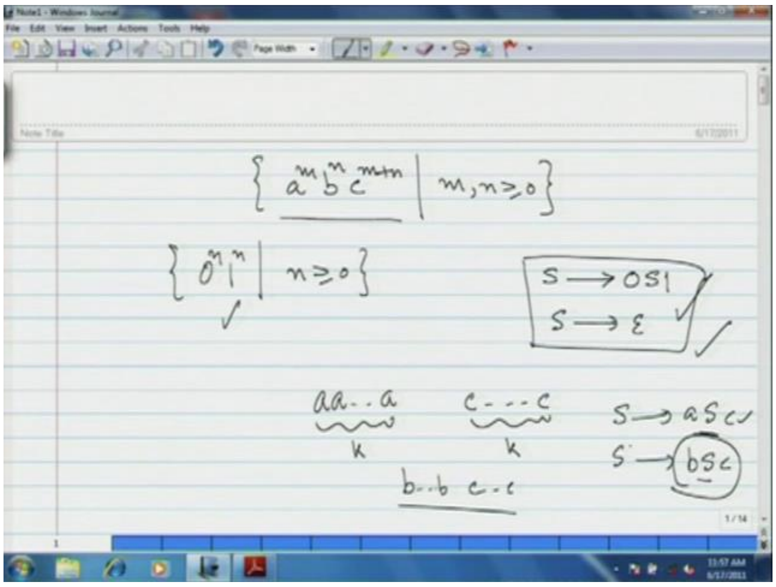
Formal Languages and Automata Theory
Prof. Dr. K. V. Krishna
Department of Mathematics
Indian Institute of Technology, Guwahati

Module - 2
Grammars
Lecture - 2
Derivation Trees

So far, we have discussed what is a contextual grammar? And the languages generated by contextual grammar, that is contextual languages, and certain examples to get the familiarity with contextual grammars.

(Refer Slide Time: 00:46)


$\{ \underline{a^m b^n c^{m+n}} \mid m, n \geq 0 \}$
$\{ 0^n 1^n \mid n \geq 0 \}$ ✓
$S \to 0S1$
$S \to \varepsilon$
$\underbrace{aa..a}_{k}$ $\underbrace{c...c}_{k}$
$\underline{b..b\ c..c}$
$S \to aSc$
$S \to bSc$

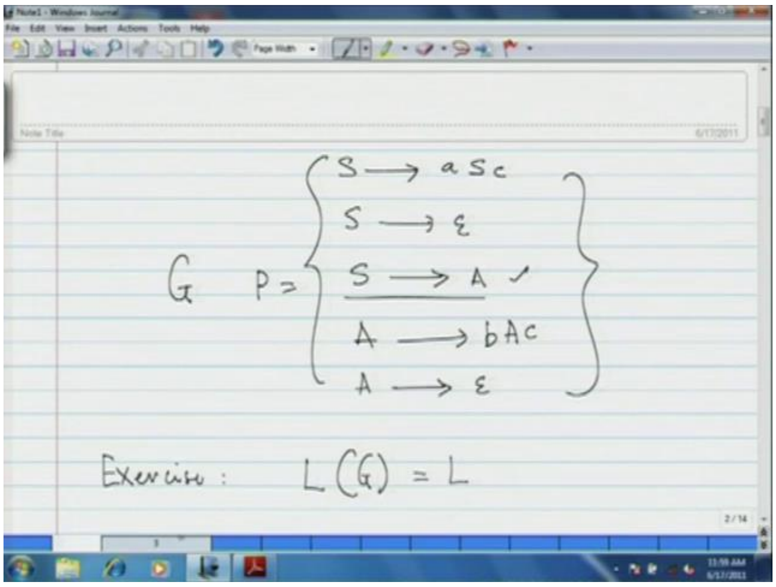
Let me give the one more example; we considered a set of strings of the form a power m and b power n and c power m plus n. Such that we consider this language, you can give a context grammar to generate this language by taking the clue of sum of previous examples. So, for example if recollect the grammar for this language, which is set of 0 power n 1 power n such that n greeter then equals to 0. The grammar that will introduce to generate this is 0 S 1 and S can be epsilon. These 2 rules generates the language 0 power n 1 power n and n greater then equals to 0.

Now, if you look at a tropical string; in this that is a power m b power m c power m plus n, what we have to get generate? The number of S followed by the number of b equals to the followed by number of C S. Now, what can will do is; if we first generate so many a S corresponding to those many C S. For example, generated here K a S, generating K C S that you can do using this type of rules. And since n can be greater than are equal to 0; if n is 0 in which case it is the language extremely this is form a power m, c power m. That is strings in this language of that form or what can do? We can continue to produce.

Now, b is in between these two certain number of b is and corresponding to which can progress this again using this kind of certain number of c. But here we have 2 things; one thing in mind that after producing a S once to start producing b S we should not again go back to produce S here. For example, if I just simply introduce S does to a S c and S goes to b S c; this will not be the business. The reason why after producing using first rule; for example our second rule here you are I do not have any restriction, that once I have S here at given point of time and produce b c or after producing b c once again using S I may produce a c in between.

So, that way there may be mixer of a s and b s; but we have in restriction that a s followed by b s then C S. So, we have to choose 1 more non terminal symbol which instruct after producing S; first we want to produce as many as you produce using the first symbol here you make call some for example a S. And then you have to give a connection to another not then the symbol once we go through the non-terminal and new non-terminal symbol there, we can be use this kind of thing producing this a S and C S. But not non-terminal symbol will not give you back to come to producing S.

(Refer Slide Time: 04:52)



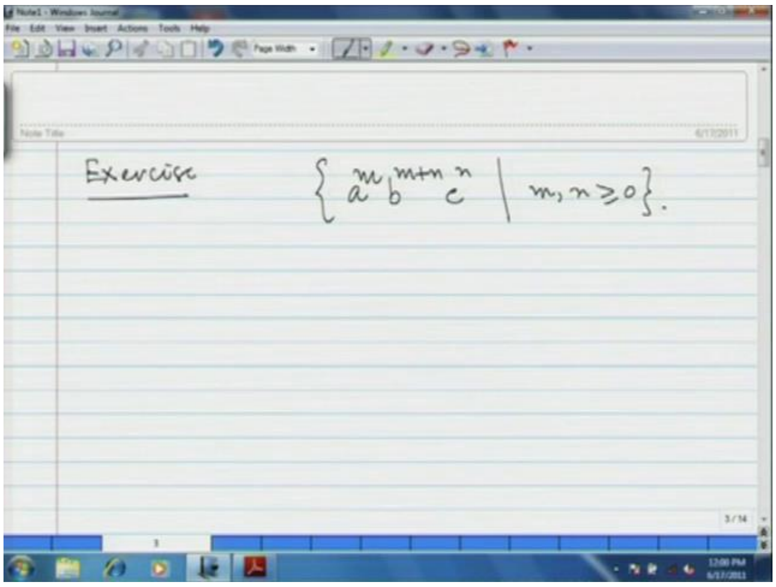
$$G \quad P = \left\{ \begin{array}{l} S \rightarrow aSc \\ S \rightarrow \varepsilon \\ \underline{S \rightarrow A} \checkmark \\ A \rightarrow bAc \\ A \rightarrow \varepsilon \end{array} \right\}$$

Exercise : $L(G) = L$

So, now let us use this information to define the grammar as just mention you can take this rule S goes to a S c; this produces as many a S and as many C S at both the size. And then you can terminate; because we are asked m n garter than equal to 0. Now, I give you non terminal symbol for example, S goes to A and this non terminal symbol will produce b S. Once we go to the non terminal symbol we would not have a connection by coming to S. Now, this A goes to say b A c, and now we can terminate again now order is very clear that this S is connecting to A.

For example, if this m is 0 we did not produce any S then A string is b power m, c power m. So, which case we can directly chose this rule come to A and produce b S and C S the terminate the derivation or in case if we have to produce nearly a S and C S were n is 0. Then, you will not come to this rule to produce b S and thus this rules exercise the will precisely gives you the language l. Now, you can prove that take it as an exercise thus if I call the grammar G; which is having this production rules prove that language generated by this grammar is as desire.

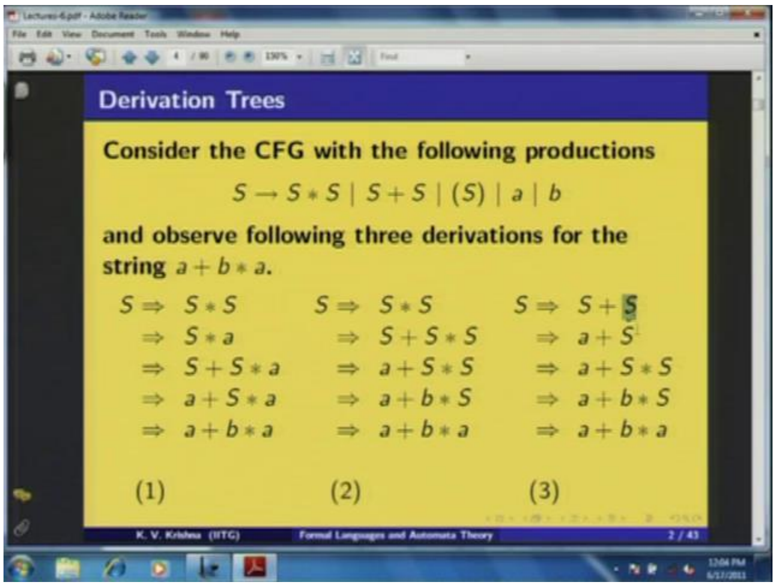
(Refer Slide Time: 07:04)



Exercise $\{a^m b^{m+n} c^n \mid m, n \geq 0\}$.

Similarly, we can take one more that a power m, now in between b power m plus n, c power n; such that m n greater than equal to 0. You can try the context programmer, which generates this language we show that these languages are called context.

(Refer Slide Time: 07:49)



Derivation Trees

Consider the CFG with the following productions

$$S \rightarrow S * S \mid S + S \mid (S) \mid a \mid b$$

and observe following three derivations for the string $a + b * a$.

