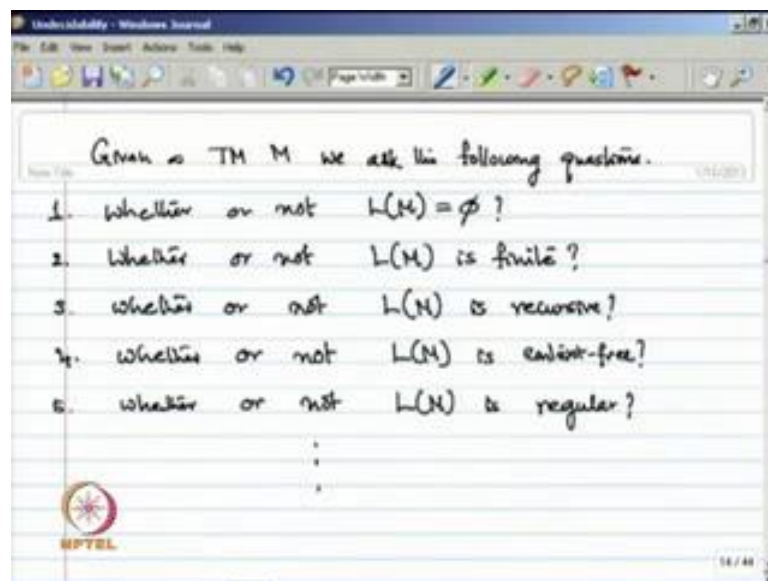


Formal languages and Automata Theory
Prof. Diganta Goswami
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module - 13
Decidability and Undecidability
Lecture - 3
Undecidability Part 2

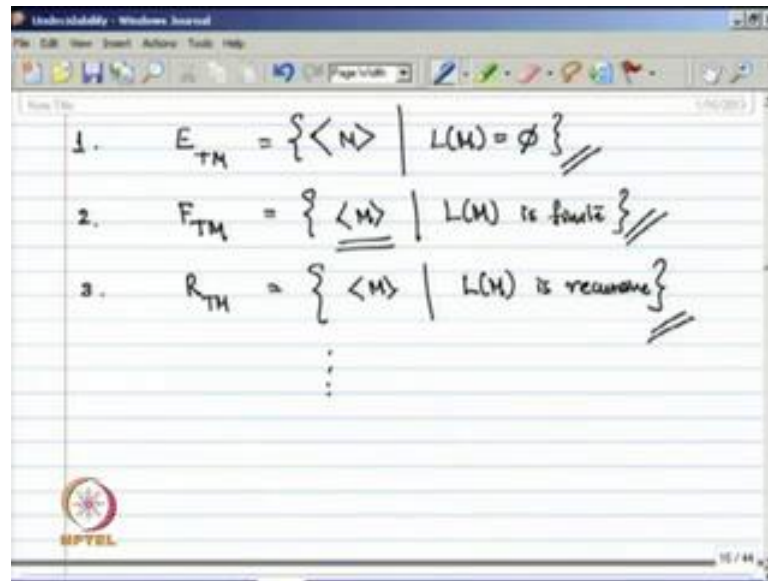
We have already proved that the halting problem is undecidable. Now we will discuss some more problems which are undecidable, and we give some important results which can be used to prove that some problems related to languages accepted by Turing machines are undecidable.

(Refer Slide Time: 00:50)



Thus consider the following problem given a Turing machine M whether or not the language accepted by M is empty, whether or not the language accepted by M is finite or recursive or this context free what is regular and so on.

(Refer Slide Time: 01:17)



The image shows a presentation slide with a blue title bar and a toolbar. The slide contains three numbered items, each defining a set of Turing machines (TM) based on the properties of their language (L(M)).

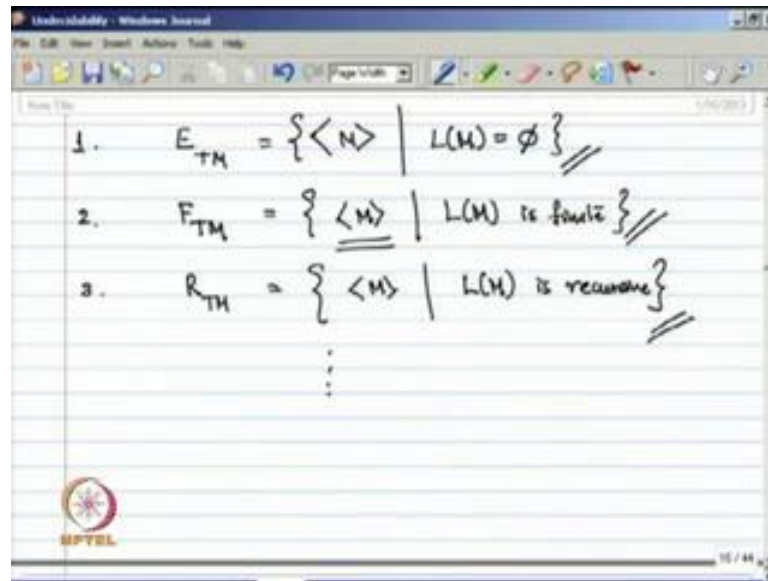
1. $E_{TM} = \{ \langle M \rangle \mid L(M) = \emptyset \}$
2. $F_{TM} = \{ \langle M \rangle \mid L(M) \text{ is finite} \}$
3. $R_{TM} = \{ \langle M \rangle \mid L(M) \text{ is recursive} \}$

Below these, there are three vertical dots indicating further examples.

The corresponding language for this problem can be written as say E_{TM} is a set of all Turing machines solves that $L(M) = \emptyset$. So, F_{TM} a set of all Turing machines and coring of all the Turing machines M , such that $L(M)$ is finite or say $L(M)$ is recursive and so on.

Now, we want to know whether these problems are decidable or undecidable, what you can do is that, for each of the cases, we can reduce halting problem to this problem, and we can show that all these problems are undecidable. But their interesting result that shows that or which can be used to show that all the problems related to languages accepted by a given Turing machine is are on undecidable, and that is stated by the theorem called rice's theorem.

(Refer Slide Time: 02:23)



The image shows a digital whiteboard with handwritten mathematical expressions. The expressions are numbered 1, 2, and 3, and each is followed by a double slash indicating it is a theorem or definition. The expressions are:

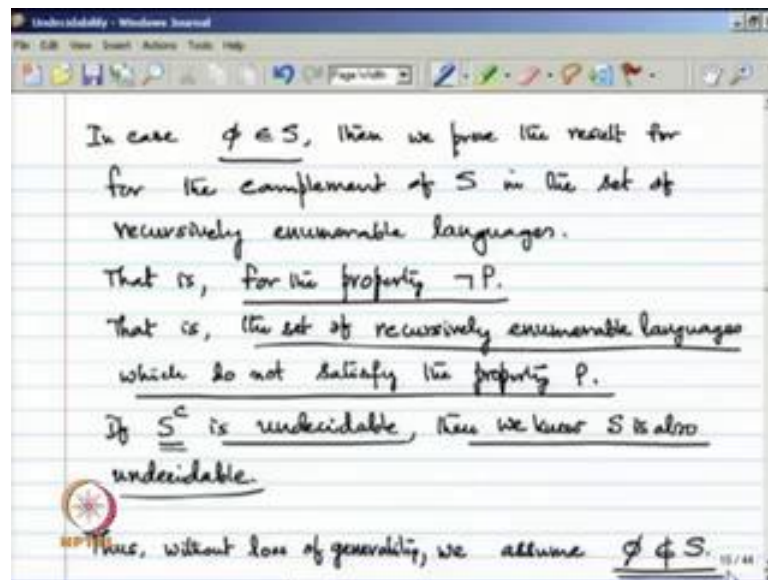
1. $E_{TM} = \{ \langle M \rangle \mid L(M) = \emptyset \} //$
2. $F_{TM} = \{ \langle M \rangle \mid L(M) \text{ is finite} \} //$
3. $R_{TM} = \{ \langle M \rangle \mid L(M) \text{ is recursive} \} //$

Below these, there are three vertical dots indicating a continuation of the list.

So, once you have the rice's theorem, we can show that, so many problems related to language accepted by the by a Turing machines; that means, recursion any more languages are undecidable. State the rice theorem, we consider a property, so a property that describes a proper non empty subset of recursively enumerable languages is undecidable; that means, suppose that p is a property that describes a proper subset of recursively enumerable language and there is a non empty subset of course.

