

**Theory of Computation**  
**Prof. Kamala Krithivasan**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture No. # 05**  
**Simplification of CFG**


Today, we shall see how to remove useless symbols from a grammar. When you write a grammar, especially for a programming language you want to write a compiler and first step is to write a grammar, you write a grammar. Without knowing yourselves, you may end up with some useless symbols, because when you write a grammar for a compiler; for a language for that matter. It will not have three or four rules; it will have hundreds of rules, then when you write hundred of rules, without knowing yourself there may be some useless symbols which will not lead you to anything. So, how to find out which symbols are useless and how to get rid of them?

(Refer Slide Time: 00:53)

**Removal of Useless symbols**

Let  $G=(V,T,P,S)$  be a CFG. A symbol  $X$  is useful if there is a derivation  $S \Rightarrow \alpha X \beta \Rightarrow w$  for some  $\alpha, \beta$  and  $w$ . Where  $w$  is in  $T^*$ . Otherwise  $X$  is useless.

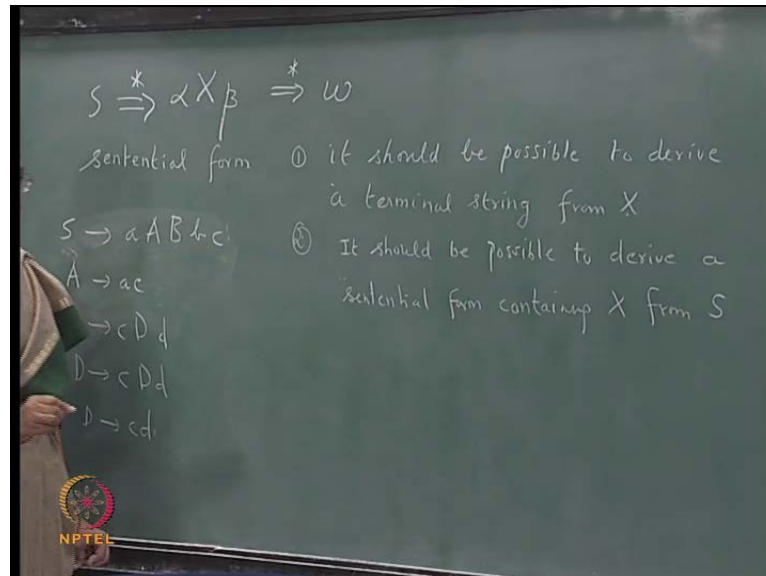
Lemma : Given a CFG  $G=(V,T,P,S)$ , with  $L(G) \neq \emptyset$ . we can effectively find an equivalent CFG  $G'=(V',T,P',S)$  such that for each  $A$  in  $V'$  there is some  $w$  in  $T^*$  for which  $A \Rightarrow w$



So, first let us see what is a useful symbol and what is a useless symbol? If you consider the grammar  $V, T, P, S$ ;  $V$  you can take as a non-terminal alphabet,  $T$  as a terminal

alphabet,  $P$  is a set of productions,  $S$  is a set of start symbol, a symbol  $X$  is useful if there is a derivation,  $S$  derives  $\alpha X \beta$  derives  $w$ .

(Refer Slide Time: 01:27)



That is a symbol  $X$  is useful if you can have a derivation of the form,  $S$  derives  $\alpha X \beta$  derives  $w$ , where  $w$  is a terminal string. So, from  $S$  it should be possible to derive a sentential form, this is called a sentential form. So, from  $S$  it should be possible to derive a sentential form which contains  $X$ , and ultimately that would also lead you to a terminal string. If this is satisfied, then the symbol  $X$  is useful, if this condition is not satisfied then the symbol is useless.

Now, if you look carefully at the definition, there are two aspects; one is, there should be a terminal portion derivable from  $X$ , it should be possible to derive a terminal string from  $X$ , this is one condition. The second condition is; it should be possible to derive a sentential form containing  $X$  from  $r$ . In other words, sometimes it is also put in as  $X$  is reachable from  $S$ .  **$X$  is reachable from  $S$**  There are two conditions, are they necessary or sufficient, these two conditions are necessary condition, but **they are not sufficient** they are not sufficient conditions, why? See, if  $X$  is a useful symbol; then from  $X$  it should be possible to derive a terminal portion of this  $w$  from  $x$ . So, it should be possible to derive a terminal string from  $X$ , and from  $S$  it must be able to reach.

But, these two conditions if you use only two of them, it may be see suppose I have something like this. I have a sentential form,  $\alpha A B \beta$  derivable from  $S$ , and this is

the only sentential form in which A will occur starting from S, suppose it is that A. And then from A it is possible to derive a terminal string X, both conditions are satisfied. But it may so happen that from B you may not be able to derive a terminal string, if you are not able to derive a terminal string from B, then even though S A is reachable and you can derive a terminal string from A, A is useless. Or, instead of putting start I will write like this, suppose I have a rule; I have a grammar in which I have a rule S goes to a A B b c, then A goes to a c.

