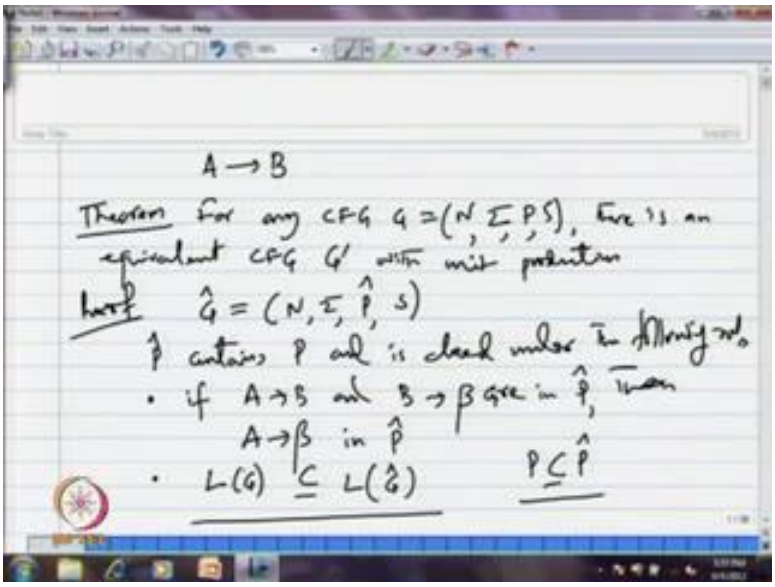
Formal Languages and Automata Theory
Prof. Diganta Goswami
Department of Computer and Engineering
Indian Institute of Technology, Guwahati

Module - 8
Simplification of CFGs
Lecture - 2
Normal Forms of CFG

In the previous lecture, we have seen how can we remove useless symbols or say unit production to simplify a grammar. In this lecture today we will first see how we can remove the unit production, a procedure is almost similar to that of your removal of Epsilon productions.

(Refer Slide Time: 00:44)

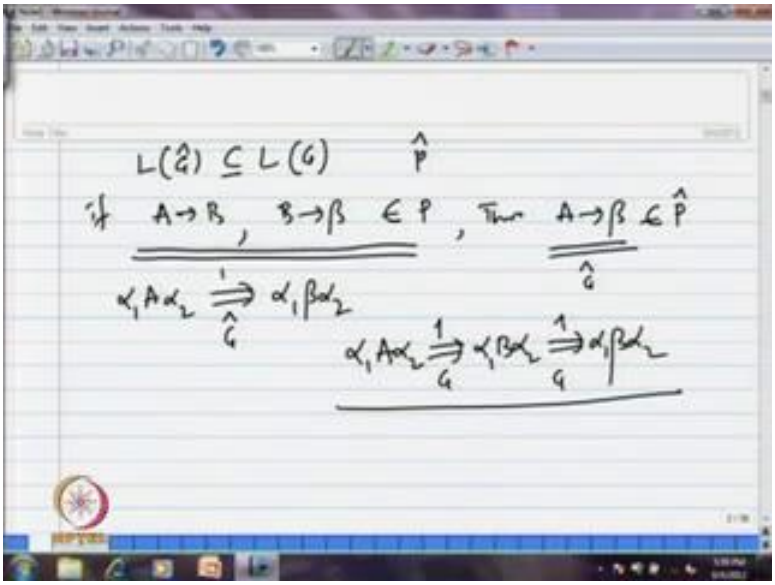


So, we said that a unit production is in the form A goes to B, where A and B are non-terminals. Now what we can do is that we say that for any CFG say G, which is N, sigma P, S. We can say that there is an equivalent CFG equivalent CFG say G dash, which has no unit production, we will prove this using similar approach that we did for removal of Epsilon production.

So, first we will construct a CFG say G hat from G we will construct a CFG set is G hat which is N, sigma, P hat, and S; that means, you are going to include some more productions. We say that P hat contains all the productions of P and it is closed under the

following rules, so the rule is that if A goes to B and B goes to beta are in P hat. Then we include A goes to beta as well in P hat; obviously, P hat will be finite, because your finite numbers of productions in P hat. Clearly you can say that L of G is a proportion of set from L of G hat, because P is a subset of P hat. So, L of G is a subset of L of G hat because P is a subset of P hat, so it is quite clear.

(Refer Slide Time: 03:17)



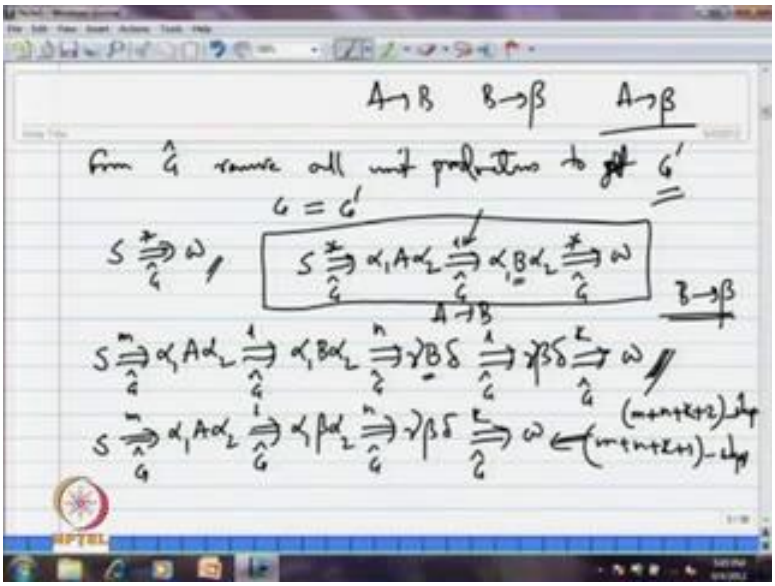
Now, we can also show that L of G hat is a subset of L of G, if you can show that you can show that G and G hat are equivalent. Now this is the case because every new rule that added to P hat is because of the above rules and that rules can be simulated using two steps. Because, we that said that if A goes to B and B goes to beta is in P or P hat say since everything is there in P will also be in P hat, so if these two rules are there in P, then A goes to beta is in P hat, so these are rules we used.

