Theory of Computation

Prof. Kamala Krithivasan

Department of Computer Science and Engineering

Indian Institute of Technology, Madras

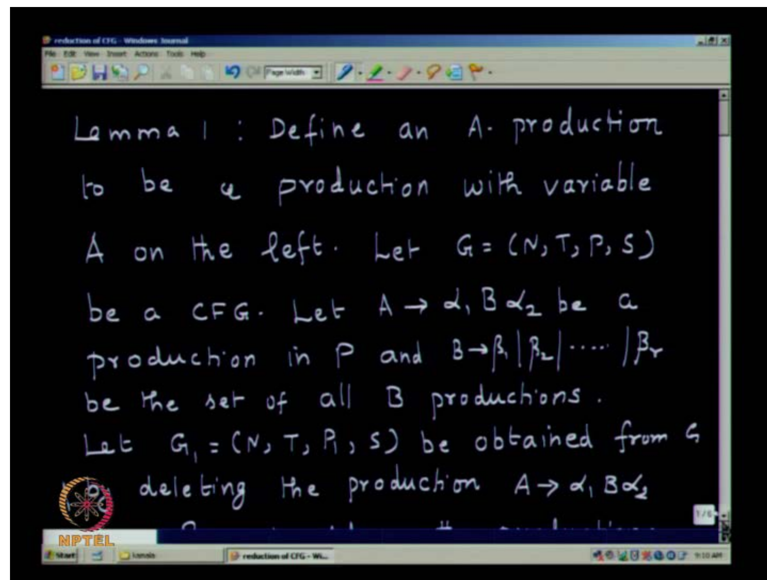
Lecture No. # 07

Greibach Normal Form for CFG

So, in the last class, we saw about the reduction of a context free grammar, so how to remove the epsilon productions, the unit productions and how to remove the useless non-terminals. We also saw what is chomsky normal form, and what is the greibach normal form, and how to convert context free grammar into chomsky normal form, it does not contain itself. Of course, for epsilon we saw that, if you want to include epsilon, then we should have a rule of the forms S goes to epsilon, S this not occur on the right hand side of any production.

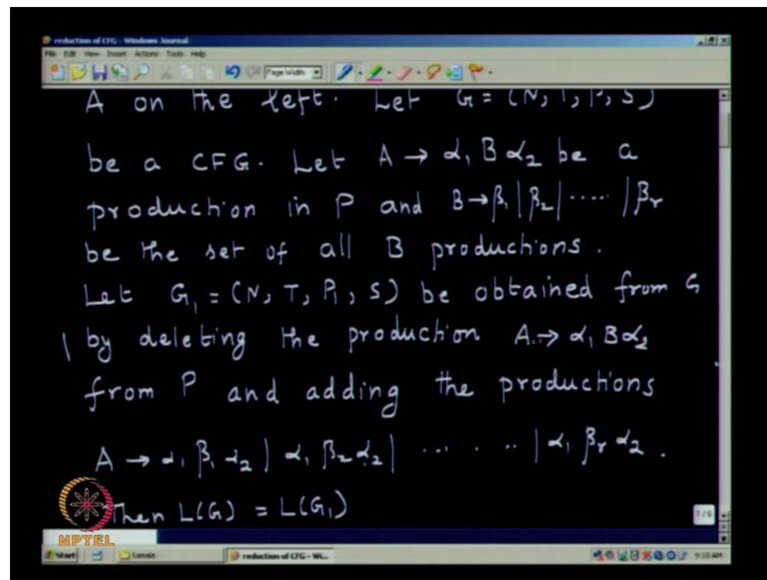
Now, we shall see how to convert a grammar to Greibach normal form for that we saw 2 lemmas, we will be making using of these 2 lemmas. Let us recall those 2 lemmas.

(Refer Slide Time: 01:03)



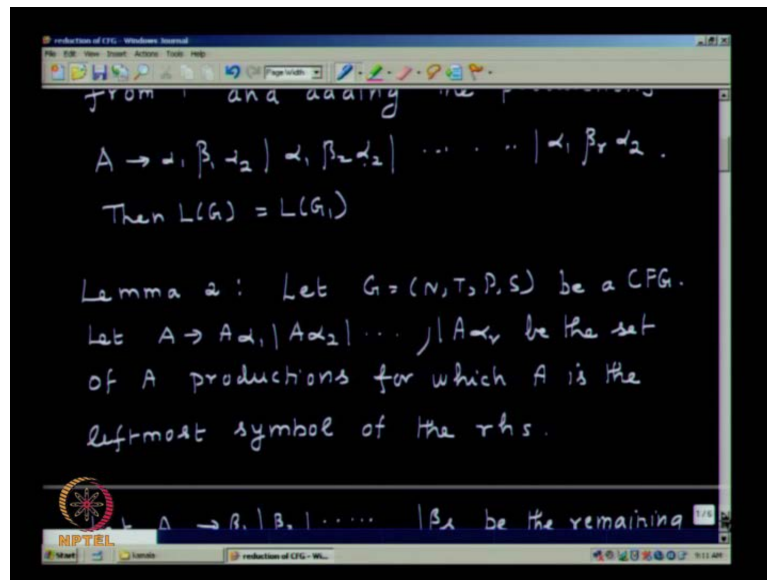
Lemma 1 says that define an A production to be a production with a variable A on the left, and if you have a grammar G is equal to N, T, P, S. Then for some reason if you want to get rid of the rule A goes to alpha 1 B alpha 2, where alpha 1 and alpha 2 are some strings. Then instead of this, you have to include a set of rules. With B on the left hand sides the set of B productions are B goes to beta 1, B goes to beta 2, B goes to beta r. Then this rule can be replaced by a set of rules, where you have that is you remove this rule, but instead add the rules.

