefore just by pumping I can generate infinitely many strings from that one string. Right. So what this string is telling me is that, remember pumping lemma says that for every string accepted by the machine M, whose length is great greater than or equal to the number of states, right.

Now what this is saying is that left open, that yes I am looking for a string whose length is greater than equal to n but is there are bound within which I am guaranteed to find it if it is there? And in fact this is the problem, at most you need to work to 2n. Why? Because the value that you pump, so let say we had a long string, bigger than 2n which is accepted by the machine M. Then clearly if you, if you recall the pumping lemma, such a string, there will be a particular V parts, now I pump down. On pumping down I become, I come into this string or I am still greater than 2 n and then I come, again I pump down.

What is this idea, why that is coming because what we pump is that most of size n, remember. That the V part, the size of V is less than equal to n. So from a large string I can keep taking out chunks of n and then since that chunk can be of size n, therefore I will get at least some strings in these strings. So this shows that this lemma can be used, so this can this lemma can be used for an algorithm, right. So what I can do, given a DFA, all I need to do is to look at its number of states that is n and then look at strings in this range and see whether any of these strings is accepted by the DFA.

But clearly this is an exponential time algorithm because a string of size n, how many strings are there of size n, exponentially many and we would like to come according to this naive algorithm you check each such string in this range, whether that string is accepted by M, accepted by M and so on. There is clearly, if you think about there is a polynomial time algorithm and whose basic idea is this that if, imagine this DFA and if there is, of course for it to have some string, there has to be a path from Q0 to one of the final states QF, right in the transition diagram graph.

(Refer Slide Time: 52:08)



Given a regular language, does it have infinitely many strings.

▷ Given a DFA M, L(M) is infin
iff there is a string x accepted by
M, $n \leq |x| < 2n$, where n
the number of states in M

And now in this graph if there is a cycle, then clearly we will be able to generate infinitely many strings through this. And this idea is not just true of this idea that I am talking of, that if I have a path from initial state to one of the final states and in that path there is a state on which there is a cycle, that, then the resultant language will be infinite. That idea is of course true even when the machine is an NFA. What about the case when the machine is an NFA with Epsilon transitions? All I have to make sure that the cycle does not consist only of Epsilon edges.

So in other words what we are saying that if in an automaton, there is a path from the initial state to one of the final states and in that path there is a vertex where there is a cycle which does not consist only of Epsilon edges, remember our context for this graph is the directed transition diagram graph. In that case that finite automaton will accept infinitely many strings and that is a necessary and sufficient condition. And using that idea we can have a polynomial time algorithm for deciding whether an automaton, finite automaton, whether it is DFA, NFA or NFA with Epsilon transitions, whether it accept infinitely many strings.

So therefore what we have done today is we have looked at some decision problems concerning regular languages and we have found not only that all these decision problems can be solved by an algorithm and these algorithms tend to be quite efficient polynomial time. And as we look at other classes of languages later on, we will see that in fact there will be many decision problems for which we will not have any algorithm at all, not to talk of efficient algorithm. So in that sense our this class, regular language class is a very nice class.