So, therefore if in the original grammar G, if we used these two rules, then or if you use this rule in grammar G hat in some derivation, then that can be simulated by using these two steps using these two productions in grammar G; that means, say alpha 1 a, alpha 2, suppose in G hat in one step. Suppose, we derive alpha 1 beta alpha 2, then this step can be simulated in G as like this say alpha 1 a, alpha 2 in one step in G, it will be alpha 1 b alpha 2 and in one more step in G alpha 1 beta alpha 2.

So, its need only one more step, so therefore, these two grammars are equivalent G and G hat are equivalent. Now, what we can do that you know similarly reason in that we

gave in case of say even follow of unit product Epsilon productions, we can say that a minimum length derivation in G hat really do not use any unit productions. And hence we can threw out all the unit production from the grammar G hat and to have a new grammar say G dash and which should be equivalent to original grammar G.

(Refer Slide Time: 05:48)



That means from G hat we remove from G hat we remove all unit productions unit productions to get a new CFG, G dash and eventually G and G dash will be equivalent. Now, what we want to show to that a minimum length derivation in G hat do not use any unit production, in that case you can throw out all unit productions from G hat to get the equivalent grammar G dash.

Suppose that a string w, we derive under G hat where w is string of terminals of only say there is a minimum length derivation in G hat and assume for contradiction that this minimum length derivation in G hat for w uses some unit production, that is this derivation will be like this. So, S derives under G hat say in zero or more steps it goes to alpha 1 a, alpha 2 of this form, then one step this A is replaced by B unit using the unit production alpha 1 b alpha 2.

And then in zero or more steps under G hat we derive w, so whatever set is at this point in this step we use the production A goes to B, this unit production used and we that said that this is a minimum length derivation. Now, we will arrive at a contradiction the point is that the occurrence of this non-terminal B, eventually must be removed disposed of by

using some production of the form say B goes to beta, the production of form must be used at some point to dispose of this non terminal B.

So, therefore, this derivation look like this, so from S in say n steps in G hat, where n is greater than or equal to zero, so this is alpha 1 a, alpha 2 say takes n steps, then in a 1 step under G hat it is replaced by B using the unit production. Then in another suppose n steps in under n steps say this alpha 1 alpha 2 are converted to some other string of terminals say gamma B delta and eventually in another one step in G hat this B is replaced by beta using this rule.

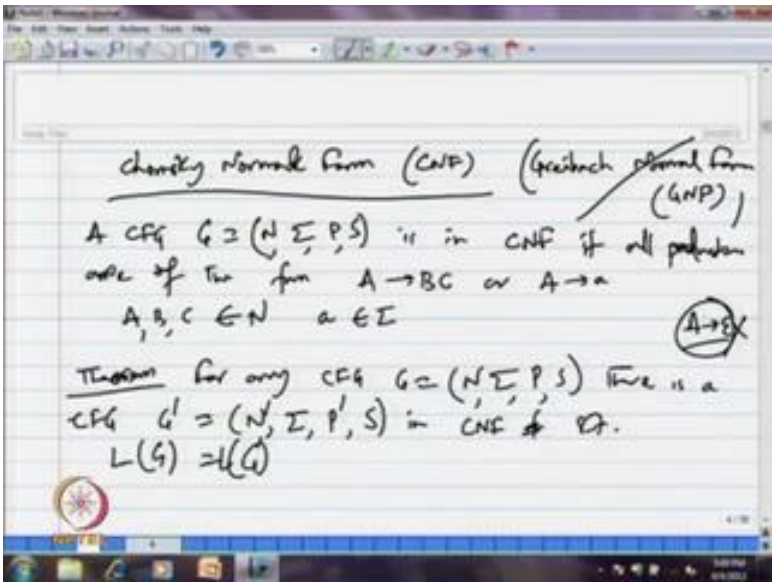
And eventually in another suppose k step under G hat we get the string a terminal say this is w, so it total wreckers m plus n plus k plus, two steps in grammar G hat and we said that this is a minimum length derivation. But according to our rule that we have followed that if a goes to beta A goes to B and B goes to beta is there in P hat and a goes to beta must also be here in P hat.

Now, what we can do is that we can use this steps starting with S in say m steps under G hat we will get alpha 1 a, alpha 2, then in another one step instead of replacing A by P, we can replace A by beta using these rules say this zero is also there. So, alpha 1 beta alpha 2 then in another m steps using the same steps, that we used earlier that will be gamma beta delta and in another k steps under G hat we will have w, so this derivation is exactly m plus n plus k plus 1 steps.

So, this takes m plus n plus k plus one step, so it takes one step less compare to the previous derivation, so our assumption that this previous derivation is a minimum length is false. So, therefore, we can easily throw out all the unit productions and can still have an equivalent grammar, by this way following this rule from G we can have G dash which is be equivalent to the original grammar G and we will not have any unit production.

Now, we will come to the discussion of normal form, we defined a normal form by imposing some restrictions on the form of productions allowed in the grammar. For many applications, it is often helpful to assume that CFG are in one or another special restriction form or normal form, we will consider took the most useful normal forms.

(Refer Slide Time: 12:06)

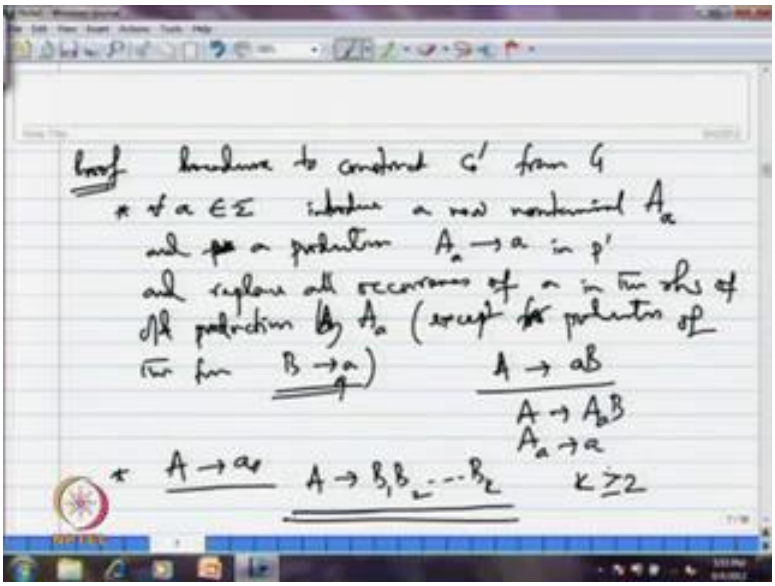


One is Chomsky normal form simply CNF and another is Greibach normal form Greibach normal form simply GNF, we will first discuss this Chomsky normal form and we will come to Greibach normal form that a later point. We say that first let us define this Chomsky normal form, we say that a CFG G N sigma P S is in Chomsky normal form or CNF. If all productions if all productions are of the form say A goes to B C or A goes to a where capital A, B, C are non-terminal S and small a is a terminal symbol.