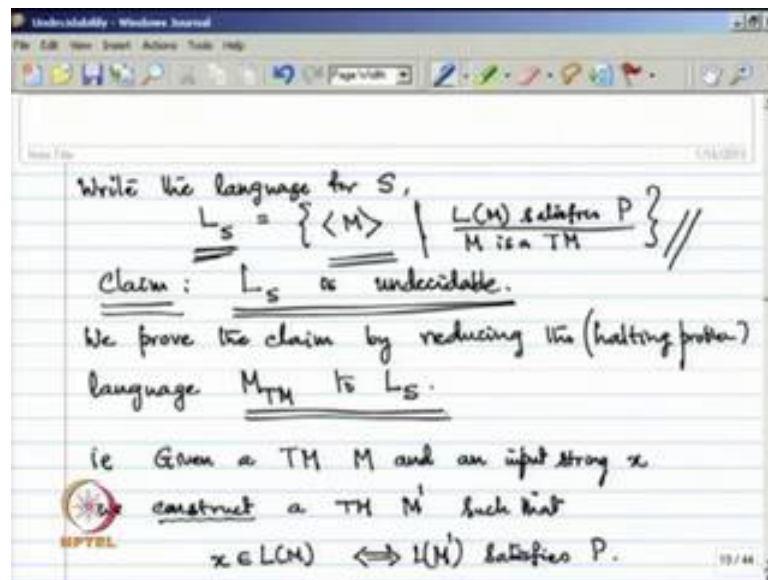
For example, say if S is a non empty subset the set of all recursion enumerable language L ; that means, some languages accepted by Turing machine such that this L satisfies P , so L is recursively enumerable and that is that satisfies the property P delta f that we have considered. So; that means, S in a proper subset of all recursion enumerable languages, without loss of generative or assume that ϕ is not in S , suppose ϕ is in S ; that means, is empty set say if any of the values ϕ , that we just consider the compliment of that problem.

(Refer Slide Time: 03:47)



Suppose, ϕ is in S , then thus consider the complement of this set S and we prove the result; that means, we consider for property which is a negation of P , because the complement because a corresponding set corresponding to a property which is the negation of P ; that means, the set of all recursively enumerable languages which do not satisfy a property P . So, once you can show that a complement of S , this is S^c is undecidable, then we also know that S is also undecidable; that means, if you solve a problem the decidable problem for a complement. We know the answer for these set also. Therefore, without loss of generality what we assume is that ϕ is not in S , because if $\phi \in S$ which are considered a complement of the set and solve the problem.

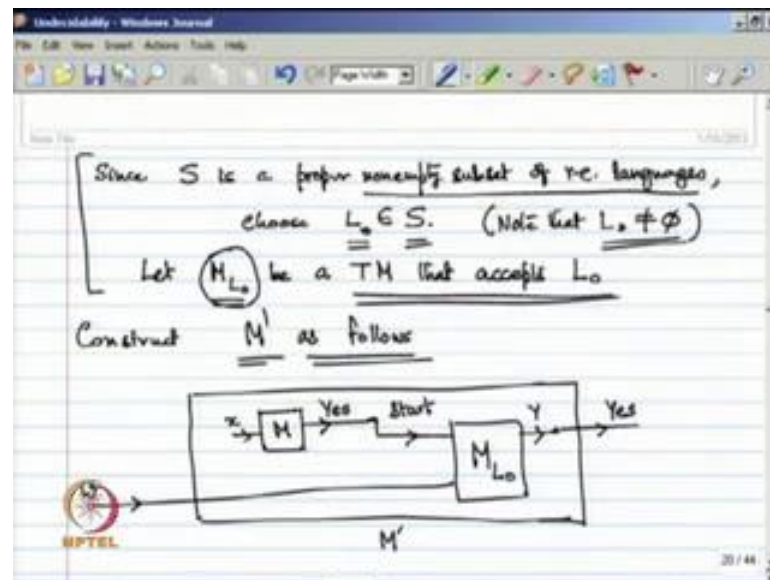
(Refer Slide Time: 04:50)



Now, the corresponding language for S can be written as we note it as L_S . The set of all Turing machines M and coding Turing machines M , such that $L(M)$, the language dimension satisfies the property P . So, that is how we describe the corresponding language for a problem? Now, our claim is that this L_S is undecidable, you know we want to prove that L_S is undecidable, and we prove this claim by reducing the halting problem; that means, the language M_{TM} to L_S . We just reduce halting problem to this problem; that means, a language M_{TM} will be reduce to L_S .

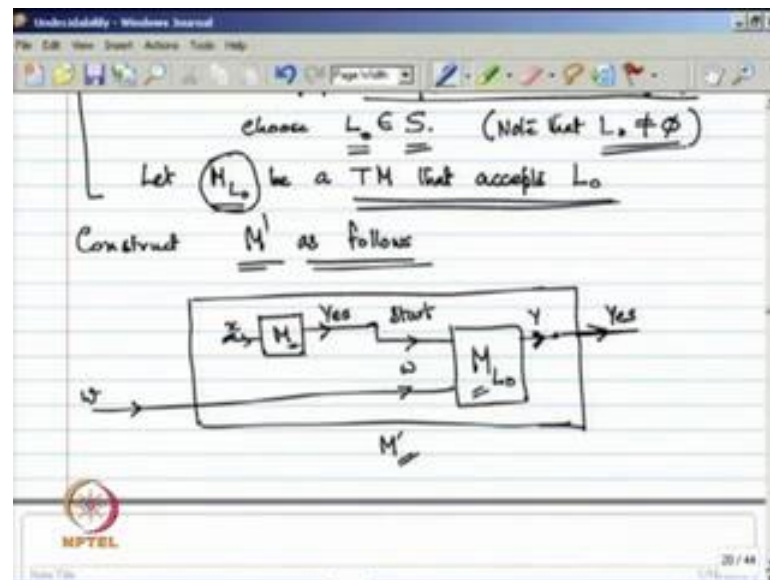
So, therefore, to do that what any to be is that given a Turing machine M and an input string x , we construct a Turing machine M' , such that x belongs to $L(M)$, if and only if $L(M')$ satisfies P , so that is the reduction. Given M an input string x , we need to construct Turing machine M' satisfying this x belongs to $L(M)$, if and only if $L(M')$ satisfies the property. In such a case, we are true.

(Refer Slide Time: 06:18)



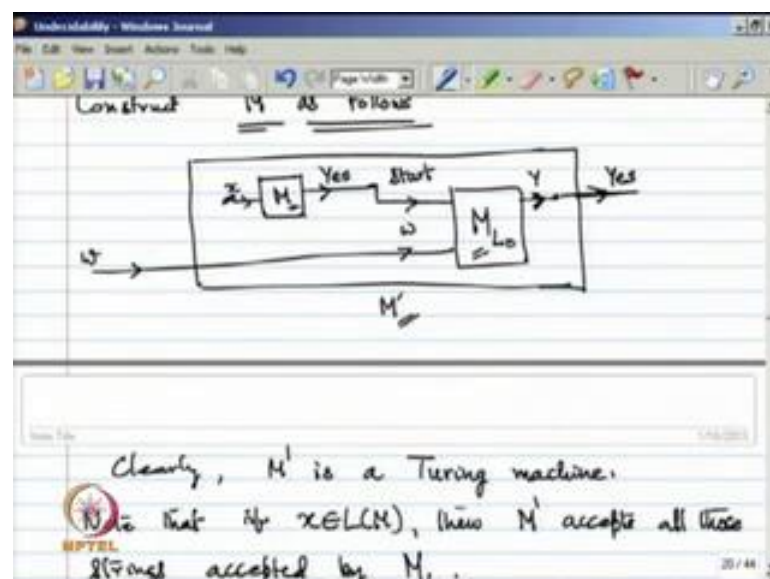
Now, we have say that S is a proper non empty subset of all recursively enumerable languages; that means, proper non subset of the class of recursive enumerable languages. So, therefore, we just consider any language L naught that belong to S , since it is not a non empty subset, such a language L_0 must exist in S , since it does not contain a ϕ . So, such a language L_0 must exist, also L_0 is not ϕ , because we say that if S contains ϕ , then we consider the compliment of the problem. So, L_0 is not ϕ . Just consider, that M_{L_0} be a Turing machine that excepts L_0 , so there is a Turing machine accepting L_0 , and that is to that is the Turing machine M_{L_0} . From this we construct the Turing machine M' as follows Turing machine M' for (Refer Time: 04:50) such that at that satisfies P .