Then some other rules, S goes to c D d, some other rules are there. Now, is the symbol A useful or not? It is possible to get a sentential form containing A from S, it is also possible to derive a terminal string from A, but if you use this rule and then replace A by a c still B will be, there is no rule with B on the left hand side. It is not possible to derive a terminal string from b. So, if a symbol is useful, then it should be possible to derive a terminal string from X, and it should also be possible to derive a sentential form containing X from S, but these two conditions in put together will not imply that the symbol is useful. So, they are necessary conditions, but they are not sufficient conditions.

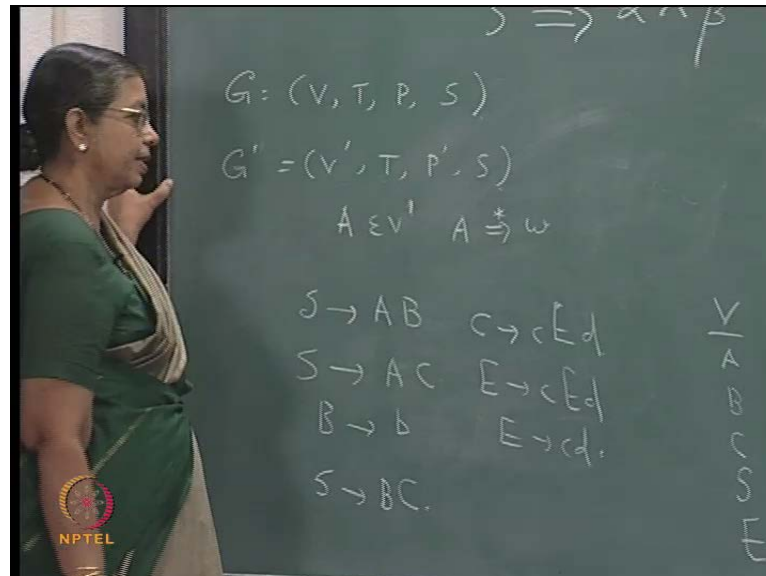
So, with this idea let us see how to remove the useless non-terminals, especially most of the time terminal symbols you would not so much bother about usefulness or not, they will be there altogether. But there may be non-terminals which are really useless and you may want to get rid of them. And any time you write a grammar, it is better to write a precise grammar, in the sense that; you try to minimise a number of production rules, try to minimise a number of a non-terminals and so on, but at the same time for example, you may have an ambiguous grammar if you use only one non-terminal, but if you use two non-terminals it may become unambiguous.

So, for some other reason you may want increase the number of non-terminals or production rules by a small factor. So, there'll be a trade off between certain things, but generally we do not want to have any useless symbols. So, first let us see one lemma, given a context free grammar  $G, V, T, P, S$ ;  $V$  is a set of non-terminals,  $T$  is the set of terminals,  $P$  is the set of production rules,  $S$  is the start symbol, with  $L(G) \neq \phi$ .

We will assume that the language generated is not empty, this is the with the loss of generality we can assume that, we can effectively find an equivalent context free

grammar  $G$  equal to  $V$  dash  $T$   $P$  dash  $S$ , such that for every  $A$  and  $B$  dash, there is some  $w$  in  $T$  start for which  $A$  derives  $w$ .

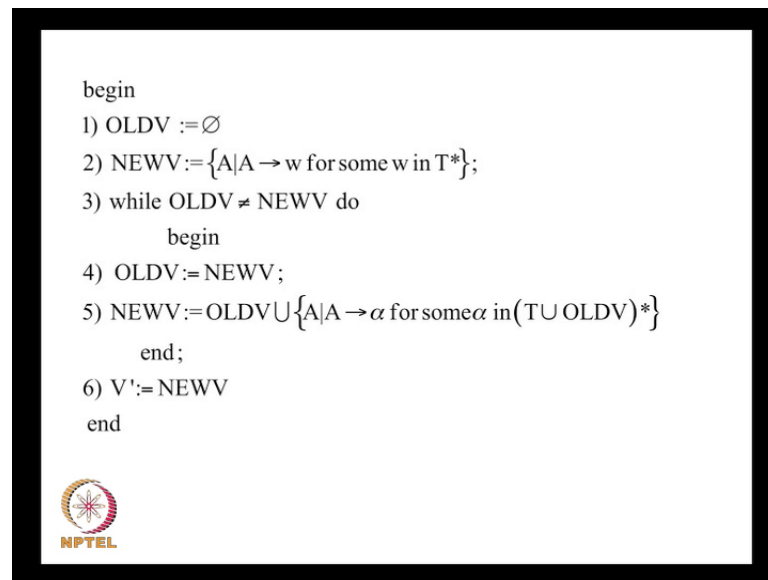
(Refer Slide Time: 08:42)



That is given a grammar the first step we take is like this. Given a grammar  $G$  is equal to  $V, T, P, S$ , you construct a grammar  $G$  dash equal to  $V$  dash  $T$   $P$  dash  $S$ . Such that, for this grammar the language generated for both are the same, it is an equivalent grammar, but in this grammar any non-terminal  $A$  belonging to  $V$  dash will have the property that, it should be possible to derive a terminal string  $w$  from  $A$ . That is what you are in the sense trying to do is, in this grammar if there are non-terminals from which terminal strings are not derivable, then you remove them.

You remove the non-terminals from which terminal strings are not derivable; in this case the terminals are not really affected. So, we use the same  $t$ , but  $V$  dash will be a subset of  $V$ , and  $P$  dash will be a subset of  $P$ . You just remove some non-terminals and productions involving them, so that the resultant grammar you have the property that every non-terminal derives a terminal string.