Please note that we do not use productions of the form A goes to Epsilon and therefore, the grammar in CNF cannot generate the string Epsilon; that means, empty string cannot be generated since it not allowed productions of this form. Now, what we say is that in the form of theorem say for any CFG G N sigma P S, there is a CFG say G dash, which is and this sigma P dash and say S in CNF form or in CNF.

Such that L of G is equivalent or identical to L of G dash; that means, G and G are equivalent; that means, for every CFG G, we can find in CFG G dash which is in CNF and which is equivalent to G. Let us see how to get that CFG G dash which is equivalent to G and which in Chomsky normal form.

(Refer Slide Time: 15:05)



That means the proof of the theorem, so without loss of generality what we can assume that the given grammar G does not have any useless symbol, does not have any unit production or Epsilon productions. Even, if it has we can remove all those productions using the method that we have already discussed. Now, we use the following procedure to construct, to construct G dash from G.

The first is that for each terminal for each terminal symbol a belong to sigma, we introduce a new non-terminal say it is a subscript a for every small a terminal symbol, we introduce a new non terminal which is not there in the grammar G. And we introduce production A goes to a in P dash, so in P dash, we will include A a goes to a, where a is the new non-terminal and replace all occurrences of a in the right hand side of all productions.

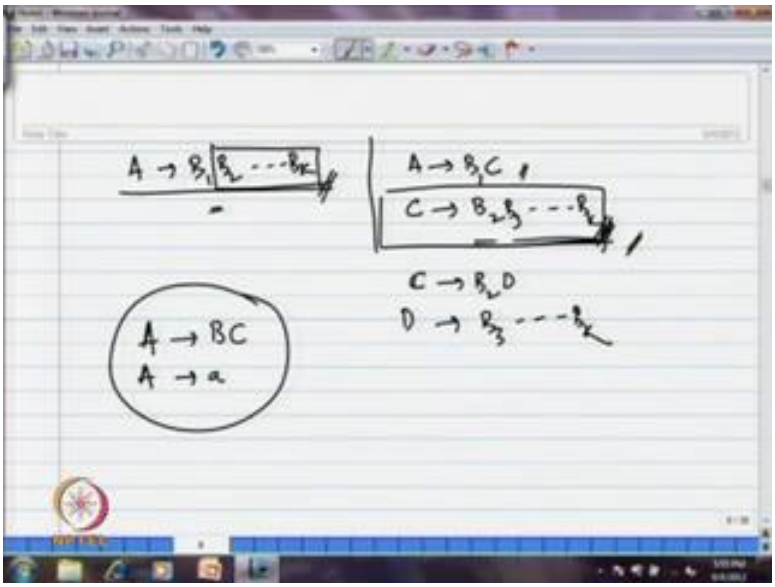
Whenever small a appears in the right hand side, when A production and then occurrence B replaced by that that occurrence we replaced by and replace all the occurrence of a in the right hand side of all productions by A a. Of course, except for the production of the form B goes to A, suppose if B goes to A is there with not talks the right hand side, but if otherwise suppose we have A productions of like this A goes to say a B, then this will be replaced by a goes to A a B and already we have included A a goes to a in the grammar.

So, this type of production with not replaced this A by a, that is what we have to say, so after this step all productions will be of the form A goes to a, where capital A is a non-

terminal and small a is a terminal symbol or it will be of the form. A goes to some B 1, B 2 up to say B k for some k greater than or equal to 2, so every production will be either of this form or this form, so it is quite clear from this construction.

And we have seen that by using this or by this construction we are not changing the language of the grammar; that means, G and G dash are equivalent, because in the new grammar whatever is generated in the old grammar any string. In the new grammar it will take one more step and whatever you can generate in this new grammar can also be generated in the old grammar.

(Refer Slide Time: 19:25)



Now, for all productions of the form a goes to say B 1, B 2 up to say B k say if we have a production of this form, so what we do we replace it by this production replaced by A goes to B 1, C. Where C is a new non-terminal that we have introduced in the grammar which are not there in the previous grammar and C goes to B 2 B 3 up to say B k; that means, they remain there of the string.

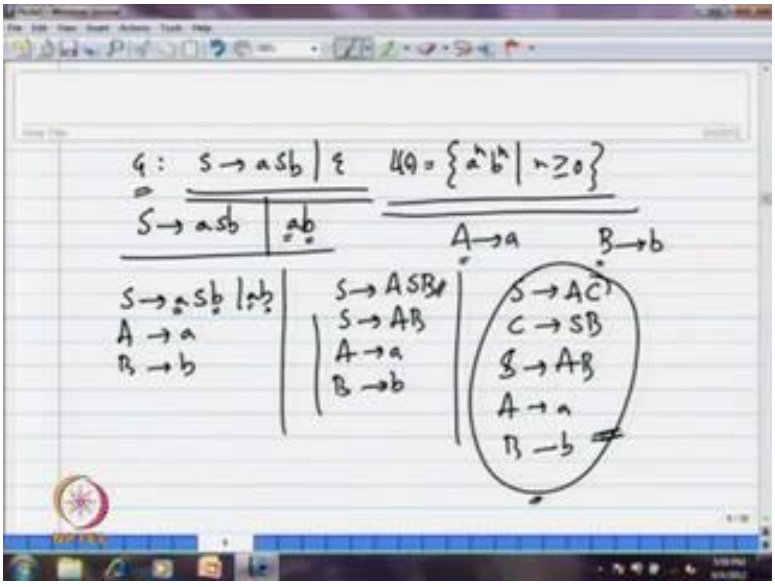
So, this part in this case also we are not changing the language of the grammar, so we have for thus this production we have introduced two new productions, so whatever is generated in this previous grammar can also be generated in this new grammar. But, it will take only one more step, similarly, I have written that we generated in this new grammar can also be generated in the old grammar.