(Refer Slide Time: 07:36)



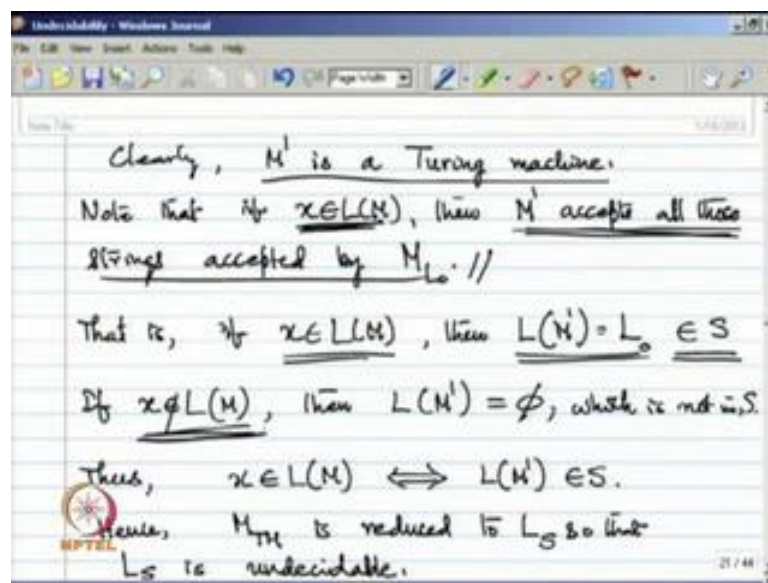
So, given any string as input w to M' , M' initially does not do anything, it first simulates M_1 on input string x . Now, if M_1 says Yes on the input string x , then we start the simulation on M_{L_0} on the given input string w , we start simulation and if M_{L_0} accepts then y acts as input to M' , if it accepts then M' also accepts it says yes, if it says no, if it says no or repeat does not say yes or it was answer yes.

(Refer Slide Time: 08:37)



Clearly M' is a Turing machine and we see that if x belongs to $L(M)$, x belongs to $L(M')$ then M' says yes, and then only we start M_0 on input string w . So, therefore, if x belongs to $L(M)$, then M' accepts all those strings accepted by M_0 , because if M_0 accepts a string, then M' says yes. So, if x belongs to $L(M)$, then M' accepts all those strings accepted by M_0 . So, this is quite clear, that is if x belongs to $L(M)$, then $L(M')$ is nothing but L_0 and which is which belongs to S . Similarly, if x does not belong to $L(M)$, then in case, when x does not belong to M , then M' does not say yes, since it does not say yes, M_0 is not started at all. So, therefore, M' will not accept anything.

(Refer Slide Time: 10:03)



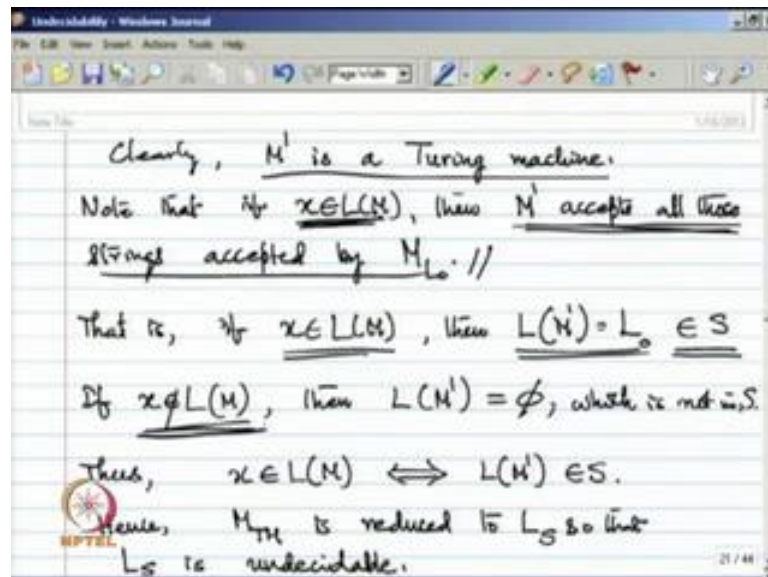
Therefore, if x does not belong to $L(M)$, then $L(M')$ must be empty, because M' does not accept, and which is not in S because you know that in S we do not have \emptyset . Thus, x belongs to $L(M)$, if and only if, $L(M')$ belongs to S , therefore this is a reduction. So, M TM here is reduced to L_S , so that L_S is undecidable.

Now, therefore setting different properties P , we now say that say if the property P may be avoided this language given recursion language is recursive or failure the language accepted by the Turing machine is regular, whether languages accepted by the Turing machine is context free, whether language accepted by the Turing machine is empty or finite and so on, we just said different properties.

And for the language accepted by Turing machine thus consider any property that defines a set which is non empty and a proper subset according to Rice's theorem, we can

say that all these are undecidable. So therefore, we can use this rice's theorem, to prove that so any property is non empty of course, and proper subset of all recursive element languages, that describes the non empty and proper set of recursion M languages are is undecidable.

(Refer Slide Time: 11:48)



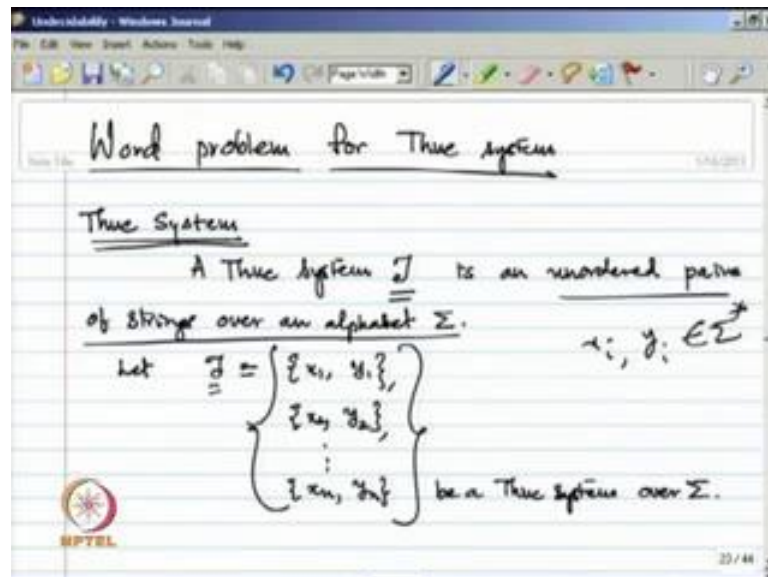
Now, we just consider a problem and take it an exercise, suppose M is a Turing machine and w be an input string, given w as input to M , will M eventually halt with empty tape, and its head on the left most cell sure is a question there is a problem. So, is this problem decidable or undecidable; that means, the recursion language is that L of H_0 is set of all strings Mw , such that M is a Turing machine like this with the elements $Q \Sigma^* Q$ naught and delta, and w is an input string.

And, if you start in the initial state, initial configuration initial state is q_0 , input is w , and head is placed it a cell just after write blanks at that separate input string w , so there is a initial configuration of the Turing machine M . So, in 0 or more steps, whether it will halt a string its head and the input tape is empty and placing in the heads on the left most cell, that is the language. So, set of all those strings Mw , such that M is a Turing machine, and Turing machine eventually halt starting with the initial configuration taking w as input, eventually halt with head on the left most cell.

So, what can done is that, we can reduce the halting problem to this problem called as problem say H_{naught} . So, accordingly the language is L of H_{naught} , we can reduce

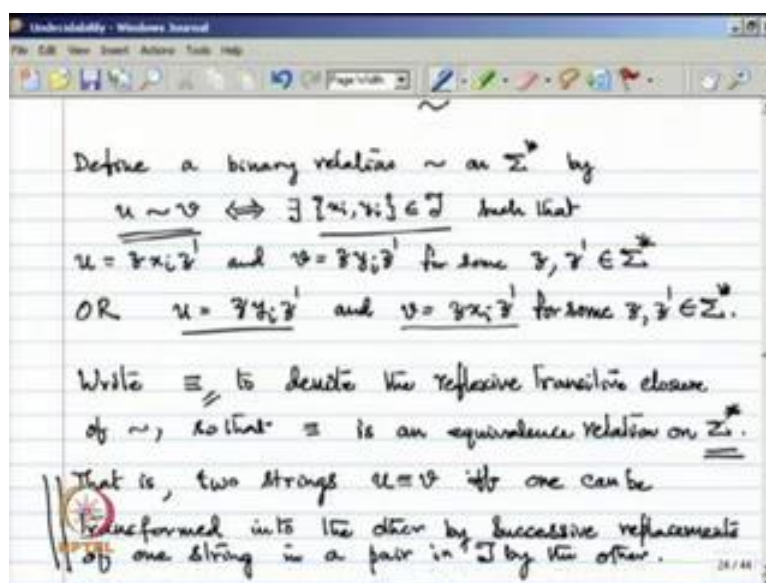
halting problem to this variant of halting problem, the halting is that you just remember that whenever the Turing machine halt in case of halting problem. So, this machine should erase its tape eventually. So, all the containable tape must be erased. (Refer Time: 14:11) Now we use this result, so show that some other process to me undecidable.