(Refer Slide Time: 01:54)



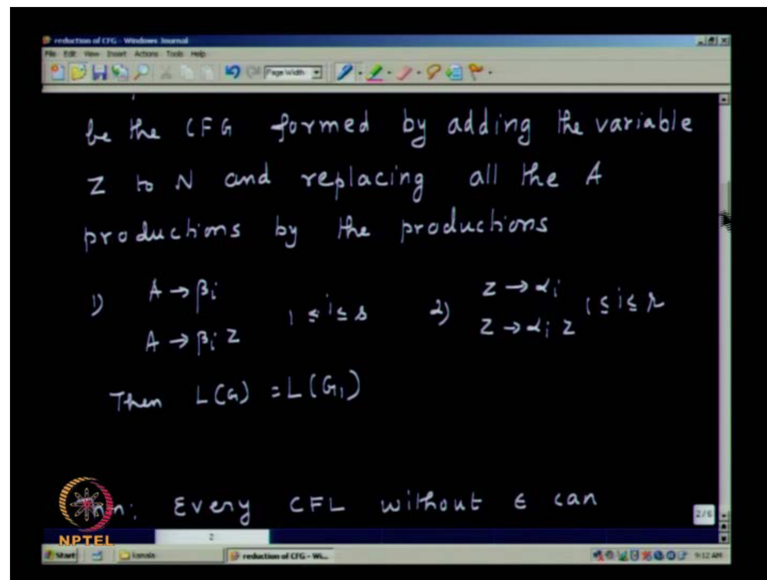
A goes to alpha 1 beta 1 alpha 2 alpha 1 beta 2 alpha 2 and so on. So, instead of this r; instead of this rule, you have add r rules of this form. The effect will be the same, because if you apply this rule and afterwards you apply a rule B goes to beta i. Then the effect of applying this and then B goes to beta a. It is achieved by using the rule A goes to alpha i beta a alpha 2. So, the language generated will not be affected, it is the same. But the only advantage you have got is, you have got rid of this rule.

(Refer Slide Time: 02:48)



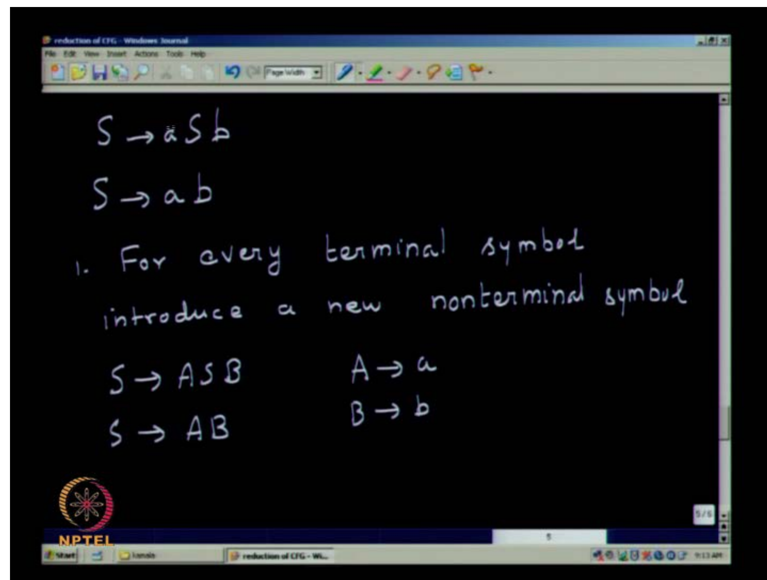
Let us see, why we need that when you convert the grammar into Greibach normal form. The next lemma is to avoid left recursion. This we have seen again, let G be a context free grammar. And A goes to A alpha 1 A alpha 2 be the set of left recursion rules with a on the left hand side. And A goes to beta 1 beta 2 beta r the non- recursive ; left recursive rules with A on the left hand side. Then this r plus s rules can be replaced by 2 r plus 2 rules of this form.

(Refer Slide Time: 03:14)



A goes to beta i, A goes to beta i is that Z is a new non-terminal introduced and i varies from 1 to s, and Z goes to alpha i Z goes to alpha i Z 2 r rules. So the r plus s rules are really replaced by 2 r plus 2 s rules. In order, when we do this, the left recursion is removed, but a right recursion is introduced. But the language generated is not affected, this also we have seen. So, we will make use of these 2 lemmas, to convert a given grammar into greibach normal form.

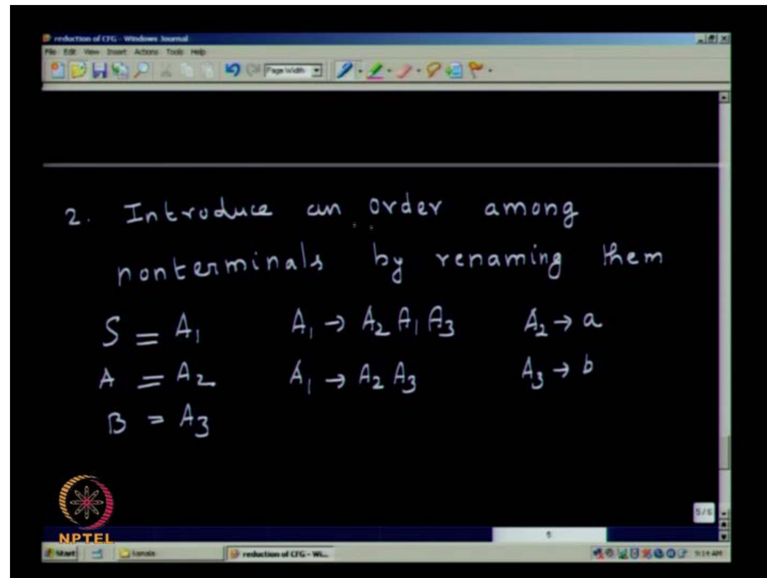
(Refer Slide Time: 04:06)



So, let us take the simple example, we have started with the simple example S goes to a S b, S goes to a b. The language generated is a power n, b power n. So, for convenience we can take languages which do not contain epsilon. So, the first step will be like in Chomsky normal form. Every terminal symbol will be replaced by a non-terminal symbol. So S goes to a S b, S goes to a b will be replaced by S goes to A S B, S goes to A B, where the A and B are new non-terminals introduced. A corresponding to small a, and B corresponding to small b.

