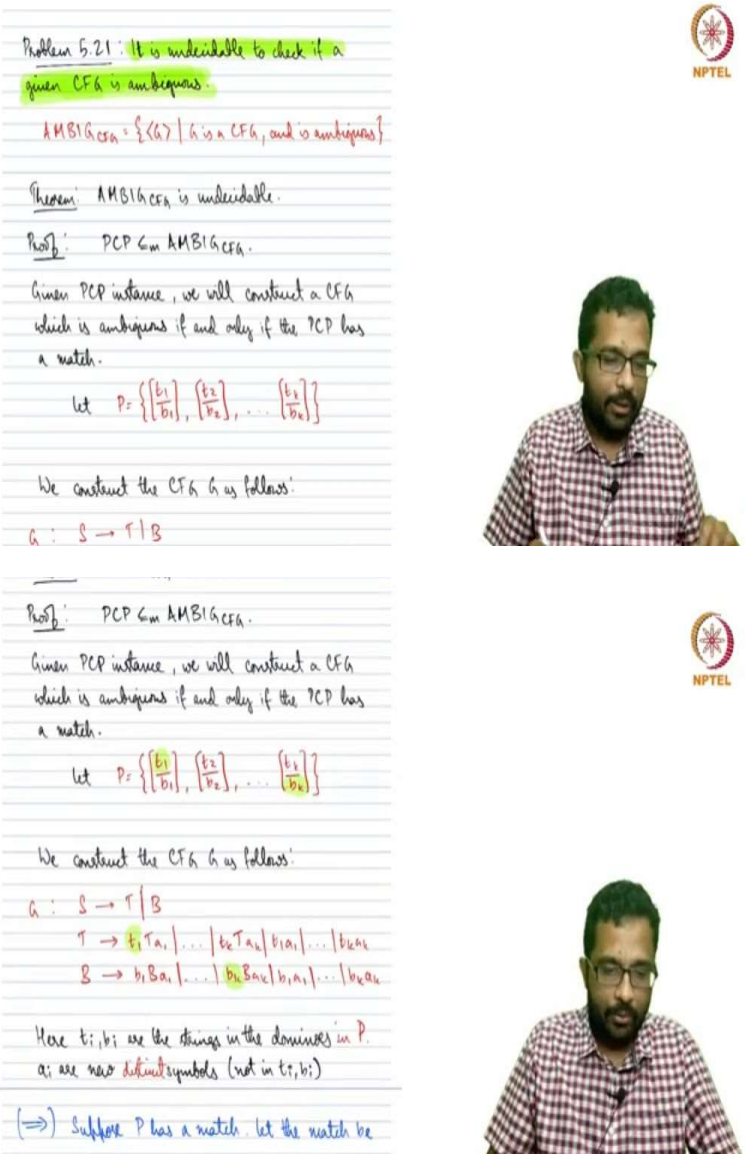
Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Checking Ambiguity in CFG is Undecidable

(Refer Slide Time: 00:16)



Hello and welcome to lecture 44 of the course theory of computation. In the previous lecture, we saw that PCP is undecidable. In other words, given a PCP instance, it is undecidable to determine whether the given instance has a match. If you recall, when we learned context-free grammar in the second chapter, we talked about ambiguity. So, we say that a grammar is ambiguous if some

string, if there is some string that can be derived ambiguously; meaning if there are multiple leftmost derivations for the same string, or there are multiple parse trees for the same string.

This was the definition of ambiguity. And we also mentioned that there are some languages, some grammars that are ambiguous; sometimes you can remove the ambiguity from the grammar. Anyway, in this lecture, we are going to see an application of the fact that PCP is undecidable. So, we are using the fact that PCP is undecidable, to show the undecidability of determining whether a given grammar a context-free grammar is ambiguous.

So, this is not there in the textbook as a text material; but it is there as a problem, problem 5.21. So, given a CFG, it is undecidable to determine whether the given CFG is ambiguous. So, we can also write it as a language; we can write it as a language $AMBIG_{CFG}$, as a language of all the ambiguous CFGs. And this is an undecidable language; meaning given a grammar it is undecidable to determine whether the grammar belongs to this class.