(Refer Slide Time: 14:24)



So, if introduce a problem which is called word problem for Thue system and eventually will show that this word problem is also undecidable, so first we define what a Thue system is, and then we define the word problem for Thue systems. The Thue system T is an unordered pair of strings over an alphabet Σ ; that means, this T is a set of ordered pairs so $x_1 y_1$, $x_2 y_2$ and so on up to say $x_n y_n$. So, all this strings $x_i y_i$ are strings over Σ . So, this belongs to Σ^* .

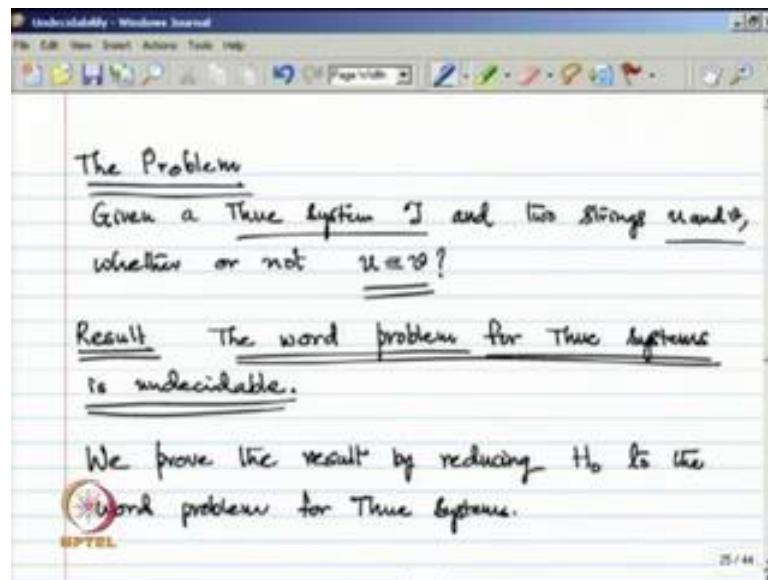
(Refer Slide Time: 15:24)



Now, we define a binary relation say it a tilde, so this binary relation tilde is defined on sigma star, we say that two strings even b or related by this relation binary relation tilde if and only if, there exist some pair $x \mid y \mid$ in that two system such that u can be written as $z \mid x \mid z$ dash and v can be written as $z \mid y \mid z$ dash for some $z \mid z$ dash and sigma star, or u can be written as $z \mid y \mid z$ dash and v can be written as $z \mid x \mid z$ dash for some $z \mid z$ dash. So, this is simple binary relation that may be defined on sigma star.

Now, we use this notation to denote the reflex reflexive transition closure of tilde. So, that this is an equivalence relation on sigma star, that is two strings even v are related by this relation; that means, u is equivalent to v , u is equivalent to b , if and only if one can be transformed into the other by successive replacement of one strings in a pair in the Thue system T by the other, it does not relate the order in any order you can apply it. So, you can study one can be transfer to the other string by some successive replacement of one string in a pair in a Thue system by the other. So, in such a case we say that u is equivalent to v .

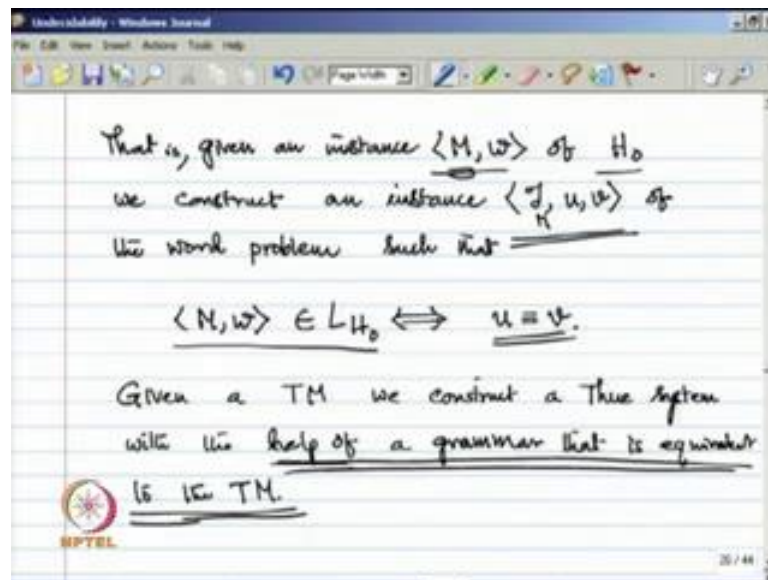
(Refer Slide Time: 17:28)



Now, we define the problem the what problem for Thue systems, the problem is that give a Thue system T , and two strings u and v for any Thue is an T , and any two strings u and v whether or not u and v are equivalent. So, that is the problem and we can show that this problem is called word problem for Thue systems is undecidable; that means, we cannot decide this word problem for Thue system.

We can prove this result by reducing that variant of halting problem that we have just introduced that H_0 to the word problem for the Thue system; that means, we will consider this problem H_0 , which is a modified variant of the original halting problem we can use this. That means, we can reduce this problem H_0 to the word problem, to show that word problem is also undecidable. We use H_0 , reducing H_0 to word problem we can showed a Thue system word problem for Thue system is undecidable.

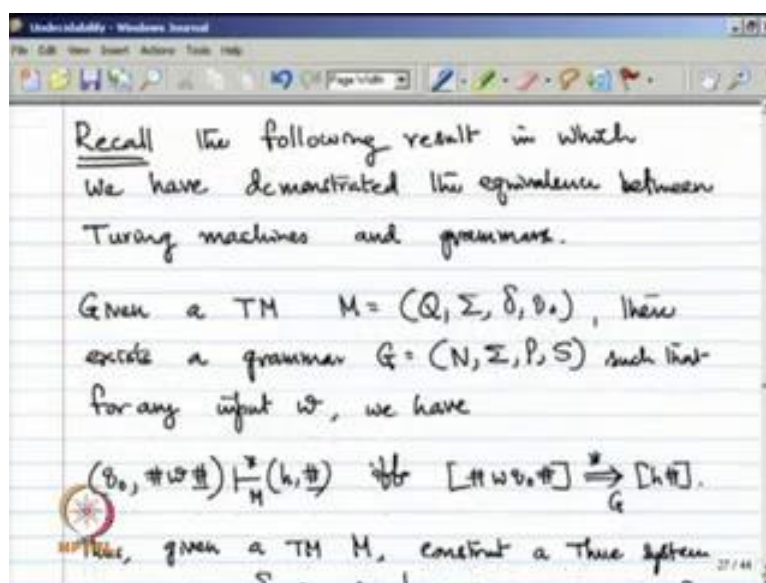
(Refer Slide Time: 18:55)



That means given an instance $M w$ of H_0 , we construct an instance of the word problem which is $T M$ corresponding Turing machine M , we construct $T M u v$, such that $M w$ belongs to L of H_0 , if and only if, u and v are equivalent, so that is what you want to do. From a given instance of H_0 , we construct an instance of the word problem; that means, a Thue system and two strings u and v , such that for any $M w$, if $M w$ belongs to $L H_0, M w$ we can say $L M_0$, if and only if, u and v are related u and v are equivalent.

So, given Turing machine we construct a Thue system with the help of a grammar that is equivalent to the Turing machine. Now, we know that already it has been given or proved that for every Turing machine, there is an equivalent grammar, and it does not shown, how to construct the grammar rules for any given Turing machine? So, that the grammar and a Turing machine are equivalent, so will use that construction to construct the Turing Thue system from the given Turing machine $T M$.

(Refer Slide Time: 20:39)



Now, recall the result in which we have demonstrated already the equivalence between Turing machines and grammars; that means, given Turing machine M . So, the Turing machine, so there exist a grammar G , such that for any input w , we have if we start in the initial configuration, if the Turing machine in 0 or more steps, eventually goes to these configuration; that means, the halt state. So if and only if, so this string corresponding to this initial configuration derives this string corresponding to the halting configuration.

So, you need to recall the result that you have already proved corresponding to in the context of proving that Turing machine and grammars are equivalent. Now, once we have that given a Turing machine M , we simply construct a Thue means Thue system by collecting all those pairs, so, $x y$ such that $x y$ is a rule in the grammar G .

So for Turing machine M , we can construct a grammar G , in the grammar G we have rules like x goes to y , so collect all those rules, from those rules you collect all the pairs $x y$. And those are the pairs that we have in the corresponding Thue machine $T M$, so corresponding Turing machine M , we have the corresponding Thue system $T M$. So, that is how we construct the Thue system for a given Turing machine.