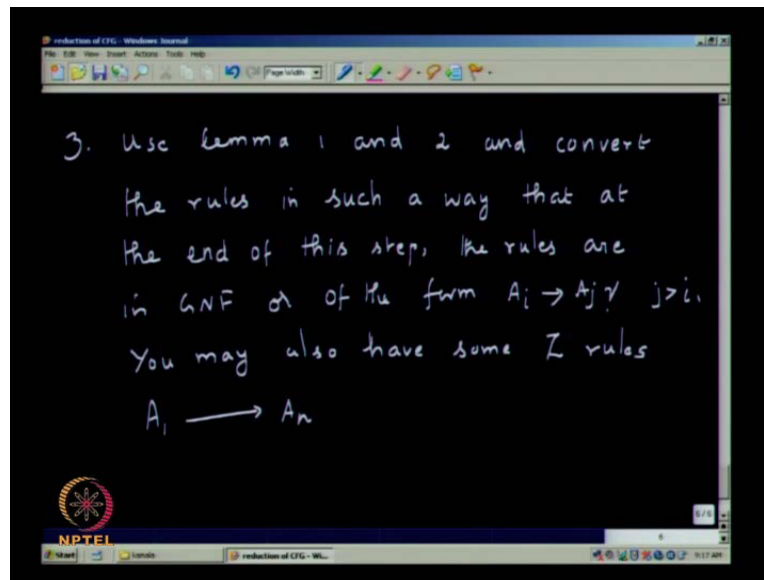
Now, the capital A goes to a, Capital B goes to b will take care of the fact that. This A and B are ultimately converted into small a s and b s which are terminal symbols. We can very easily see that, this grammar generates a power n, b power n, this also generates A power n, B power n. Initially it generates capital A power n capital B power n. Then all the A s are made into are converted into small a s and all capital B s are converted into small b s. So the language generated, it is not going to be affected.

(Refer Slide Time: 05:20)



That is a first step. Then the second step is you, among the non-terminals, you introduce an ordering call them as A 1 A 2 A 3. So, in this example; for example, you can take S to be A 1 A to be A 2 and B to be A 3. Then the rules will be of this form, A 1 goes to A 2 A 1 A 3, A 1 goes to A 2 A 3 and then A 2 goes to a, A 3 goes to b. This is you introduce an ordering among the non-terminals, again this is just renaming of non-terminal so, the language generated is not going to be affective.

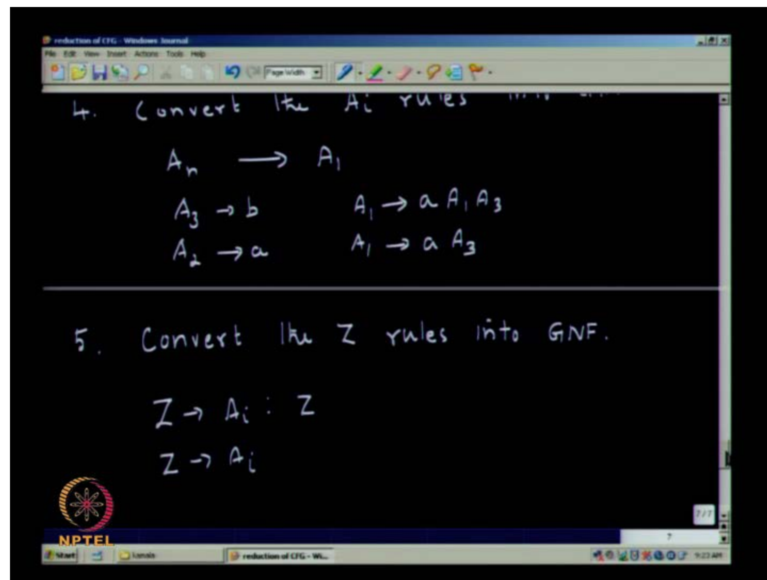
(Refer Slide Time: 06:09)



Now, after doing this third step is use lemma 1 and 2 and convert the rules in such a way that at the end of this step, the rules are of the form in G N F or of the form A i goes to A j some gamma, where j greater than i. You may also have some Z rules that is a Z, the Z s are the new non-terminals introduced. So, you use lemma 1 and lemma 2 in an appropriate manner, we can take an example and see. And at the end of this step, either the rules will be in greibach normal form or the A i rules will be of the form, A i goes to A j gamma, where j is greater than i. This is the first symbol on the right hand side, the A j is such that j will be greater than i. When you do this, you may introduce some Z rules also, Z is the set of Z 1 Z 2 you may be introducing.

Let us see, How to do this? For this, you go from actually A 1 to A n. First you convert the A 1 rules, then you convert the A 2 rules and so on. Actually in the example which we have considered, this step is not necessary. Look at the rules at the end of the second step A 1 goes to A 2 A 1 A 3. This is 2 is greater than 1, A 1 goes to A 2 A 3, this is greater than 1, A 2 goes to a, A 3 goes to b are in Greibach normal form. So, the third step in this example is not necessary. Some other example, it may become necessary.

(Refer Slide Time: 09:10)



Then the fourth step is convert the A i rules into greibach normal form. In this step you will go from A n to A 1. First you will see the non-terminals are A 1 A 2 A n, then by this condition the last A n rules will be in greibach normal form. Because you cannot have something greater than that on the right hand side. At the end you will find that the A N rules will be in greibach normal form. So, make use of them in A n minus 1 rules, then make use of them, A n minus 2 rules and so on, keep on back substitute and at the end of the fourth step, all the A i rules will be in greibach normal form.

