# Formal Languages and Automata Theory

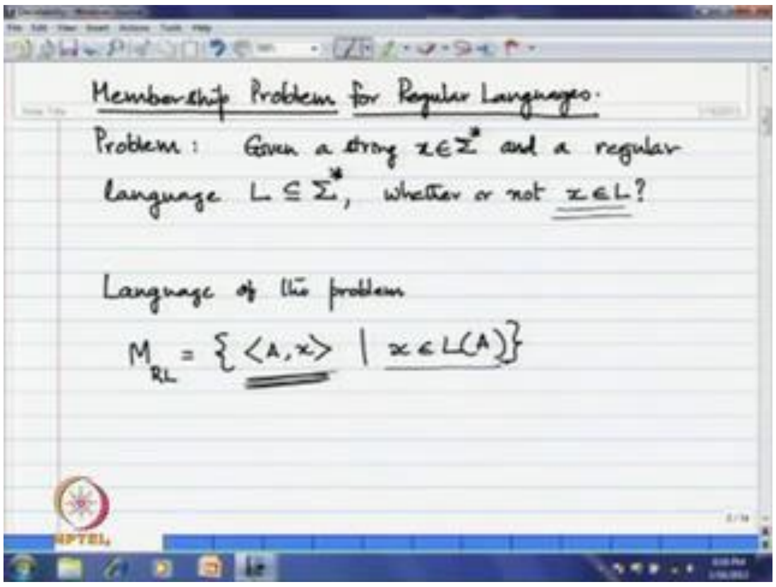
**Prof. Diganta Goswami**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Guwahati**

## Module - 13
## Decidability and Undecidability
## Lecture - 01
## Decidability

In this lecture will discuss Decidability, already we have introduced Turing machine as a model of general purpose computer. The notion of algorithm has also been defined in terms of Turing machine by means of source turing thesis, will now see that some problems can be solved algorithmically while others cannot. So, it gives us the limits of algorithm suability, in decidability will concentrate some example of languages that are decidable by algorithms.

That is there are Turing machines to decide those languages of course, while giving Turing machines as algorithms. That means, these are the Turing machines, rather than spelling out the full specification of Turing machines; that means, states, transition functions, etcetera, we provide only a high level execution, where we use in this prose to describe an algorithm, ignoring the implementation details. That means, we do not mention how the Turing machine managed is step or head, we now consider some example languages that are inside able, we consider languages to represent various commercial problems just for our convenience.

(Refer Slide Time: 01:50)



Membership Problem for Regular Languages:

Problem: Given a string $x \in \Sigma^*$ and a regular language $L \subseteq \Sigma^*$, whether or not $\underline{x \in L}$?

Language of the problem

$$M_{RL} = \{ \underline{\langle A, x \rangle} \mid \underline{x \in L(A)} \}$$

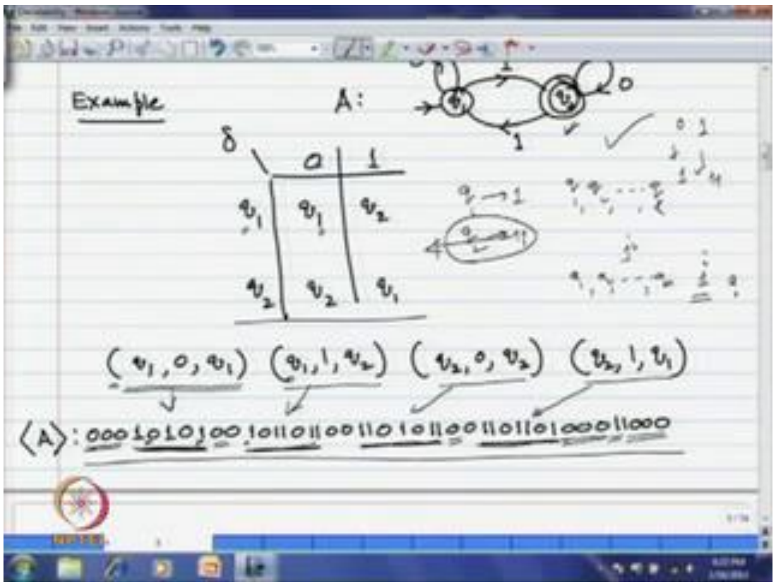
So, first we see problems from regular language itself, so first let us consider membership problem of for regular languages. So, the problem is that given a string x belonging to sigma star for some sigma and a regular language L for what it alpha with whether or not the string x belongs to L, so this is a problem. So, this problem the corresponding language for this problem can be stated as M R L, which is the set of all strings like this.

So, A x such that a is A DFA and x belongs to L of A, so why we are searching here A to be a DFA. Because, since we understood where as x belong to L or not, we need to give an algorithm; that means, we need to design Turing machine to decide this problem. Now, we have to give a description of the language to the Turing machine, how this description this language should be given to the Turing machine, as input is the main problem because, language may be infinite.