So, this way we have reduced the length of the right hand side here by one non-terminal by one symbol, we continue this process that means, again we introduce a new non-terminal and from for this production. Again, you introduce a new non-terminal and you write it as say C goes to B 2 D and D goes to B 3 up to b k, where D is a new non-terminal and it is not going to sense the language of the grammar.

So, this way we keep on introducing a new non-terminal and until the length of the right hand side of each production becomes equal to 2; that means, in right hand side we will have only two non-terminals. So, eventually every production will be will be of the form A goes to B C eventually because eventually it has a terminal.

Since, the length of the right hand side is always this is finite it will be of from a goes to B C or it will be of from A goes to a. So, we will get only this kind of productions in the grammar and at that point it will be in Chomsky normal form just for example, we introduce or construct a grammar in CNF for a given language.

(Refer Slide Time: 21:51)



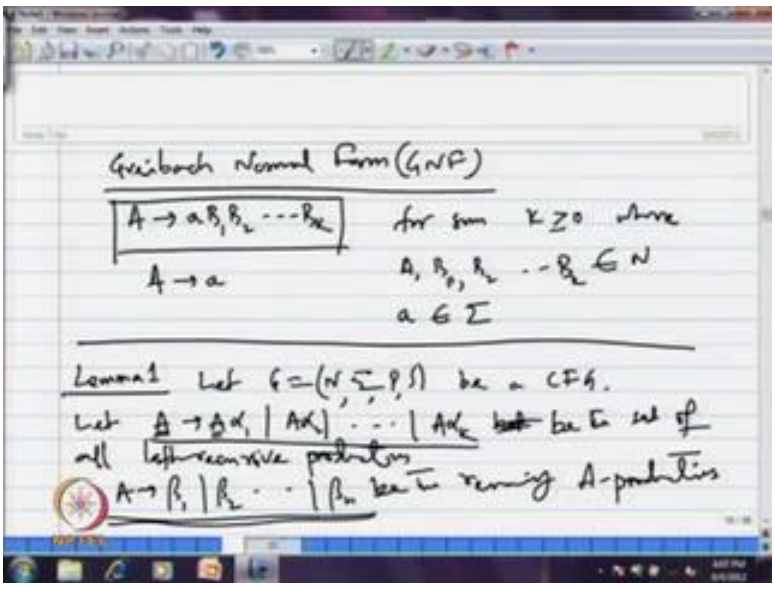
Say the set of G is given like this S goes to a S B Epsilon, this generates the language a triple n, b triple n, n greater than or equal to 0, so this is the language generated by G grammar. So, L of G, so is a set of strings numbers of is followed by same numbers of B, now since this contains an Epsilon productions, so we can remove this Epsilon productions by introducing is a S B or S goes to a b. So, we introduce a new production S goes to a b to remove this Epsilon production that we have already discussed.

Now, this is not in CNF because both of the productions S goes to a S B or S goes to a b is not in the form that to be in CNF, so what we do we introduce non terminal say should have been small B since written about B only. Now, we introduce two non-terminal S say A and B for each of the terminals over here, so A goes to a, we introduced in the grammar and B goes to b we introduced in the grammar. That means, the new grammar we will have the following rules S goes to a S B, a b, A goes to a and B goes to b.

Now, every occurrence of this a and b in the right hand side of production will be replaced by this non-terminal S, so from this now we get S goes to a S B, S goes to a b, A goes to a and B goes to a. Now, last three productions are in the form CNF, but this production is not in the form CNF, so corresponding to this we introduce a new non-terminal say C and we write it like this a S goes to A C. Then, C goes to S B, where C is a new non-terminal and we retain all the earlier productions A goes to A B goes to B.

Now, we see that in this every production obeys the rule that have given for productions of CNF and this is equivalent to the original one, but of course, this cannot generate the string Epsilon. Other in that, every string will be identical or same string can be generated; that means, generated in G as well therefore, this is a grammar which is in CNF for the given language A triple m, B triple m. Now, let us consider the other normal form which is called Greibach normal form.

(Refer Slide Time: 25:25)



First let us define Greibach normal form or GNF, we say that is CFG G is in GNF, if all productions in the grammar is order of the form A goes to A, B 1 B 2 up to B k for some k greater than or equal to 0, where A B 1 B 2 up to say b k. All these symbols are non-terminal S and symbol a is a terminal symbol; that means, every productions should be of this form, where the first symbol that appears in the right hand side is a terminal symbol.

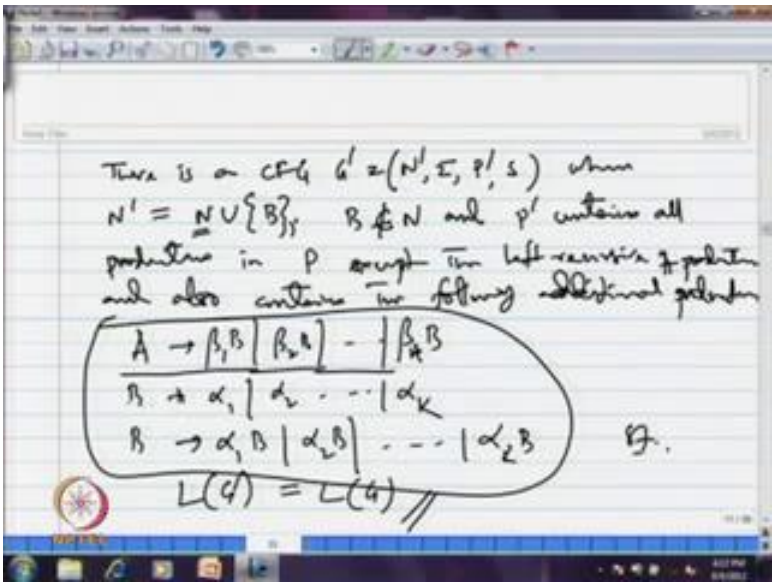
And the remaining all symbols it may be zero or more symbols there all the symbols should be non-terminal S symbol, so every productions should be of this form. So, here you see that k may be zero since k may be 0, therefore production of the form A goes to a is also adopted. On letting, we do not allow is the Epsilon production, similarly to your CNF, we will now show every CFG G can be transformed which equivalent CFG G dash.