(Refer Slide Time: 10:02)



The steps are very simple; you use this algorithm to find the set of non-terminals. So initially, you look at rules of the form  $A \rightarrow w$  where  $w$  is a terminal string. Try to form  $V'$  like this, look at the productions in  $P$  and look at symbols where the right hand side is a terminal one, put those non-terminals here. Then look at those rules, where the right hand side will consist of terminal symbols and symbols already present there; that means, from each one of them it is possible to derive a terminal string. So, from this non-terminal it will be possible to derive a terminal string, so include that here.

Then again look at these symbols, see the right hand side is made up of terminals and these symbols, and then if the left hand side is different from what is already there include that, keep on doing that until no more things can be added. That is what the algorithm says, old  $V$  is empty then new  $V$  is  $\{A \mid A \rightarrow w \text{ for some } w \text{ in } T^*\}$  start, and while old  $V$  is not equal to new  $V$ . To begin old  $V$  is equal to new  $V$ ; new  $V$  will be some more symbols added old  $V$  union  $\{A \mid A \rightarrow \alpha \text{ for some } \alpha \text{ in } T \cup \text{old } V\}$  start. Then when old  $V$  becomes equal to new  $V$ ,  $V'$  will consist of that.

So, let us take a very simple example, and see its very simple idea. So, start with  $S \rightarrow AC$ ,  $B \rightarrow b$ ,  $C \rightarrow \dots$  Some grammar is there like this which has got non-terminals,  $V$  has the symbols;  $V$  has  $A B C$ , not  $A B C$  it is  $S$ .  $V$  has  $S A B C E$ , these are the non-

terminals. We will have the following non-terminals, how will you construct? First, look at the rules at the right hand side, small a b c d are terminals since b c d they are terminals symbols, non-terminals are capital A B C S E. The language generated is nonempty, because S goes to B C, from B we can derive, from C you can derive C E d and then E c d. So, it is at least B C square d square you can derive from this, so it is a nonempty language.

Now, let us see how to find the useless non-terminal and remove it? First, look at the rules where you have only symbols from the terminals. So, here you have only b, so B has to be included here, and here you having just c d, so E has to be included here. So, the first step V dash it is of course, in the algorithm it is put as new V and old V, so I will put as new V and old V. Initially, old V is empty and new V consists of these two symbols, then by the algorithm old V is becomes new V. So, B E, and new V is what you already have, and what symbols you are going to add? If the right hand side consists of terminals and B and E, if the right hand side consists of terminal symbols and B or E then you will include the left hand side.

So, here look at this, this has terminals and E alone, E is on the left hand side, but E is already there. Here look at the rules, c E d, E is already there, c and d are terminal symbols, so include C here, so the next step C will be included. And next stage old V becomes equal to new V, and new V; try to find out whether there is a right hand side which involves terminals and B E C. Look at this one; this has B and C on the right hand side, so S is on the left hand side, so S will be added here. Now, after adding S, again next iteration S old V will have S and new V will have this, and just find out whether any more symbols can be added? You cannot add anymore symbol now, because from A it is not A is not on the left hand side.

So, it is not possible to have A. A is not on the left hand side, so it is not possible to derive any symbol from A. Even, if you have something like this a goes to A some a A, it is not leading to a terminal string, so A will not be included. So, at this stage old V becomes equal to new v, so these are the symbols. So, the new grammar you remove the non-terminal which does not lead to a terminal string, here A does not lead to a terminal string, so remove A, and the rules involving A on the left or on the right. So, remove this rule, remove this rule, remove this rule, remove all the rules involving A is a symbol from which you cannot derive a terminal string, so you have to remove that symbol and


all the rules involving A. So,  $V'$  is this, **this is  $V'$**  it is a subset of  $V$ ,  $V$  has all these symbols but  $A$  is useless so you remove  $A$ ,  $V'$  consists of the rest of the symbols. And originally  $P$  had so many rules, you remove those rules involving  $A$ .

So, like this from a grammar  $G$ , see you must note that the language generated is not affected by this; you are only removing the symbol which is not leading to a terminal thing. So, the language generated is not affected,  $L_G$  is equivalent to  $L_{G'}$ . So, from a grammar  $G$  you are removing symbols which do not lead to terminal symbols, and get a grammar  $G'$ . Here you must note that  $V'$  is a subset of  $V$ ,  $P'$  is a subset of  $P$ . So, we have taken care about the first condition; let us say about the second condition. It should be possible to derive a sentential form containing the symbol from the start symbol otherwise, it is a useless symbol.