So, need to have some finite prediction of the language, so that we can give it as an input. So, there are different kinds of ((Refer Time: 03:27)) for example, so we have a regular expression by using regular expression we can give or represent the n language or we can also use a finite automata for example, say DFA to describe the language. So, here will consider finite automata to describe the language, so therefore, this A x this is the string, where A is the encoding of the DFA and x is a string.

So, the language of problem is the set of all strings like this A x in encoding, such that A is a DFA and x belongs to L of A. Now, since we have said that when we to give an encoding of A; that means, that DFA if first consider how to encode a given DFA A.

(Refer Slide Time: 04:36)


Example
A:
δ
0
1
$q_1$
$q_2$
$(q_1, 0, q_1)$ $(q_1, 1, q_2)$ $(q_2, 0, q_2)$ $(q_2, 1, q_1)$
⟨A⟩: 000101010100101101100110101100110110100011000

Let us see by encoding of DFA or coding of a DFA, so if A is a DFA then corresponding encoding we written as A which in angular bracket. So, that is why we have written here, ((Refer Time: 04:50)) in the language as A x which in angular bracket, so these are encoding, consider for example, this DFA, where we have 2 steps, q 1 and q 2 and the inputs are from the alphabet 0, 1. So, this is a corresponding Turing function delta, ((Refer Time: 05:20)) set q 1 and only one final state it is q 2.

From the transition function, we know all the statement will be represent triplets, say first one q 1 and input 0 it goes to q 1. So, we represent like this q 1 0 q 1, second one is q 1 on 1 goes to q 2, so q 1 1 q 2 like that we can also represent order two transitions. Now, we give a encoding of this DFA using strings over 0 and 1, so to start with the lubricant we represent using three 0's. And then since we have finite number of states will be using, so if we have q 1 q 2 up to say q k, we use a sequence of ones to represent the states.

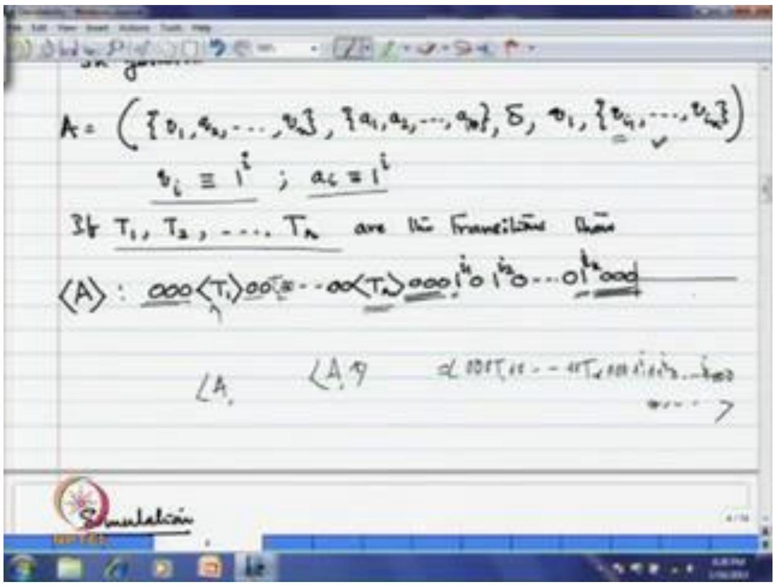
That means, 1 to the power i will be used to represent the state q i, similarly to represent the symbol from alphabet, again we use sequence of 1's only. For example, say a 1, a 2 say a m these are the symbols from alphabet, then will be using 1 to the power i to represent the symbol a i. Therefore, we can now encode every transition like this, so after three 0's we denote the lay bracket, so q 1 will be denote as 1 to the power i; that means, 1.

Now, next we will use 0 as a separator after using 0 here as separator, we use single symbol, say 1 to the power i; that means, only 1 to denote 0 and then again 0 is the separator, then 1 to represent q 1. So, 10101 this the code for a corresponding transition q 1 0 q 1, then this transition will separate from the other transitions by two 0's over here. And then for this transition q 1 will be represented as by 1 separate by 0, then this symbol 1 will be given by two 1's, because there are two symbols 0 and 1, 0 will be represented by 1 and 1 will be represented by 1 1.

And then q 2 will be represented by two 1's 1 to the power 2 because, you have two sets q 1 and q 2, for q 1 we have single 1 and for q 2 we have two 1's. So, this is the code for this particular transition, next it will be separated by an two 0's and this the code accordingly for this transition two 0's again to separate and this will be the code for a corresponding transition over here. Then again will use three 0's to indicate that we have no more transitions over here, all that transitions have been encoded.

And then we indicate the set of final states after this, so since q 2 is the final state, so use two 1's according to our notations for q 2, we have two 1's. And then we use three 0's to indicate that there is no more final states over here, so this is the encoding for the DFA A.

