

Theory of Computation
Professor. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Pumping lemma for Context Free Languages

(Refer Slide Time: 00: 16)

Pumping lemma for CFL's

- Similar to pumping lemma for regular languages.
- Necessary condition for a language to be context-free.
- We can use pumping lemma to show that languages are **not context-free**.
- By Bar-Hillel, Perles and Shamir (1961)

Pumping lemma (Theorem 2.34): If A is a CFL, then there is a number p (pumping length) where if s is any string in A , $|s| \geq p$, then there exists a partition $s = uvxyz$, satisfying

- We can use pumping lemma to show that languages are **not context-free**.
- By Bar-Hillel, Perles and Shamir (1961)

Pumping lemma (Theorem 2.34): If A is a CFL, then there is a number p (pumping length) where if s is any string in A , $|s| \geq p$, then there exists a partition $s = uvxyz$, satisfying

<ol style="list-style-type: none"> 1. For each $i \geq 0$, $uv^i xy^i z \in A$. 2. $v > 0$ (either $v \neq \epsilon$ or $y \neq \epsilon$) 3. $xy \leq p$. 	$s = xyz$ 1. $xy^i z \in A$ 2. $ y > 0$ ($y \neq \epsilon$) 3. $ xy \leq p$
--	---

Compare with pumping lemma for regular languages.



Hello, and welcome to Lecture 25 of the course Theory of Computation. In this second chapter so far, we have seen context-free languages. And we have seen two ways to realize them, context-free grammars and push down automata. And we saw that they are equivalent and we

saw examples for grammars as well as push down automata. In this section, which is the last section of this chapter, we are going to see pumping lemma for context-free languages.

So, there are lots of similarities with the pumping lemma for regular languages that we had seen at the end of Chapter 1. So, what is pumping lemma? It is a necessary condition for a language to be context-free. So, just like the pumping lemma for regular languages was a necessary condition for a language to be regular, this is a necessary condition for a language to be context-free.

And, so which means if a language is context-free, the conditions of the pumping lemma has to be satisfied. And if we are able to show that the conditions of the pumping lemma are not satisfied then we can infer that the language is not context-free. So, we can use pumping lemma to show that the languages are not context-free. So, you can use it to show languages are not context-free. We cannot use it to show that languages are context-free. To show languages are context-free, we need to construct a PDA or CFG or something, but we can use pumping lemma to show that it is not context-free.

And once again this was discovered by Bar-Hillel, Perles and Shamir in '61. In fact, the pumping lemma for regular languages also was a special case on this. So, this is by the same people. So, the structure or the statement of the pumping lemma and the way it is applied, all of these are very similar. So, there are only some minor differences in the way the pumping lemma for context-free languages differs from pumping lemma for regular languages. Many things are very similar. So, we will highlight the differences, even in the statement of the pumping lemma itself and in the way we apply it.

So, pumping lemma says that if a language A is a context-free language, then there is a pumping length p , such that if you take any string s in A whose length is at least p , s is of length at least p , then we can partition s into five parts $uvxyz$ such that the following three conditions are satisfied. So, if A is a context-free language there is a pumping length, and if you take any string in the language which is of length at least the pumping length, then you can partition that string into five parts $uvxyz$ meaning as a concatenation of these five strings such that the following conditions are met.

Condition 1 is that uv^ixy^iz is in A , for all i . Condition 2 is that the length of vy is strictly greater than zero. This means that both of them cannot have zero length. Or in other words, both

of them cannot be empty strings. Either v has to be a non empty string or y has to be not an non empty string. And finally, we have the length of vxy which is the middle part is at most p . So, these are the three conditions. So, let us just contrast them with the conditions of pumping lemma for regular languages.

In the case of regular languages, we had to partition s as uvw , into three parts, not five parts. And Condition 1 was that $wv^i w$ is in A , for all i . Condition 2 was that length of v was strictly greater than zero, meaning v was not an empty string. I do not recall if I used xyz or uvw . I think I used xyz , so maybe I will just replace it with xyz , s is equal to xyz , Condition 1 was that $xy^i z$ is in A , 2 was that this length of y is strictly greater than zero, meaning y is not empty, and 3 was that the length of xy is at most p , the length of it xy is less than or equal to p . These were the three conditions in the pumping lemma for regular languages.