(Refer Slide Time: 18:49)

Lemma : Given a CFG  $G=(V,T,P,S)$  we can effectively find an equivalent CFG  $G'=(V',T',P',S)$  such that for each  $X$  in  $V' \cup T'$  there exist  $\alpha$  and  $\beta \in (V' \cup T')^*$  for which  $S \Rightarrow \alpha X \beta$

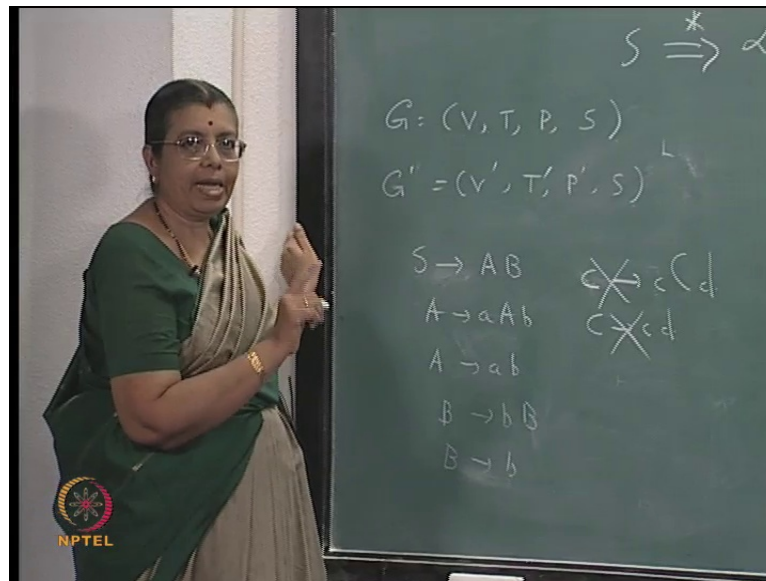
Theorem: Every nonempty CFL is generated by a CFG with no useless symbols.



So, the second portion is like this, given a context free grammar  $G$  is equal to  $V T P S$ , we can effectively find an equivalent context free grammar  $G'$  equal to  $V' T' P' S'$ , such that for each  $X$  and  $V' \cup T'$  there exists  $\alpha$  and  $\beta$ ,  $V' \cup T'$  start for which from  $S$  it is possible to derive  $\alpha X \beta$ .

So, we are looking into the second condition now, given a context free grammar  $V, T, P, S$ ; we are trying to find a grammar from where? **Yes**, it is possible to derive every symbol.

(Refer Slide Time: 19:56)



So, let us see how to do that? Now, you note that we have  $T'$ ,  $T'$  may be a subset of  $T$ . First, let me explain with an example, consider this example  $S$  goes to  $AB$ ,  $A$  goes to  $aB$ . A grammar something like this, here what are the non-terminals? The non-terminals here are  $S, A, B, C$ , the terminal symbols are  $a, b, c, d$ . Now, from this we want to construct another grammar  $G'$ , where the non-terminals will be a subset of  $V$ , terminals will be a subset  $T'$  will be a subset of  $T$ , such that from  $S$  it should be possible to get a sentential form in which each one of them are correct.

Now, start with  $V'$  and  $T'$  like this,  **$V'$  and  $T'$**  first put  $S$  in this **first put  $S$  in this**, then look at the rules with  $S$  on the left hand side, and look at the terminal symbols and non-terminal symbols occurring on the right, here  $S$  goes to  $AB$ . So, this  $A, B$  are non-terminals you include them. Usually, we denote capital letters for non-terminals and small letters for terminals. So,  $A, B$  are included in  $V'$ , then look at the rules with  $A$  on the left hand side and  $B$  on the left hand side, look at the right hand side. On the right hand side again you are getting  $A$  and  $B$  capital letters. So, non-terminals are  $A$  and  $B$  which are already there, but terminal symbols  $a$  and  $b$  are occurring here, so you will be adding  $a$  and  $b$ .

Now, look at the left hand side again, there are no more symbols non-terminal symbols added. So, no more rules you need to consider, you find that  $C$  is not reachable from  $S$ , from  $S$  it is not possible to get a sentential form containing  $C$ . So, the rule is useless

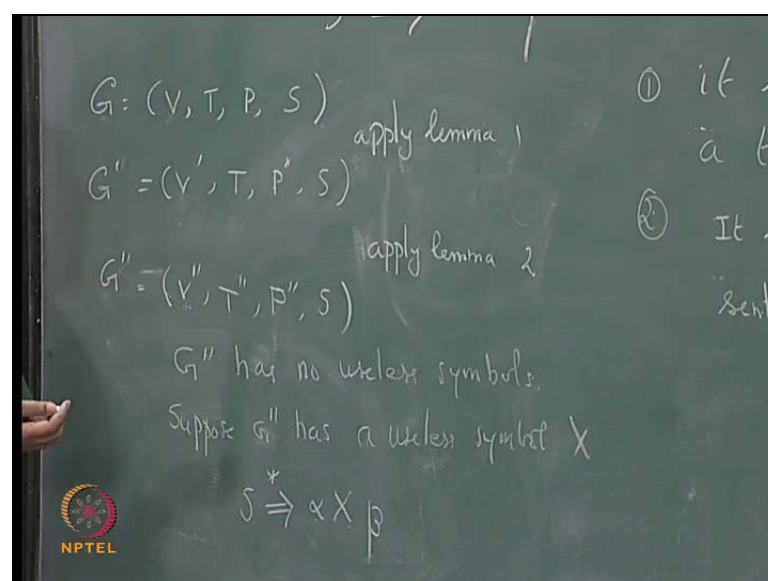


unless you from the start symbol you are able to reach that non-terminal, that non-terminal is not going to contribute anything. The language generated consists of strings derivable, terminal strings derivable from the start symbol. So, here from the start symbol it is not possible to reach C, so C is not going to contribute anything to the language generated by the grammar. And it involves rules small symbols c and d terminal symbol; they are also not going to be in the language generated.