(Refer Slide Time: 10:09)



$A = (\{q_1, q_2, \dots, q_n\}, \{a_1, a_2, \dots, a_m\}, \delta, q_1, \{q_{i_1}, \dots, q_{i_k}\})$

$q_i \equiv 1^i \; ; \; a_i \equiv 1^i$

If $T_1, T_2, \dots, T_r$ are the transitions then

$\langle A \rangle : 000\langle T_1 \rangle 00 \langle T_2 \rangle \cdots 00 \langle T_r \rangle 000 1^{i_1} 0 1^{i_2} 0 \cdots 0 1^{i_k} 000$

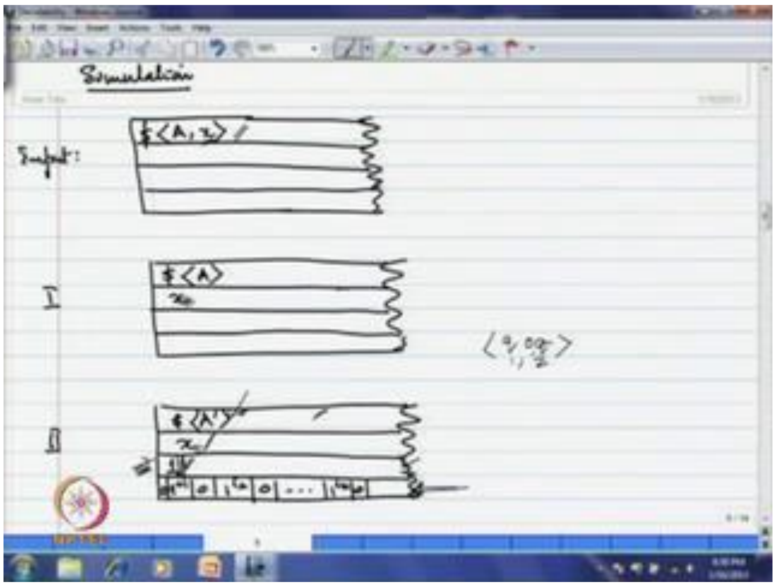
$\langle A,$ $\langle A, x \rangle$ $\langle 000 T_1 00 \cdots 00 T_r 000 1^{i_1} 0 1^{i_2} \cdots 1^{i_k} 000$ $x \cdots \rangle$

So, in general if we have a DFA having finite number of states q 1 to q n, finite number of symbols a 1 to say a n, delta is a transition function, q 1 is a initial state and a set of finite states over here q i 1, q i 2 up to q i k. What we do is that, q i will be represented by 1 to the power i this thing and a i will be represented by 1 to the power i this things. And if T 1, T 2, T r are the transitions, then the DFA A is represented by three 0's, the code of T 1, two 0 then the code of T 2 again two 0 the code of T 3 and, so on up to the code of T r the r’th transition.

Then three 0's to indicate that there is no more transitions and then 1 to the power i 1 to indicate that is the final state q i 1 is the final state 0, 1 to the power i 2 q i 2 is the final state and, so on up to q i k to indicate that q i k q i 1 q i 2 up to q i k, these are the final states and eventually three 0's to indicate the end, so that is how we represent or encode any given DFA. So, once we have encoded what we do, we use a Turing machine, so this encoding of A along which after the encoding what we have.

So, this is the encoding of A we have 000 T 1 00 and, so on 00 T r 0001 to the power i 1 0 1 to the power i 2 0 up to your 1 to the power i k 000 and after this we placed the string x whatever it may be. So, it just again a sequence of 0's and 1's 0 1 or whatever is sequence of 0's and 1's, so this overall will give us the string A x, so therefore, that is how we give the input to the Turing machine.

(Refer Slide Time: 12:59)



So, we consider first simulation we consider a 4 step Turing machine, on the 1'st step we place the string encoding on Turing machine along with the string. That means, encoding of A and x together, where dollar is a first symbol and the 2'nd step we put, so initially the what a Turing machine can do, it can look for ((Refer Time: 13:41)) the first group of three 0's, second group of three 0's, and finally last group of three 0's. Once it gets a last group of three 0's, the third group three 0's, it knows that there is no more final sets and whatever is there after this will be just string x.

So, once it can identify the string x it can copy, let see x from the 1'st step to the 2'nd step. So, in a number of steps that Turing machine can copy, the second component; that means, the string to the 2'nd step, now in the 4'th step, the Turing machine again can find out or identify all the final states of that DFA and it can copying on a 3'rd step the final states of the DFA. And a 3'rd step will be considered as a current step or; that means, the step which is used for the simulation the Turing machine.