So instead of five pieces, we split into three pieces. First part was that $xy^i z$ is in A for all i . So, here it is five parts and we have, just to contrast, we have, instead of $xy^i z$, we have $wv^i xy^i z$. So, the second and the fourth parts are going to be repeated, how many ever times, as per the count of i . And here, in regular languages we had the length of y is strictly greater than zero.

Here, we are saying that vy is strictly greater than zero, meaning when we had y should be non empty in regular languages here, we are saying either y or v should be not empty. And in regular languages we have length of xy at most p , here we have vxy was at most p . So, there it was xy which was the initial two parts, here which is vxy which is the middle three parts. So, there are some similarity. This is, the structure is certainly similar. But in the actual statement itself there are some small differences

(Refer Slide Time: 07:07)

Proof idea

- In regular languages, we used finiteness of DFA.
- In CFL's, we use finiteness of grammar.

If we choose a long string s , then the derivation will be long (must have many steps). In such a long derivation, some variable must repeat.

Suppose T is the starting variable and $T \Rightarrow^* s$.
By repeat of variable, we mean the following.

$$T \Rightarrow u R z$$
$$(R \Rightarrow v R y) \Rightarrow u (v R y) z$$

By repeat of variable, we mean the following.

$$T \Rightarrow u R z$$
$$(R \Rightarrow v R y) \Rightarrow u (v R y) z$$
$$\Rightarrow u v x y z$$

R
 \downarrow
 $v R y$
 \downarrow
 x

In the above derivation, we have $R \Rightarrow^* v R y$.
We can derive $v R y$ from R again as per the rules of the CFG.

That is we could have

$$T \Rightarrow u R z \Rightarrow u (v R y) z \Rightarrow u v (v R y) y z$$



In the above derivation, we have $R \Rightarrow vRy$.
 We can derive vRy from R again as per the rules of the CFG.

$\rightarrow uxz, uvvxzyz, uvvxzyyz, \dots$

all these strings may be derived from the CFG.
 That is we could have

$T \Rightarrow uRz \Rightarrow u(vRy)z \Rightarrow uv(vRy)yz$
 $\Rightarrow uvv(vRy)yyz$
 $\Rightarrow uvvxzyyz$

$T \Rightarrow uRz \xRightarrow{*} u(vRy)z \Rightarrow uv(vRy)yz$
 $\xRightarrow{*} uvvxzyyz$

$T \Rightarrow uRz \xRightarrow{*} uxz$



So, let us see why this is true. So, in pumping lemma for regular languages, we use the finiteness of the DFA. So, we said that A is regular. So, there is a finite automata DFA for A . And then we said that if the length of the string is greater than the number of states then it has to revisit some state. And this revisiting some state was turned out to be the middle part of the string. So, now when you revisit a state, you can revisit it again and again and again. You can go through the same loop. This was the high level idea of the proof of pumping lemma for regular languages.

Basically, if the length of the string is long enough, some states have to be revisited because you have only a finite number of states and by using pigeon hole principle. In case of context-free languages, we use the finiteness of the context-free grammar. Not the DFA, not the PDA but the finiteness of the grammar. Basically, the idea is this, if a string is long then we may have to apply the rules many times because every rule may have a limited length of string appearing in the right hand side.

So, if the string is long, we may have to apply the rules many times. So, the derivation may be long. And in a long derivation, because you have only fixed number of variables, some variable must come again. So, some variable must repeat. And this is what we exploit to get the pumping lemma. So, if the string is long, the derivation has to be long, when I say derivation has to be long, I mean many steps. And if there are many steps, we cannot have all distinct variables because the number of variables is bounded. So, some variable must come again.