So, c and d are useless, C is the useless non-terminal. The terminal symbols small c and small d they are useless, and the capital C which is a non-terminal is useless. So, V dash will consist of only S A B and terminal symbols are only a and b, and these productions are involving C and c d are removed. The useless symbols are removed, and the productions involving them either on the left or on the right are removed. So, V dash is a subset of V, T dash is a subset of T, and P dash is a subset of P. You are not adding any non-terminal at any stage please remember that, you are only removing some of them.

So, the next thing is every nonempty context free language, you can generate by a context free grammar with no useless symbol, so you can see that both conditions should be satisfied. That is, from every non-terminal it should be possible to derive a terminal string, and every symbol should occur in a sentential form starting from the start symbol. How do you do that?

(Refer Slide Time: 25:03)



We consider two lemmas actually, so start with the context free grammar  $G$  is equal to  $(V, T, P, S)$ . Then consider the first lemma, apply lemma one; that is, you make sure that from every non-terminal it is possible to derive a terminal string, then you get a grammar  $(V', T', P', S')$ .

These two grammars are equivalent, but from this grammar from every non-terminal it is possible to derive a terminal string. Then from this apply lemma 2, you get something like  $(V'', T'', P'', S'')$ , but from this grammar every symbol in  $V''$  or  $T''$  is reachable from  $S''$ , starting from  $S''$  it should be possible to get a sentential form containing every symbol in  $V''$  and  $T''$ . We must remember that  $V'$  is a subset of  $V$ ,  $V''$  is a subset of  $V'$ . So obviously, by transitivity  $V''$  will be a subset of  $V$ , and here  $T$  and  $T'$  are equal, here  $T'$  is a subset of  $T$ ,  $T''$  is a subset of  $T'$ , and  $P''$  is a subset of  $P'$ , and  $P'$  itself is a subset of  $P$  this you must. So, what you are doing is, starting with the grammar  $(V, T, P, S)$  first you are applying lemma 1 to see that every non-terminal derives a terminal string, then you are applying lemma 2 to show that every symbol is reachable from  $S$ .

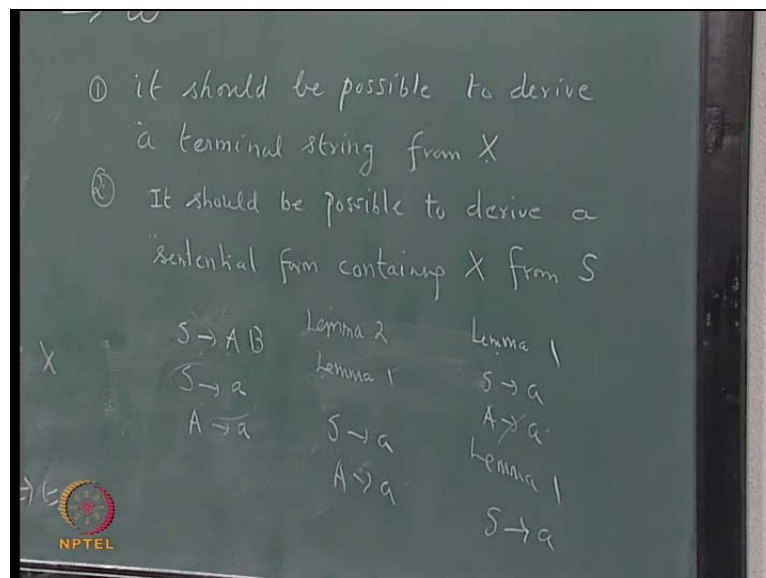
Now, you claim that this  $G'$  has no useless symbols,  **$G'$  has no useless symbols** we will prove reductio absurdum proof. Suppose,  $G'$  has a useless symbol  $S'$ , **suppose it has a useless symbol  $X'$** , then  $G'$  is obtained from  $G$  by applying lemma 2, so every symbol here is reachable from  $S'$ . So,  $G'$  has that property, so from  $S'$  it should be possible to get something like this. From  $S'$  it should be possible to get  $X'$ , then only that is how you have obtained  $G'$ . Now, every symbol here either non-terminal or terminal, especially non-terminals, whatever non-terminals are occurring in  $\alpha$  or  $\beta$  they are in  $V'$ , because  $V''$  is a subset of  $V'$ , all symbols in  $V''$  or in  $V'$  also. So here, the symbols occurring will be from  $V''$  and  $T''$  only, so whatever non-terminal is appearing here that will be in  $V'$  and in  $G$  what is the property?

Every non-terminals derives a terminal string. So, it should be possible to derive a terminal string from every one of this, and  $P''$  is subset of  $P'$ ,  $V''$  is subset of  $V'$ ,  $V'$  is a subset of  $P$  and so on. So, from every symbol here it should be possible to derive a terminal string. So, you are able to get a derivation like this which means that  $X'$  is useful and it is not useless as you assumed. So, therefore  $X'$  is useful,  **$X'$  is useful**. So this assumption is not correct, so  $G'$  do not have useless symbols. So,

from a context free grammar you may be able to remove the useless productions and the useless non-terminals. So, at present it may not be very I mean, you may think why it is; why you should do it and all that. We will convert something to Griebach normal form and Chomsky normal form, and when we do that the number of productions increase, number of new non-terminals will be introduced. Ultimately, we will find that some are useless; you can remove some of them.