(1)	(2)	(3)
$S \Rightarrow S * S$	$S \Rightarrow S * S$	$S \Rightarrow S + S$
$\Rightarrow S * a$	$\Rightarrow S + S * S$	$\Rightarrow a + S$
$\Rightarrow S + S * a$	$\Rightarrow a + S * S$	$\Rightarrow a + S * S$
$\Rightarrow a + S * a$	$\Rightarrow a + b * S$	$\Rightarrow a + b * S$
$\Rightarrow a + b * a$	$\Rightarrow a + b * a$	$\Rightarrow a + b * a$

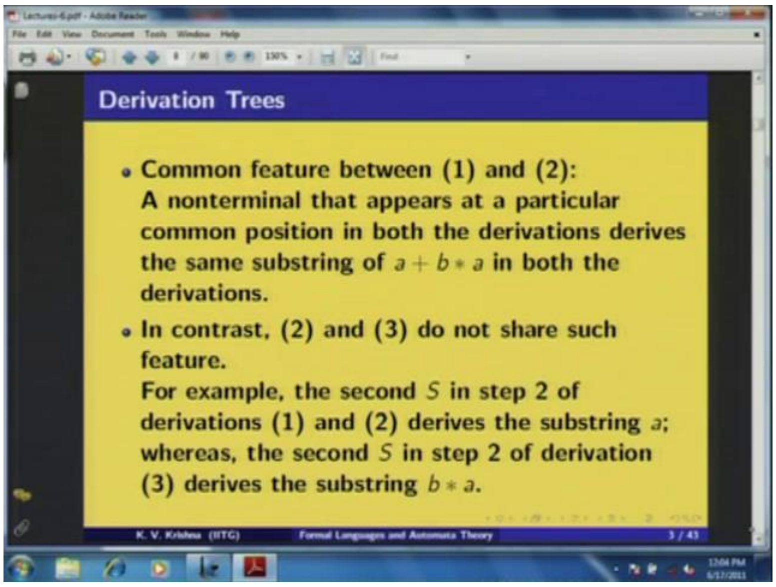
K. V. Krishna (IITG) Formal Languages and Automata Theory 2 / 43

Now, I will introduce a concept called derivation tree, which is useful to understand more about this context languages; because the context languages are generated by context grammar. And this context grammar any string in the languages generating the context grammar this derivation. And to understand more properties of about the derivations this derivation trees are very much useful. In this context let me consider the following example consider the context grammar which has having production is given S goes to S star S or S plus S for bracket S or little a or little b what is the language generated by this is grammar?

If you do little homework and quickly understand that arithmetic expressions with the operations star and plus over symbols a and b can be generated using this production rules. That means the grammar this context grammar generates arithmetic expressions over a, b with the operators plus and additional, multiplication. Now, what we do? Let us observe the following 3 derivations for the string is a plus b star a; using this rule first derivation a plus b square a this gives S star S, then this S I am terminating with a because S goes to a is a symbol.

Now, you consider for this S plus S and then you terminate this S by a; and then this S if we terminate the b. So, using this production rules I have one derivation which produces a plus b square a. Similarly, I have the derivation 2 here and derivation 3 here. These 3 different derivations we can see clearly that first rule here is same; but here in the second step it is different. And in the first step itself is difference with any of these 2; so that way these 3 different derivations are different.

(Refer Slide Time: 10:23)



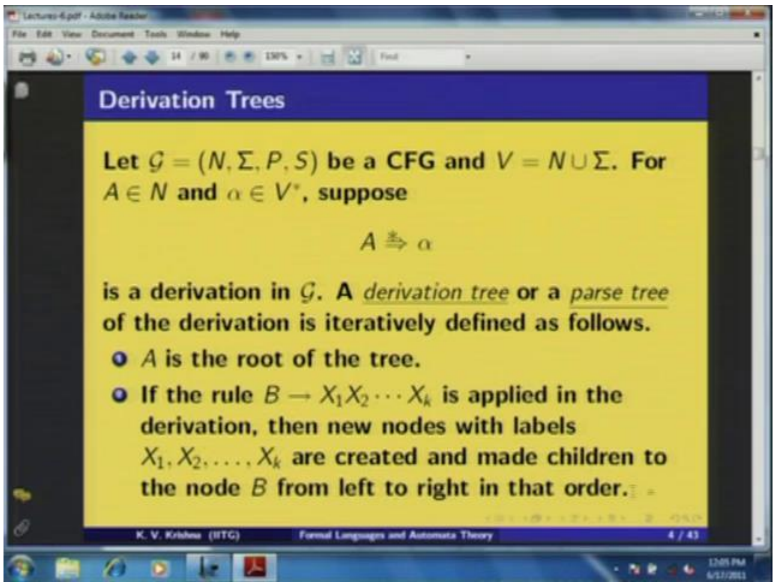
Derivation Trees

- Common feature between (1) and (2):
  A nonterminal that appears at a particular common position in both the derivations derives the same substring of $a + b * a$ in both the derivations.
- In contrast, (2) and (3) do not share such feature.
  For example, the second S in step 2 of derivations (1) and (2) derives the substring a; whereas, the second S in step 2 of derivation (3) derives the substring $b * a$.

K. V. Krishna (IITG) Formal Languages and Automata Theory 3 / 43

But there are common futures that you can understand between the derivations 1 and 2 what are the common future that we can understand here? What is if a particular step we can find non terminal symbol at a particular place? The terminal string produced by the particular non terminal symbol is same as the non-terminal the sub string of this produced by this non-terminal symbol. For example, in this 2 derivations 1 and 2 the second S; if you consider here in the second step this is producing a sub string is a or a plus b star a. And there is a context feature between 1 and 2 with 3 that is this second S in the step 1 non-terminal symbol S it is producing this b star; this 2 are producing sub string a whereas this second non-terminal symbol is producing b star a. Now, this is construct future that I have explained.

(Refer Slide Time: 11:47)



Derivation Trees

Let $\mathcal{G} = (N, \Sigma, P, S)$ be a CFG and $V = N \cup \Sigma$. For $A \in N$ and $\alpha \in V^*$, suppose

$$A \overset{*}{\Rightarrow} \alpha$$

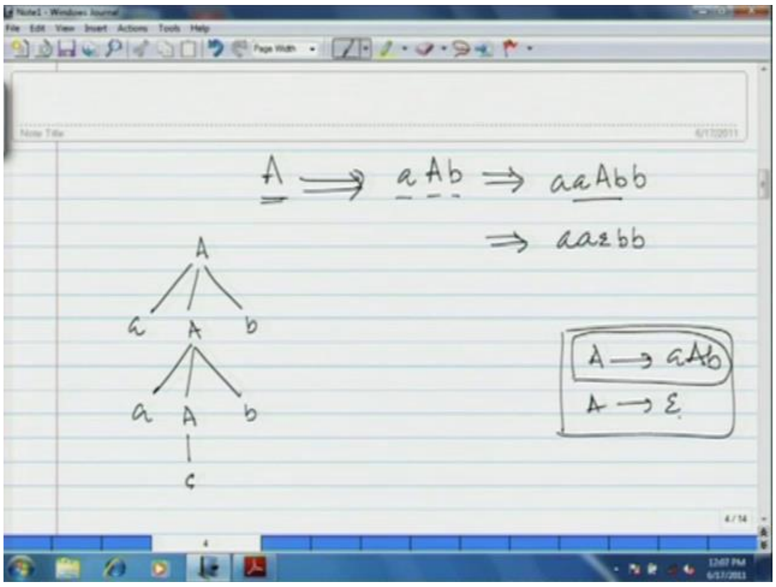
is a derivation in $\mathcal{G}$. A *derivation tree* or a *parse tree* of the derivation is iteratively defined as follows.

- A is the root of the tree.
- If the rule $B \rightarrow X_1X_2 \cdots X_k$ is applied in the derivation, then new nodes with labels $X_1, X_2, \ldots, X_k$ are created and made children to the node B from left to right in that order.

K. V. Krishna (IITG) Formal Languages and Automata Theory 4 / 43

Now, to take about this kind of equableness between derivations the common futures that we have mention or the context future to distinguish of this will not be making use of it motion called derivation tree. Now, let me define formally the motion of derivation tree; let G be at context CFG has given here with V is N union sigma. Now, consider a derivation; let, A gives alpha infinite steps and alpha V star. Now, A derivation tree or A parse tree of a derivation given here is defined iteratively as follows. For the tree we take this non-terminal symbol as the root and they are finitely rules that we are applied to produce alpha here. So, if you consider the rule, B goes to X 1, X 2, X k; if this is applied in the derivation what do we do? Then, new notes with the tables X 1, X 2, X k are created and made children to the node B left to right in that order.

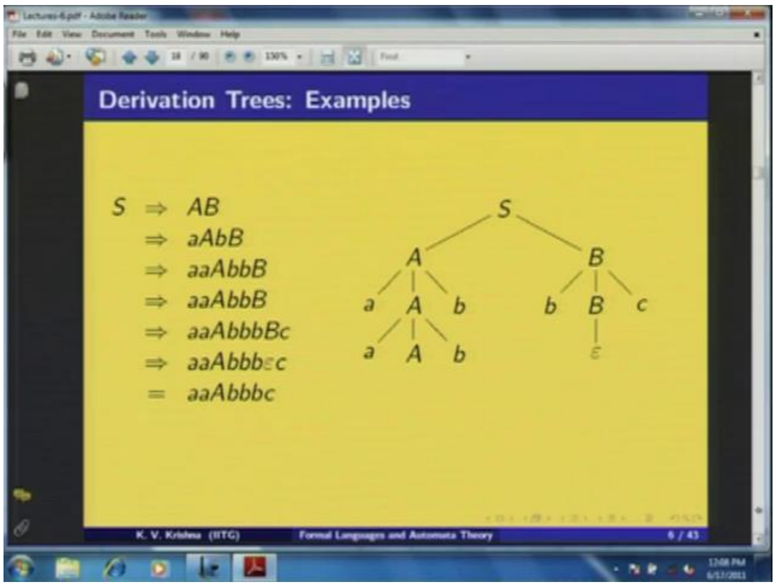
(Refer Slide Time: 13:14)



That means for example, A a A b we have applied and once again I have applied this a a A b b. And applying the rule if you consider this derivation here the rules I am using this rules, if you considered this derivation what is that we have introduced that will be defined A is root of the tree, this A is the root of the tree. And now we have apply the production rule A gives a A b this protection rule. So, we will create this 3 as labels of children of A like this and create the connections like this. And then once again I have he applied the rules A a A b; now, for this we have this 3 children and then the rule A epsilon is introduced.