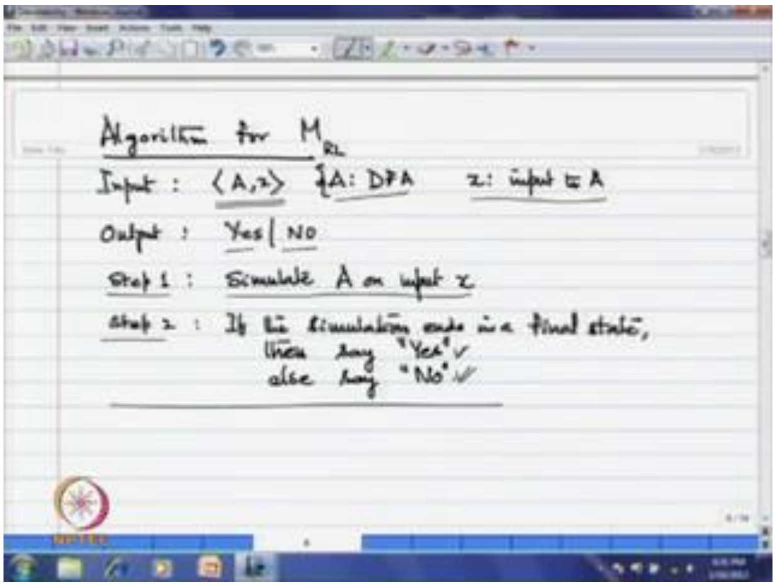
Now, will give the algorithm for M R L to do that we need to know how the Turing machine will simulate the behavior of the DFA. and it will say yes, whenever the DFA eventually accepts the string x and it will say no, whenever DFA does not accept the string x. So, the Turing machine will first see put the initial step; that means, the 1 because, q 1 is the initial step that is why it will put 1 in the 3'rd step ((Refer Time: 15:45)).

Then looking at a symbol the first symbol of x, so it knows that it is in step current step is q 1, it will first see what is a current symbol from the 2'nd step. Therefore, from the 1'st step, which encoding the Turing machine, which contains the transitions of the DFA, it will see what is your q 1 the first symbol of x is suppose say 0. It will see, whether the third component in the transition which is available in the 1'st step. So, accordingly if it is q 1, it will print 1 over here in the 3'rd step, otherwise if it seek it is q 2, then will print two 1's over here on the 3'rd step.

To indicate that the DFA has sends it a step 2 q 2, according to transition function given on the 1'st step in the encoding. So, this will be continued in this way looking at the symbols on the 2'nd step, the current state in the 3'rd step and any transition matching the current state and the current symbol, it will keep on changing the state in the 3'rd step. So, eventually whenever the input string is exhausted, then the Turing machine check the concern of the 3'rd step.

So, it will indicate what is the current state, if that current state you see wherever it is a final state from the 4'th step, where are that state belong is a state, which there on a 4'th step or not, if it is it knows that the input is exhausted and it is entered in a final state. So, therefore, the DFA should accept; otherwise the DFA will reject that is given string x, so that is how the Turing machine will simulate the behavior T DFA using a 4 step Turing machine.

(Refer Slide Time: 18:21)



Algorithm for $M_{RL}$

Input : $\langle A, x \rangle$ {A : DFA  x : input to A

Output : Yes | NO

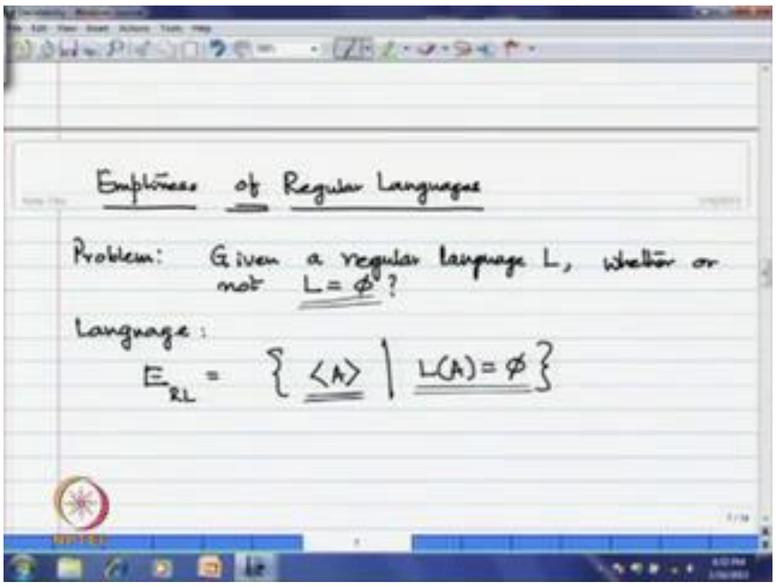
Step 1 : Simulate A on input x

Step 2 : If the simulation ends in a final state,
then say "Yes" ✓
else say "No" ✓

So, once you know how it goes we can now give the algorithm for the language M R L, so in this case, the input is of this form A x where is A DFA and x is the input to the DFA A. So, x is the encoding is given to the Turing machine, so Turing machine will either output yes or no, so if it outputs yes; that means the DFA A accepts the string x, if it outputs no, it means that the DFA A it rejects the string x, so in step 1 will simulate the DFA A on input x.

So, it is now quite clear how the Turing machine will simulate, the DFA A on input x using it is 4 steps. In step 2, if the simulation ends in a final state, then the DFA will say yes output is yes; otherwise it will say that no, so this is the algorithm for M R L. So, therefore, M R L is decidable; that means, the membership much problem of regular languages is decidable.