Now, here note that starting from  $V$ , we have applied lemma 1 to get  $G$  prime, and we have applied lemma 2 to get  $G$  2 dash. These lemmas should be applied in that order only, you cannot apply lemma to first and then lemma 1 next, you cannot change the order.

(Refer Slide Time: 31:32)



What happens if you change the order? You may not be able to remove all the useless productions for example, consider this grammar  $S$  goes to  $A B$ ,  $S$  goes to  $a$ ,  $A$  goes to  $a$ . A grammar which has got three non-terminals;  $S$  goes to capital  $A$  and capital  $B$ , they are non-terminals. Three non-terminals are  $S$ ,  $A$  and  $B$ , only one terminals small  $a$ . What is the language generated by this grammar? The language generated is only one string  $a$ , that is only  $L G$  consists of only one string  $a$ . So, it is enough if we have this rule alone, it is not necessary to have the other two.

So, let us see what happens when you apply lemma one first and lemma two next, and lemma two first and lemma one, what happens? First apply lemma 2, that is from  $S$


include symbols which are reachable from S, A B is reachable so you will keep this, a is reachable, a is reachable, so you will not remove anything. If you apply lemma 2 first, everything is reachable from S, so you will not remove anything **right**. Then apply lemma 1, which are the symbols which lead you to terminal strings? S and A lead you to terminal strings, so this will be removed. So, you will end up with S and A. If you apply lemma 2 first and lemma 2 first second, if you apply lemma 2 first and then lemma 1 next; that is, this condition first and this condition, you will end up with a grammar having two rules; S goes to a, A goes to a, but A goes to a is useless is not it? Capital A is also useless, but that will not come out.

What will happen if you apply lemma 1 first and then lemma 2? If you apply lemma 1 first, from every symbol it should be possible to derive a terminal string. You can see that from S and A you can derive, from B you cannot derive. So, this rule and B has to be removed, this rule has to be removed B has to be removed. So, at the end you get S goes to a, A goes to a after applying lemma 1. Then you apply lemma 2; that is, every sentential form, every symbol should occur in a sentential form reachable from S. You see that capital A you cannot reach from S now, **you cannot reach capital a from S**, so this has to be removed, so you end up with just S goes to a, only one rule. So, applying these two rules in this order is these two lemmas in this order helps, the other way round it does not help sometimes. Sometimes it may work out cannot say. So, now we have seen how to remove a useless symbol from a context free grammar.

(Refer Slide Time: 35:45)

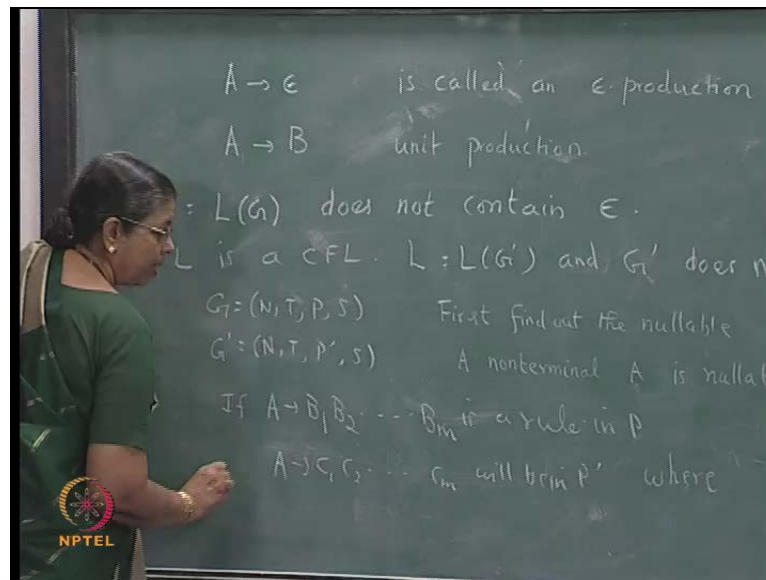
Removal of  $\epsilon$  and Unit Productions :

Theorem : If  $L=L(G)$  for some CFG  $G=(V,T,P,S)$  , then  $L-\{\epsilon\}$  is  $L(G')$  for a CFG  $G'$  with no useless symbols or  $\epsilon$  - productions.



Next is removal of epsilon productions and unit productions. We want to remove epsilon production. A rule of the form  $A \rightarrow \epsilon$  is called an epsilon production **is called an epsilon production**. A rule of the form, a non-terminal going into another non-terminal is called a unit production.

(Refer Slide Time: 36:00)



You would like to avoid such productions,  $A \rightarrow \epsilon$ ,  $A \rightarrow B$  you would like to **...** Why should you avoid such productions? First of all that length increasing property, length of the left hand side is less than or equal to length of the right hand side, it is violated by rules this form.