So, now to do that what we do we first introduce two lemmas; that means, given any these lemmas will be helpful in constructing a CFG in GNF for any CFG G, so given any CFG G which contains some left recursive productions. We can always construct an equivalent CFG G dash say removing those left recursive productions by right recursive productions.

For, any grammar G if it contains some left recursive productions we can replace those by right recursive productions and that new grammar will be equivalent to the original grammar. we will first prove this we called it say lemma 1, let G equal to N sigma P S be a CFG and let A goes to A alpha 1, A alpha 2 like this. We have k numbers of left recursive productions say these are the set of all left recursive productions, this is called left recursive because this terminal non-terminal A appears in the left side.

So, which is there in the label appears the left side of the right hand side is called left recursive there are set of all left recursive productions and let of course, there are left recursive A productions, where A is the left hand side there maybe some other left recursive productions where it may be B productions or C productions and like that and let A goes to beta 1, beta 2 up to say beta N be the remaining A productions. So, there may be two kinds of a productions, this is the left recursive type and this is the remaining type. So, we assume that there are the remaining a productions that we have in the grammar.

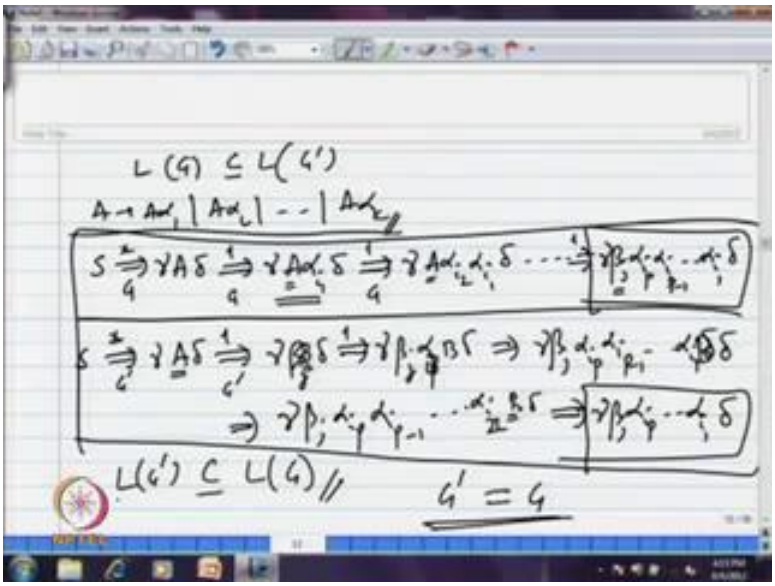
(Refer Slide Time: 29:51)



Then, there is a CFG G dash say it is N dash sigma P dash S, where N dash is N union B, we introduce a new non-terminal B which is not there in the previous set of non-terminals that means, B does not belong to N and P dash contains all productions in P except the left recursive productions. We will remove all left recursive A productions and also contains the following additional productions by removing left recursion A productions, we include the following additional productions.

So, A goes to beta 1, b beta 2, b up to say beta N, B goes to alpha 1, alpha 2 up to alpha k, this is n and B goes to alpha 1 B alpha 2 B and alpha k B, such that L of G dash is equal to L of G; that means, G and G dash are equivalent. So, we have removed all left recursive a productions and introduced a new non-terminal B and these new productions have been introduced in the grammar. Now, we can show that these two grammars are equivalent, we first show that L of G is a subset of L of G dash.

(Refer Slide Time: 32:29)



Now, what we can see is that A goes to A alpha 1, A alpha 2, A alpha k, there productions were there in the grammar G, which are not there in G dash, but if in the in the derivation in G if we use at any point these kind of productions which are not there in the G dash that can be simulated by a step of by few steps in G dash. So, whenever we used these kind of productions eventually this A goes to A alpha 1, A alpha 2 or in general if A goes to A alpha i 1 is used.

So, this A has to disposed of by using A productions of this form, some A goes to beta J, so therefore, the derivation will be like this say S goes to in G some gamma A delta in zero or more steps. Then, in one step in G, we replace A by A alpha i 1, suppose A alpha i 1 delta, then under one step in G, we replace this A by again say A alpha i 2 say gamma A alpha i 2, so alpha i 1 is there in the previous step delta.

So, continuing this eventually every time in one step we are using A productions, we will go up to alpha i 1, alpha i 2 up to say alpha i P and eventually this left side A has to be replaced by some beta J, so we say that gamma some beta J, then alpha i p minus 1 and so on. So, alpha i 1 and then delta, so this is how we need to proceed now this same thing can be derived in G dash as well like this say S goes to gamma A delta.

Now, instead of using A goes to A alpha I, because this kind of production we do not have in grammar G hat G dash, so we use in G dash, so in one step in G dash, we use a goes to beta J instead we use A goes to beta J, so gamma beta J delta. Now, in the next