Now, in the same order we can make these children from left to right. Because if we applied the rules B goes X 1, X 2, X k is a derivation that a particular step; for B we make X 1, X 2, X k children and if we keep them same order.

(Refer Slide Time: 15:12)



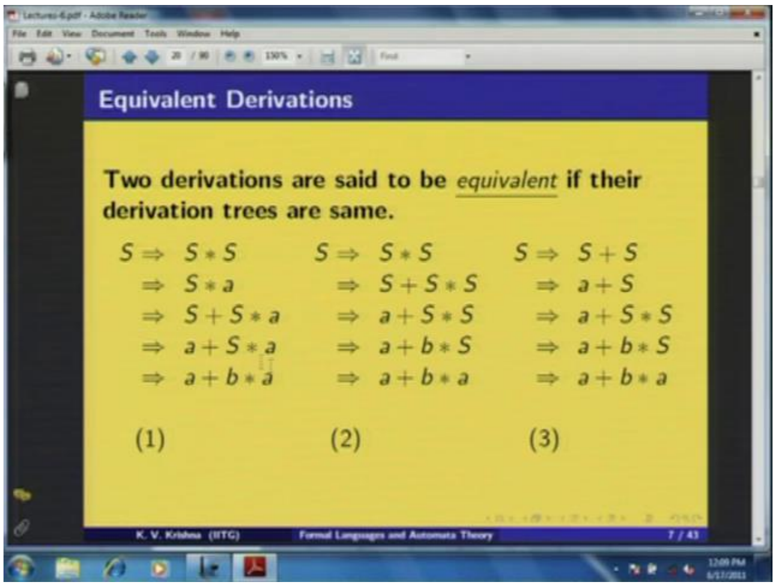
Derivation Trees: Examples

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow aAbB \\ &\Rightarrow aaAbbB \\ &\Rightarrow aaAbbB \\ &\Rightarrow aaAbbbBc \\ &\Rightarrow aaAbbb\varepsilon c \\ &= aaAbbbc \end{aligned}$$

K. V. Krishna (IITG) Formal Languages and Automata Theory 6 / 43

As just explained with S A here; the derivation tree for that is here. And if you consider this derivation S goes to A B the rule is applied. And then in the next step we have applied the rule A goes to a capital A B. And then once again we have applied the rule this and if you consider the corresponding we can verify that is the derivation tree for this derivation.

(Refer Slide Time: 15: 49)



Equivalent Derivations

Two derivations are said to be *equivalent* if their derivation trees are same.

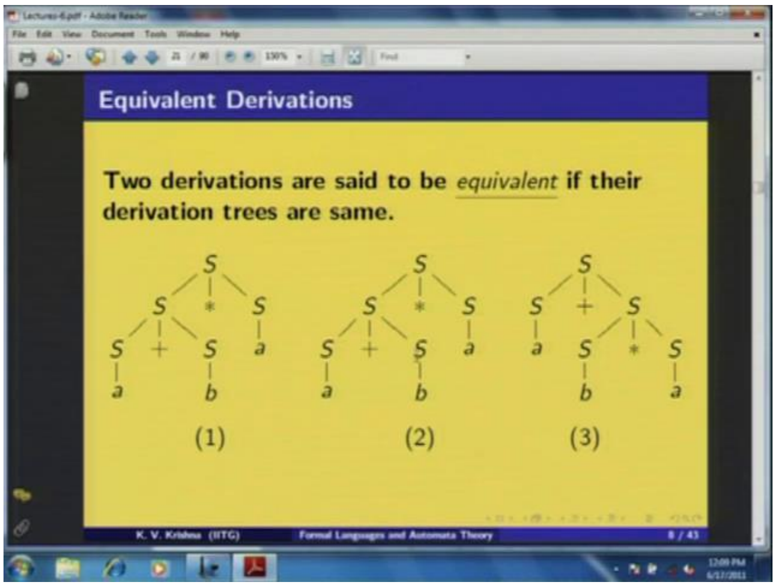
$$\begin{aligned} S &\Rightarrow S*S \\ &\Rightarrow S*a \\ &\Rightarrow S+S*a \\ &\Rightarrow a+S*a \\ &\Rightarrow a+b*a \end{aligned} \qquad \begin{aligned} S &\Rightarrow S*S \\ &\Rightarrow S+S*S \\ &\Rightarrow a+S*S \\ &\Rightarrow a+b*S \\ &\Rightarrow a+b*a \end{aligned} \qquad \begin{aligned} S &\Rightarrow S+S \\ &\Rightarrow a+S \\ &\Rightarrow a+S*S \\ &\Rightarrow a+b*S \\ &\Rightarrow a+b*a \end{aligned}$$

(1) (2) (3)

K. V. Krishna (IITG) Formal Languages and Automata Theory 7 / 43

Now, using the concept derivation tree or parse tree we will talk about the equivalence between the derivations from we said two derivations which are said to be equivalent; if there derivation trees are same. And again I will recollect this derivation.

(Refer Slide Time: 16:23)



Equivalent Derivations

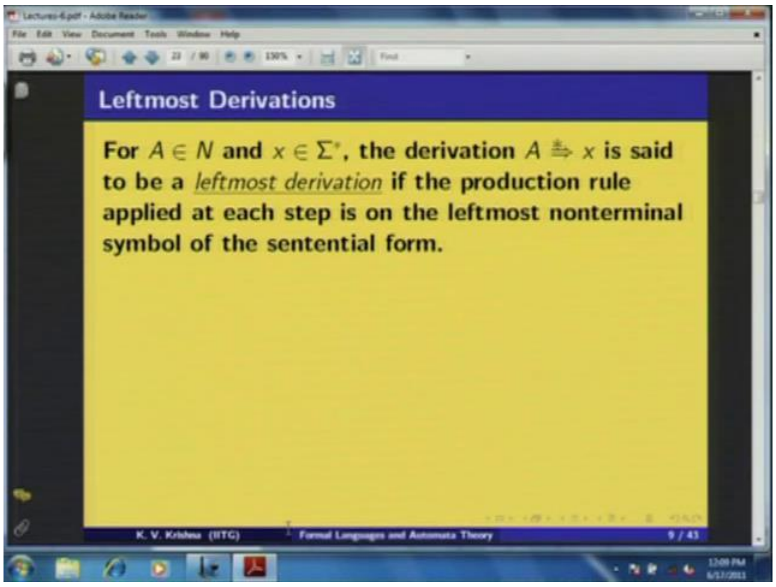
Two derivations are said to be *equivalent* if their derivation trees are same.

(1) (2) (3)

K. V. Krishna (IITG) Formal Languages and Automata Theory 8 / 43

Now, if we draw the derivations trees for these 3 derivations; you can see for that first 2 derivations these 2 trees are exactly same and for the third derivation the tree is different form 1 and 2. And thus we can understand that this 2 derivations that derivation number 1 and 2 are equal lent were as the third one is not equal lent to any of the first 1 and 2.

(Refer Slide Time: 17:00)



Leftmost Derivations

For $A \in N$ and $x \in \Sigma^*$, the derivation $A \overset{*}{\Rightarrow} x$ is said to be a *leftmost derivation* if the production rule applied at each step is on the leftmost nonterminal symbol of the sentential form.

K. V. Krishna (IITG) Formal Languages and Automata Theory 9 / 43

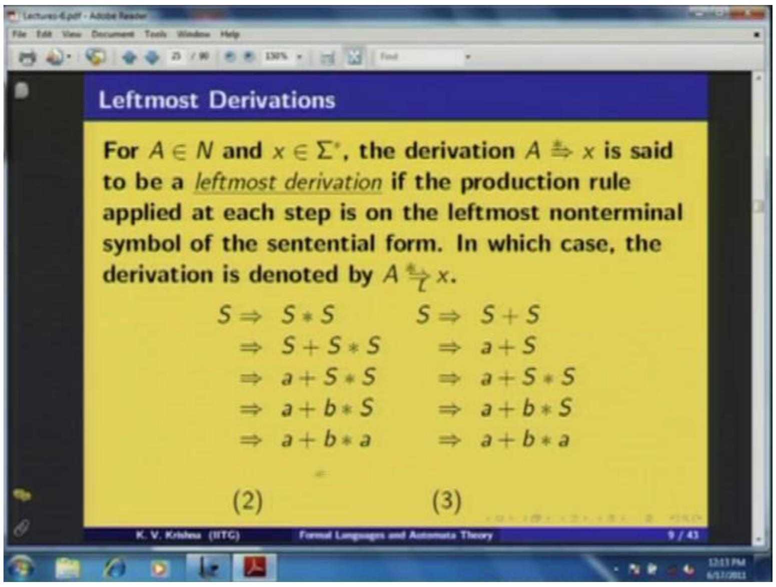
Now, this concept related to equal lent you we can capture through the motion left most derivation. So that means essentially what we have understood in the previous case about the equal lent 2 derivations are equal lance if we have derivation tree. That means the concept of derivation tree equal lance was assign between equal lance derivations. Now, can be any have derivations a canonical representation of derivation which is unique in some sense for equal lent derivations. In that context, we are introducing this motion called left motion derivations. So, this left most derivation defined as follows if we consider derivation A gives X and X is the terminal string infinitely many steps through a context programmer.

Now, we say this is left most derivation A derivation A gives X. If we say left most derivation, if the production rule apply in each step is on the left motion on symbol of the conditional form. When you are getting a derivation you have applied rules in each step now what is the condition here? For example, these derivations when you look at of course here in each center conditional form you have only one non-terminal symbol. So, whatever production here I am applying that is the left most non-terminal symbol. But if you consider this example in this derivation you have the choice there are 2 non-terminal steps A and B. So, to say derivation is left most derivation the production rules that you apply at particular non-terminal symbol.

In the particular non-terminal symbol that is occurrence first sentential form first one, left side left most non-terminal symbol. For example, in this step we have applied on the first non-terminal symbol A; so, this step is satisfying that. And if you continue like this and now you see there is a non-terminal symbol A here and there non-terminal symbol B here in this step. So, in this step we have not applying production rule on this non-terminal symbol; were as we have applying production rule on this non terminal symbol B and here also you can observe that. On B we have applied these production rules to produce b B c that means this step, this step is violating the condition of left most derivation.