So, in proofs when this property is violated, the proof becomes lengthy and so on in many cases. So, if you avoid such rules, the proof you can give very easily, that is one idea. The second thing is, why do you want to avoid unit productions,  $A \rightarrow B$  non-terminals? In fact, in some of the examples, like when you wanted to find a power  $n$ ,  $b$  power  $n$ ,  $c$  power  $m$  union,  $a$  power  $n$ ,  $b$  power  $m$ ,  $c$  power  $m$ . The rules we wrote was  $S$  goes to  $S_1$ ,  $S$  goes to  $S_2$ , and from  $S_1$  you derive this, from  $S_2$  you derive this and so on. Is not it? So, such unit productions were easy to write, but why would you avoid them, why do not you why should you not?

The reason for this avoiding unit production is like this. In the compiler every time when the parsing takes place, the reduction either bottom up or top down it is taking place. So, when you find that when you reduce using a rule, a semantic rule will be semantic

routine will be called and a code will be generated. So, if these unit productions really do not do anything, like you had  $E$  is equal to  $E$  plus  $E$  like that. So, these unit productions really do not contribute to any proper code but unnecessarily you will be using them and each time you will be calling a semantic routine and so on. So, if you use too many of that unit productions  $A$  goes to  $B$ , the compile time will be more, compiling of a program will be more.

But, in order to avoid that we will see how to avoid the unit productions and how to avoid the epsilon productions. When you avoid the unit productions the number of rules may increase, but the number of rules increases very much, then also its a disadvantage. At each stage you may have to check which rule is applicable for reduction and so on, there are too many choices and so on, then that time will be more. So, the idea is to reduce the compile time by removing the unit production if the number of rules increases too much, it depends on the example, particular example. If it is somewhat reasonable then you will avoid unit production, if it is not reasonable then it may rather be advantageous to use the unit production, but anyway unit productions can be avoided, epsilon productions can be avoided and so on.

But, one thing you must remember, if you avoid epsilon productions, the language will not have epsilon, the language generated will not have epsilon. If you want to have epsilon, you must have at least one rule  $S$  goes to epsilon, if the language consists of epsilon you must at least have one rule having  $S$  on the left hand side epsilon on the right hand side, and you must also make sure that  $S$  does not occur on the right hand side of the unit production, in order to add that.

So, at this stage we will assume that  $L(G)$  does not contain epsilon. You have  $L(G)$ , you have a language which does not contain epsilon, if it contains epsilon we have to do some modification as I told you. Now,  $L(G)$ ... so  $L$  is CFL, and  $L$  does not contain epsilon, then you can see that  $L$  can be generated. If  $L$  is equal to  $L(G)$  for some CFG  $G$ , then  $L$  minus epsilon is  $L(G)$  dash, that is; all strings in  $L$  except epsilon can be generated by a grammar  $G$  dash with no useless symbols or epsilon productions. So, you have to remove the  $L(G)$ ...  $(( ))$  without loss of generality we assume that  $L$  does not contain epsilon, and we have to show that  $L$  is equal to  $L(G)$  dash and  $G$  dash does not have epsilon productions.

Now, how do you go about doing this? G is equal to say N, T, P, S, then G dash is same N T, but rules will change. How do we construct G dash? First, find out the nullable non-terminals **first find out the nullable non-terminals**. What is a nullable non-terminal? A non-terminal A is said to be nullable, if it is possible to derive epsilon from that non-terminal. There is a difference between considering this rule and that for example, from A you may not have a rule like this, but we may have something like A goes to B, C, and then from B goes to epsilon you may have, C goes to epsilon you may have, you may not have A goes to epsilon. In that case B is null able, C is null able, A even though on the right hand side you did not have epsilon, A is also nullable because we are able to derive epsilon from A, we are able to derive the empty string from A, so in this case A, B, C are all nullable.

How can you find out the nullable non-terminals? First, look at the non-terminals like this which are of this form, again you include you put nullable like this, initially it will be empty. Then look at the rules where on the right hand side you have epsilon, include the non-terminals. After doing that look at the rules where on the right hand side you have symbols consisting of the symbols already occurring here, then from each one of them you can derive epsilon. So, you will include the left hand side, then again repeat the process until no more symbols can be added; for example, you may have something like this, you may have a grammar having number of rules, in that I have A goes to BD, D goes to BC, C goes to epsilon, B goes to epsilon, I may be having like this.

So, null able non-terminals when you find, you have right hand side epsilon for B and C, so first you will include B and C, then look at the right hand side which is having only B and C, this B and C is occurring. So, from D you can drive BC, and from B and C again you can derive epsilon, so from D you can derive epsilon, so you will be adding D to this. Now, again look at this rule, you are having BD on the right hand side, B and D are already occurring here. So, from B and D it is possible to derive epsilon, so from A also it is possible to derive epsilon, so you have A. Keep on doing this until no more symbols are added, the procedure will ultimately stop anyway.