And this we will exploit to show that these conditions should be met. So, in other words, suppose T is the starting variable and T derives s . And what we mean is this. So, at some stage, let us say, you got some string like, this uRz where R is some variable and u and z are some strings which consists of both variables and terminals or let us say u and z are entirely terminals, and R is a variable.

Then let us say, because the string is long, this R yields vRy again. So, which means that the R yields something that has an R again. So, if the string is long, the derivation has to be long, and some variable R must derive itself at some point. So, that is what I have written here. R derives vRy . And then the middle R derives x . So, that is what is happening, and this is what we will exploit for the pumping lemma.

The key point is this, the fact that R derives itself. So, this is what we will use. So, now one may ask why not derive R . So, we got vRy from R but we could use the same thing again. So, for instance we can do this, we said T gives uRz and R gives vRy . So, the middle R gives vRy . Now, what if this R again gave vRy .

And we may do this again. So, you have uvv at the beginning and yyz at the end, and this middle R , is going to give vRy . So, we may use that rule. So, if R is able to derive vRy as per the rules of the context-free grammar, we can derive, we can apply it again and again. So, this means I can get something, some string like $uvvxyyyz$ or I may be able to get something like this let us say T gives uRz , now instead of R , I can put vRy , and again instead of this new R , I get vRy again and now I am going to replace this findla R by x , we know it is allowed. above the R was replaced by x . So we get $uvvxyyz$. Or I could do something else, I could do something like T gives uRz . This is one possible derivation. And now directly, instead of deriving vRy , I can derive x from R . So, I could get uxz . So, notice that now by these three derivations, these three derivations, I got three strings. One is $uvvxyyz$, two is $uvvxyyyz$, another one is uxz .

So, what we are saying is that if some R derives itself, the part v and y , I could repeat it, in the original string v and y repeated or came up just once, but I could have it two times or three times or even zero times. So, the three strings that we got here are, uxz or $uvvxyyz$ or $uvvxyyyz$

and so on. So, all of these things can be derived from the starting variable as per the rules of the grammar. So, all these strings may be derived from the CFG.


So, just because we know that R derives $v R y$ and R derives x , we can combine them in different ways to get these things. And that is pretty much the high level idea of the proof of the pumping lemma. So, the only things that we need to formalize here is, I just said a hand wavy thing like if the string is long, the derivation is long and then some variable must be repeated.

So, once we kind of quantify these things, what do we mean by the string is long, and what do I mean by the derivation is long, and how can we, conclusively, say that some variable must derive itself. So, these things, once we kind of formalize, the proof is there. That is the proof.

(Refer Slide Time: 15:24)

Proof: let b be the maximum number of symbols on the RHS of a rule. If the height of the parse tree is $\leq h$, then the length of the derived string is $\leq b^h$.

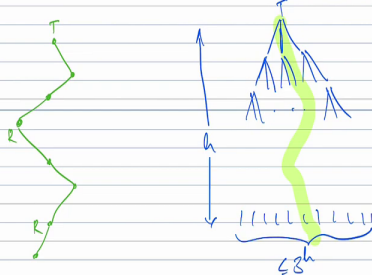
If $|V|$ is the number of variables in G , let $p = b^{|V|+1} \geq b^{|V|} + 1$. So any string of length $\geq p$ has parse tree height $\geq |V| + 1$.



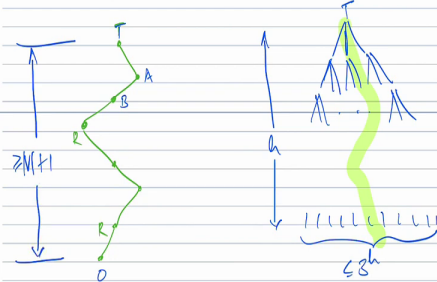
$$|s| \leq b^h$$

If $|V|$ is the number of variables in G , let

$b = b^{|V|+1} \geq b^{|V|} + 1$. So any string of length $\geq b$ has parse tree height $\geq |V| + 1$.



$\Rightarrow P$ has parse tree height $\geq |V| + 1$.



\rightarrow Choose s , $|s| \geq b$.