So, let us take this example, which we have been considering. Look at this rule, you have to the fourth step, you have to go from A n to A 1. So, look at A 3, this rule is in greibach normal form. Look at A 2 this is in greibach normal form, the A 1 rules are not in greibach normal form so, you have to convert them to greibach normal form. Use the lemma 1 so, you want to get rid of this rule so, instead of A to wherever there is A 2 rule you have to use them. That is lemma 1 instead of A 2, you use this rule that means instead of this rule you will be writing A 1 goes to small a a 3. Similarly, in this rule use lemma 1, you get rid of this rule, but substitute for this A 2, A 2 goes to A is a rule you are having.

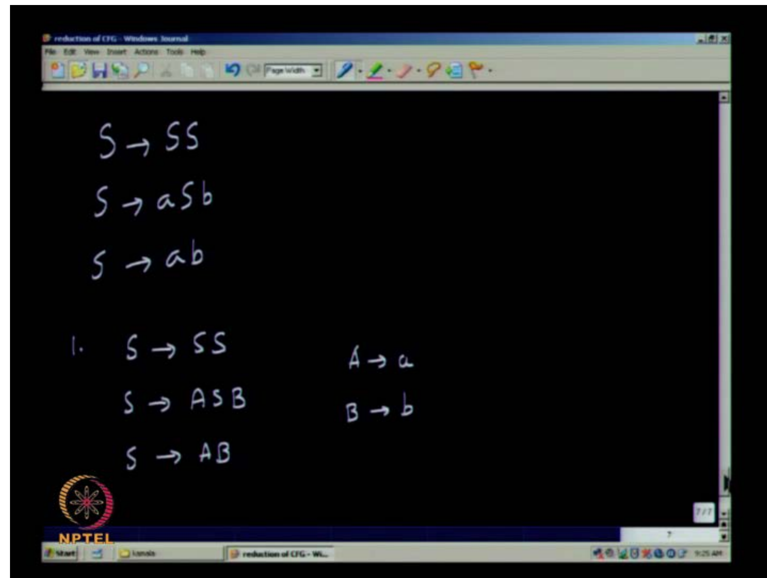
So, here also this A 2 you will be replacing by the right hand side that is A. So, these 2

rules will be replaced by replacing the A 2 by A. So, this example, the fourth step becomes A 3 goes to b will be there, A 2 goes to a will be there. Then A 1 instead of A 2 you write a A 1 A 3, A 1 goes to a A 3. Now, throughout the conversion either you will be using lemma 1 or lemma 2. And we have already seen that those 2 lemmas the language generated it is not affective.

Now, you find that this grammar is in greibach normal form this rule on the right hand side. You are having only a terminal symbol, these 2 rules we are having a terminal followed by a string of non-terminals. So, the grammar has been converted into greibach normal form, with these 4 rules. But you can easily see that, this useless symbol A 2 is an useless symbol. Now, because A 2 we have already replaced so, this rule is useless and symbol is also replaced useless. So, 1 2 3 rules are enough, after conversion you may end up with some useless symbols, useless productions. You can remove them in the usual way so, this is the fourth step.

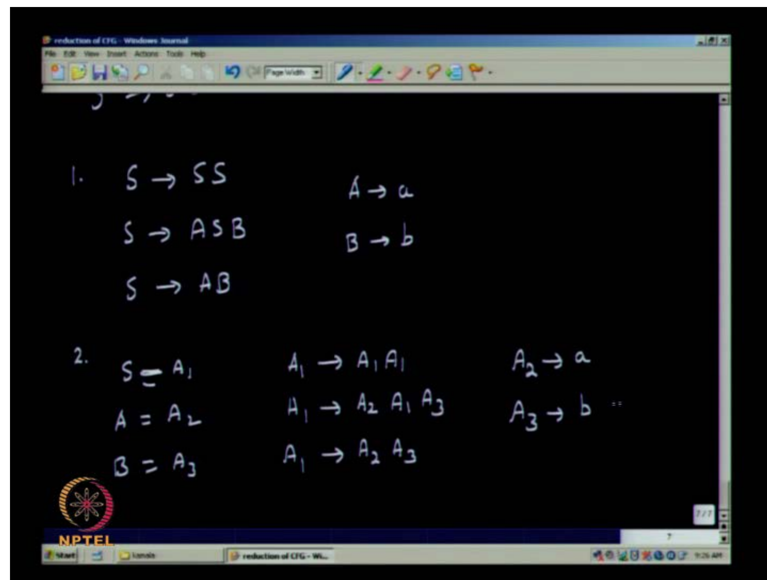
The fifth step, which is not necessary in this example, is convert the Z rules into G N F. By the way, the Z rules are, they will be of the form. Z goes to some A i, something Z in the n Z or Z goes to some A i they will be of this form, the first symbol will not be A i anyway. So, again making use of lemma 1, at this stage all the A i rules are in greibach normal form. So make use of them and make instead of this A i put the right hand side of those rules. So in that case all the Z rules will be converted to greibach normal form. In this particular example, it is not necessary at the fourth stage itself, it is converted into greibach normal form.

(Refer Slide Time: 14:30)



Let us take one more example, where this fifth step will be necessary. So, let us take as almost similar example, (No Audio From 14:26 to 14:40) what is the language generated? The language generated is the dyke set, which is the well formed string of parenthesis. Now, let us convert these, the earlier one we had only these two rules, now we have one additional rule S, S goes to S S. So, the first step is, what is the first step? For every terminal symbol, introduce the new non-terminal. So, the rules will become S goes to S S, S goes to A S B, S goes to A B, A goes to a, B goes to b, this is the first step.

