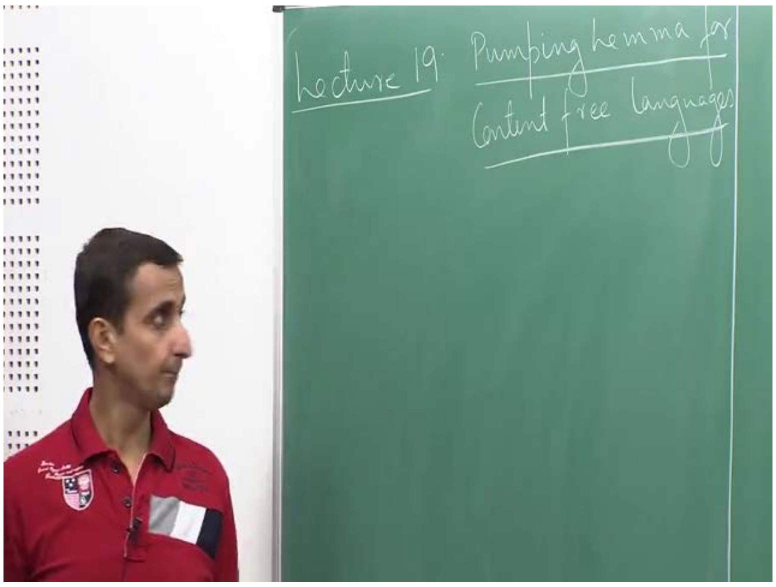
Theory of Computation

Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 19
Non-CFLs, Pumping lemma

Welcome to the 19th lecture. Similar to regular languages, there are limitations to the power of context free grammars in the sense that there are languages that are not context free languages.

(Refer Slide Time: 00:15)



And there is a pumping lemma; there is a version of the pumping lemma for context free languages that one can used to show that a language is not a context free language. So, today we are going to discuss this pumping lemma for context free languages. So, the agenda for today is we will try to motivate the statement of the pumping lemma. So, before I describe the statement of the pumping lemma for context free languages, let us try to see how the proof goes I mean how the statement actually comes through.

(Refer Slide Time: 01:15)

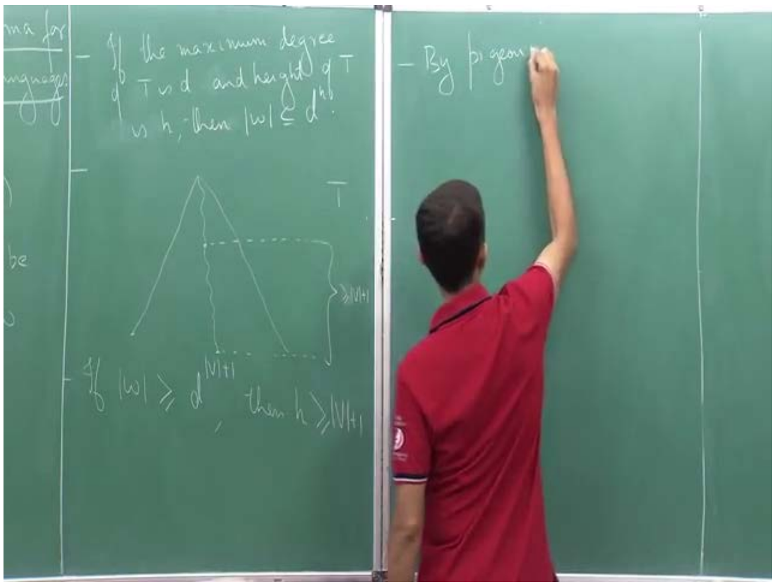
Lecture 19 Pumping Lemma for Context free languages

Let L be a CFL and G be a CFG s.t. $L = L(G)$ let $w \in L(G)$ and let T be the smallest parse tree of w w.r.t. G

So, let L be a context free language, and G is a grammar for L. So, what can we say about a string w or if I what can we say about the parse tree of a string. Let w be a string in L of G, and let T be the smallest parse tree of w with respect to G. So, when I say smallest parse tree, it is the parse tree, which has the least number of nodes; and if you have a string for which there are multiple parse trees which have the least number of nodes, you can pick any one it does not matter.