And the proof is by using a reduction from PCP. So, given a PCP instance will build a context-free grammar, such that the context-free grammar is ambiguous if and only if the PCP instance is a match. So, I am talking about PCP. So, let us see the reduction. The reduction itself is not that difficult to see and even the correctness is also not that difficult to see; so let P be this.

So, where setup dominos t1 b1, t2 b2, up to tk bk, where t1 b1 all come from some alphabet.

$$P = \{\left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right]\}$$

They are strings from some alphabet. The context-free grammar is constructed as follows. The first rule is that S yields T or B. So, S can give rise to T or S can give rise to T; so, basically these are two branches. So, S derives T, then you can only use rules for T; because T never gives rise to B. So, what are the rules?

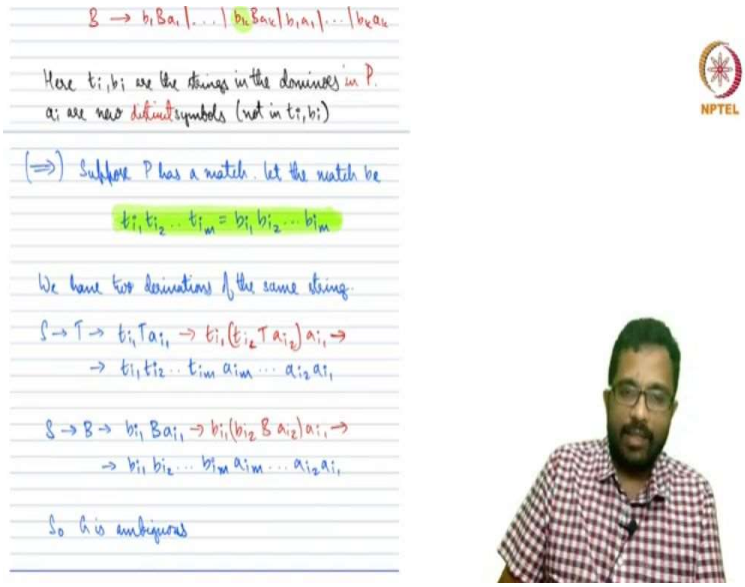
So, T gives $t_1\ T_{a_1}$, where this t_1 is the same as this t_1 in the PCP instance; so, t_1 is a string. And similarly, $t_2\ T_{a_2}$ and so on up to $t_k\ T_{a_k}$; so, $t_1\ T_{a_1}$, $t_2\ T_{a_2}$ up to $t_k\ T_{a_k}$. And finally, and also $t_1 a_1$, $t_2\ a_2$ up to $t_k\ a_k$ this part, where there is no variable involved; this is one branch of computation.

The other branch involves the variable B; so, it is similar. So, where B gives rise to $b_1 \mid B_{a_1}$, where b1 is corresponding to the b1 from the PCP instance.So, $b_1\ B_{a_1}$, $b_2\ B_{a_2}$ so on up to $b_k\ B_{a_k}$; so bk corresponds to the bk in the PCP instance. So, b1 corresponds to b1 here, b2 corresponds to the b2 here and so on; everything corresponds to.

So, T1, T2 et-cetera are the strings in the PCP instance; dominos in P. And a1, a2 up to ak are all new symbols; they are not strings, they are just symbols. And they are all distinct new symbols. Maybe I should write that even a1, ak are all new and distinct symbols; meaning a1 is different from, a2 is different to ak. So, we have k new symbols a1, a2 up to ak; and these are the rules S gives T or B. And once we get T, then $t_1\ T_{a_1}$ up to $t_k\ T_{a_k}$; then $t_1\ a_1$, $t_2\ a_2$ up to $t_k\ a_k$.

Then, B gives $b_1\ B_{a_1}$ up to $b_k\ B_{a_k}$, so, these are the rules of the grammar. So, the grammar is very easy to construct once we have the PCP instance; we do not need to check whether they are matching or anything. Now, all that remains to be shown is the correctness of this reduction. So, we are told this given a PCP instance, how to construct the grammar; so grammar is very easy to construct. What remains to be shown is PCP instance has a match if and only if the grammar is ambiguous. So, now let us see that.