(Refer Slide Time: 22:48)

Now, for any input w of M , consider

$$u = [\#wq_0\#] \text{ and } v = [h\#].$$

We show that

$$\langle M, w \rangle \in L_{H_0} \iff u \equiv v \text{ in } T_M //$$

So that the problem H_0 is reduced to the word problem

Note that

$$\langle M, w \rangle \in H_0 \iff (q_0, \#w\#) \xrightarrow{*}_M (h, \#)$$

$$\iff [\#wq_0\#] \xrightarrow{*}_G [h\#]$$

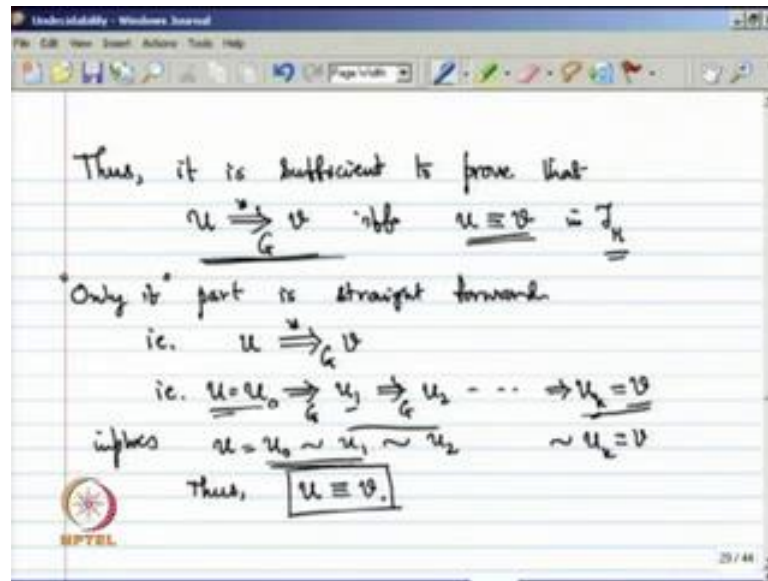
ie $u \xrightarrow{*}_G v.$

Now, for any input w of M , we consider two strings, so u v is considered to be left square bracket has w Q_0 has right square bracket and v is left square bracket H for H is the halting state has right square bracket. Now, you want to show that the string M w belong to L of H_0 , if and only if u and v are equivalent in the corresponding Thue system T_M , that we construct from M . So, that the problem H_0 is reduce to the word problem, showing that the word problem for Thue system is also undecidable.

Now, we just note that M w belongs to H_0 , these should have been L of H_0 , if and only if, the Turing machine starts in the initial state taking w as input, it process the input string eventually it enters in halt state; that means, from this configuration Q_0 has w , has in 0 or more steps in M , it arrives the configuration when H is the state, and head is pointing to the blank cell.

So, L w belongs to L of H_0 , if and only if these are situation, if this is the case, according to our equivalence of Turing machine M grammar. We know that these string, corresponding to this initial configuration, we will derive in 0 and more steps, these string corresponding to the halting configuration; that means, since we have say that this string is not denoted by I said u . Already, we have said that this a string u and this a string v , so therefore, this says that u the string u derives 0 are most option G the string v . So, M w belongs to L of H_0 , if and only if, u derives the string v for u is this string, and v is this string.

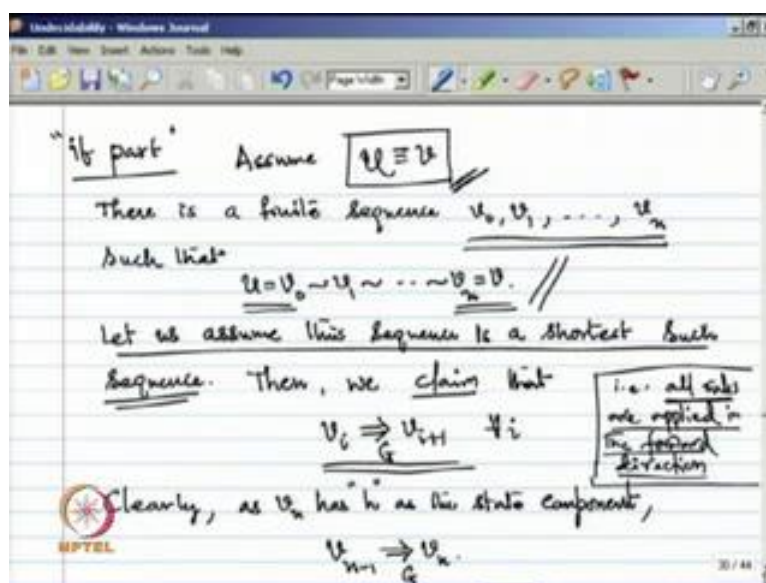
(Refer Slide Time: 25:57)



Thus, in a sufficient proved at u derives v in G, if and only if, u and v are equivalent in the Thue system. So, we have just simplify the problem or restate it. Now, in this case which as observed at only if for is very straight forward; that means, u derives v in G in 0 or more steps.

Suppose that is a case, in such a case what will have is that there is a sequence of derivations sequence steps in revision, where u_0 derives u_1 in one step, u_1 derives u_2 in one step and shown up to say u_{k-1} derives u_k in one step, while u_0 is u and u_k is v, therefore u derives v in 0 or more steps in grammar G, so in such cases, so therefore, since grammar rules and the pairs in Thue system are parallel. So, what you can say is that u_0 is related to u_1 by binary relation \sim the u_1 is related to u_2 by the same binary relation and so on. So, therefore, according to our definition of their equivalent relation, we know that u is equivalent to v. So, this part only part is quite straight forward.

(Refer Slide Time: 27:34)



The only difficult part is that the if part; that means, assume that u is equivalent to v . So in such a case, if u is equivalent to v , then you need to show that u derives b in 0 or more steps that is what you understood. Suppose u is equivalent to b , then there must exist a finite sequence v_0, v_1 or to say v_n , such that v_0 is u , v_0 and v_1 are related, v_1 v_2 are related by a binary relation tilde and so on, v_{n-1} on v_n is related by binary relation tilde the v_n is v . So, they must exist such a sequence, according to the definition of this since u and v are equivalent.

Now let us assume, this sequence is a shortest sequence because there may be many such sequences assume that out of all those this is this sequence is a shortest; that means, there is no such and this such that there is a sequence v_0, v_1 up to say v_n dash, where v_0 is where to be v_1 , v_1 will to be v_2 and so on up to v and thus so on relate to v , and thus where v not is u , and v and thus is say v , and this is less than n . So, this cannot happen because you say that this is a shortest sequence. So, n is always less than any sets and thus if subsequence exist, so this is our assumption.

So, if that is the case we claim that v_i derives v_{i+1} for all i in this sequence; that means, what you say that all rules in the derivation are applied in the forward direction; that means, v_i derives v_{i+1} is not a case that v_{i+1} derives a v_i . So, first you show that all the rules that the derivation are always applied at every state it is applying the forward direction.

Now, the first into note is that v_n has H as the state component because we have defined v_n to be since v_n equal to v , and we have say that v is so we have say that v is a string left square bracket H has right square bracket. So, this H is a halting state is there in v we nothing but the halting state H is there in v , which is v_n which is nothing but v . So, therefore, v_n means one must derive v_n that is at this step when v_n is one derives v_n in grammar G . The rule must be applied in the forward direction, because from halting state we do not go to any other state from other state only we can go to the halting state. So, therefore the last step in the derivation, the rule must be applied in the forward direction, so that is our claim. Now, suppose that so, since we claim that v_i derives v_{i+1} for all i that is about we want to prove that, if that is not a true there must be some step where this is halted.