(Refer Slide Time: 19:43)



Emptiness of Regular Languages

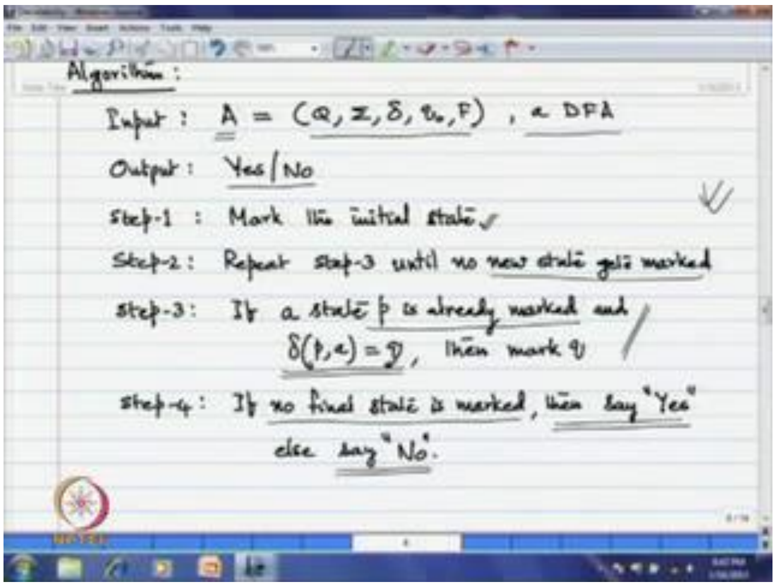
Problem: Given a regular language L, whether or not $L = \phi$?

Language:

$E_{RL} = \{ \langle A \rangle \mid L(A) = \phi \}$

Now, let us see the emptiness problem of regular language, so in this case the problem is that given a regular language L, whether or not L is empty. So, the corresponding language for this problem is that E R L, which is set of all strings A, where is A DFA and this is an encoding of the DFA and L A is phi; that means, it does not accept any string. So, what is now we do not give the Turing machine specifically all the states are like that, but simply will give the algorithm.

(Refer Slide Time: 20:30)


Algorithm :

Input : $A = (Q, \Sigma, \delta, q_0, F)$ , a DFA

Output : Yes / No

Step-1 : Mark the initial state

Step-2 : Repeat step-3 until no new state gets marked

Step-3 : If a state p is already marked and $\delta(p, a) = q$, then mark q

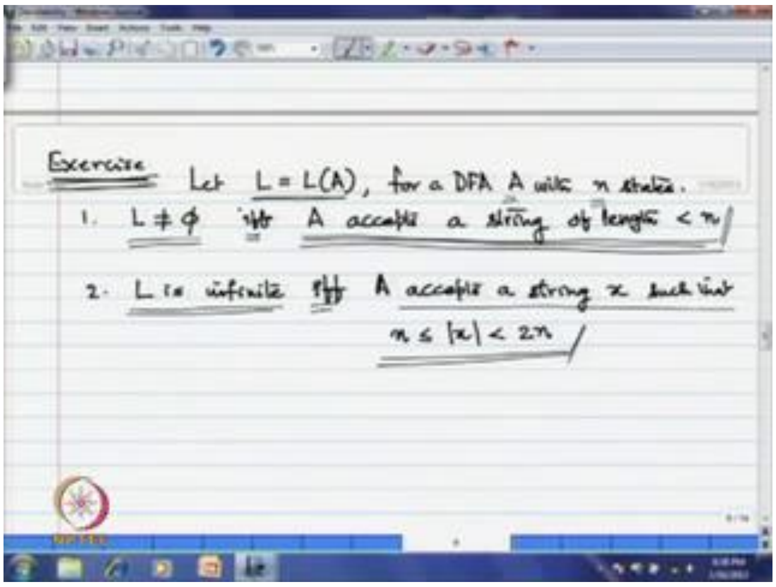
Step-4 : If no final state is marked, then say "Yes" else say "No".

So, in this case the input is DFA A, which is having the state Q, input symbol sigma, delta is a trans function, q naught is the initial state and F is the set of states. So, which a DFA is the input and the output is again yes or no depending on whether L of A is phi or L of A is not phi. In the first step what the investor can do it will mark the initial state of the DFA, so there is single initial state it will mark that state.

In step 2 the Turing machine will repeat step 3 until no new state gets marked, so step 3 is that if a state p is already marked, suppose p is already marked state. Then if delta p a equal to q, so this can be determined from the 1'st step, where the description of the or the encoding of the DFA is available, if delta p a is equal to q then it will also mark q. Because, p is already marked on some symbol it goes to q, then it will mark q as well, so these step we continued until no more, no new state gets marked.

So, eventually when it comes out from step 3, if no final state is marked it means that, the DFA cannot enter in any of the final states. In such a case it will say yes; that means, it will not accept any string; otherwise it will say no; that means, the DFA accepts some string. That means, the language of the DFA is not empty, now there is an alternative way to solve the same problem; that means, to reside the same problem; that means, the emptiness problem.

(Refer Slide Time: 22:35)



Exercise: Let $L = L(A)$, for a DFA $A$ with $n$ states.