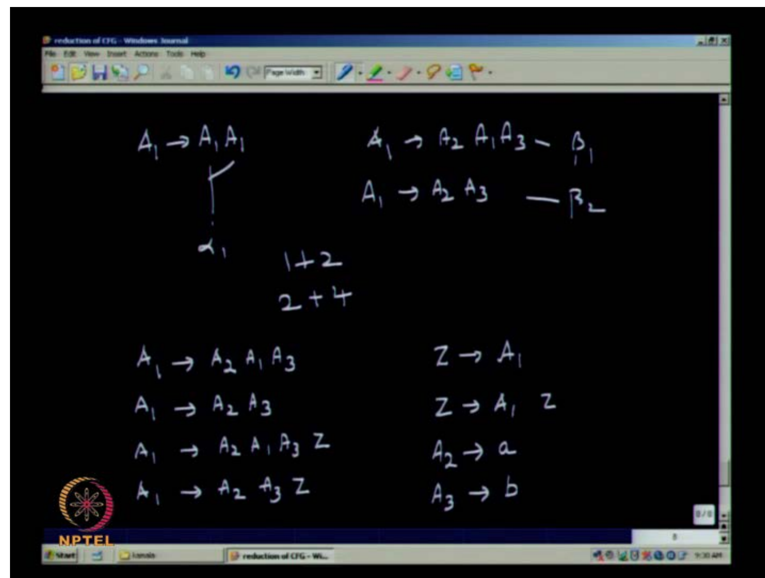
(Refer Slide Time: 15:48)



Then the second step is introduced an ordering among the non-terminals. So, make S as A 1, A as A 2, B as A 3. So, the rules will become now, A 1 goes to A 1 A 1, A 1 goes to A 2 A 1 A 3, A 1 goes to A 2 A 3. Wherever you have S, you have put A 1, wherever you have capital A you have put A 2, wherever you have capital B you have put A 3. Then these rules will become A 2 goes to a, A 3 goes to b. Now, the third step becomes necessary. Because the condition that A i goes to A j, where j is greater than i is satisfied by these 2 rules, but not by this rule.

This rule left hand side, you have A 1 right hand side also the first symbol is A 1. So you have to use something to convert this, now what do you do for that? Now, you note that this is the left recursive, which one you will use? Sometimes you have to use lemma 1, sometimes you have to use lemma 2. At this stage to convert the A 1 rules since, such a way that, the first symbol on the right hand side is A 2 or A 3. You have to use A lemma 2, because there is a left recursive rule A 1 goes to A 1. The first symbol is again the same as A1 so, there is a left recursive rule. So, how do you go about doing this?

(Refer Slide Time: 17:45)



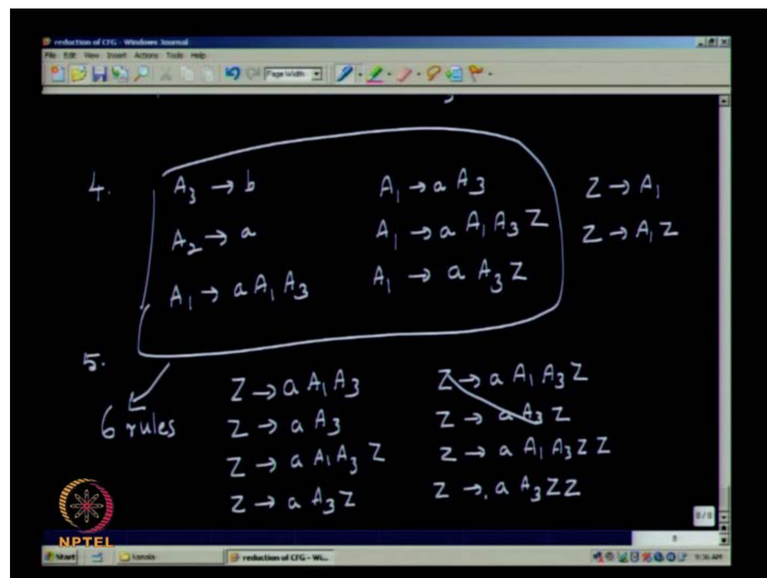
Now, you have A 1 goes to A 1 A 1 as a left recursive rule. And the non-recursive left recursive rules are A 1 goes to A 2 A 1 A 3, A 1 goes to A 2 A 3. In the lemma if you put alpha 1 is this, beta 1 is this, beta 2 is this. So, there is 1 plus 2 rules here, instead we have to replace them by 2 plus 4 rules, 6 rules. So, what are they? The rules are A goes to beta i and A goes to beta i Z. So, you will have A 1 goes to, you will be introducing the new non-terminal Z. So, A 1 goes to A 2 A 1 A 3, A 1 goes to A 2 A 3. This is of the form A goes to beta i. Then you have A 1 goes to A 2 A 1 A 3 Z, A 1 goes to A 2 A 3 Z.

So, instead of these two rules, you are getting four rules now, instead of this one rule you must get two rules. What is alpha 1? Alpha 1 is this A 1, not this together A 1 alpha 1, is it not? It left recursion. So, alpha 1 is A 1, the rules will be of the form Z goes to alpha 1, Z goes to alpha 1 Z, but what is alpha 1 is A1. (No Audio From: 19:53 to 20:04) A 1 (No Audio From: 20:12 to 20:21). So, instead of the three rules, we have six rules. Apart from that the other two rules will be there, A 2 goes to a, A 3 goes to B. So, at the end of step 3, we have 8 rules. The A i rules satisfy the condition that they are of the form A i goes to A j something, this is 2 is greater than 1 2, is greater than 1, and so on here, they are in greibach normal form. Apart from that we are having 2 Z rules also.

Now, the step four, you convert the see in step three; we have gone from A 1, then A 2

then A 3. A 1 we have done so that, it is converted into the required form, A 2 and A 3 are already in greibach normal form, so we did not have to do anything about that. In step 4, we have to convert all the A i rules into greibach normal form. For that first we start with A 3, then A 2, then A 1. So, at the end of step three, all the A n rules will anyway be in greibach normal form. So, See A 3 will be in greibach normal form. A 2 also is in greibach normal form. Now, we have to convert this into greibach normal form. Again for this, which one you will use lemma 1 or lemma 2? Lemma 1, because on the right hand side, you are having A 2, this could be easily replaced by the small a, so, in each of this four rules, you can replace A 2 with A.