(Refer Slide Time: 32:05)

Let i_0 be the largest number such that

$$v_{i_0} \xrightarrow{G} v_{i_0+1} \parallel (0 \leq i \leq n-1)$$

Then, we have $v_{i_0+1} \xrightarrow{G} v_{i_0+2} \parallel$

Also, we have $v_{i_0+2} \xrightarrow{G} v_{i_0+3} \parallel$

But, since M is deterministic, the rules in G give us that $v_{i_0} = v_{i_0+2}$

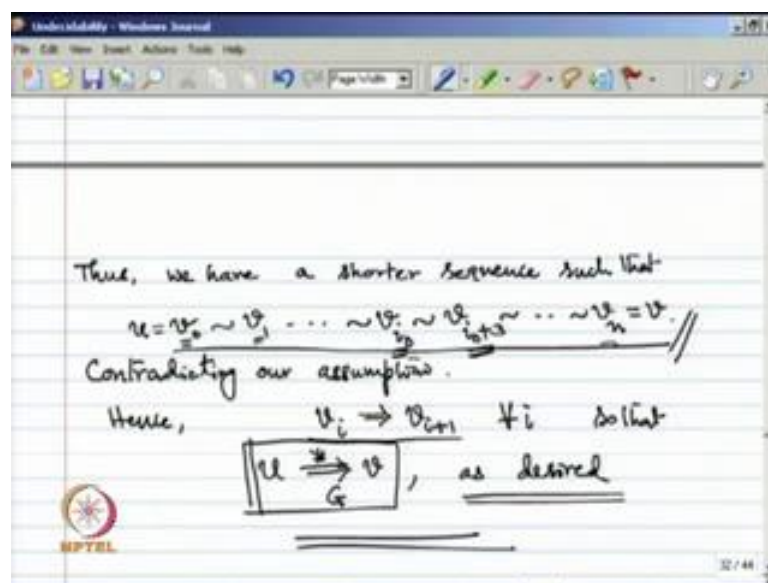
Suppose that i_0 is the largest number where i_0 is basically may be greater than equal to 0 or less than n minus 1 because for n minus 1 we have already assume that there will be applied now for duration. So, suppose that i_0 is a largest number. So, for this is highlighted; that means, v_{i_0} does not derive v_{i_0+1} in G , then we must have that v_{i_0+1} derives v_{i_0} in G ; that means, rule is applying the by quadration.

But already, we know that v_{i_0+1} derives v_{i_0+2} in one step, because i_0 is the point i_0 is the number largest number, why they we use file result, beyond that w is not

halted. So, therefore, $v_i 0 \text{ plus } 1$ must derive $v_i 0 \text{ plus } 2$ in G that must be the case and we have say that the rule is at this point. So, therefore, $v_i 0 \text{ plus } 1$ derives $p_i 0$.

Now, while consider Turing machine, if we assume that Turing machine is deterministic then what we have is that the grammar rule must be unique. So therefore, v since $v_i j$ plus 1 derives $v_i 0$ also $v_i j$ derives $v_i 0$ also $v_i j$ process from derives $v_i 0$ cross two non step. So, $v_i 0$ and $v_i 0 \text{ plus } 2$ must be the same string, because the Turing machine M can be assume to revenuestic and hence the rules will be unique.

(Refer Slide Time: 34:02)

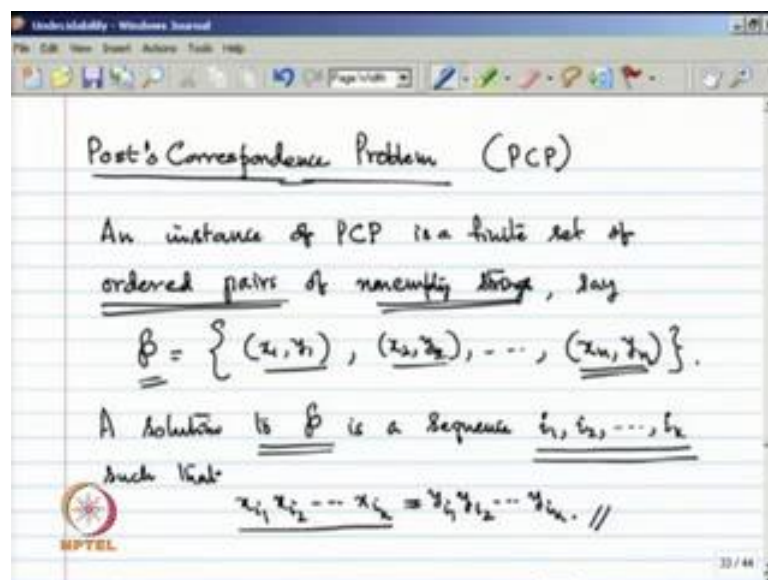


So, therefore, now we have the sequence say u is v_0 , v_0 let to be u_1 and so on. So, $v_i 0$ is related to now in set of say $v_i 0 \text{ plus } 1$ since $v_i 0 \text{ plus } 1$ derives $v_i 0 \text{ plus } 2$, and $v_i 0 \text{ plus } 1$ derives $v_i 0$ and these two are identical. So, therefore, we have the sequence now, where $v_i 0$ and $v_i 0 \text{ plus } 3$ are related by tilde and so on up to v_n by v_n is p .

Now, in this case since from $v_i 0$, we get $v_i 0 \text{ plus } 3$, these two related or original assumption that the sequence v_0, v_1 up to $v_n, v_0, v_1 v_2$ up to v_n , this is a shorter sequence that whatever this is the assumption that must be first because in the in that sequence, so what we have v_0, v_1 say $v_i 0$, then $v_i 0 \text{ plus } 1$, then $v_i 0 \text{ plus } 2$, then on $v_i 0 \text{ plus } 2$. So, this must be longer sequence corresponding to the sequence that we have just introduced. So, that must be any longer sequence corresponding this results all are sequence.

So, this contradicts our original assumptions that successive shorter sequence exist. So, therefore, v_i must derive v_{i+1} for all i . So, therefore, even v must be related u , u must derive v in 0 or more steps in G , applying the forward rule at every state. So, that is what we want to prove; that means, u derives v in G in 0 more steps, if and only if, u and v are equivalent, so we have shown the both size if for n and if for n . So, therefore by reducing this variant, or the halting problem to this what problem for Thue system. We have shown that what problem for the what problem for Thue system is also undecidable. Similarly, we can show many other problems to me on undecidable by using this kind of reduction.

(Refer Slide Time: 36:58)

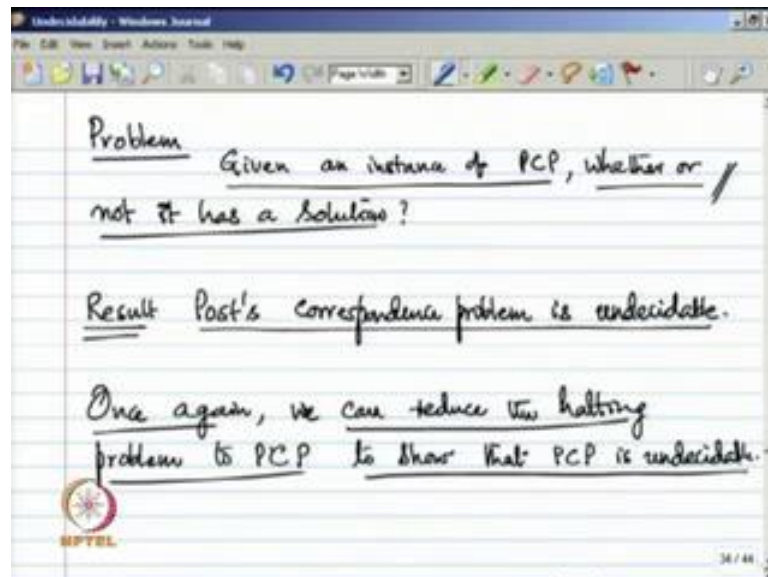


Will now introduce another problem, where is called a similar to what problem in Thue system is called post correspondence problem, and will show that post correspondence problem is also undecidable. Let us first this define an instance of PCP there is post correspondence problem, an instance of PCP is a finite set of ordered pairs. So, in this case these pairs are all ordered finite set of ordered pairs of non empty strings say it is a post in sense of post correspondence problem to P is $x_1 y_1, x_2 y_2, x_3 y_3$ and so on $x_n y_n$. So, it is a finite set of reversal strings over Σ and this pairs are ordered.

Now a solution to P are is a sequence, so listen to these instant of P instead a PCP is a sequence $i_1 i_2$ up to i_k , such that if you take the corresponding strings from the ordered pairs. The first element and the second element and concatenate they will be identical;

that means, $x_{i1}x_{i2} \dots x_{ik}$, concatenate $x_{i1}x_{i2}$ upto say x_{ik} , and concatenate $y_{i1}y_{i2}$ upto to y_{ik} , there said $x_{i1}x_{i2}$ upto x_{ik} is equal to $y_{i1}y_{i2} \dots y_{ik}$. So, if such a sequence x is satisfying these condition we say that this is a solution to the instance the PCP.