(Refer Slide Time: 06:02)



Suppose the PCP instances has a match; let us see what is the ambiguity. Let the match be given by repeating the dominos or placing the i_1, i_2 up to i_m. So, i_1 could be anything from 1 to k, i_2

could be anything from 1 to k and so on up to i_m; so, this is such as new subscripts. So, when you line up this i_1^{th}, i_2^{th} domino and so on; the top string will be t_{i1} , t_{i2} up to t_{im}; and the bottom string will be b_{i1} , b_{i2} up to b_{im} .

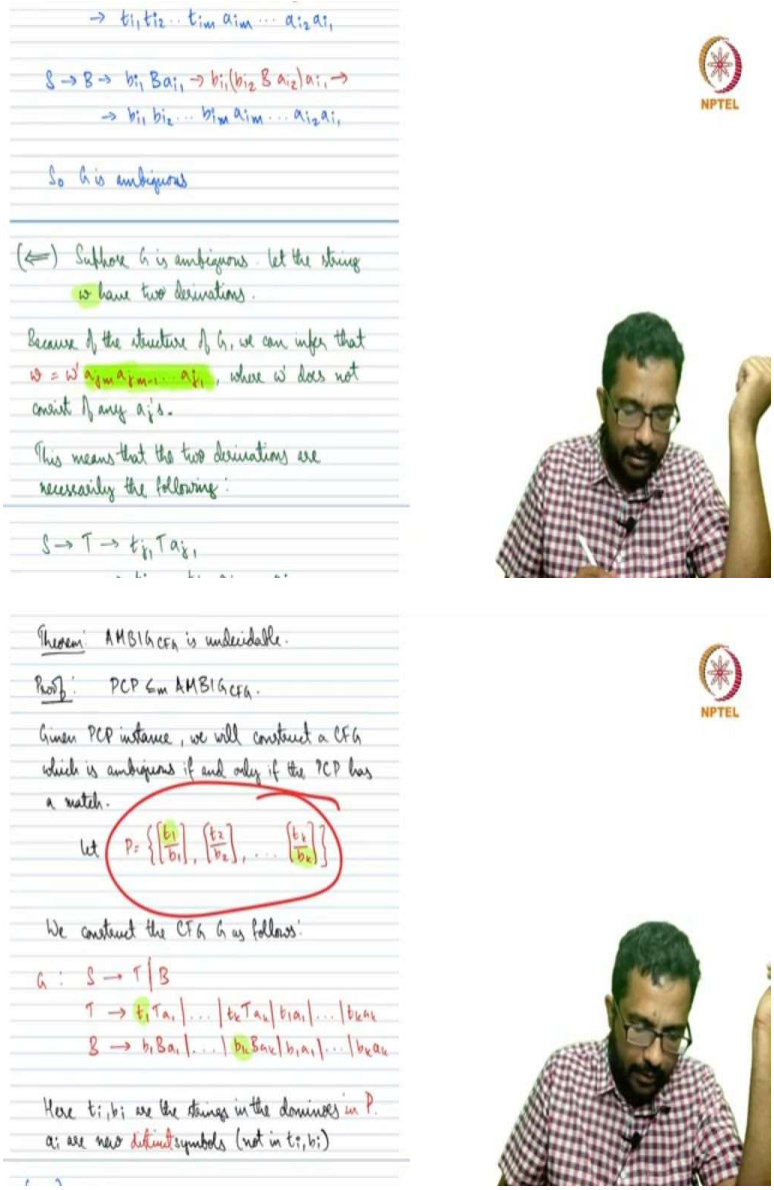
So, since they are a match, it means that the top string must equal to the bottom string t_{i1} , t_{i2} up to t_{im} should be equal to b_{i1} , b_{i2} up to b_{im}. Now, we will demonstrate two derivations of the same string. So, what are the derivations? First is that S gives T . So, we know, so we use T to give $t_{i1}T_{a_{i1}}$. So, let us say if i_1 is 2, we will first give t_2T_{a2}. So, if i_1 is 1, it is t_1T_{a1}; and then we will use that to give $t_{i2}T_{a_{i2}}$ and so on till we get.