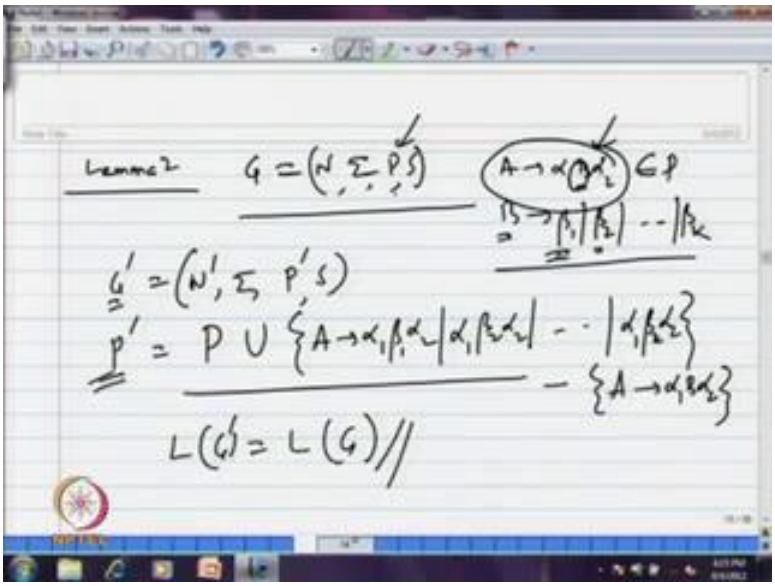
step we can use the production, that we have already added to that kind of production A goes to beta J, A goes to beta 1 B, beta 2 B and beta J B, so we will have beta J B,

So, in next step it will be we can use the production of the form where B goes to alpha 1, alpha 2 up to alpha k, we can use B goes to alpha P, when B goes to alpha 1 B, alpha 2 B like that so we can use gamma beta J, this B goes to say alpha P B delta. Then, next step again we can use B goes to alpha P minus 1 B and we can continue this like this beta J. So, you will have alpha or say this is alpha i P, alpha i P minus 1 up to say alpha i 1 and then B delta.

And then, eventually you will have gamma beta J alpha i P alpha i P minus 1 up to alpha i 1 B delta, then in one step we can replace this B by so you go up to say alpha i 2 and in one step we can replace it by B goes to alpha 1 to dispose of this B. So, it is gamma beta J alpha i P up to say alpha i 1 and then delta, so the same sentential form we have arrived at in this case also only thing is that the length of derivation of this case is more.

Similarly, any derivation in G is also a derivation in G dash because, this is a subset of G is a subset of G dash, now to show that L G dash is a sub set of L G, we need to follow just the reverse process that we have already given over here. By using the reverse process of this we can show that L G dash is a sub set of L G as well, that I can show that g dash and g are equivalent. So, even though we used this transformation to remove the left recursive productions and replace it by right recursive productions, the two grammars will be equivalent.

(Refer Slide Time:39:28)

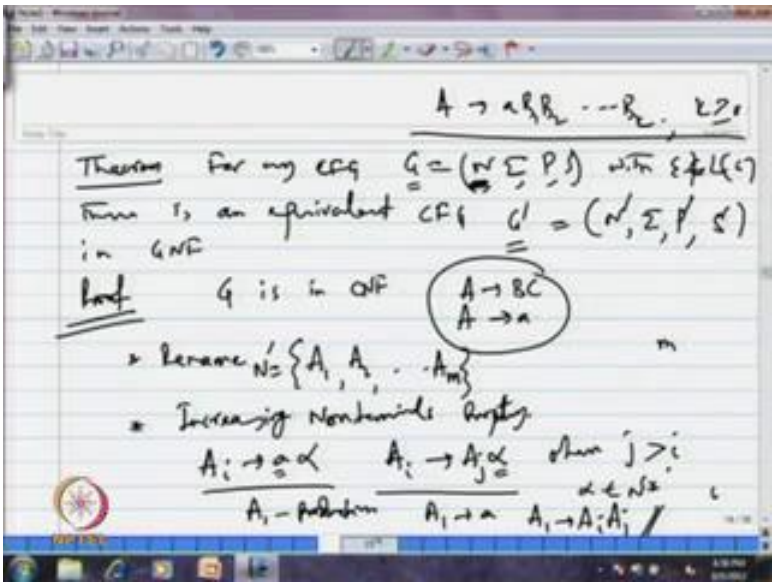


Then, we introduce under lemma what is called lemma 2, so where suppose there is a CFG G with N sigma P S and say A goes to alpha 1, B alpha 2 is in the production and B goes to beta 1, beta 2 up to say beta k, these are set of all b. Productions which are which are there in P, then we can have construct from G, the grammar G dash which is N dash sigma P dash and S; such that P dash contains all those productions which are there in already in P.

And then all the productions of the form A goes to alpha 1, beta 1, alpha 2; that means, this B is replaced by this beta 1 or A goes to alpha 1, beta 2, alpha 2, this B is replaced by this B 2 and like that it will go up to alpha 1 beta k alpha 2 and then we remove this production; that means, we subtract from this set. This particular production A goes to alpha 1, B alpha 2; that means, we removed this production and replaced this occurrence of B in the middle by all the right hand side of these B productions.

So, you get a new grammar which contains these kinds of productions, we want to show that L of G dash and L of G are equivalent, so it is quite easy to see that whatever we can simulate in G can also be simulated in G dash, it requires only one more step and whatever we can generate in G dash can also be generated in G. So, we left it to the reader to prove it, so it is quite simple to show what we do now is that we use these two lemmas to transform any grammar which is not in Greibach normal form to Greibach normal form.

(Refer Slide Time: 42:07)



We say that for any CFG G N sigma P S with Epsilon not in the grammar, there is an equivalent CFG G dash say N dash sigma P dash say S dash which is in GNF that means, all productions of the form A goes to small a, B 1, B 2 up to B k for some k greater and equal to 0. So, given any grammar G, we can always construct an equivalent grammar G dash which is in GNF.

So, we will consider instead of giving the proof of this theorem I will explain how can you construct the equivalent grammar in GNF for this grammar with help of one example. So, the construction step what we assume is that first we assume that the grammar G is in CNF, if it is not in CNF we can always convert it by using the process that we have already given.