(Refer Slide Time: 22:30)



So, at the end of step 4, A 3 goes to b, will be there A 2 goes to a, will be there then A 1 goes to a, we will see the rules and then write. A 2 A 1 A 3, A 1 goes to, instead of A 2 you write a A 1 A3, A 1 goes to a A 3, A 1 goes to a A 1 A 3 Z, A 1 goes to a A 3 Z. The two Z rules will be there, Z goes to A 1, Z goes to A 1 Z, those two rules will be there. So, at the end of step 4 you have these rules. Now, note that all A i rules have been converted into greibach normal form.

Step 5 is to convert the Z rules to greibach normal form. So, step 5, these 6 rules will be there, and maybe I need not write them again. These 6 rules will be there. These rules we

have to convert to greibach normal form, again you have to use lemma 1 in this for this a 1 you have to substitute from these 4 rules. Whatever is on the right hand side, you want to get rid of this rule so, Z goes to instead of A 1, use this right hand sides. Z goes to A 1 Z instead of A 1, use these right hand sides. So, you will get Z goes to a A 1 A 3, Z goes to a A 3, Z goes to a A 1 A 3 Z, Z goes to a A 3 Z. Now, for this rule if you do the same thing you will get, Z goes to a A 1 A 3 Z, Z goes to a A 3 Z, Z goes to a A 1 A 3 Z Z for this A 1.

I am substituting all these four, right hand sides Z goes to a A 3 Z this Z comes here. So for this A 1 you can substitute all the four. For this A 1 also you can substitute all the four so, you get 8 rules. But note that this is the same as this, it is getting repeated. These two rules are repeated so, need not have that, it is the same. So at the end, you are getting 6 plus 6, 12 rules. Now look at these 12 rules. At the end of fifth step you are having 12 rules. These rules are all greibach normal form these rules are also in greibach normal form. So, you have converted the given grammar into greibach normal form. So, the steps involved in conversion to greibach normal form are first step will be for every terminal symbol introduced a new non-terminal symbol.

Then, rename the non-terminals and introduced an ordering among them, call them as A 1 A 2 A 3, this is just to bring the A i rules into greibach normal form. Then the third step you can use lemma 1 and lemma 2, and convert the rules in such a way that at the end of the step you will all the A i rules will be of the form, A i goes to A j gamma, where j is greater than i. While doing this, you may introduced some Z symbols, new symbols you may be introducing and in this step, first you will convert the A 1 rules then the A 2 rules, then the A 3 rules and so on up to the n th rules.

At the end of this step all the A n rules will be greibach normal form. So, in step 4, what you do is convert the A i rules in the greibach normal form, for which you will go from A n to A 1. First you look at the A 1 rules then A m i A n minus 1 rules and so on. So, at the end of step 4 all the A i rules will converted into greibach normal form, but the Z rules may not be in greibach normal form. So, the fifth step is use again, lemma 1 to convert the Z rules into greibach normal form, use substitution. So, this is what we have done in this example so, at the end this converted into greibach normal form. The language

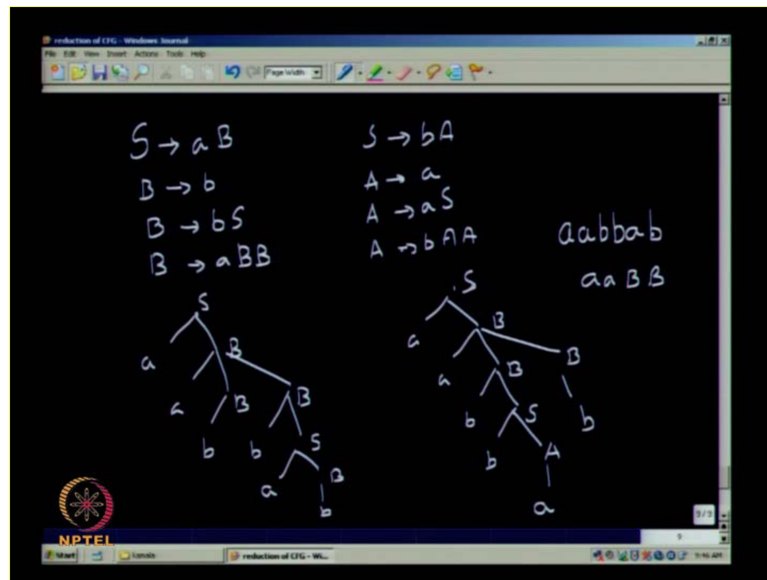
generated it is not affected; see it is easy to see in this example.

Look at this example, this one, this is the dyke set. It is easy to see that dyke set, is it not? It has got three simple, three rules, and it is very simple. Whereas, at the end you are ending up with that 12 rules and by looking at these twelve rules, it is there is no way you can immediately say that it is dyke set, is it not? You would not know, what is the language and it is a slightly complicated. So, but how can you prove that the language generated is the same as the earlier 1? Because throughout the conversion, what have we used? We have used only lemma 1 and lemma 2, nothing else. And we have already shown that, if you use 2 lemmas, those 2 lemmas the language generated is not affective.

So, at the end, you end up with the greibach normal form, it generates the same rule same language. Now, what is advantage? Why should you convert something into greibach normal form? As I told you, the thing is it is easier to for parsing. And when you want to show the equivalence between pushdown automata, and context free grammar, taking the grammar in greibach normal form helps. Also when you write a grammar for a compiler as far as possibly, if you write the rules in greibach normal form that parsing portion will be easy. There are construction of the table will be easy. So, we have seen two normal forms.

One is the Chomsky normal form and another is the greibach normal form. There are other normal forms something is called operator precedence normal form, binary standard form. There are other things also, but we will not go into that. Each one has its own advantage, because something will some for some theorem that may be helpful and so on, even greibach normal form for proving some theorems. If you assume that the grammar is in greibach normal form, it is very easy. We will; we shall see where it is used in due course. Now, coming back to ambiguity, we have seen, what is an ambiguous grammar? What is the ambiguous language? What is an inherently ambiguous language? And so on.