So, maybe I will just write one more step; t_{i1} is already derived, then maybe $t_{i2}T_{a_{i2}}$ and so on till we get t_{i1} , t_{i2} up to t_{im} , followed by a_{im}, $a_{im\,-\,1}$ up to a_{i2}, a_{i1}. So, we can get this string over here. Very similarly, we can use S gives B; B gives $b_{i1}\,B_{a_{i1}}$. And then I can use similarly $b_{i1}($ $b_{i2}\,B_{a_{i2}})a_{i1}$ and so on. Finally, I have b_{i1} b_{i2} b_{im} a_{im} a_{i2} a_{i1} .

And I claim that these two strings the underlying strings are the same. Why is that? Because the first part t_{i1} up to t_{im} and b_{i1} up to b_{im} are the same; because of the assumption that P has a match, the PCP instance has a match. So, this part is the same. And the latter part is also the same; because it is a_{im} up to a_{i1} and a_{im} up to a_{i1} , this is literally the same; so, this is basically the same. Whereas this, for this part, we are using the fact that the PCP has a match.

So, we have one string with two derivations; and these are clearly two derivations because the first step itself is different, S gives T and S gives B. So, this means that the same string has two derivations; so that is the definition of the grammar being ambiguous. So, that completes the proof that PCP has a match implies that the grammar is ambiguous; now the reverse direction.

(Refer Slide Time: 09:37)



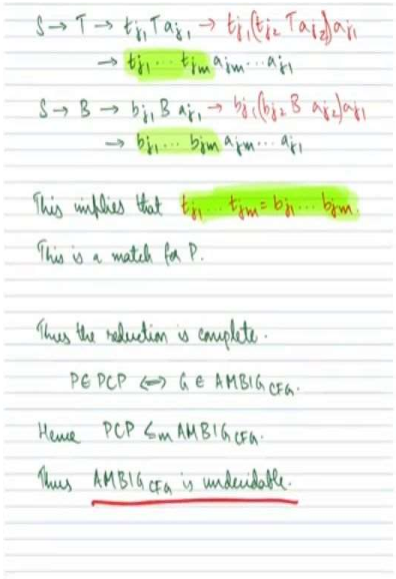
If the grammar is ambiguous, we have to show that the PCP has a match. So, let there be some string which has two derivations; let the string be w, let it have two derivations. Now, notice the structure of the grammar here. Whenever we apply a rule, let say we apply a rule, then some a1 or a2 or something gets appended towards the end of the string; whichever rule, this rule or this rule or this rule, or this rule or any any rule, where we accept the first step of the derivation, where S gives T | B.

Everything else involves the creation of a_1, a_2, a_3, a_k something; one of these ai's should be produced at the end. So, because of the structure of the grammar, we may assume that the string ends with a string of a's. So these are the all the a's; the a's do not come in between, they only come at the end. So let the string of a's be a_{j_m} , $a_{j_{m-1}}$ up to a_{j_1} ; so I am highlighting that.

So, let this the string, let the string derived ambiguously end with a_{j_m} , $a_{j_{m-1}}$ up to a_{j_1} . So, let us write this formally. So, let w be written as w' followed by this; let w' is free of any $a_{j's}$. It only consists of the original alphabet which was used to build these dominos. So, w' only has the original alphabet; none of these a_1, a_2 up to a_k. And followed by this string is a_{j_m} , $a_{j_{m-1}}$ up to a_{j_1} .

Now, if any derivation ends with this; so, now, w is derived ambiguously means the ending set of derivations. The only way that we could produce this string w, because again, because of the structure of the grammar is the following.

(Refer Slide Time: 11:55)



$S \to B \to b_{i_1} B a_{i_1} \to b_{i_1}(b_{i_2} B a_{i_2}) a_{i_1}$

$\to b_{i_1} \cdots b_{i_m} a_{i_m} \cdots a_{i_1}$

This implies that $t_{i_1} \cdots t_{i_m} = b_{i_1} \cdots b_{i_m}$.

This is a match for P.

Thus the reduction is complete.

$P \in PCP \iff G \in AMBIG_{CFG}$.

Hence $PCP \leq_m AMBIG_{CFG}$.

Thus $AMBIG_{CFG}$ is undecidable.

This also marks the end of computability theory.

Proof: $PCP \leq_m AMBIG_{CFG}$.