\rightarrow Choose a parse tree for s with the smallest number of nodes. This tree has height $\geq |V| + 1$. So the tree contains a root-leaf path of length $\geq |V| + 1$. This path has $\geq |V| + 2$ symbols,



→ Choose s , $|s| \geq b$.
 → Choose a parse tree for s with the smallest number of nodes. This tree has height $\geq |V|+1$. So the tree contains a root-leaf path of length $\geq |V|+1$. This path has $\geq |V|+2$ symbols, that is $\geq |V|+1$ variables and a terminal.
 → Since there are $\geq |V|+1$ variables, some variable must repeat in this path. Choose R to be a variable that repeats in the lowest $|V|+1$ variables of the path.



So let us go to the formal proof. Let b be the maximum number of symbols on the right hand side of any rule. If you look at all the rules of the grammar, A gives xyz or A gives 0 A 1 or something, so look at the right hand side and see how, what is the length of the right hand side, how many symbols does it contain? And I say symbols, I am talking about both variables and terminals. So, let us call it b .

For instance, if it is a Chomsky normal form, we know that any rule, the right hand side has at most two variables, or a single terminal. So, then we know that b is 2. In case of a general context-free grammar, I am just calling it b . So, it could be 3, it could be 10, it does not matter. We just want a finite number which we can define as per the grammar because a grammar has a fixed number of rules. You look, go through all the rules and see which rule has a maximum number of symbols on the right hand side and you call that b .

And now, let us note one thing that if the height of the parse tree is at most h , then the length of the derived string cannot be more than b^h . Let us quickly see why.

Suppose b was 3. So, in one step of the derivation, like from the starting variable, let us say T , I could get three different symbols and then let us say some of them are terminals, some of them are variables. So, the worst case is everyone is a variable. They can give rise to let us say three more variables. And this can keep going. And if the height is let us say h , at the end, we will generate a string of length 3 power h , because if you look at it after the first level, there are three symbols, if you look at a second level, there are nine symbols. So, every time it gets multiplied

by 3, but it need not get multiplied by 3 but it may get multiplied by something smaller, but then we want to give an upper bound. So we said that the maximum number of variables or terminals on the right hand side is 3. It could be 2, it could be 1, but this is an upper bound. So, the worst case is every rule gives three variables, or three terminals or whatever. So, now we get 3 power h.

So, if the height of the parse tree is h, then the length of the string is at most 3^h . So in this case, every rule gives rise to at most 3 symbols on the right hand side. If it was now, instead of 3, it was b, then we would get b^h . So, now, let us say the number of variables is given by the cardinality of V, here V(capital V) is the set of variables. So, $|V|$ denotes the number of variables.

Now, the pumping length that we get is just $b^{|V|+1}$. Clearly this is at least $b^{|V|} + 1$. This $b^{|V|+1}$ is pumping length (p).

So, for any string of length at least the pumping length, to derive that string, the height of the derivation tree must be at least $V+1$. So, any string of length at least p, the parse tree has to be of height at least $V+1$. Why is this? Because if the parse tree has height V or less then the length of the string will be $b^{|V|}$ or less. But by the choice of the string, the length of string is at least p, and we know that p is at least $b^{|V|+1}$. So, the height of the parse tree has to be at least $V+1$, if not more. So what do I mean by height of the tree? By height of the tree what I mean is, if you look at the tree, there is some path that is of length $V+1$ from the top of the tree to the leaf of the tree, from the root of the tree which is the top to the bottom of the tree which is a leaf.

So, there is a path of length of at least $V+1$. So all the root to leaf paths cannot be of length less than or equal to V, there has to be at least one path, root to leaf path, of length greater than or equal to $V+1$. So, it has length greater than or equal to $V+1$ means that all through the paths all the intermediate symbols are going to be variables because if you get a terminal, let us say small a, then the small a cannot derive anything more. So, it stops there. So, we know that all the, all the labels in the, in this path are going to be variables except for the last label which will be some terminal, let us say 'a' or '0' or something.