(Refer Slide Time: 38:55)

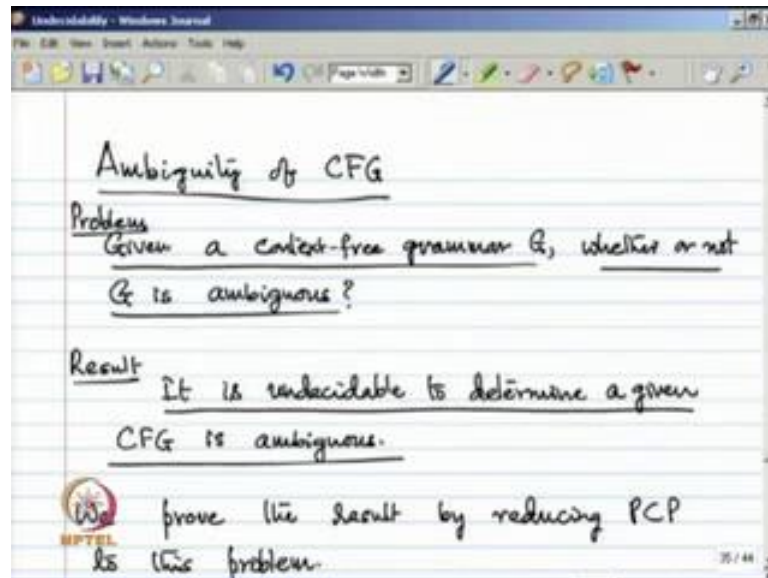


Now, the problem is given an instance of PCP whether or not it has solution that is our problem. So, can we decide it the result is that post correspondence problem is undecidable, so we can show that the post correspondence problem is also undecidable. To show this first for it do we consider a modified for some the post correspondence problem, because given an instance of PCP related to find out the sequence $i1$ $i2$ up to say ik satisfied that condition; that means, $x_{i1}x_{i2}$ up to x_{ik} must be equal to $y_{i1}y_{i2}$ up to y_{ik} .

The problem is how to start, or just index $i1$ thus difficult to find out may be if we consider suppose that we need to start at say index 1 and then you find all those remaining the sequence; that means, if the case that $x_1x_{i1}x_{i2}$ up to say x_{ik} equal to y_1 start with the index 1 $y_{i1}y_{i2}$ up to say y_{ik} , so this is solution to the NPCP. So, this modified version is said to be modified PCP, simply MPCP we can show that MPCP is decidable if PCP is decidable. So, you can reduce this problem by reduction you can show this result.

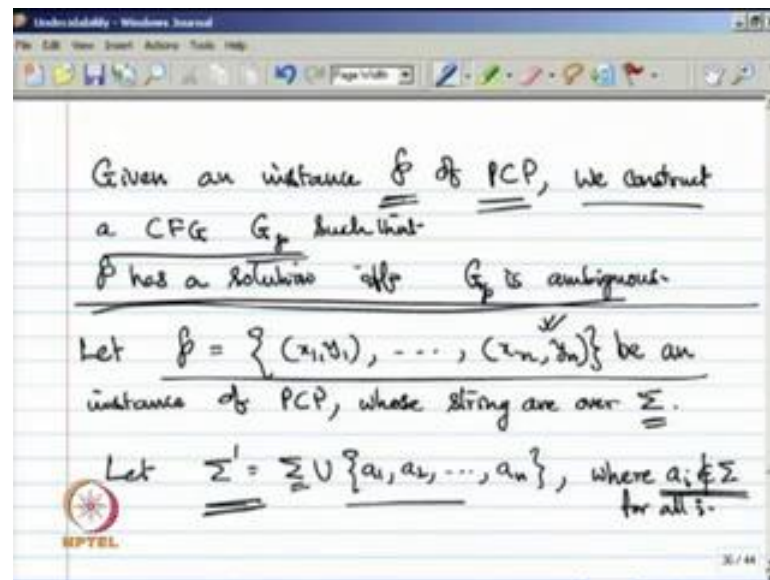
Now, this halting problem again can be reduced to this MPCP to show that MPCP is undecidable. So, you leave it an exercise you can give a time to prove the result. So, once again we can reduce the halting problem to PCP, in fact to MPCP to show that PCP is undecidable. Now, this result that PCP is undecidable can be used to show that some problems, so let to CFG are also undecidable.

(Refer Slide Time: 41:56)



For example, just consider a problem of ambiguity of CFG that means given a context free grammar G whether or not G is ambiguous can we still given ambiguous CFG can we decide whether G is ambiguous. The result is that is undecidable to determine a given CFG is given, CFG is ambiguous, so we cannot decide over any given ambiguous CFG is ambiguous. Now can use this PCP, to show that and make it to problem of CFG is undecidable; that means, we reduce PCP to this problem.

(Refer Slide Time: 42:46).



Now, the reduction for any to do is that given an instance P of PCP we construct a CFG G_P from this instance P of PCP we construct an instance CFG G_P such that this PCP has a solution if and only if the corresponding G_P is ambiguous. So, these a reduction if can show this we know that and make it a problem of CFG is also undecidable

Now, let see the reduction, how can you reduce it? Consider that P is an instance of the PCP, where all the strings $x_i y_i$ are strings over Σ , some other forward Σ . We just consider another alphabet say Σ' this where it contains all the in symbols from Σ , and Σ' union and some additional symbols $a_1 a_2$ up to a_n , where the numbers of such symbols is equivalent to the number of such pairs in this PCP is that is. So, $x_1 y_1$ up to we have $x_n y_n$, so we have n numbers of additional symbols in Σ' will contain all the symbols of Σ as well as. So, additional symbols; that means, a_i does not belong to Σ for all i and there n numbers of such symbols where n is on number of pairs in the PCP.

(Refer Slide Time: 44:36)

Set $G_1 = (\{S_1\}, \Sigma, P_1, S_1)$, where

$$P_1 = \left\{ \begin{array}{l} S_1 \rightarrow a_i S_1 x_i \\ S_1 \rightarrow a_i x_i \end{array} \right\} \quad (i=1,2,\dots,n)$$

Note that $L(G_1) = \{a_1 a_2 \dots a_m x_1 x_2 \dots x_m \mid m \geq 1\}$

Similarly, let $G_2 = (\{S_2\}, \Sigma, P_2, S_2)$, where

$$P_2 = \left\{ \begin{array}{l} S_2 \rightarrow a_i S_2 y_i \\ S_2 \rightarrow a_i y_i \end{array} \right\} \quad (i=1,2,\dots,n)$$

Note that $L(G_2) = \{a_1 a_2 \dots a_m y_1 y_2 \dots y_m \mid m \geq 1\}$

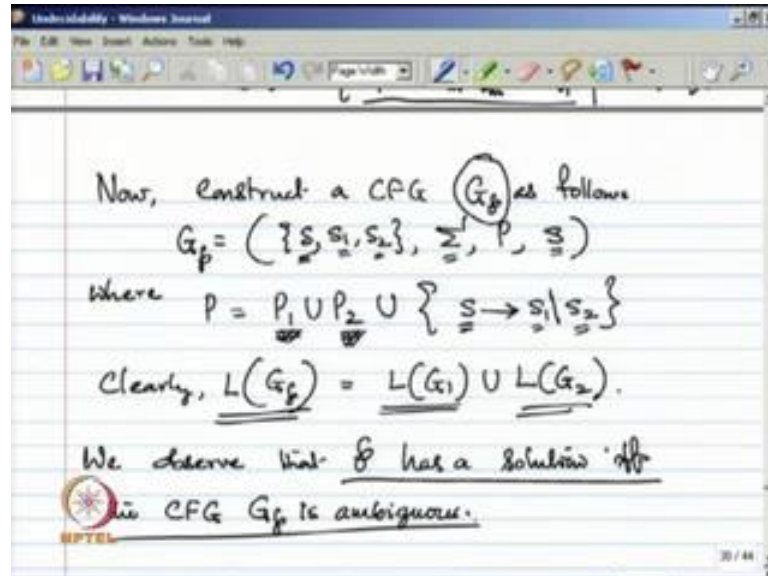
Now, we construct a grammar say G , G contains a single non terminal S is alpha by the sigma this, star symbol is; obviously, it must be S , and a set of productions P . So, P contains to the set productions, S goes to $a_i S x_i$ for all i , i equal to 1 through n and S goes to $a_i x_i$, so there only two rules, that we have for all $a_i x_i$ of for all for all i .

Obviously, we will see that the language of the grammar G , I will be of the form $a_i 1$, $a_i 2 a_i m$, then $x_i m x_i 2 x_i 1$ for some M greater than or equal to 1, because since we have to star symbol, it has some a from the alphabet, and that corresponding x_i will be there in the other side and in between we have this S . This S again we have to replace S by the same will may be then we have another a_i for some i and a corresponding x_i 1 eventually we have terminated by this two this S has to terminated by this two, we reduce a $a_i x_i$. So, therefore, every string must be of this one.