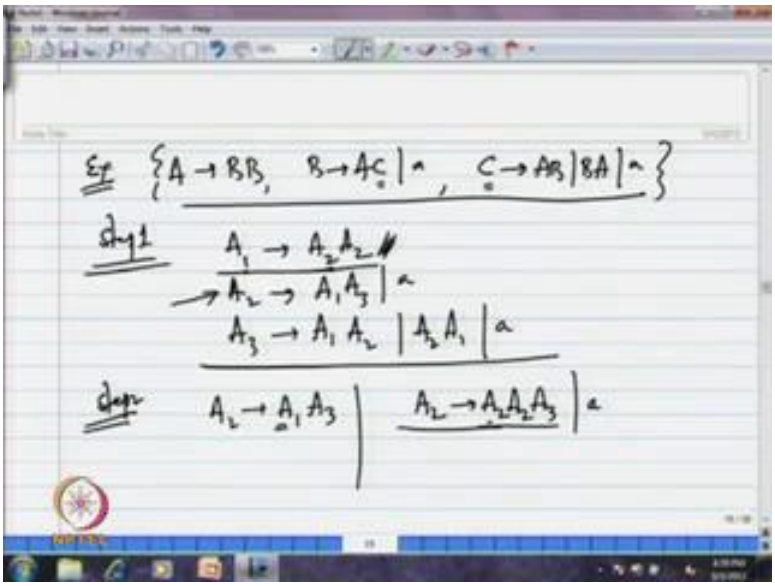
So, therefore all productions B of the form A goes to B C or A goes to a and in the construction process first rename all variables or all non-terminal S, suppose that there are m numbers of non-terminal N, in this N numbers of non-terminal is m which are say and those can be renamed to have say A 1, A 2 up to say A m. So, there are m numbers of non-terminals, these are set of non-terminals, so this N dash will have these non-terminals, so we have renamed all those non-terminals without changing the language of the grammar.

Then in the second step what we will do we process the productions in P, such that they satisfy a property called increasing non-terminal property and this increasing non

terminal property is defined like this we say that all productions are increasing non-terminal property. If it is the forms A i goes to A alpha, where it starts with a terminal symbol or A i goes to A J alpha, where J is greater than i and where alpha is basically string of non-terminals.

To enforce the increasing non-terminal property, what we do we start with A 1 production of the form A 1 goes to a and A 1 goes to some A i is because of all productions should be of C form according since it is in CNF. Now, here we can apply this lemma 1 and lemma 2 to bring it to a GNF, we will illustrate it with a help of example.

(Refer Slide Time: 46:04)

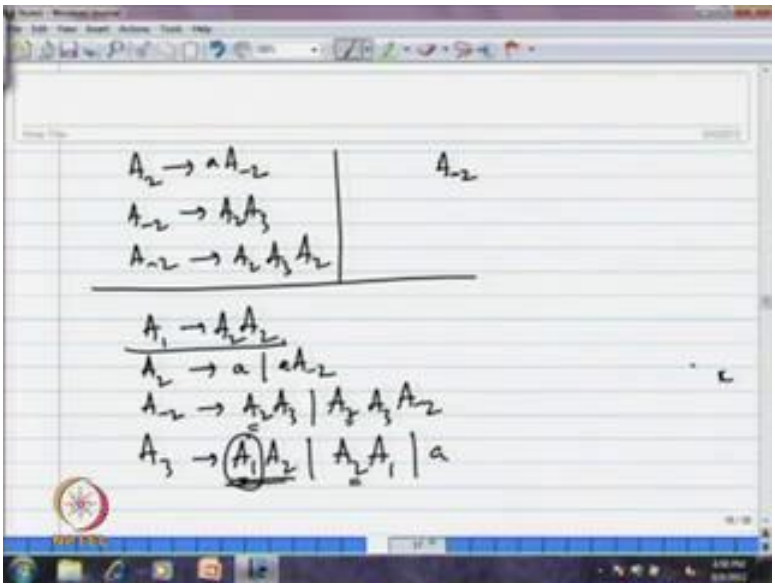


For example, say we consider this grammar A goes to B B, B goes to A C, C goes to A B, B A, suppose that this is given grammar we have these productions, so in step 1, we rename the non-terminals and suppose A written as A 1 and B is say A 2. So, we will have it is A 2, A 2 then B is A 2, A 2 goes to A 1, where C is A 3 and this A and A 3 goes to because this C is A 3, A 1, A 2 or A 2, A 1 for this 1 and this A.

So, we have renamed and by renaming we have got all these productions of the original productions, then in step 2, we consider first this A 1 production, we see that this A 1 production is already in increasing non-terminal property. Because, there is only one A 1 production is A 1 goes to A 2, A 2 and the subscript here A 2 is greater than this subscript over here A 1.

Now, we process this A 2 production, so when we process these A 2 productions to enforce this increasing non-terminal property what we do so first apply lemma 2 to A 2 goes to A 1, A 3. When, we apply lemma 2 over here we get A 2 goes to we replace this A 1 by the right hand side of this A 1 production, that is it is A 2, A 2, because we know that already you A 1 production is in satisfies increasing non-terminal property. So, it will be A 2, A 2, A 3 and the other production A 2 production is A 2 goes to A. Now, we apply lemma 1, because this is a left recursive production A 2 goes to A 2, A 2, A 3, so when you remove this left recursion.

(Refer Slide Time: 48:42)

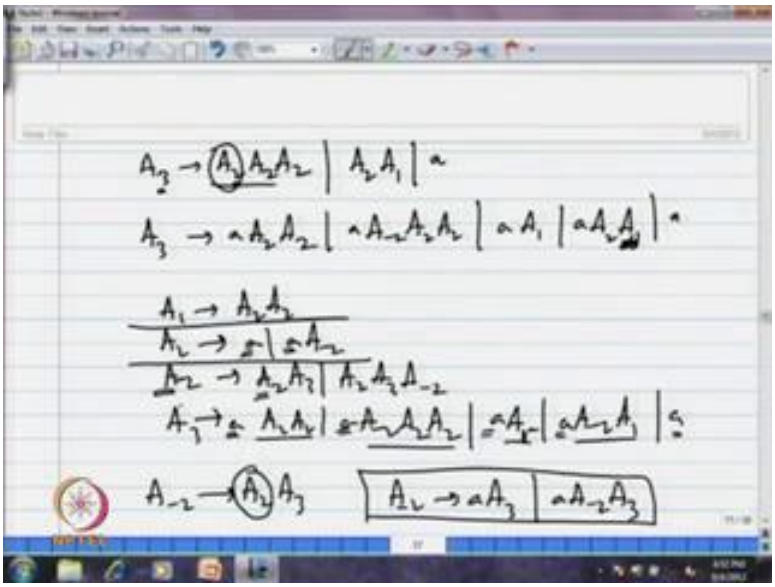


Then, we get here we introduce a new non-terminal say A minus 2 corresponding to non-terminal A 2 and corresponding production we will have A 2 goes to a A minus 2, A minus 2 goes to A 2, A 3 and A minus 2 goes to A 2, A 3, A 2. So, in this case all these productions satisfy this increasing non-terminal property and hence the resulting grammar will be A 1 goes to A 2, A 2 already A 1 production satisfy increasing non-terminal property.