Because in the left most derivation as we have mentioned at each step we are applying the production rule, we have to apply the production rule on the left most non-terminal symbol; then, we say left most derivation. So, in that sense this is not the left most derivation.

(Refer Slide Time: 20:14)



Leftmost Derivations

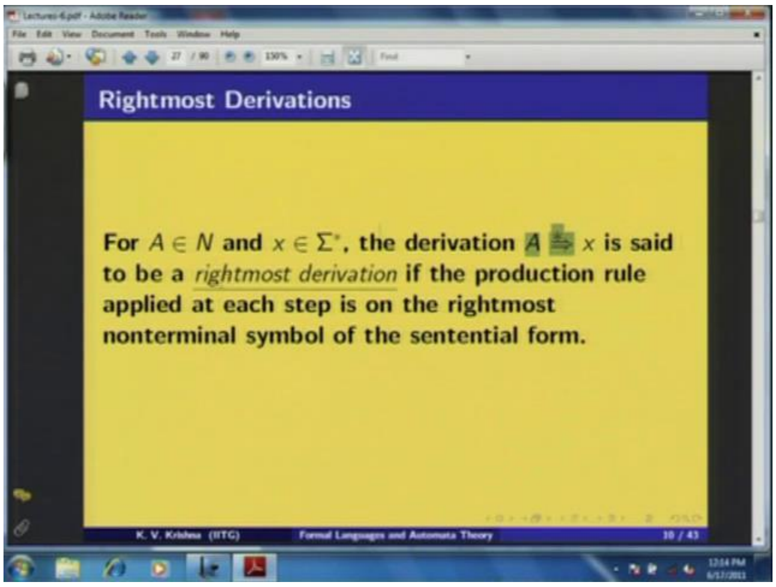
For $A \in N$ and $x \in \Sigma^*$, the derivation $A \overset{*}{\Rightarrow} x$ is said to be a *leftmost derivation* if the production rule applied at each step is on the leftmost nonterminal symbol of the sentential form. In which case, the derivation is denoted by $A \overset{*}{\underset{l}{\Rightarrow}} x$.

$$\begin{array}{llll} S \Rightarrow & S*S & S \Rightarrow & S+S \\ \Rightarrow & S+S*S & \Rightarrow & a+S \\ \Rightarrow & a+S*S & \Rightarrow & a+S*S \\ \Rightarrow & a+b*S & \Rightarrow & a+b*S \\ \Rightarrow & a+b*a & \Rightarrow & a+b*a \\ & (2) & & (3) \end{array}$$

K. V. Krishna (IITG) Formal Languages and Automata Theory 9 / 43

So, A derivation if we say the left most derivation that is denoted by a subscript l. For example, as I have explained in the previous case also this derivation 2 is a left most derivation. Because this is S star S we have applied production rule on this we have got X plus S; then, S is terminated by here, then this S is terminated here. And derivation 3 also left most derivation; because we have applied the production rule first S here and then after this is the non-terminal symbol. On this non-terminal symbol we have applied this production rule, and now here is a non-terminal symbol and that we have applied. So, this derivation and both the derivation 2 and 3 in the previous example both are left most derivation; were as the derivation is one, it is not left most derivation.

(Refer Slide time: 21:26)


Rightmost Derivations

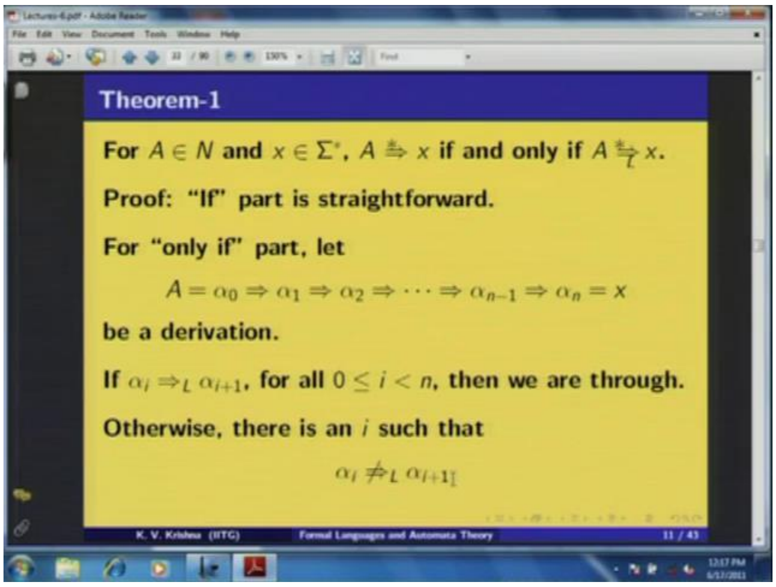
For $A \in N$ and $x \in \Sigma^*$, the derivation $A \Rightarrow x$ is said to be a *rightmost derivation* if the production rule applied at each step is on the rightmost nonterminal symbol of the sentential form.

K. V. Krishna (IITG) Formal Languages and Automata Theory 10 / 43

And, now a similar concept will write to the derivation can be introduced that means a derivation A gives X infinitely many steps we discuss a terminal string X, we say this is right most derivation. If each production rule will the production rule will have applied each step it is the right most non-terminal symbol of the sentential form. In which case we say it is right most derivation? We see the similarities in the left most derivation and right most derivation. In case of left most derivation we have applying production rule on left most non-terminal symbol in each step. And in case of right most derivation we have applying the production rule are the and right most non-terminal symbol is in each step.

You see the similarity between these 2 and because of the similarities of these 2 whatever the results that we give for the left most symbol derivation; 1 we can similarly produce right most derivations also. So, in the following simply consider only left most derivations and similar results can be to produce as an exercise for the right most derivation as well. There is a rotation similar to left most derivation; if it is right most derivation, we use the subscript r for this derivation for that derivation.

(Refer Slide Time: 22:47)



Theorem-1

For $A \in N$ and $x \in \Sigma^*$, $A \overset{*}{\Rightarrow} x$ if and only if $A \overset{*}{\Rightarrow}_L x$.

Proof: "If" part is straightforward.

For "only if" part, let

$$A = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n = x$$

be a derivation.

If $\alpha_i \Rightarrow_L \alpha_{i+1}$, for all $0 \leq i < n$, then we are through.

Otherwise, there is an i such that

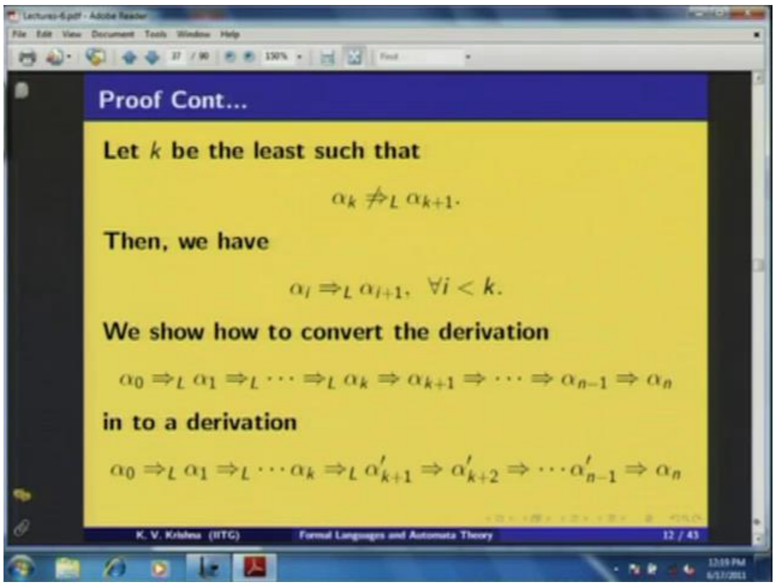
$$\alpha_i \not\Rightarrow_L \alpha_{i+1}$$

K. V. Krishna (IITG) Formal Languages and Automata Theory 11 / 43

Now, let me give you a theorem that every derivation that produces terminal string that can I have equal lent to left most derivation. What I mean here is if you have a derivation A gives X; where X is the termination string. Then, you can produce X through A left most derivation starting from A and conversely. Of course you can quickly understand that important straight forward; because every left most derivation is derivation. Therefore, if we can produce A terminal X and l infinitely ministers through a left most derivation; of course you produce the X infinitely ministers then if part is straight forward. Now, let me look at the only if part so what we do? You consider the derivation A gives X infinitely in this steps; let me called A is alpha naught and that gives alpha 1is first step and second step is may be alpha 2 and so on.

In n steps we are producing X. Let me consider the derivation as this what we do we have to show there is an equal lent left most derivation for this. That means it produces X using in each step and left most non terminal symbol we are applying production rule. If what are the derivations that you have consider in which you have this kind of property. That means on left most non-terminal symbol only we have to apply the production rule in each step then it is a left most derivation then there nothing we have to prove. Then, otherwise at particular step we have not applied to production rule on left most non-terminal symbol and produce alpha i plus 1 that means there is an i alpha it produces alpha i plus 1. But not through the production rule applied on the left most non-terminal symbol that is this.

(Refer Slide Time: 25:12)



Proof Cont...