(Refer Slide Time: 31:50)



So let me take this example, which we have already considered S goes to a B, S goes to b A, B goes to b, B goes to b S, B goes to a B B, these examples we considered earlier in the class. A goes to a, A goes to a S, A goes to b A A, the language consisted of since having equal number of a(s) and b(s) epsilon is not included, is this grammar is ambiguous or not. Consider the string aa bb ab, we can see that this particular string can be generated in two different ways. Start this S now, there is one thing which will make use of later see when you have S and A combination that is S you have, and you have to generate the three (( )). When you have S and A, you must use this. When you have S and B you can use this rule that is fixed.

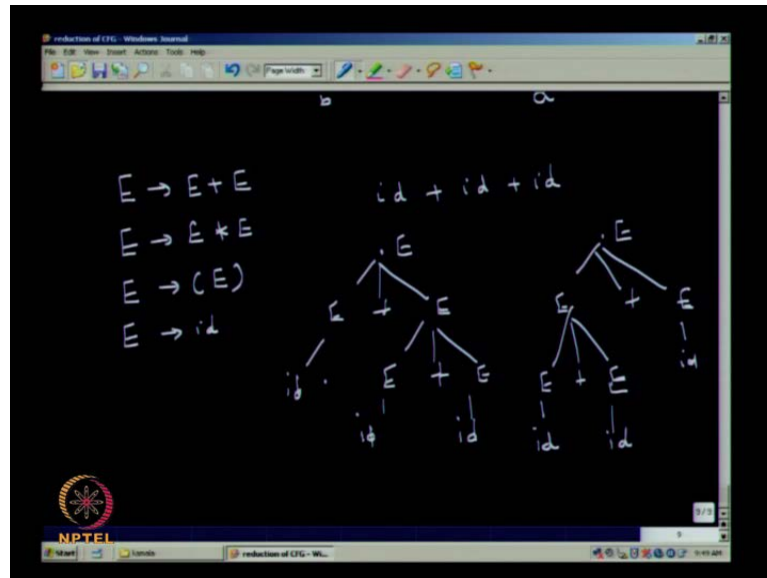
Similarly, when we have B and A, we have to use this rule. If you have B and B, you can use either of them. If the non terminal in the first symbol they tell you by looking at the first symbol in this, you can able to say which rule is used. Then such a grammar is called LL one grammar and that is very useful in recursive decent (( )). This is not a LL one I hope you have (( )) problem, but S and a combination this is the string to be generated so, first you have S and a. So, you have to rule, use this rule then B and a next symbol to be generated is a so, you have B and a combination so, we have to use this rule only a B.

But for this B, you can either use b and terminate or in another way this two steps are the same, you can use b S here. Then from this b you have to generate b a b so, you will get b S a B b. Now, here from this S, you have to generate B so B a, this will go to a, this will go to b. So, the string generated is a a b; a a b b a b here also, a a b b a b, but the derivation trees are different. So, this grammar is ambiguous, but this language is not inherently ambiguous. You can give an unambiguous grammar for this we can consider how to give an unambiguous grammar like this. Now, the first as I told you at this stage, first 2 steps are the same, you generate a a and then capital B B, a a B B you generate. Then from these 2 b (s), you have to generate this.

We know that if, a string is derivable from S, it has got equal number of a (s) and b (s). If a string is derivable from B, it has got one more B then it has a (s). If a string is derivable from a, it has got one more a, then it has b (s0. Now, from this B, you have to derive a string which is got one B more than a. And from this B also you have to derive a string which is got one more B then it has c's. Now, this b b a b portion, you can split like b bab. That is, this also has one b and zero a's, and this has got 2 b (s) and one a (s) or you can split it as bba b. This portion will have two b (s) and one a (s) this has one.

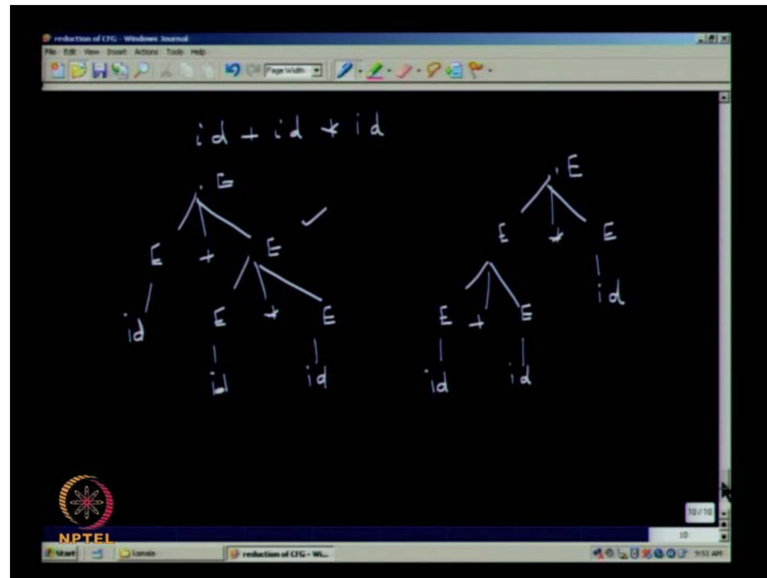
So, this b b a b, we can split in two ways, two substrings, each one of them having one b more than the number of a (s). And because of that possibility, this derivation from b b is done in two different ways. Now, if you want to construct an unambiguous grammar, you must avoid that, just think about how to do that?