Given PCP instance, we will construct a CFG which is ambiguous if and only if the PCP has a match.

Let $P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots \left[\frac{t_k}{b_k}\right] \right\}$

We construct the CFG G as follows:

$G: S \to T \mid B$

$T \to t_1 T a_1 \mid \ldots \mid t_k T a_k \mid t_1 a_1 \mid \ldots \mid t_k a_k$

$B \to b_1 B a_1 \mid \ldots \mid b_k B a_k \mid b_1 a_1 \mid \ldots \mid b_k a_k$

Here t_i, b_i are the strings in the dominoes in P.
a_i are new distinct symbols (not in t_i, b_i)

($\Rightarrow$) Suppose P has a match. Let the match be

Theorem: $AMBIG_{CFG}$ is undecidable.

Proof: $PCP \leq_m AMBIG_{CFG}$.

Given PCP instance, we will construct a CFG which is ambiguous if and only if the PCP has a match.

Let $P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots \left[\frac{t_k}{b_k}\right] \right\}$?

We construct the CFG G as follows:

$G : S \to T \mid B$

$T \to t_1 T a_1 \mid \ldots \mid t_k T a_k \mid t_1 a_1 \mid \ldots \mid t_k a_k$

$B \to b_1 B a_1 \mid \ldots \mid b_k B a_k \mid b_1 a_1 \mid \cdots \mid b_k a_k$

Here t_i, b_i are the strings in the dominoes in P.
a_i are new distinct symbols (not in t_i, b_i)

Here t_i, b_i are the strings in the dominoes in P.
a_i are new distinct symbols (not in t_i, b_i)

($\Rightarrow$) Suppose P has a match. let the match be

$$t_{i_1} t_{i_2} \cdots t_{i_m} = b_{i_1} b_{i_2} \cdots b_{i_m}$$

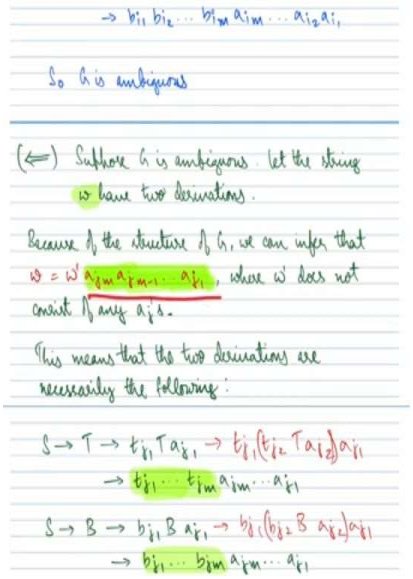
We have two derivations of the same string.

$S \to T \to t_{i_1} T a_{i_1} \to t_{i_1}(t_{i_2} T a_{i_2}) a_{i_1} \to$
$\to t_{i_1} t_{i_2} \cdots t_{i_m} a_{i_m} \cdots a_{i_2} a_{i_1}$

$S \to B \to b_{i_1} B a_{i_1} \to b_{i_1}(b_{i_2} B a_{i_2}) a_{i_1} \to$
$\to b_{i_1} b_{i_2} \cdots b_{i_m} a_{i_m} \cdots a_{i_2} a_{i_1}$

So G is ambiguous

($\Leftarrow$) Suppose G is ambiguous. let the string



So, there are two possibilities; one is that S gives T. And T gives $t_{j1}\, T_{a_{j1}}$, because a_{j1} has to be at the end; so this first division has to be this. And then it has to be $t_{j1}\,(t_{j2}\, T_{a_{j2}})\, a_{j1}$, where this has to be the next derivation and so on. Finally, we get this string t_{j1} up to t_{jm} , followed by a_{jm} up to a_{j1} . Similarly, the other derivation possible, because of again because of the structure of the grammar is that the ending part is a_{j1} ; the ending symbol is a_{j1} .

So the first, the other possibilities S gives B and B gives b_{j1} B_{aj1}; and again b_{j1}, b_{j2}, B_{aj2}, a_{j1}. And finally we get this string b_{j1} up to b_{jm}, a_{jm} up to a_{j1}. So, these are the two strings. So, and by assumption, this is the same stream derived in two different ways. And we assume that a_{jm} up to a_{j1} is a common suffix; and that is why we got these two derivations.