So, take a grammar, take a string in the language of that grammar and let T be the smallest parse tree of that string with respect to the grammar. So, what can we say about T? So firstly, so there are few things that we saw last time. So, suppose if we look at any path from the root to the leaf a of a parse tree, it consists of variables see if the root has let say k vertices then there are k minus 1 variables and the last node is either a terminal or epsilon, so that is true for any path. So, if you have a path which has k vertices then this is the property.

(Refer Slide Time: 03:22)


- If the maximum degree of T is d and height of T is h, then $|w| \leq d^h$!
T
$\geq |V|+1$
- If $|w| \geq d^{|V|+1}$, then $h \geq |V|+1$
- By pigeon

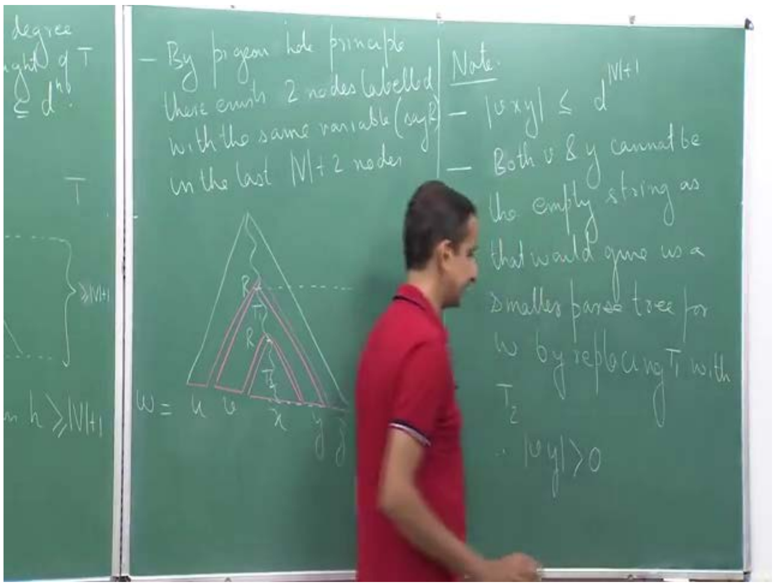
And if the maximum degree of T is d and the height of T is h. So, then actually what you can argue is that the number of leaf nodes is at most d to the power h, because from the root, I have d mini children at most d mini children and for each of those d mini children I can have another d children. So, it is d square. Finally, when I reach the leaf, it is at most d to the power h. If the maximum degree of T is d, and height of T is h then the cardinality of w is at most d to the power h, because w is nothing but the concatenation of all the leaves from left to right. Once again, we had seen this earlier.

Now consider the following. Let us look at this tree. Let us draw a picture. Let say that this is the tree that I have; this is my tree T. And let us look at the path in this tree from the root to a leaf node, which has a length at least h. Let us look at the longest path in this tree. We know that the longest path. So, what can we say about the longest path. So, if we take w as a string, which has length greater than d to the power V plus 1, so where V is the set of variables. What do we have, so we already have the w is less than or equal to d to the power a. In addition, if we assume the w is w is greater than or equal to this then what this implies is that the height of the tree is at least V plus 1. So, this just follows simple equality.

Let us take a path in this tree of height at least V plus 1. And let us look at the last V plus

1 vertex. So, may be this is the vertex, and I have this vertex and if I look at this length, so let say this length is greater than or equal to V plus 1. So, if I look at the last V plus edges, well, I do not want vertices exactly; I want edges. If I look at the last V plus 1 edge, this path will have at least V plus 2 vertices. And the last node of this tree can be a variable, it can be a terminal or it can be epsilon, but we know that all the other nodes of this path must be a variable. So, this path has at least V plus 1 variable.