So, in the path of length $V+1$, we have all the labels as variables except one, which is at the bottom. So, let us say a path of length 3 has four vertices or four dots. So, it may have T,A,B and

some symbol 0. So, it has three variables and a terminal. So, a path of length $V+1$ has similarly, at least $V+1$ variable and one terminal.

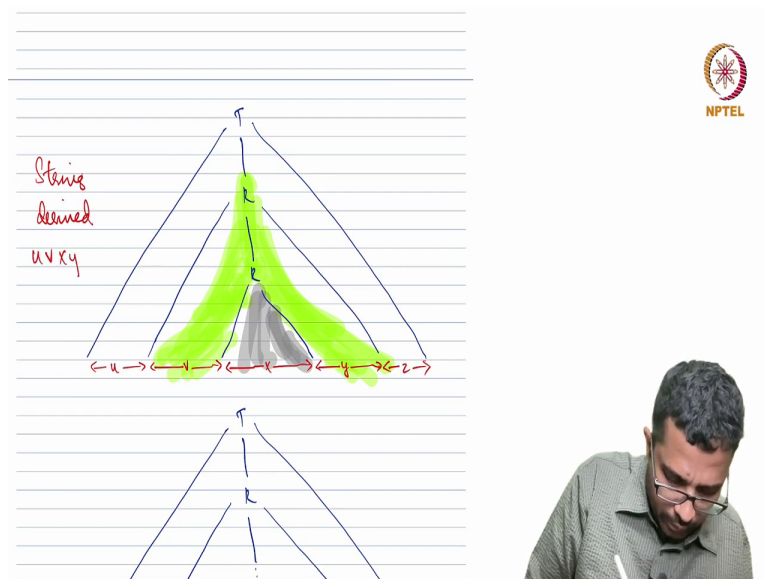
So, in this path of height $V+1$, so, there are at least $V+1$ variables and one terminal. But we know that the grammar has only V variables, size of V by definition is the number of variables.

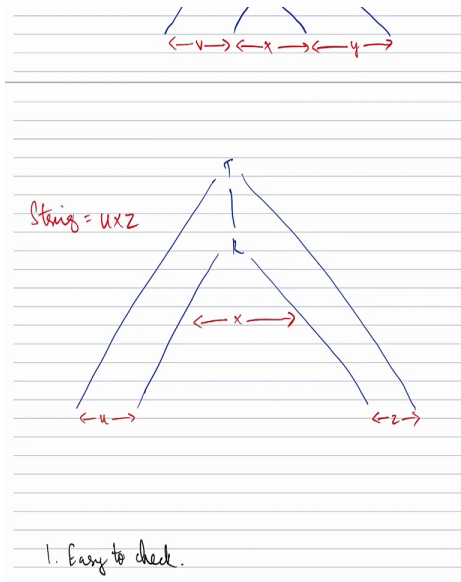
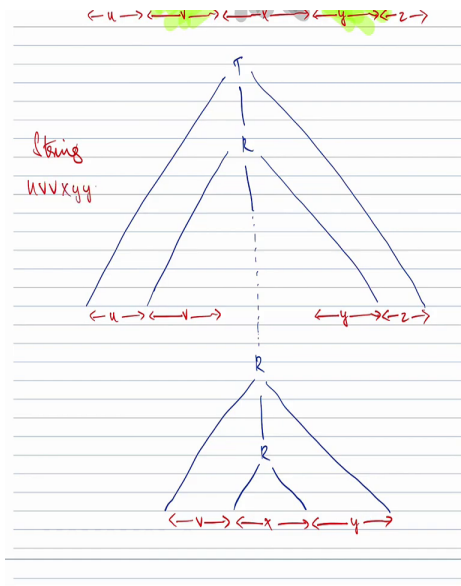
So, which means that some variable must appear, at least one variable should be there that appears more than once. And let R be that variable. Pick R to be such a variable that appears more than once. And if there are multiple such variables, you look at the bottom, you look at the bottom of the tree. From the bottom, you look at $V+1$ variables, not from the top, from the bottom, and from the bottommost $V+1$ variables, you choose a variable that repeats.