Let k be the least such that

$$\alpha_k \not\Rightarrow_L \alpha_{k+1}.$$

Then, we have

$$\alpha_i \Rightarrow_L \alpha_{i+1}, \ \forall i < k.$$

We show how to convert the derivation

$$\alpha_0 \Rightarrow_L \alpha_1 \Rightarrow_L \cdots \Rightarrow_L \alpha_k \Rightarrow \alpha_{k+1} \Rightarrow \cdots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n$$

in to a derivation

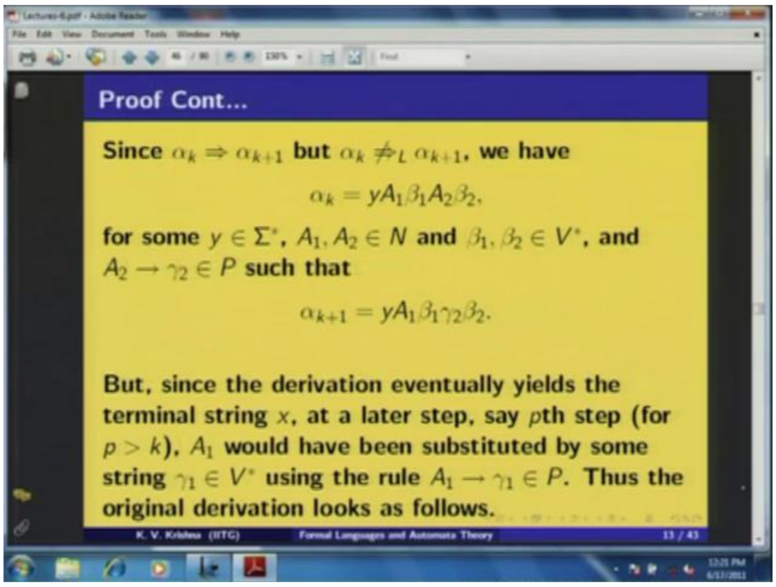
$$\alpha_0 \Rightarrow_L \alpha_1 \Rightarrow_L \cdots \alpha_k \Rightarrow_L \alpha'_{k+1} \Rightarrow \alpha'_{k+2} \Rightarrow \cdots \alpha'_{n-1} \Rightarrow \alpha_n$$

K. V. Krishna (IITG) Formal Languages and Automata Theory 12 / 43

Now, what do you consider let k be the least for which this is satisfied that means till k we have alpha i produces alpha i plus 1 through that which is the part of applied on which is alpha left most non-terminal symbol we are sectional alpha i. And there after particularly at this place we have the condition failed. Now, what will you do? We will show a mechanism, which can make, which can you a derivation for which k plus 1 this is k plus 1 step. Because in the first step we are getting alpha 1 we are k plus 1 getting k plus 1; we have k plus1 step also this derivation which is applied on the left most non-terminal symbol.

So, if I can give a mechanism which can convert the k plus 1 step also as a the left most as which part of the left most derivation. Then, we can apply an induction to produce the equal lent derivation that means derivation, which produces X in a step. That means what will give a conversation of this derivation an equivalent 1 to produces the alpha n; with this alpha k plus 1 step you have applied left most non-terminal symbol. There after some string that is alpha dash k plus 1; sorry alpha dash k plus 1 alpha dash k plus 2, alpha dash k plus n and so on alpha dash n minus 1 alpha n that is that is x.

(Refer Slide Time: 27:00)



Proof Cont...

Since $\alpha_k \Rightarrow \alpha_{k+1}$ but $\alpha_k \not\Rightarrow_L \alpha_{k+1}$, we have

$$\alpha_k = yA_1\beta_1A_2\beta_2,$$

for some $y \in \Sigma^*$, $A_1, A_2 \in N$ and $\beta_1, \beta_2 \in V^*$, and $A_2 \to \gamma_2 \in P$ such that

$$\alpha_{k+1} = yA_1\beta_1\gamma_2\beta_2.$$

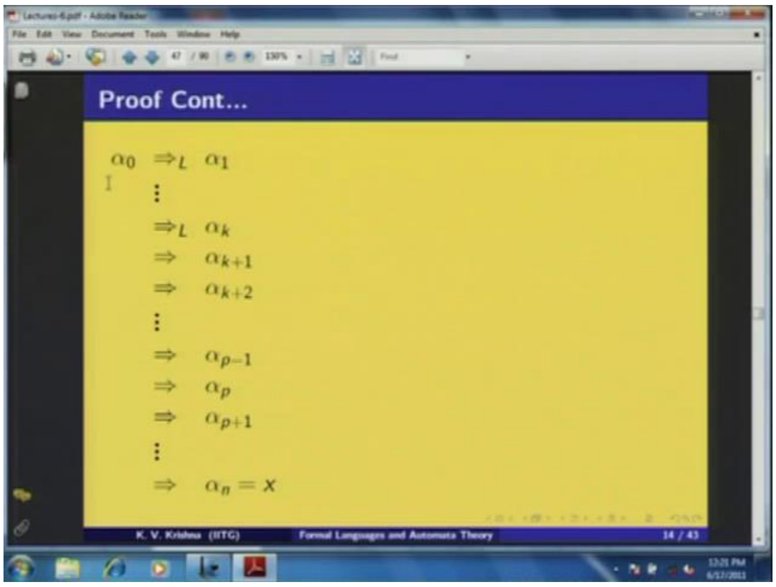
But, since the derivation eventually yields the terminal string x, at a later step, say pth step (for $p > k$), A_1 would have been substituted by some string $\gamma_1 \in V^*$ using the rule $A_1 \to \gamma_1 \in P$. Thus the original derivation looks as follows.

K. V. Krishna (IITG) Formal Languages and Automata Theory 13 / 43

So, what do we have an information alpha gives alpha k plus 1; but it is not obtained we have applied the production rule on the left most non-terminal symbol that means alpha k is at the form sum termination y. And A 1 is the first non-terminal symbols in the termination form that is the left most non-terminal symbol. And after that the left most non-terminal symbol is A 1 some non terminal symbol A 2 determine1that is element of V star and some 1 element of the symbol A 2 and some beta 2 and element of V star. I can see here 2 non-terminal symbols; just to mention that in alpha k is the left most non-terminal symbol were as we have applied to the production rule to get alpha k plus 1. And the non-terminal symbols A 2; this A 2 is the second non-terminal symbol from the left side then the beta 2 as well.

Now, y is terminal string and A 1 is the and A 1, A 2 is the non-terminal symbols beta 1, beta 2 are in elements of V star. And we have applied production rule say A 2 goes to gamma 2 to get alpha k plus 1. Now, what we have understood here in this derivation eventually we are getting terminal alpha n terminal string. Thus, at a particular step in the derivation we have to apply he production rule A 1.Because of A 1 is the non-terminal symbol to converting into the terminals string we have to apply somewhere production rule. So, let me say that means the peak step thus and peak step we are in the applying the A 1 goes to gamma 1.

(Refer Slide Time: 28:51)



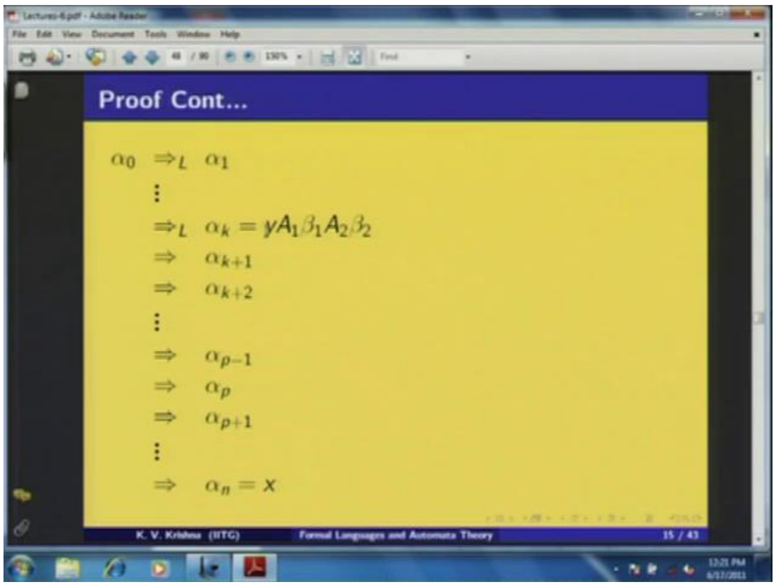
Proof Cont...

$$\begin{aligned}\alpha_0 &\Rightarrow_L \alpha_1\\ &\vdots\\ &\Rightarrow_L \alpha_k\\ &\Rightarrow \alpha_{k+1}\\ &\Rightarrow \alpha_{k+2}\\ &\vdots\\ &\Rightarrow \alpha_{p-1}\\ &\Rightarrow \alpha_p\\ &\Rightarrow \alpha_{p+1}\\ &\vdots\\ &\Rightarrow \alpha_n = x\end{aligned}$$

K. V. Krishna (IITG) Formal Languages and Automata Theory 14 / 43

Now, what do we do; so, in this situation the derivation looks at as follows are alpha naught that is a gives alpha 1 is first step this so on in alpha k applied on the left motion non-terminal symbol.

(Refer Slide Time: 29:11)



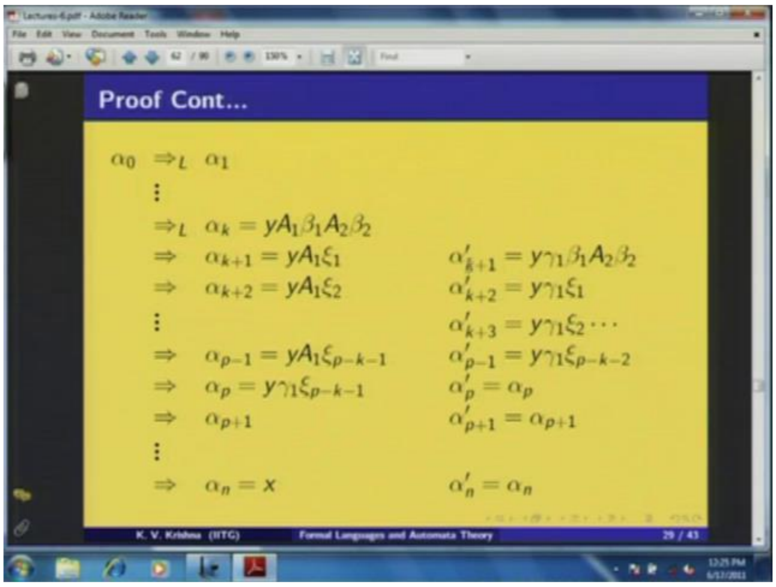
Proof Cont...

$$\begin{aligned}\alpha_0 &\Rightarrow_L \alpha_1\\ &\vdots\\ &\Rightarrow_L \alpha_k = yA_1\beta_1A_2\beta_2\\ &\Rightarrow \alpha_{k+1}\\ &\Rightarrow \alpha_{k+2}\\ &\vdots\\ &\Rightarrow \alpha_{p-1}\\ &\Rightarrow \alpha_p\\ &\Rightarrow \alpha_{p+1}\\ &\vdots\\ &\Rightarrow \alpha_n = x\end{aligned}$$

K. V. Krishna (IITG) Formal Languages and Automata Theory 15 / 43

And, step k plus 1 at we have this choosing and so on we producing X this is the formal derivation at we have… With the situation alpha i k form as have an explained. And we have applied production rule on a to 2 replace gamma there to the alpha k plus 1. Let me call size 1 it is beta 1 and gamma 2, beta 2 let me call is psi1 and I have mention it is termination y it as we are not going to apply anything a 1 is the first non-terminal symbol. And that as I have mentioned applying the production rule on a 1 in this step only in the peak step only. So, that way the production rule in between these step that you have to apply on psi 1 to produce the psi 2 and then psi 3 and so on.