So, this $a_i 1$ the last symbol the last string has to be $x_i 1$, $a_i 2$ corresponding to these we have $x_i 2$ and so on, up to $a_i n$ when the no terminate, so it means x_i . So, that is how we get a strings of the language is an advisable, similarly G_2 , another gamma G_2 can be constructed. So, which S_2 is a star symbol, and sigma raised a alphabet S_2 has to be the star symbol here, and P_2 is a safe same similar kind of productions P_2 the set of all production at this S_2 goes to $a_i S_2 y_i$ and S_2 goes to $a_i y_i$ for all i . So, therefore, the

language generated by this contained the strings of this kind $a_1 a_2 \dots a_n y_i, n y_i$
 $m-1$ up to y_n, y_1 for m greater than or equal to 1.

(Refer Slide Time: 47:30)



Now from this G_1 and G_2 , we construct a grammar G which is called G_P corresponding to the instance PCP, the G_P as follows it contains besides S_1 and S_2 , one additional start symbol S which is a start symbol of the grammar over the same alphabet Σ .

And P contains all the rules of G_1 , all the rules of G_2 that is $P_1 \cup P_2$, and then from the start symbol S may have to go to start symbol G_1 or the start symbol G_2 ; that means, clearly the language of G_P is nothing but $L(G_1) \cup L(G_2)$ because from start symbol of S you go to either S_1 or to S_2 and then you used the corresponding rules from G_1 or from G_2 , so that $L(G_P)$ is union of $L(G_1)$ and $L(G_2)$. Now observe that P has a solution if and only if, CFG G_P is ambiguous. So, this will prove our claim.

(Refer Slide Time: 48:41)

Let x_1, x_2, \dots, x_k be a solution to P .
 i.e. $x_1 x_2 \dots x_k = y_1 y_2 \dots y_k$.
 Now, observe that the string
 $w = a_{i_k} a_{i_{k-1}} \dots a_{i_1} x_1 \dots x_k$
 has two leftmost derivations in G_p , as below.

$$\begin{aligned} S &\Rightarrow S_1 \Rightarrow a_{i_k} S_1 x_{i_k} \\ &\Rightarrow a_{i_k} a_{i_{k-1}} S_1 x_{i_{k-1}} x_{i_k} \\ &\vdots \\ &\Rightarrow a_{i_k} \dots a_{i_2} S_1 x_{i_2} \dots x_{i_k} \\ &\Rightarrow a_{i_k} a_{i_k} a_{i_1} x_1 x_2 \dots x_{i_k} = w \end{aligned}$$

Now, assume that x_1, x_2 up to x_k be a solution to P , if that is a case $x_1 x_2 \dots x_k$ must be equal to x_1, y_1, y_2 up to y_k . Now, observe that the string $a_{i_k} a_{i_{k-1}} \dots a_{i_1}$, then $x_1 x_2$ up to x_k has two left most derivation in G_p as shown below. We start it S in one stable go to S_1 in one step, you apply S_1 goes to $a_{i_1} S_1 x_{i_1}$, the next step you apply the rule $S_2 a_{i_k} \dots a_{i_1} S_1 x_{i_1} \dots x_{i_k}$ and so on. Eventually, we apply that terminated by as goes to $a_{i_1} x_1$ and hence this is nothing but w .

(Refer Slide Time: 49:54)

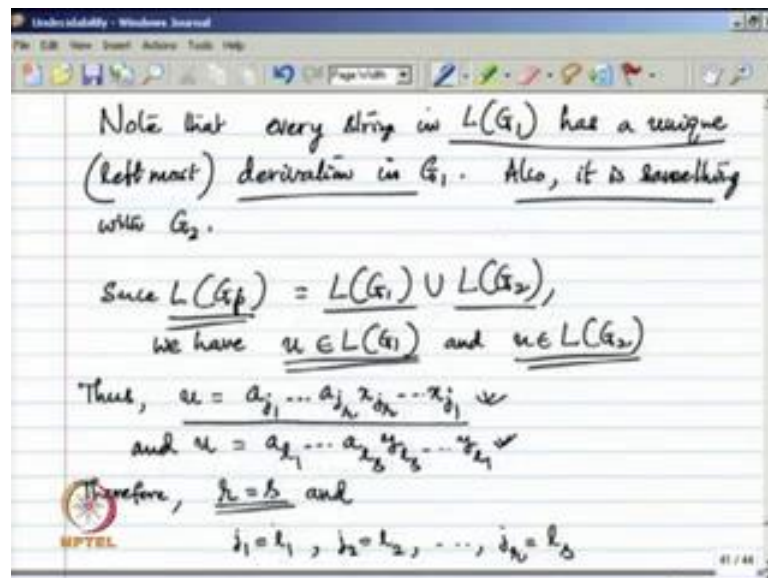
$$\begin{aligned} S &\Rightarrow S_2 \Rightarrow a_{i_k} S_2 y_{i_k} \\ &\Rightarrow a_{i_k} a_{i_{k-1}} S_2 y_{i_{k-1}} y_{i_k} \\ &\vdots \\ &\Rightarrow a_{i_k} \dots a_{i_2} S_2 y_{i_2} \dots y_{i_k} \\ &\Rightarrow a_{i_k} \dots a_{i_2} a_{i_1} y_1 y_2 \dots y_{i_k} = w \end{aligned}$$

Thus, G_p is ambiguous.

Conversely, assume G_p is ambiguous.
 Then, there exists a string $u \in L(G_p)$
 with two leftmost derivations in G_p .

Now, similarly we can show that in one step from S we go to S_2 , then apply the rules first step $a_i k S_2 y_i k$, next step $a_i k S_2 y_i k$ minus 1 when solve terminate with it $a_i l y_i l$, but these one these string and these string both the identical, because already we have say that $i l i 2$ at $i k$ the solution PCP hence this part $x l i l x 2 i l x i 2 x i k$ and $y i l y i 2 y_i k$. This two are assign and first part is over sign $a_i k$ up to $a_i l$ $a_i k$ up to a_i one therefore, both had w , but distinctly this two are this one and this one are two different left most revision for the same string w and hence the grammar is ambiguous. Similarly, if you assume that conversely if you assume that G_P is ambiguous then you must adjust a string u that belongs to G_P , which has two left most derivations must have because you have say that G_P if empty case.

(Refer Slide Time: 51:13)



Now, since every string in $L(G_1)$ has the unique left most derivation in G_1 , also it is same thing for G_2 every string in $L(G_2)$ has unique left most derivation in G_2 . And, since $L(G_P)$ is union of $L(G_1)$ and $L(G_2)$, we have for u belong to $L(G_1)$ and u belongs to must belong to $L(G_2)$. Therefore, you must be some a_{j_1}, a_{j_2} up to a_{j_r} for some r then $x_{j_i} x_{j_i}$ mean show up to x_{j_1} , similarly u must be a_{l_1}, a_{l_2} up to a_{l_s} , then $y_{l_s} y_{l_s}$ minus 1 up to y_{l_1} . And therefore, r n s must be same, and j_1 must be equal to l_1 . So, that this two are identical, similarly j_2 and l_2 must be identical, and next two were identical up to a_{j_r} must be equal to a_{l_s} ; that means, j_r equal to l_s .

(Refer Slide Time: 52:37)

The image shows a digital notepad with handwritten mathematical derivations. The top part shows the construction of a string u from a sequence of indices i_1, \dots, i_n and a sequence of strings x_1, \dots, x_n . It states that $u = a_{x_{i_1}} \dots a_{x_{i_n}}$ and that $u = b_{y_{j_1}} \dots b_{y_{j_n}}$. Therefore, $u = v$ and $i_1 = j_1, i_2 = j_2, \dots, i_n = j_n$. The bottom part shows that $x_{i_1} \dots x_{i_n} = y_{j_1} \dots y_{j_n}$, which implies that i_1, i_2, \dots, i_n is a solution to the PCP instance P .

$$u = a_{x_{i_1}} \dots a_{x_{i_n}}$$

$$u = b_{y_{j_1}} \dots b_{y_{j_n}}$$

$$\text{Therefore, } u = v \text{ and } i_1 = j_1, i_2 = j_2, \dots, i_n = j_n //$$

$$\text{Hence, } x_{i_1} \dots x_{i_n} = y_{j_1} \dots y_{j_n} //$$

$$\text{So that } i_1, i_2, \dots, i_n \text{ is a solution to } P. //$$

Hence $x_{j_1} x_{j_2} \dots x_{j_n}$ must be equal to $y_{j_1} y_{j_2} \dots y_{j_n}$, so that j_1, j_2, \dots, j_n is a solution to PCP. So, this shows that the PCP instance that the instance of the instance P a PCP, P has solution if and only if the grammar G grammar G P that a construct from the instead a PCP, P is ambiguous. So, therefore, ambiguous says it determining ambiguous of CFG of any other CFG is undecidable, since we have given a reduction from the PCP to this make it a problem of CFG.