(Refer Slide Time: 07:40)



– By pigeon hole principle there exists 2 nodes labelled with the same variable (say R) in the last $|V|+2$ nodes

Note:

– $|vxy| \leq d^{|V|+1}$

– Both v & y cannot be the empty string as that would give us a smaller parse tree for w by replacing T_1 with T_2

$\therefore |vy| > 0$

Therefore, by pigeon hole principle, there exists a node, there exists 2 nodes labeled with the same variable say R. In the last, V plus 2 nodes, so let me draw this picture once again. So, I have my tree. I have this path such that the entire path has length more than V plus 1. There must be a node over here which is labeled by a vertex R, and there is another node over here which is again labeled via vertex R such that this guy to this guy. So, this entire path has length at most V plus 1 edge. So, if I look at the last there will be something like this.

Now, let us considered the following sub trees, so I divide my tree into the following structure now, so I will look at the sub tree that is rooted at this R; I will look at the sub tree that is rooted at this R. So, may be this is some part over here. And I will look at the remainder of the tree. So, the remainder of the tree is now let us give some names to these things. So, whatever so the concatenation of the strings on this left most piece, I will call this x; concatenation of the strings over here, I will call this y, change this a little bit.

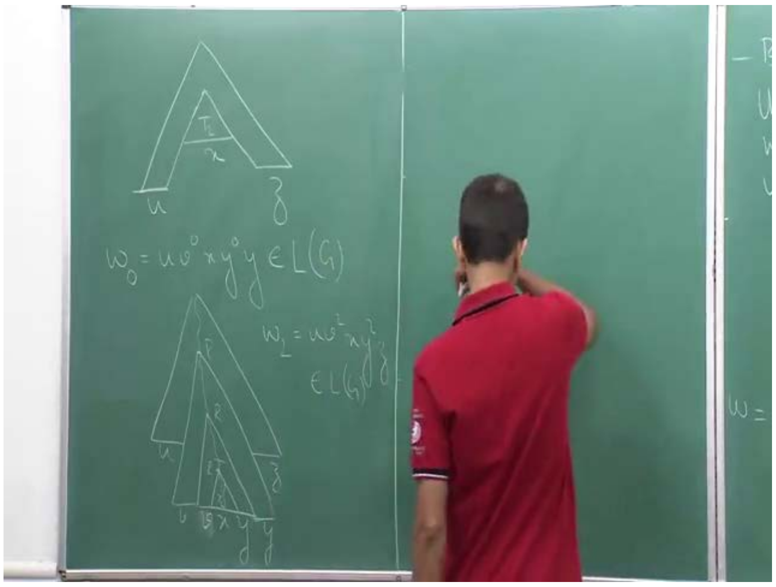
So, here I will call this as u; I will call this as v; the concatenation of these strings, I will call this as x, y and z. So, my entire string was w. So, this was a parse tree for w. So, I can write w as a concatenation of five strings u, v, x, y, z, where x is the string that is being generated by the sub parse tree that is rooted at this bottom R; v, x, y is the string that is being generated by the parse tree that is rooted at this top part; and u, v, x, y, z is the string w which is being generated by the root node.

So, what can we say about these pieces. So, the first point is since I had taken this R to be a, so of course, when I look at the last V plus 1 edges where I have V plus 2 vertices, I can have many such a pairs which get repeated, let say I will I will look at the last such pair. So, because I look at the last such pair, what I can say is that the height from this node to the leaf is at most V plus 1. So, because the height is at most V plus 1, what I have is that the length of v, x, y whatever is been generated by this parse tree that is rooted at the top part is at most the degree raise to the power V plus 1.

What can we say about v and y? So, obverse that both V and y together they cannot be empty, because if both v and y are the empty strings. What I could have done is that a I instead of keeping this entire parse tree, I could have replaced this parse tree rooted at the top R with the parse tree rooted at the bottom R. So, if I give them some names, so let say that this top parse tree is T 1 and the bottom parse tree is T 2, I could have replaced T 1 with T 2.