1. $L \neq \phi$ iff $A$ accepts a string of length $< n$

2. $L$ is infinite iff $A$ accepts a string $x$ such that $n \leq |x| < 2n$

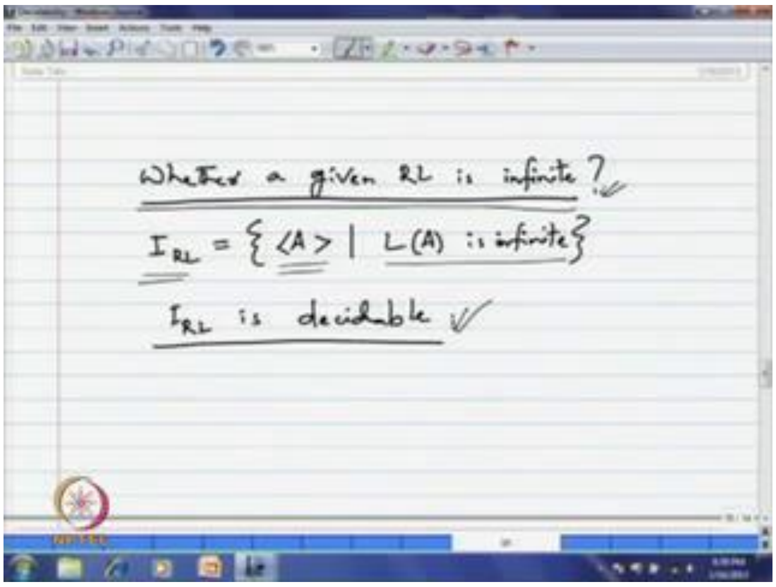
The alternative way we can prove it by means of these two theorems are concentrated from these theorems, suppose the Del DFA A which n states and L is language accepted by the DFA. Now, we can prove that L is not equal to phi if and only if A accepts the string of length less than n, so this we have kept as exercise, you can prove it easily by using pumping lemma. Already pumping lemma for a regular language have been discussed and they can apply that pumping lemma for regular languages to prove that n is not empty if and only if the DFA A accepts some string of length less than n.

Similarly, the second theorem says that L infinite if and only if it accepts a string x whose length is greater than or equal to n or less than twice n, where n is the number of states of course. So, we can use the first theorem to prove that or to decide whether the language of the DFA is empty or not; that means, emptiness of the regular languages, whether it is decidable or not. So, in this case the Turing machine basically can list out all are strings of length less than n.

So, there finite numbers of strings like that of length at most n and all these things can be generated on one tape of a Turing machine. And then the Turing machine can simulate the DFA on that string, so we continue for to generate the strings of length up to n and then test it or run the simulation or simulate the behavior of the DFA, whether it accepts or not. So, eventually if the DFA accepts any string of length at most n or less than n, then it can determine, it can decide that, it can output that, L not equal to empty otherwise it empty.

So, similarly since there are finite numbers of strings of length greater than or equal to n and less than twice n because, number of states the DFA is finite. So, Turing machine can always generate all those strings on one of it is step 1 by 1 systematically and then verify by simulating the DFA on that string, whether it accepts or not. If any one of the strings is accepted by the DFA whose length is greater than or equal to n and less than twice n, then it can decide that the language of the DFA is infinite.

(Refer Slide Time: 26:02)



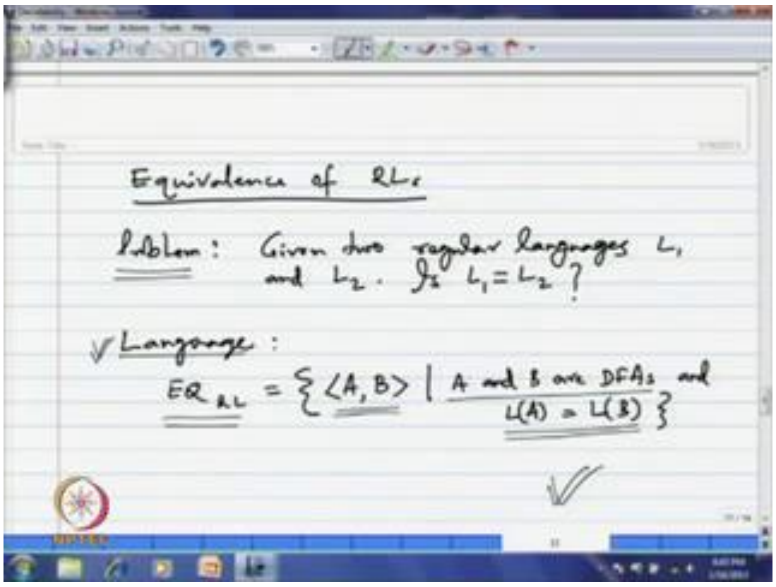
Whether a given RL is infinite? ✓

$I_{RL} = \{ \langle A \rangle \mid L(A) \text{ is infinite} \}$

$I_{RL}$ is decidable ✓

So, whether given regular language infinite; that means, the corresponding for that this problem, the corresponding language is I R L they encoding a DFA A, such that A is a DFA and L A is infinite. So, you can show that I R L is decidable by constructing a Turing machine, which will output yes or no accordingly.