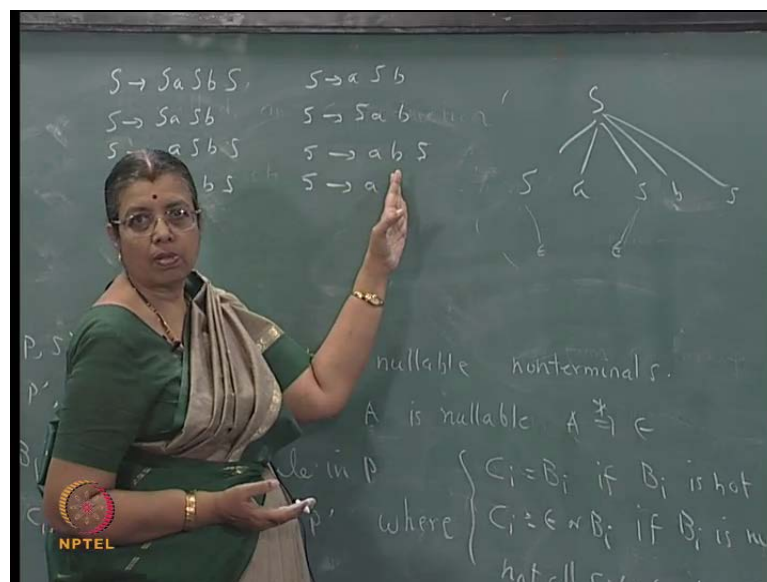
So, first find out all the null able non-terminals, then what do you do? G dash has the same non-terminals same terminals, only the production rules change **right**. So, how do you change the production rules? If A goes to... for each rule you consider like this. You

take one by one the rules in P, for one rule you may have several rules in P dash, the number of rules in P dash will increase.

So, take a rule like this in P, for this what sort of rules you will add in P dash? If this is a rule in P,  $A \rightarrow C_1 C_2 \dots C_m$  will be in P dash. It is not a single rule it will represent a set of rules. What is that? What is a set of rules? Where  $C_i$  is equal to  $B_i$  if  $B_i$  is not nullable, that is if it is a terminal symbol or a non-terminal which is not nullable, you keep it as it is. But if it is nullable, if  $B_i$  is nullable, if it is a nullable non-terminal you have two choices; either you keep it or you remove it, both possibilities you have to consider. But not all  $C_i$ 's can be epsilon. You cannot make all the  $C_i$ 's epsilon then; that means, right hand side is epsilon is not it? So, that possibility you should not consider.

So, what you do is; first, you find out the nullable non-terminals, then for the rules each one you take from P, and if the rule is of this form if it is a non-terminal which is not nullable you keep it, if it is a terminal you keep it, but if it is a nullable non-terminal you have two possibilities, either you can keep it or you can remove it.

(Refer Slide Time: 49:46)



Take for example the simple grammar, this generates epsilon also this is a Dyck set generator, generates the Dyck set well formed strings of parenthesis taking a as a left parenthesis, B as the right parenthesis, it generates epsilon also.



Now, the language without epsilon wants to generate avoiding epsilon rules. So, for this is avoided, for this rule  $S$  is nullable non-terminal, so for this rule how many rules you will have? You have each  $S$  can be present or not present. So, you will have  $S$  goes to  $S$  a  $S$  b  $S$  all are present, then two of them can be present one need not be present, two of them may be absent and all of them may be absent. So, in this one you can replace  $S$  by  $S$  or epsilon, so if I replace  $S$  by  $S$ , if I keep all the three  $S$ 's I get this, if I replace this  $S$  by epsilon I get this, if I replace this  $S$  by epsilon I get this, if I replace this  $S$  by epsilon I get this, if I replace two of them by epsilon I get that and so on. So, this one rule is replaced by so many rules and this rule is avoided.

Now, the resultant grammar this does not have epsilon on the right hand side, it does not have epsilon production, but you must feel convince that this generate the same language. How do you know that it generates the same language? You can give a step by step proof by induction on the number of steps of derivation and so on, formal proof can be given. Even in the other two lemmas, I have explained with examples, the concept was very clear, it was not very difficult. In this case slightly more difficult, you can use a formal proof using induction I will not go into that. But you must understand that, if you are applying a rule  $S$  goes to  $S$  a  $S$  b  $S$  at some stage in the derivation tree, and afterwards you are replacing this by  $S$  epsilon and so on, and this by epsilon say.

The result you can have by applying instead of doing this, you could have applied a b  $S$  right, you could have applied this rule. So, all possibilities once you apply this rule, in the next rule you may use epsilon in these three stages or you may not use at all, if you do not use at all you use this rule again you will get the same thing, but the next stage suppose you use epsilon in one or two places, instead of doing it in two steps like that you could have use one of these rules. So, the effect is the same, the language generated is the same or not changing this. So, given a context free grammar you can remove the epsilon productions, I shall consider the removal of unit production in the next class and you can remove the useless symbols.

But, you must note that, first the order in which you do is like this. First you remove the epsilon productions then you remove the unit production, because the removal of epsilon productions may introduce unit productions. So, remove the epsilon production first, then remove the unit productions, then apply lemma 1, then apply lemma 2; that is, lemma 1 is every non-terminal should derive a terminal string, lemma 2 is every symbol

is reachable from  $S$ . This is the order in which you have to do, so we will consider the removal of unit productions like this. But if you want to include epsilon in the language what you have to do is like this. If epsilon belongs to  $L(G)$ , then add the rule  $S \rightarrow \epsilon$  and make sure  $S$  does not occur on the right hand side of any production, this is the way  $((\epsilon))$ . So, removal of unit productions, and some normal forms like Chomsky normal form and greenback normal form, we shall consider in the next class.