And I would have got a smaller parse tree, because this length is the parse tree T 2 is smaller than the parse tree T 1. So, I would have got a smaller parse tree which would have generated the same string w because v and y are empty. So, therefore, both v and y cannot be the empty string as that would give us a smaller parse tree for w by replacing T 1 with T 2. Therefore, what I can say is that the length of v, y together is strictly greater than 0.

(Refer Slide Time: 15:22)



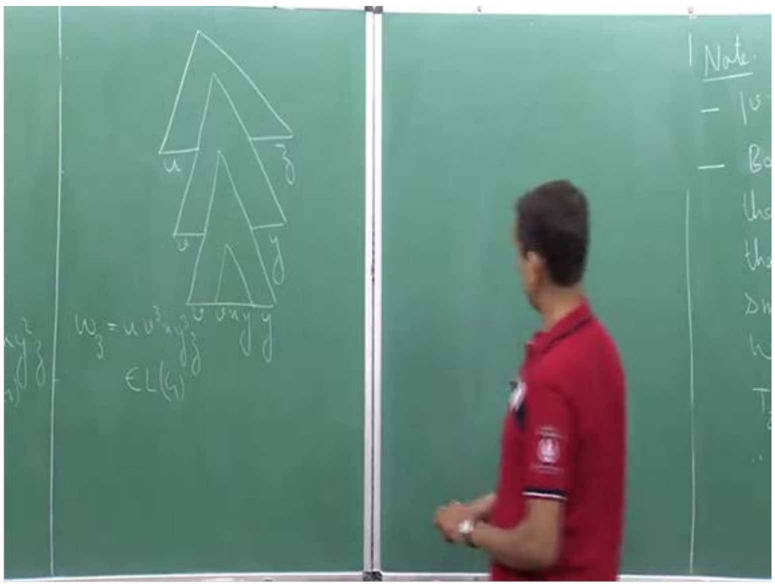
Now, let see what other strings are there in this language, what other strings I can generate using this parse tree. See if I look at this parse tree suppose, if I replace T 1 with T 2, suppose if I do replace it, suppose if I look at this parse tree, and now I replace T 1 with T 2. So, I have T 2 over here. So, this part of my string was u, this part is x, and this part is z. So, I get a string w, let us call it w 0, which is equal to u v to the power zero, v is absent, x y to the power 0, y is also absent z, which is also in the language of the grammar, because I have a parse tree with respect to the same grammar.

If I look at another parse tree, where I let say replace T 2 with T 1 suppose if I replace T 2 with T 1. So, this was my T 2 earlier. Now, I have u, and this part is my T 1 once again. So, I have v; and then once again I have v, x, y; and then once again I have y and then I have z. What is the string that I get? So, I get a string let us call it w 2, which is nothing but u this is v square x y square z which again belongs to L of G. So, this was my vertex or again this is also a vertex R which is what I am doing.

Now, I very conveniently decided to do this replacement. So, here I am doing this replacement; here also I am doing this replacement, but why is it that I am able to do this replacement. The reason why I am able to do this replacement is because both these nodes, so both if I go back to this parse tree, both these parse trees T 1 and T 2, they have the same root.