So, there could be multiple variables that repeat, but you pick a variable that repeats in the bottom $V+1$ variables of that path. So, this is the choice of the variable. This is how we choose R . So, this is the key thing. Because the string is long. So, again, let me just say it once again, we took a string of length at least p , where p is $b^{|V|+1}$. Because the string is long, the height of the parse tree is at least $V+1$, and because the height of the parse tree is at least $V+1$, some variable is there that repeats, and let that variable be R .

And if there are multiple such variables that repeat, you choose such a variable that repeats from the bottom bottom $V+1$ symbols. So, this choice will matter in some aspect of the proof.

(Refer Slide Time: 25:48)





So, the picture that we have, we have the starting variable T here. At some point, R is derived. So, when R is derived, the part to the left of R is u and the part to the right of R is z . So, if you look at the derivation till that part, T gives uRz . But we know R derives itself again, and let us say R derives, vRy . And the second R derives x . So, now the thing is that we could replace this gray triangle, with a copy of this, and, this bigger triangle because it is R that is deriving. R can derive vRy .

So, I could, instead of the gray triangle (refer above figures), I could replace the entire triangle (green + grey). So, now if you see the derivation, the string that is derived is $uvvxyyz$. I can

again do this. Again, this small triangle can be replaced by this entire triangle. So, I will get v repeating three times and y repeating three times, $u v v v x y y y z$.

Or I could even replace the bigger triangle with a smaller triangle. The bigger triangle where R gives $v R y$, I can replace with the gray triangle where R simply gives x . So, in this case the entire string is $u x z$.

So, now you can see for v and y , how many ever times I want, I can repeat them. So, the number of times v and y repeat is the number of times we use this replacement rule. We used R gives $v R y$ once and again R gives $v R y$. So, we used two times. So, you get $u v v x y y z$.

In next case we never use R gives $v R y$. So, you do not have any v or y . So, that is the proof of the first part of the pumping lemma, which says that for each i , $uv^i xy^i z$ is in the language, which is what we have shown. So, we can kind of divide the string in such a way and get v and y such that if you repeat v and y the same number of times, all those strings are in A .

So, we chose R to be the variable that repeats from the bottom. On the first derivation of R , what comes before R is u and what comes after R is z . And when R derives itself, what comes before R , that is v , and what comes out after R is y . And finally, in the second instance of R , what is derived by R is x . That is how we get $u v x y z$.

So, the first part of, the first statement that we can split s into $u v x y z$ such that for all i , $uv^i xy^i z$ is in A , is verified. And now we have to show that vy is non-empty and vxy is at most p . So, let us see why that is true.

(Refer Slide Time: 31:52)

1. Easy to check.

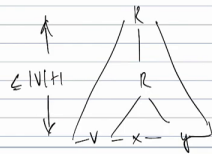
2. If $v=y=\epsilon$, then we can replace the parse tree with a smaller parse tree by pumping down. This contradicts the minimality of the tree.

3. By the choice of R , R is in the bottom $|V|+1$ nodes. So $R \Rightarrow vxy$ has height $\leq |V|+1$.

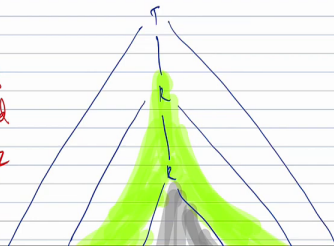
So $|vxy| \leq b^{|V|+1} = p$, the pumping length.