(Refer Slide Time: 30:01)



Proof Cont...

$$
\begin{array}{lll}
\alpha_0 \Rightarrow_L \alpha_1 & & \\
\vdots & & \\
\Rightarrow_L & \alpha_k = yA_1\beta_1A_2\beta_2 & \\
\Rightarrow & \alpha_{k+1} = yA_1\xi_1 & \alpha'_{k+1} = y\gamma_1\beta_1A_2\beta_2 \\
\Rightarrow & \alpha_{k+2} = yA_1\xi_2 & \alpha'_{k+2} = y\gamma_1\xi_1 \\
\vdots & & \alpha'_{k+3} = y\gamma_1\xi_2 \cdots \\
\Rightarrow & \alpha_{p-1} = yA_1\xi_{p-k-1} & \alpha'_{p-1} = y\gamma_1\xi_{p-k-2} \\
\Rightarrow & \alpha_p = y\gamma_1\xi_{p-k-1} & \alpha'_p = \alpha_p \\
\Rightarrow & \alpha_{p+1} & \alpha'_{p+1} = \alpha_{p+1} \\
\vdots & & \\
\Rightarrow & \alpha_n = x & \alpha'_n = \alpha_n
\end{array}
$$

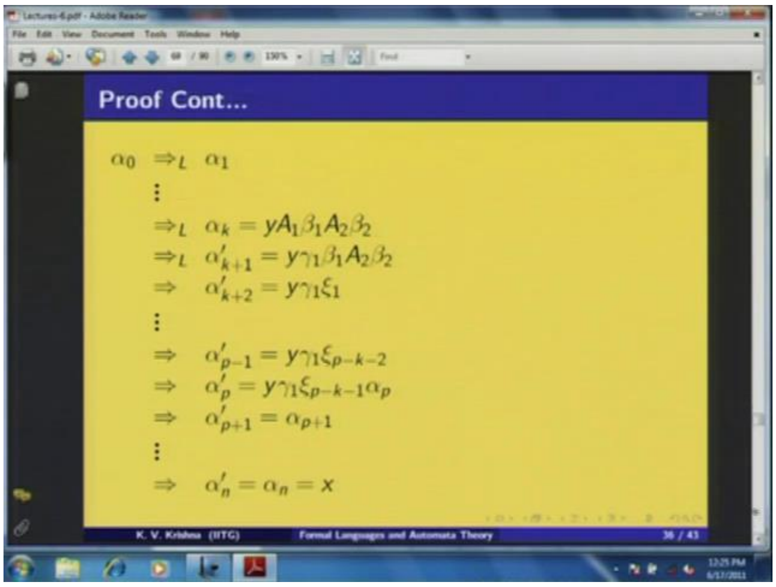
K. V. Krishna (IITG) Formal Languages and Automata Theory 29 / 43

At, this stage we have got in this step we can quickly understand that this is psi p minus k minus 1; that p step we have. And now on a 1 we have applied the rule a goes gamma 1 and the peak step let this get alpha p their after we have that is alpha p plus 1 and so on. Now, what do we do; I give a mechanism to give l here that means whatever this string alpha k plus 1 here that want to call now alpha k plus 1 dash that is this rule we preponed in this step to get L here. Let me now alpha k plus 1 dash is y gamma 1, beta 1, A 2, beta 2 that means I have production rule A 1 goes to gamma1. In this step k plus 1 itself then 1 what do we do this I will set now alpha k plus 2 dash to be I will apply in the in place of A 2, gamma 2.

So that means this step postpone here and this preponed to place here. And thereafter there will adjust that means any were this is i 1 alpha dash k plus 1 is next step from psi 1 is getting psi 2 that you will get here. And then if you continue here I already have to gamma 1; so, what I will call alpha p minus 1 dash to be y gamma 1 and what does here psi p k minus 2 because that is preponed. Now, alpha p dais is this 2 are same because of preponed within them what are the steps? The numbers of steps are same.

So, that is the alpha p. So, alpha p plus 1 dash is I will continue same that is alpha p plus 1 and I will continue the what are the steps that is alpha n dash could be alpha X. So, if I set these strings you can clearly see that alpha k produces alpha k plus 1 dash through production rule which are applied on the left most non-terminal symbols. Now I replace alpha k plus 1 dash in place of this I can give l here. That means I have k plus 1 step here which are production rules applied here on the left most non-terminal symbols. So, I will move this string here to get this derivation and understand through this derivation alpha k plus 1 dash produces alpha k plus 2 dash and alpha k plus 2 dash produces alpha plus 3 dash and so on. There after continue this what do I have is this are the k plus 1; I can move here and alpha k plus 2 and they are the things.

(Refer Slide Time: 33:10)



Proof Cont...

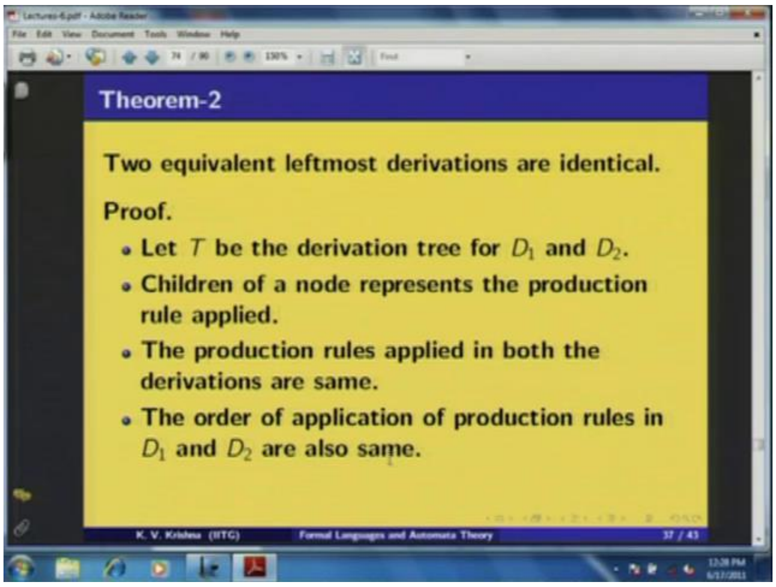
$$
\begin{aligned}
\alpha_0 &\Rightarrow_L \alpha_1 \\
&\vdots \\
&\Rightarrow_L \alpha_k = yA_1\beta_1A_2\beta_2 \\
&\Rightarrow_L \alpha'_{k+1} = y\gamma_1\beta_1A_2\beta_2 \\
&\Rightarrow \alpha'_{k+2} = y\gamma_1\xi_1 \\
&\vdots \\
&\Rightarrow \alpha'_{p-1} = y\gamma_1\xi_{p-k-2} \\
&\Rightarrow \alpha'_p = y\gamma_1\xi_{p-k-1}\alpha_p \\
&\Rightarrow \alpha'_{p+1} = \alpha_{p+1} \\
&\vdots \\
&\Rightarrow \alpha'_n = \alpha_n = x
\end{aligned}
$$

K. V. Krishna (IITG) Formal Languages and Automata Theory 36 / 43

So that I have derivation in which I have applied left most production rules in left most non-terminal symbol will k plus 1 step. So that means these mechanism by proponing this step I could produce from k steps; we have applied this left most protection rule and the left most non-terminal symbol to k plus 1 steps how to do that I have explain.

Now, we want to induction X 1gives to k plus 2 steps and here to 1 more point here understand that the number of steps is not increase or decrease the same number of steps. So that what will happens till n steps what we have complete this and those is fixed. And thus thin this limit next k plus 2 and same mechanism and k plus 3 and so on. Will n here the fixed number of steps by applying the inductively we can produce this derivation that is left most derivation to produce X from A. Thus, we have theorem that means any derivation which gives any form a gives infinitely steps corresponding to which we can give a left most derivation; if produces x that is same terminal string from A and conversely of course.

(Refer Slide Time: 34:38)



Theorem-2

Two equivalent leftmost derivations are identical.

Proof.

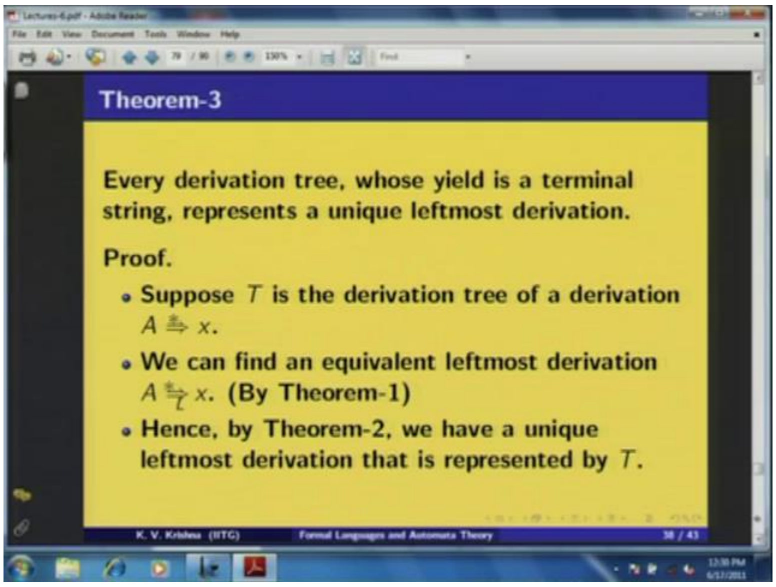
- **Let T be the derivation tree for D_1 and D_2.**
- **Children of a node represents the production rule applied.**
- **The production rules applied in both the derivations are same.**
- **The order of application of production rules in D_1 and D_2 are also same.**

K. V. Krishna (IITG) Formal Languages and Automata Theory 37 / 43

Now, let me give you a one more small result that is 2 most quality left most derivations are identical. So, that means here what we have to show that equal leant left most derivation in means we have same derivation tree, and there identical means we have to say in each step what are the production there an applying in the first derivation same thing of applying in the second derivation. It will essay to understand; because these two are consider 2 derivations left most derivation D 1 and D 2. Since, there are equal leant we have common tree. Let me call that as T.