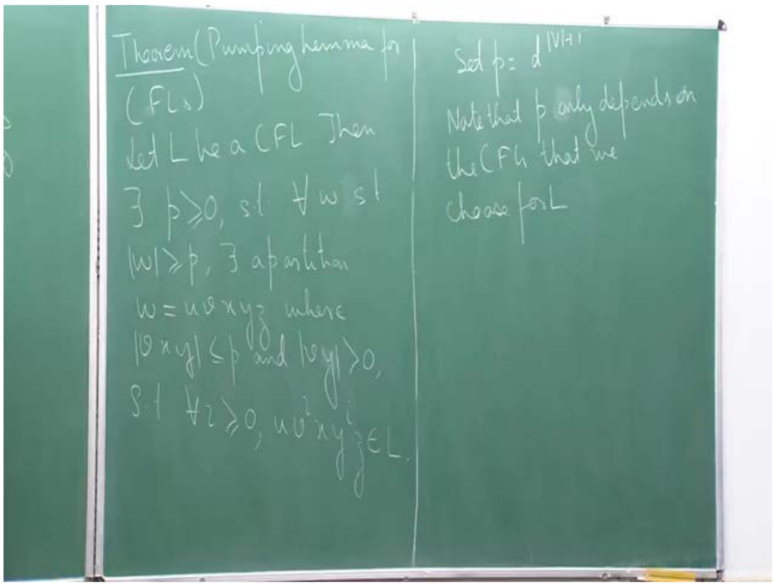
So, they have the same root R, because they have the same a root R or this the same variable as their root nodes, therefore, I could replace one tree with the other tree and get another parse tree, which generates some other string in the same language, so that is a key points. So, it is we are using the fact that both these parse tree have the same variable as their root. Now, I get string w to which is in L of G. Once more so this is my w 2, so inside this tree also there is a portion for T 2. So, I have this path; there is this vertex for T 2. And once again I can replace this T 2 with T 1.

(Refer Slide Time: 19:31)



So, what happens if replace this T 2 with T 1. So, what I get is, so I have this and I have this, and then I have a T 1 over here with T 2. So, I have u, so do I have. So, I have u v. So, u, v, again v, and then again I have v, x, y, y, y and then z. So, I get another string let us call it w 3, which is u, v cube, x, y, cube z, once again which is in the language of the grammar.

(Refer Slide Time: 20:51)



Theorem (Pumping lemma for CFLs)

Let L be a CFL. Then $\exists\, p \geq 0$, s.t. $\forall w$ s.t. $|w| \geq p$, $\exists$ a partition $w = uvxyz$ where $|vxy| \leq p$ and $|vy| > 0$, s.t. $\forall i \geq 0$, $uv^ixy^iz \in L$.

Set $p = d^{|V|+1}$

Note that p only depends on the CFG that we choose for L

Now, we are in a position to state the pumping lemma. Let L be a context free language then there exist p greater than or equal to 0, such that for all w such that length of w is greater than p, there exist a partition of w into five parts u v x y z. Where v, x, y is at most p, and v y is greater than 0, such that for all I greater than or equal to 0 u v to the power i x y to the power i z belongs to L. So, the only thing that we have to fix is what is the p that we get. So, the p that we choose given a language L give a context free language L in this case is so we set p equal to d to the power V plus 1.

So, if I have context free language then there is a grammar for that context free language. So, given that grammar I know what the number of variables in that grammar is and I also know what the degree of any parse tree will be the maximum degree. So, the maximum degree of any parse tree will be the maximum number of symbols that I have on the right hand side of any rule, so that will decide what d is I fix p accordingly.

Now if I take any string which has length more than d to the power V plus 1 then we have just shown that if I look at the parse tree of that string, if I look at the smallest parse tree of that string, I can partition it into u v x y z such that v x y is at most p. So v is whatever is generated by the sub tree T 1, and v y is greater than zero such that no matter what i greater than 0, I take, I can always construct a parse tree for that string which implies that u v to the power i x y to the power y z is in the language.

Let me just make a note. So, note that p only depends on the context free grammar that we choose for L. So since L is a context free language, there is some context free grammar for the language. So, what we will show next time is we will look at the contra positive form of this pumping lemma, so which will essentially allow us to prove that a language is not context free. We look at the contra positive form.

In the contra positive form, if I take any language L, then for all p there will be a w with length more than p. Such that if I take any partition with this property there exist i, such that this string will not be in the language and that would proof that the language is not context free. So, we will look at this next time, and also we look at some examples of non context free languages. So, I will stop here today.

Thank you.