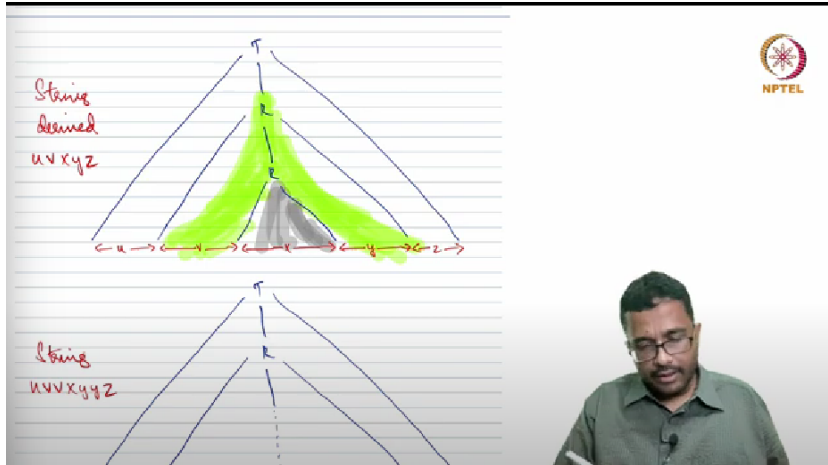


→ Since there are $\geq |V|+1$ variables, some variable must repeat in this path. Choose R to be a variable that repeats in the lowest $|V|+1$ variables of the path.

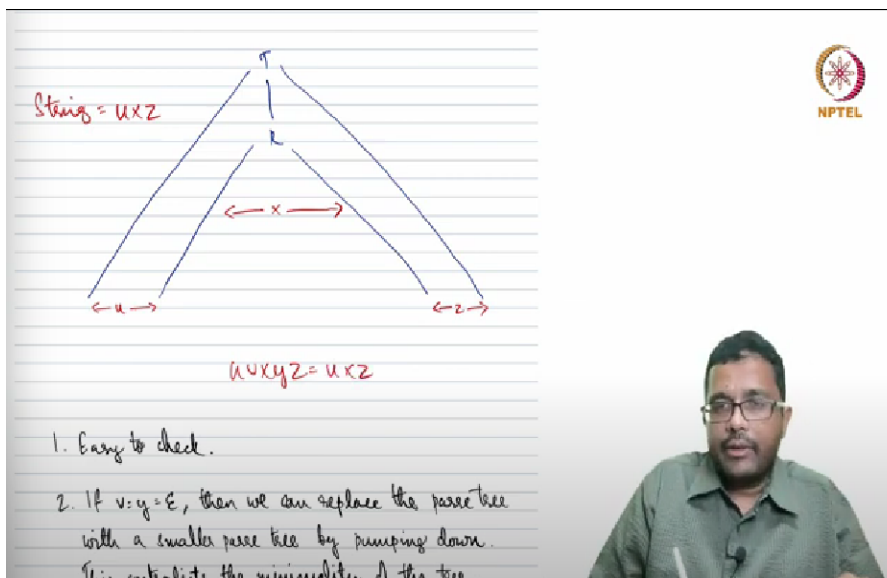


String derived
 uv^kxy^kz





Suppose v and y were both empty, then that means in the parse tree that we have (above), $uvxyz$ is equal to uxz . So, now which means this parse tree that I have drawn here (below),