Now, understand that in the common tree, tree is same understood children of each note will be protection rule will apply in D 1 and D 2. So, since we have common derivation tree the production rule which we have used in D 1 and D 2 they are essentially same. And now the production rule applied on both are same. Now, you understand order of the protection rule is also same the reason is both are left most derivations. And on each step, in each step we have applied the protection rule in the left most non-terminal symbol and the protection rules that we are applied are the same. And thus we understand that the order of protection rules D 1 and D 2 are also same and hence they are identical. So, equivalent left most derivations are identical.

(Refer Slide Time: 36:13)



Theorem-3

Every derivation tree, whose yield is a terminal string, represents a unique leftmost derivation.

Proof.

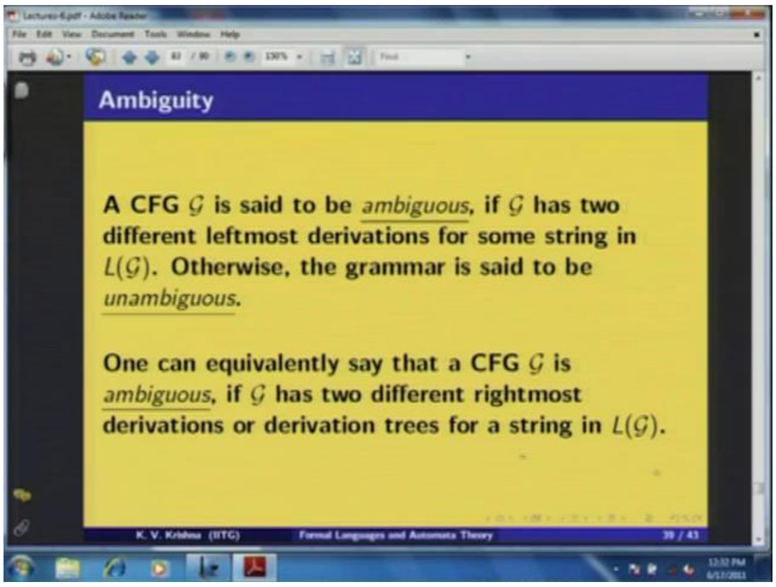
- Suppose T is the derivation tree of a derivation $A \overset{*}{\Rightarrow} x$.
- We can find an equivalent leftmost derivation $A \overset{*}{\underset{L}{\Rightarrow}} x$. (By Theorem-1)
- Hence, by Theorem-2, we have a unique leftmost derivation that is represented by T.

K. V. Krishna (IITG) Formal Languages and Automata Theory 38 / 43

Now, I used this result to produce this following important theorem looking for. Every derivation tree whose yield is a terminal string, represents the a unique left most symbol. So, that means the derivation tree concepts is equivalent intersection if find through derivations tree and the canonical form the in derivation is left most derivation as it had expression; that is expressed through this result left most derivation we have for each derivation tree which produces the terminal string of course.

We prove for this left most derivation tree for the derivation A gives X infinitely. Now, we can find equivalent left most derivation; because the theorem 1what I was explain till now for every derivation we can have an equivalent left most derivation. Now, if you have 2 equivalent left most derivations they are identical; that is what we have observe in theorem 2. So, by theorem 2 we have unique of left most derivation represented by T. So, thus what you have every derivation tree which produces a terminal string which reproduces represents a unique left most derivation.

(Refer Slide Time: 38:29)


Lectures-6.pdf - Adobe Reader
File Edit View Document Tools Window Help
K. V. Krishna (IITG)
Formal Languages and Automata Theory
39 / 43

Ambiguity

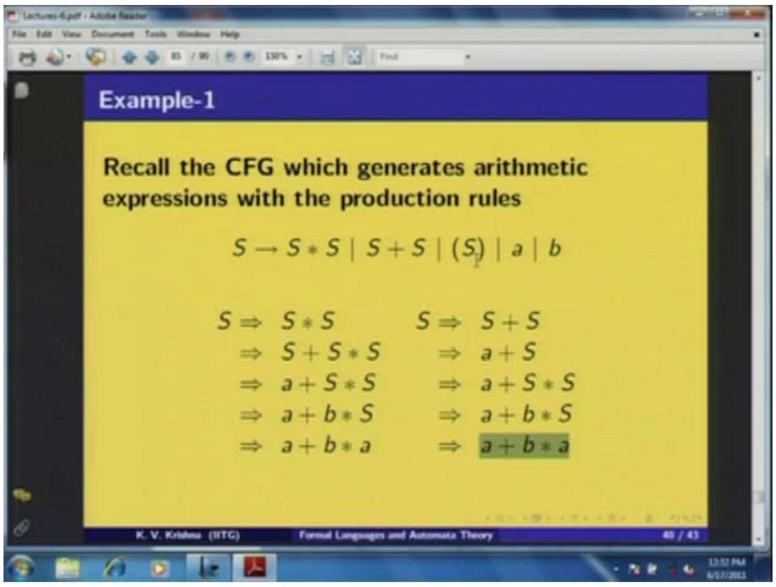
A CFG $\mathcal{G}$ is said to be ambiguous, if $\mathcal{G}$ has two different leftmost derivations for some string in $L(\mathcal{G})$. Otherwise, the grammar is said to be unambiguous.

One can equivalently say that a CFG $\mathcal{G}$ is ambiguous, if $\mathcal{G}$ has two different rightmost derivations or derivation trees for a string in $L(\mathcal{G})$.

Now, I will introduce a new concept called ambiguity this is an important application for the concept of left most derivation. And the concept of ambiguity as very much practically important, particular in the context of compiler with in because of compiler deign and the grammar that one an expect is big grammar and context grammar. Let me define the concept of ambiguity of a context grammar A CFG is to be an ambiguous has 2 different left most derivation for some string if L of (G). Otherwise, we say the grammar is unambiguous.

That means for every string they as to be unique left most derivation they cannot be left most derivation for a string in the language. So, we have expressed through this steps of left most derivation as I had mentioned. Once we have equivalent a grammar is ambiguity if G has 2 different right most derivations or derivation trees. Because the concept of derivation tree or the uniquely can be capture through left most derivation. So, that way one can see if equivalent if spring or 2 different derivatives then we say the grammar is ambiguity this is an equivalent formulation.

(Refer Slide Time: 39:46)



Example-1

Recall the CFG which generates arithmetic expressions with the production rules

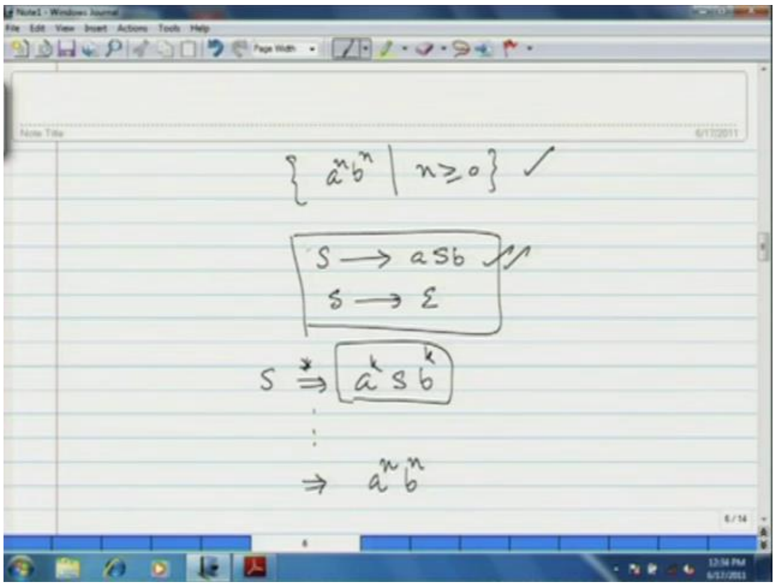
$$S \to S * S \mid S + S \mid (S) \mid a \mid b$$

$$\begin{aligned} S &\Rightarrow S * S \\ &\Rightarrow S + S * S \\ &\Rightarrow a + S * S \\ &\Rightarrow a + b * S \\ &\Rightarrow a + b * a \end{aligned} \qquad \begin{aligned} S &\Rightarrow S + S \\ &\Rightarrow a + S \\ &\Rightarrow a + S * S \\ &\Rightarrow a + b * S \\ &\Rightarrow a + b * a \end{aligned}$$

K. V. Krishna (IITG) Formal Languages and Automata Theory 40 / 43

Now, let me give you one example; let us recall the grammar that produces arithmetic expressions. Now, as you understand, then we have already mention that a plus b star a this string we have 2 different left most derivations. And thus the grammar given here is ambiguity for 1 string we have 2 different left most derivations.

(Refer Slide Time: 40:36)



$$\{ a^n b^n \mid n \geq 0 \} \checkmark$$

$$\boxed{\begin{aligned} S &\longrightarrow aSb \\ S &\longrightarrow \varepsilon \end{aligned}}$$

$$S \overset{*}{\Rightarrow} \boxed{a^k S b^k}$$

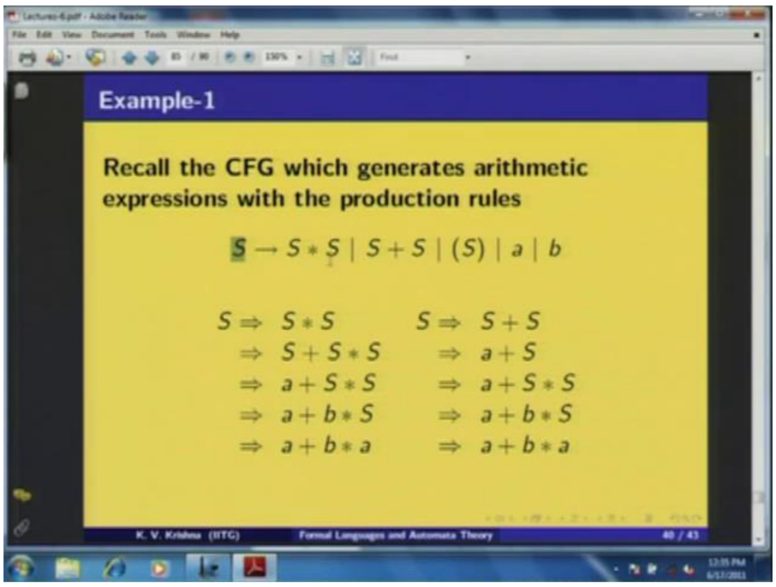
$$\vdots$$

$$\Rightarrow a^n b^n$$

Now, the grammar given for a power m and b power m and c power n and greater than equals to 0 for this the production rules are given to you. Now, since there is only 1 non-terminal symbol and right hand said of this production rule what is the derivation that you have here which of this form after k number of steps defiantly infinite steps, we can produce like this. And then we have terminating are finitely this steps what we have want a power m, b power m, of m for k steps will have so many steps and finitely this steps.