(Refer Slide Time: 26:34)



Equivalence of RLs

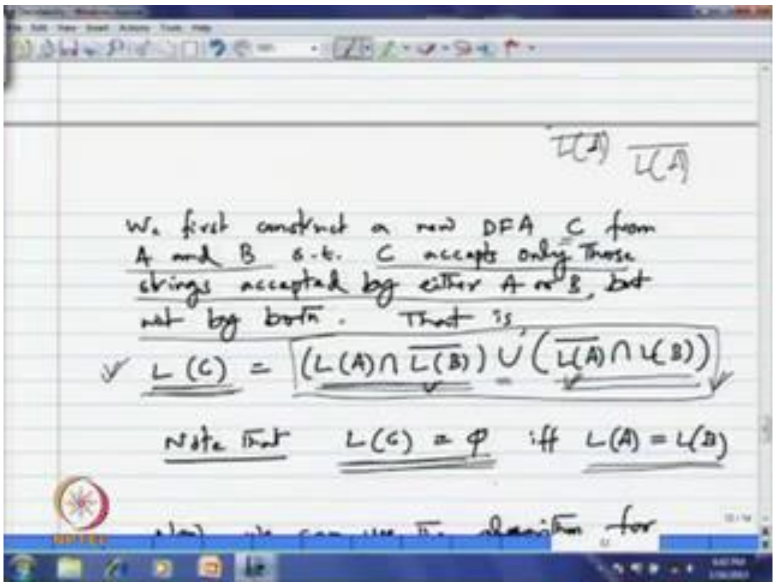
Problem: Given two regular languages $L_1$ and $L_2$. Is $L_1 = L_2$?

✓ Language:

$EQ_{RL} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \}$ ✓

Now, let us see the problem of equivalence up to regular languages, the problem is that given two regular languages L 1 and L 2 is L 1 equal to L 2 whether they are identical, the corresponding language is that is equivalence of regular language given two DFA's A and B, which are encoding of DFA A and B. So, A and B are DFA's and L A equal to L B, so that they accept the same language or not, so is a corresponding language problem.

(Refer Slide Time: 27:21)



$\overline{L(A)}$ $\overline{L(A)}$

We first construct a new DFA C from A and B s.t. C accepts only those strings accepted by either A or B, but not by both. That is,

✓ $L(C) = \boxed{(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))}$ ✓

Note that $L(C) = \phi$ iff $L(A) = L(B)$

Now we can use the algorithm for

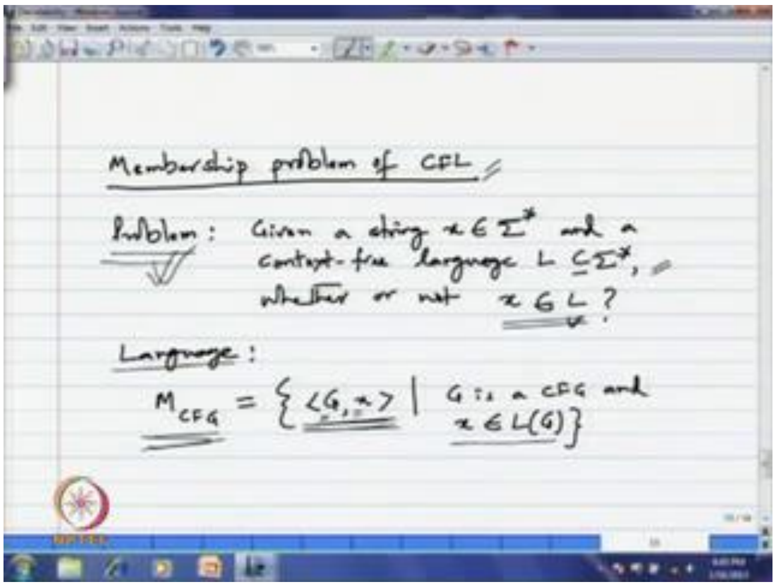
Now, to solve this problem or to decide this language will first construct a new DFA, which is say C from this given DFA A and B. Such that C accepts only those strings accepted by either A or B, but not by both; that means, language of C's written as L of A intersection compliment of L B union compliment of L of A intersection L of B, this what is called the symmetric difference A and B. Now, given a DFA A we know how to construct the DFA, which accepts the compliment of L of A, say A is a DFA, L of A is the language accepted by the DFA.

So, compliment of L of A will be accepted by that DFA, where we just inter sense the final states and non final states, all the non final states will be final states and the final states will be non final states. So, that new DFA will accept the compliment of the previous DFA, so therefore, from B we know how to construct the DFA, which accepts the compliment of L B. Similarly, from A we know how to construct the DFA, which accepts the compliment of L A.

Similarly, we have given the procedure to construct intersection of two DFA's, similarly the union of two DFA's. So, therefore, we know how to construct the DFA C, which is symmetric difference of A and B, now if we see carefully we find it language of C is phi if and only if the language accepted by A and language accepted by B are identical. So, therefore, this problem now ileuses the problem whether L A equal to L B ileuses the problem of finding wherever L C equal to phi or not.