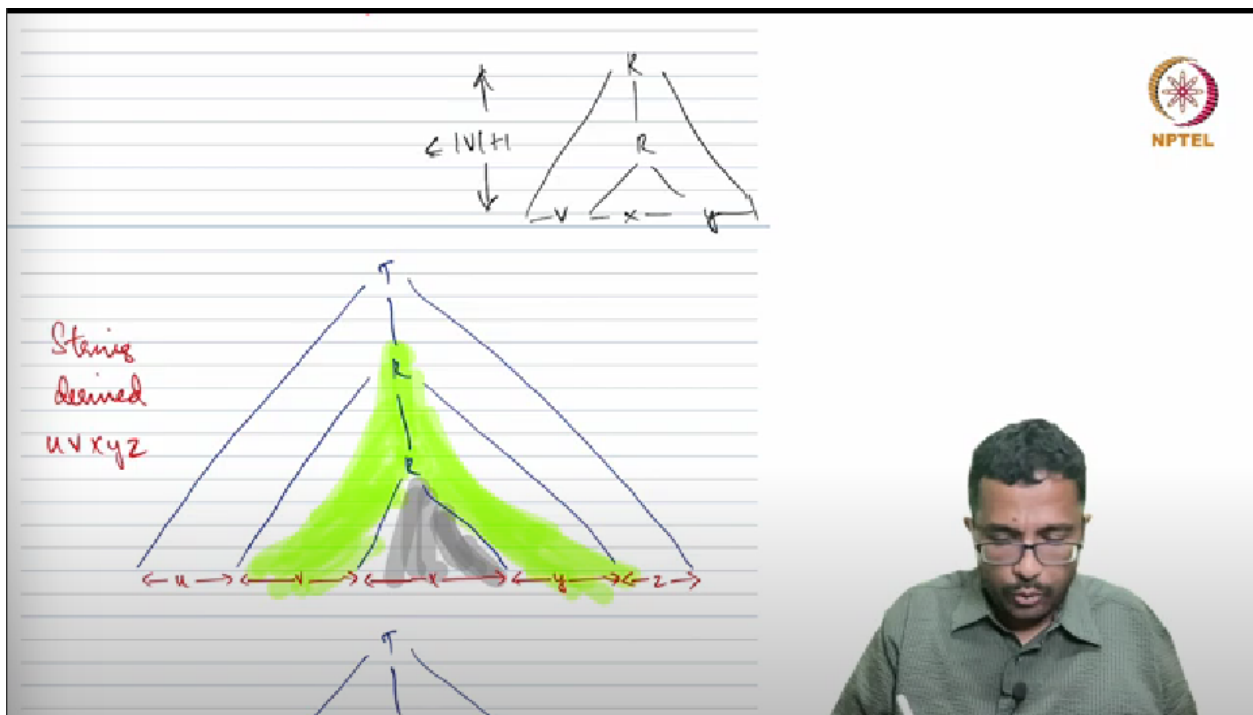
this parse tree is a simpler way to derive the same string. This is a bigger parse tree because it has to derive, it has to allow R to derive itself and so on.

So, which means this is a smaller parse tree. But by our assumption, the parse tree that we chose was one that was the smallest. But the parse tree that we chose was one which had the smallest number of nodes. So, if v and y are both empty, that contradicts the choice of the parse tree. Hence, v and y cannot be both empty. And now we have to show that vxy is of length at most the pumping length. Let us see why, and we chose R to be something that repeats. So, this is by the choice of R . We chose R to be a variable that repeats in the lowest $V+1$ variables of the path. The

height of the entire subtree of R (green + grey) is no more than $V+1$. We know that in the, even in the $V+1$ variables, there is some repeats. This means that for any tree of height h , we already said this, for any tree of height h , the length of the string is at most b^h .

Which means that R has height at most $V+1$, so length of vxy is at most b^{V+1} . And what is b^{V+1} ? b^{V+1} is nothing but our pumping length. So, this is exactly what we want.

So, that is it, that completes the proof of the pumping lemma. So, just to give the high level picture once again, if a string is long, meaning longer than b^{V+1} , then the parse tree has to be of height greater than $V+1$, which means some, there has to be some path where some variable repeats. And now in that path, you choose the variable that repeats. So this picture(below) is the one that guides the choice of the strings $uvxyz$.



So, from the starting variable to the repeating variable, what is the string that comes before the repeating variable? That is u . What is the string that comes after the repeating variable? That is z . And now the repeating variable R derives itself. So R derives some string then next R derives some other string. So, string that comes before (second R) is v , and the string that comes after (second R) is y . So, R derives $v R y$. And finally, the second, in the second repeating of R , what

is the string that derives, that is derived by R , that is x . And this defines the five variables, five strings $uvxyz$, and the three conditions also, we verified.

And that is the proof of the pumping lemma. Again, once again, pumping lemma for context-free language is very similar to the pumping lemma for regular languages in the way it is stated and structured, and also in the way it is applied. Just that, the difference is in the conditions itself will be used to proof that a language is not context-free.

But the way we argue, the structure of the proof etc is going to be very, very similar. And I think that, we have run enough time, and that completes the proof of statement and proof of the pumping lemma for context-free languages. And we will see some examples of this in action in showing that some languages are not context-free in the next lecture, which is lecture number 26.

Thank you.