And this necessarily means that, so since a_{jm} up to a_{j1} is common; the w' part has to be this part, the first part t_{j1} up to t_{jm} and b_{j1} up to b_{jm}. This means that this part is the same. The highlighted parts has to be the same, which is what I have written here, t_{j1} up to t_{jm} is b_{j1} up to b_{jm} . Now, once we have that, that is a match for the PCP instance.

So, we first have the j_1^{th} domino, then j_2^{th} domino up to j_m^{th} domino. So that is it, because the grammar was ambiguous we; or from the ambiguity of the grammar, we got a match for the PCP instance. So, that completes the second direction of the reduction or the correctness. So, this implies that the original PCP instance P is a yes instance of PCP, if and only if the grammar is

ambiguous. And that gives us the reduction from the PCP to the ambiguous CFG, which means that ambiguous CFG is undecidable.

In other words, given a grammar, given a context-free grammar, it is undecidable to determine whether it is ambiguous. So that is what I wanted to share in lecture 44. Given a context-free grammar, it is undecidable to determine whether it is ambiguous; and it is by a straightforward reduction from the from PCP.

So, given a PCP instance, we create a grammar; the grammar is very easy to construct. So, we have new symbols involved; but apart from that everything is very straightforward. S gives T | B; so three variables alone, starting variable S and T and B; and then we make these rules. And where t_1 up to t_k, b_1 up to b_k are as in the PCP instance; and a_1 up to a_k are new symbols. They are all distinct symbols. And we then showed that if the PCP instance has a match which say this, then we demonstrated two derivations of the same string, showing ambiguity of G.

Now, if G was ambiguous for the other direction, we may assume by the structure of the grammar, that the suffix of the string so derived could be something like a_{j_m}, $a_{j_{m-1}}$ up to a_{j_1}. Once you assume that this is the suffix, the rest of the derivation is fairly is kind of constrained by the structure of the grammar. So, the only two derivations possible that gives us this suffix is this as well as this.

And that gives us the t_{j1} up to t_{jm} is equal to b_{j1} up to b_{jm}, which gives us a match for the PCP instance. So, ambiguity implies a match and match implies an ambiguity; hence, the reduction is proper. And that gives us that that gives us the reduction from PCP to ambiguous CFG; so ambiguous CFG is undecidable. And that completes lecture 44. This also completes the part on computability theory. This also completes, this also marks the end of computability theory which is chapter 5, so we started.

Computability theory was spread over three chapters 3, 4 and 5. We started with defining Turing machines, types of Turing machines, Church-Turing thesis; then we saw decidable languages, then we saw undecidable languages. Then, we saw the HALTING problem is undecidable; then we saw reductions, which was a new way to show decidability and undecidability.

Then, we saw many more languages that were undecidable. Then, we saw Rice's theorem, which was a way to show many other languages are undecidable. And then we saw PCP which is a simple problem, the statement of the problem does not involve any Turing machines or DFAs, or any such technical things. And but then surprisingly, that simply stated problem was undecidable.

There is no algorithmic approach to determine whether there is a match. And using the fact that PCP is undecidable, we now saw that the undecidability of checking whether a given context-free grammar is ambiguous. So, these other things that we saw in computability; we saw what is decidability, what is computability. So, computability means what is computable. So, and we saw that there are things that are not computable.

We saw that there are languages that are not even Turing recognizable; and we saw many decidable as well as undecidable languages. So, that completes computability theory; but however, our topics will continue. So in the next lecture, we will start complexity theory. So, now we will change our change gears a bit; meaning we will no longer be interested in the nitty-gritty, or we will no longer be interested in whether it is computable et-cetera.

Now, from now on, we will only deal with computable things. Or only deal with decidable things; but we will be more careful about how much resources it takes. So, complexity theory is more about how much resources, how efficiently can we some compute something? How much time does something take? How much space does something take? Basically, it is going to classify problems based on resources that we need.

So, for all that and more, you can check out the next lecture, lecture number 45. And as far as lecture number 44 is concerned, that is all that I have. Thank you.