Now, what we can do, we can now just use the same algorithm for emptiness that we have already given over here ((Refer Time: 30:15)). So, the algorithm for emptiness to determine whether L C equal to phi or not, so therefore, if L C equal to phi then L A equal to L B otherwise L A is not equal to L B. So, later you can determine whether the problem of equivalence of regular languages, we can say that the problem on equivalence of regular languages are decidable.

(Refer Slide Time: 30:57)



Membership problem of CFL

Problem: Given a string $x \in \Sigma^*$ and a context-free language $L \subseteq \Sigma^*$, whether or not $x \in L$?

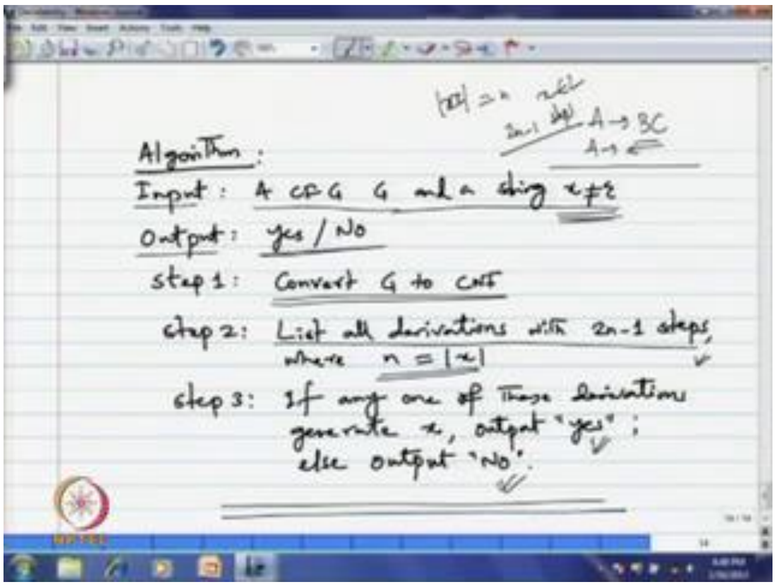
Language:

$M_{CFG} = \{ \langle G, x \rangle \mid G$ is a CFG and $x \in L(G) \}$

So, after concentrating the problems from regular languages, let us now consider the membership problem of contrast free languages, that in the problem is that given a string x over sigma star for some sigma and a contrast free language L over that sigma. Whether not x is a member of L, the same problem similar to the membership problem of regular languages, one thing and here is regular language is CFL contrast free. The corresponding language is that is M C F G, in this case we represent the language by giving the corresponding contrast free grammar.

In case of regular languages we used DFA as input, so in this case we used or the corresponding grammar G as input. So, in this case whenever you we say that we used angular bracket, it shows that it is encoding of the G and given string x, the way we have given the encoding for the DFA can also be extended to give an encoding of any given grammar. So, therefore, we assume that such a encoding exist is possible, so this G x is the string, the set of all strings G x, the language sets set of all strings G x, where G is a grammar C F G and x is a string. And here ((Refer Time: 32:31)) such that x belongs to L of G, so this is a corresponding language for this given membership problem of CFL, now let us see whether it is decidable.

(Refer Slide Time: 32:48)



$|x| = n$

$2n-1$ steps $A \to BC$

$A \to a$

Algorithm:

Input: A CFG G and a string $x \neq \epsilon$

Output: yes / No

step 1: Convert G to CNF

step 2: List all derivations with $2n-1$ steps, where $n = |x|$

step 3: If any one of these derivations generate $x$, output "yes"; else output "No".

We can show that it is decidable by giving an algorithm, now in this case what we can do in a for algorithm the input you concentrate as a CFG G and the string x. But, here we assume that the x is not equal to epsilon, if a string is epsilon then you can in determine whether it belongs to the grammar G or not by looking at the productions. Then, you can only find out whether that epsilon belongs to the language or not by similarly looking the production.

Now, the Turing machine simply output yes or no depending on whether the x belongs to the language or not. In the 1'st step we convert the CFG G to CNF, we know that in CNF all productions are of the form A goes to B C; that means, the rather sides we can have only two known terminals or A goes to say a single symbol, this kind of productions are allowed in CNF. So, since this is the case, if we have a string say x of length n we can show that to derive this string x of length n using this grammar, if x really belongs to L then it will take no more n 2 n minus 1 steps.

So, makes n also steps may take is 2 n minus 1 because, the right hand side of every production is a blind at most two where both are non terminals. So, after converting the grammar G to CNF because, this algorithm we have already given in the contest of your simplification algorithm contrast free grammars. So, will list or Turing machine will list all derivations which 2 n minus 1 steps, where n is number length of the string.

Once it is done, once it knows all the derivations of length with 2 n minus 1 steps, if any one of these derivations generate this string x, then the Turing machine will output yes. Otherwise; that means, none derives this string x, the Turing machine will output no; that means, it is not accept it. So, therefore, the membership problem of CFG or contrast free language is decidable, since we have a algorithm to decide it.