And, we observe this steps in this derivation it is a power k, S and b power k; and you have only one non terminal symbol and is derivation is continuing because of this rule 1 only and rule 2 is only used to terminate the derivation. So, that the typical sensational form in any derivation of this is grammar in the form is a power k and b power k and only one non terminal symbol. And which only will be applying protection rule thus this is the left most derivation. And this is the only derivation possible with a power m, b power m a string of this language and that we can understand that this is unambiguous grammar.

(Refer Slide Time: 45:23)



Example-1

Recall the CFG which generates arithmetic expressions with the production rules

$$S \rightarrow S * S \mid S + S \mid (S) \mid a \mid b$$

$$\begin{aligned} S &\Rightarrow S * S \\ &\Rightarrow S + S * S \\ &\Rightarrow a + S * S \\ &\Rightarrow a + b * S \\ &\Rightarrow a + b * a \end{aligned} \qquad \begin{aligned} S &\Rightarrow S + S \\ &\Rightarrow a + S \\ &\Rightarrow a + S * S \\ &\Rightarrow a + b * S \\ &\Rightarrow a + b * a \end{aligned}$$

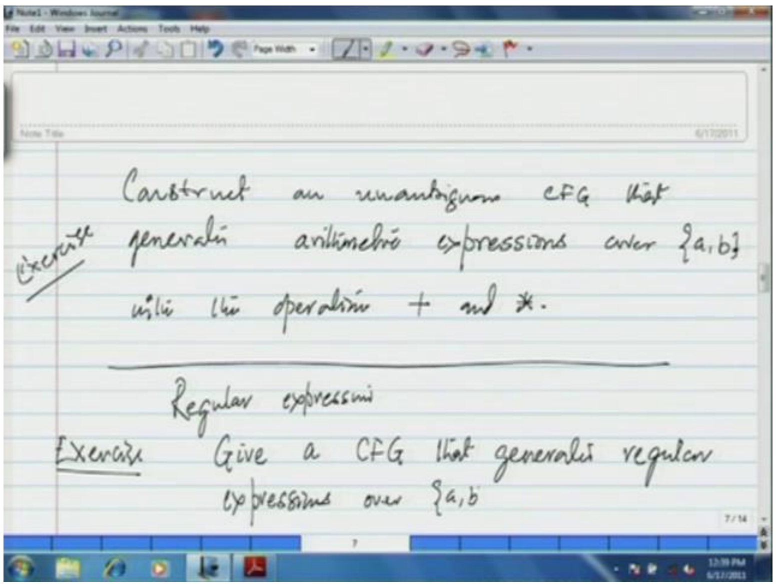
K. V. Krishna (IITG) Formal Languages and Automata Theory 40 / 43

Now, one can thing about unambiguous this grammar to produce arithmetic grammar; because here the syntax the grammar is given in this way from. But the problem where it is coming a plus b star a; we have denote 2 different derivations are the first producing the S star S or S plus S. So, this operator whatever that you are getting here star and plus we do not have any precedence here; we can produce this first or this first here and we have interchange. And because of left most derivation symbol either this rule or we have applied either of this rule to get 2 different left most derivations.

Now, if you insist on the operators so operator procedures; what is the operation that you have to apply first that is how in programming languages one consider that if I write a plus b star a; if you look at semantics of this the meaning of that. If you add a and b then you multiply this which what are the resultant value meaning it will defer from; when you take first multiple b and a then you add b, a, a plus b star a. So, this 2 meanings they defer; so that way in this syntax if can you take care of that situation.

That means 2 different expression, and 1 different expression if cannot different meaning if it has the same syntax then such an ambiguity can be avoided grammar; so, if can you give such grammar that means operator proceeding can be followed. But in the syntax we cannot say that this had to applied first, this had to be applied next; that you can do by introducing certain parenthesis. So, here you can produce the arithmetic expressions also; for example, using the same grammar you can produce a plus then within the bracket b star a. But at the operator precedence you should have to produce everywhere brackets. So, thus one can try to give a context grammar by choosing the concept operating precedence. But here you have introduced brackets to give an unambiguous grammar to produce arithmetic expression you can take it as an exercise.

(Refer Slide Time: 45:02)



exercise

Construct an unambiguous CFG that generates arithmetic expressions over {a, b} with the operations + and *.

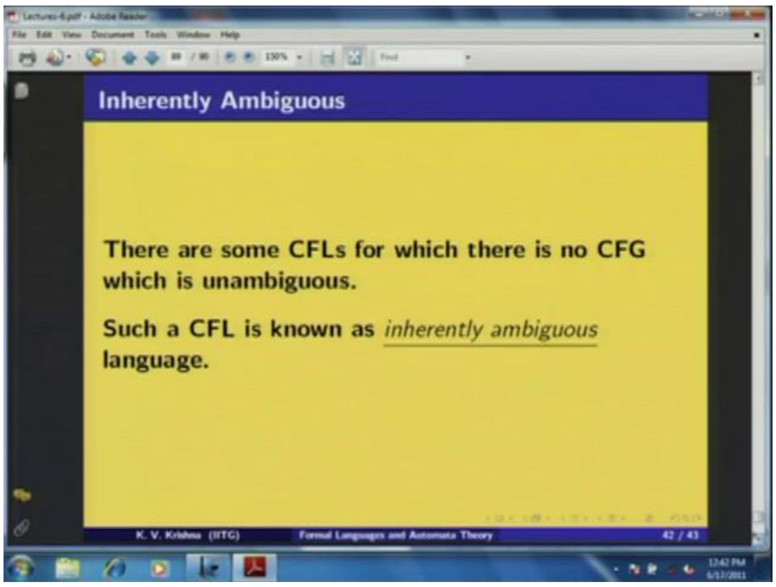
Regular expressions

Exercise Give a CFG that generates regular expressions over {a, b

Construct unambiguous CFG that produces, that generates arithmetic expressions over 2 elements sets a, b; of course, by the binary operations multiplication and addition. We can consider that is operations plus and star to try this exercise. And now you know already regular expression; the context of regular languages, this regular expressions are introduced to you they are similar to arithmetic expression. You can give a context grammar that produces regular expression over some alphabet that are operation there are union, star and contention.

So, give a grammar, so another exercise is that generates regular expressions. The generation regular expression over from alphabet may be consider same a, b. And in this context we can look for a CFG which is ambiguous. And further the look for a CFG that unambiguous grammar to produce regular expression for both the versions if possible. Form the versions ambiguous and unambiguous.

(Refer Slide Time: 47:37)



Inherently Ambiguous

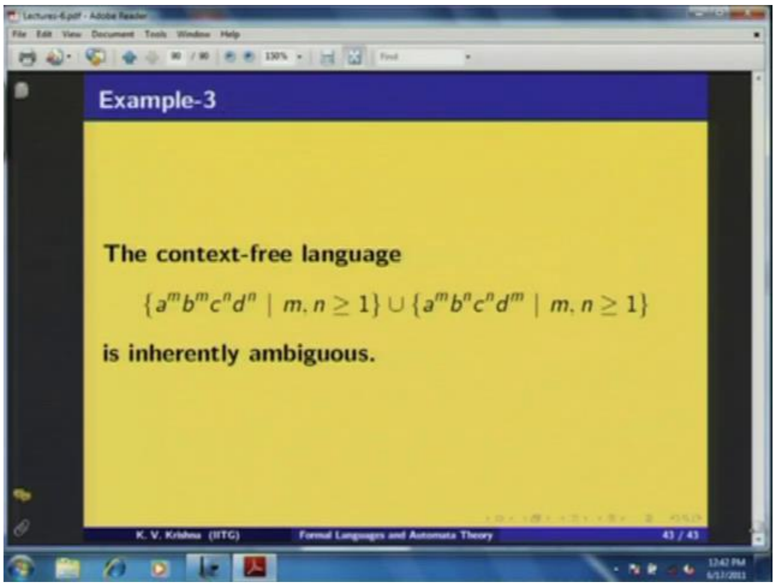
There are some CFLs for which there is no CFG which is unambiguous.

Such a CFL is known as *inherently ambiguous* **language.**

K. V. Krishna (IITG) Formal Languages and Automata Theory 42 / 43

In case arithmetic expression what do you understand we have an unambiguous grammar, but we can also produce an unambiguous grammar that gives arithmetic expression. But there are certain context languages for which we have to able any context grammar that is unambiguous. So, now that the unambiguous language is the one for which you cannot have un enormously CFG is called inherently language context languages.

(Refer Slide Time: 48:25)



Example-3

The context-free language

$$\{a^m b^m c^n d^n \mid m, n \geq 1\} \cup \{a^m b^n c^n d^m \mid m, n \geq 1\}$$

is inherently ambiguous.

K. V. Krishna (IITG) Formal Languages and Automata Theory 43 / 43

I give an example without proof this point of it is; if we consider the context language a power m, b power m, c power n and d power n; for m n greater than 1 union, a power m, b power m, c power n and d power n, for m n greater than 1. You can quickly understand that this is a context language; you can give a context grammar for this, because we have already discussed languages for this type. Now we have two components here; so, consider this component. And you know this concept discuss using the philosophy the production rules given for a power m, b power m are in beginning of this lecture. We can give production rules and context type to produce these components; similarly, this and understanding this is a context language.

And, whatever is the grammar is produce you can try and understanding that it is ambiguous grammar as well. Because I had stated here that is inherently ambiguous this languages is inherently ambiguous language. But proving that this is inherently ambiguous; we required little more details that further introducing. So, at that point of time will see why this is inherently ambiguous? But for the time being we just take this is an example and inherently ambiguous language.