Now, A 2 productions are also satisfy increasing non-terminal property, so A 2 goes to A, A minus 2 and A minus 2 goes to A 2, A 3 and A 2, A 3, A minus 2 because here the right hand side the left moves non-terminal is A 2 and here. So, it is A 2 but here it is A minus 2, so subscript here is 2 greater than minus 2 and we have already A 3 productions of the form A 1, A 2 and A 2, A 1 and A.

Now, this A 3 productions do not satisfy this increasing non-terminal property and hence we consider this A 3 productions, Now, So, that we can satisfy increasing non-terminal property.

(Refer Slide Time: 50:13)



So, we have A 3 goes to A 2 by applying again lemma 2 to A 3 productions and introducing a non-terminal say A minus 3, now this in this case we need not apply this lemma 1 because there is no any left recursive productions. Because, A 3 goes to A 1, A 2 and A 3 goes to A 2, A 1, so we need to simply apply lemma 2, instead of lemma 1, so that it satisfies increasing non-terminal property.

So, therefore, we can write it as A 3 goes to A 2, A 2, A 2 since we have A 3 goes to A 1, A 2 this A 1 can be replaced by the right hand side of this A 1 production. So, this is A 2, A 2 and hence we get instead of A 1 A 2, A 2 and already we have this A 2 over here and the other production is it is A 2 ,A 1 and A.

Now, again we apply lemma 2 because this is A 2 this and this is A 3, these two is less than 3, it has to be greater than or equal to 3 according to our increasing non-terminal property. So, therefore, this again is A 2 can be replaced by the right hand side of this A 2 production and eventually we will get A 3 to be a A 2, A 2, a A minus 2, A 2, A 2, a A 1 and a A minus 2, A 1 a minus 1, this is A 1.

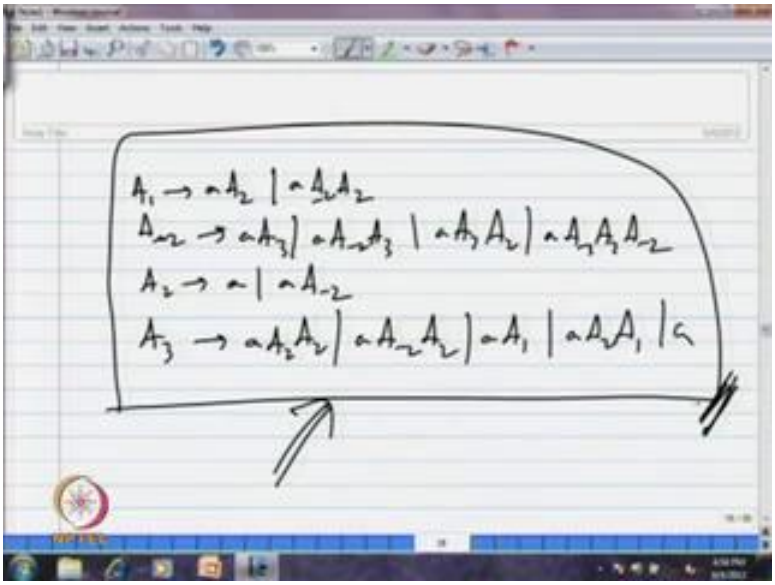
In fact, now all the productions you will see that we will be in m i n p all productions satisfy this i n p and the resulting grammar can be written as A 1 goes to A 2, A 2. Then, A 2 goes to a A minus 2, a A minus 2 then A minus 2 goes to A 2, A 3, A 2 A 3 A minus 2 and A 3 goes to a A 2, A 2 a A minus 2 A 2, A 2, a A 1 and a A minus 2, A 1 and a.

Now, all A 3 productions and A 2 productions are already in GNF, because this should have been A 3 goes to because A 3 goes to it starts with the terminal symbol, start at terminal symbol, start at terminal symbol, terminal symbol and terminal symbol and remaining string is the string of non-terminal state. We introduced already the grammar and similarly A 2 production also this is already in GNF all thing is that this A 1 production and A minus 2 productions they are not in GNF, because they start with a non-terminal.

So, now again I can apply lemma 1, lemma 2 to A minus 2 to convert it to GNF form; that means, this A 2 can be replaced by the right hand side; that means, we have the production of the form A minus 2 goes to A 2, A 3. So, this A 2 can be replaced by the right hand side of this A 2 production that we have already over here.

So, it is A and a A minus 2; that means, this can be replaced by A minus 2 goes to a A 3 or a A minus 2, A 3, so in such a case these two will be in GNF. So, if you be repeatedly apply this lemma 2 to this A 1 and A minus 2 production, eventually all the productions will be in GNF.

(Refer Slide Time: 55:17)



And we will have the resulting grammar as A 1 goes to a A 2, a A minus 2, A 2, A minus 2 goes to a A 3, a A minus 2, A 3, a A 3, A 2 and a A minus 3, A 3, A minus 2, then A 2 goes to A and a A minus 2 and A 3 goes to a A 2, A 2, a A minus 2, A 2, a A 1, a A minus 2 A 1 and a. So, this is the resulting grammar by applying lemma 2 to A 1 and A minus 2 productions and we see all productions are in GNF

So, this is how by applying repeatedly lemma 1 and lemma 2 lemma 1 to remove left recursion and lemma 2 to convert it to GNF, so that the left hand symbol of the right hand symbol right hand side is a terminal symbol. So, we can eventually get the grammar to be in GNF form and since at every step we apply either lemma 1 or lemma 2 and in when we apply lemma 1 or lemma 2. The resulting language does not change, so therefore, the resulting language in C equivalent in this grammar, which is in GNF will not be change. Therefore, this is an equivalent grammar corresponding to the original grammar, so that is how we transform any given grammar to GNF form.