(Refer Slide Time: 38:00)



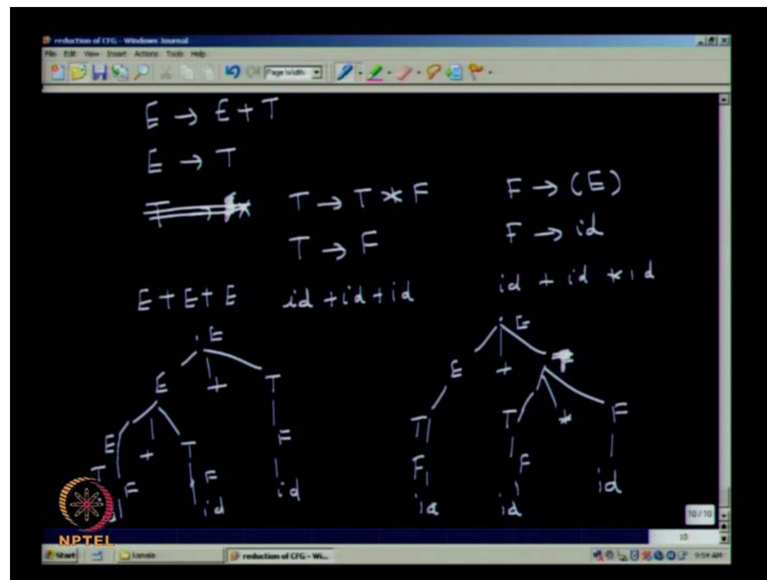
Now, as in expressions like you have E goes to E plus E, E goes to E star E, E goes to E, E goes to identifier. Now, let me take the identifier, we will keep it as it is. So, expressions of the form i d plus i d plus i d can be generated by this, but they will be ambiguous you can have two different derivation trees like, you know E goes to E plus E E plus E i d i d i d. Or E goes to E plus E, this goes to i d, this again goes to E plus E and this goes to i d, this goes to i d, can have two different derivation trees. And so, there will be problem, but if you keep a method that you will have to have only the leftmost. The first this has to be the evaluation is from left to right, then this will be performed first and then this with a result, this will be added. So actually, this will be the one with that will be taken. These two will be added first with the result, this will be added, this will not be taken if, the evaluation is from left to right, you have to do that.

(Refer Slide Time: 40:05)



If, you have something like i d plus i d star i d, then E goes to E plus E E star E and E goes to identifier, identifier, identifier. You can also have E goes to E star E, E goes to identifier E plus E identifier, identifier, which one we will consider? Here, even if you take the evaluation from left to right usually star has a higher precedence. So, this will be performed first and then the addition so, this tree will be taken and not this tree. So, you can have that grammar and have this set of a thing, that is evaluation has to be from left to right and then there is precedence among the operators and so on. And you want to overwrite the precedence use the parenthesis. But you can write it the same effect by consider see, this is a term and then each one is a factors. So, instead of having those rules, you can have set of rules of this form.

(Refer Slide Time: 41:45)



E goes to E plus T, E goes to T expression plus term, because if the evaluation is from left to right, you always have expression plus the term T goes to term. I will remove this rule T goes to T star F, T goes to F, F goes to E, F goes to i d, F is factor. So, what will be the derivation tree for the 2 things? If you have E plus E plus r i mean i would i d plus i d plus i d, you will use E goes to E plus T, then again E plus T, T goes to F, F goes to i d, T goes to F, F goes to i d here, E goes to T, T goes to F, F goes to i d. This will be the only derivation tree and this makes sure of the left to right evaluation process.

Then when you have that i d plus i d star i d, what sort of a derivation tree you will have? E goes to E plus E, this E goes to E plus E plus T, T goes to T star F, E goes to T, T goes to F, F goes to i d, T goes to F, F goes to i d, F goes to i d. This is the only way you will have, we cannot have it another way. So, for every string of this form or whatever forms it is involving identifiers and operators plus r star. There will be a unique derivation tree, this grammar with consists of this. This makes sure that the evaluation is left to right, here also. Because of this, the evaluation, if you have continuous star the evaluation will be i d star i d star i d star, the evaluation will be from left to right.

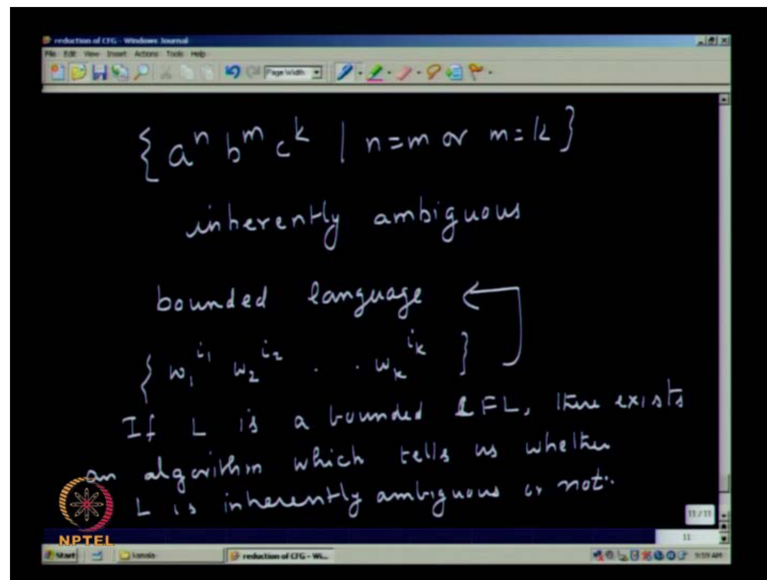
But then star will be performed first before the plus is because the term is evaluated first T star F you have E is the highest one, and then from that expression you go to term

consists of factors. So, expression consists of terms and the terms will be evaluated from left to right, terms consists of factors, will also be evaluated from left to right. But factors are evaluated, then terms are evaluated, then expression is evaluated. That gives the priority operation is taken, star has higher precedence that plus that is taken care of.

So, you can split this in such a way, that you get an unambiguous grammar and this grammar has six rules. So you see that this is an unambiguous grammar and it is very convenient to use this unambiguous grammar for generating the arithmetic expression, because that is very convenient and for code generation also, that will be very useful. So, you can convert an ambiguous grammar into an unambiguous one. Provided the language generated is unambiguous. You can see, if you have an inherently ambiguous grammar, whatever grammar you give, it will be ambiguous only. But if it is a unambiguous language, you can convert an ambiguous grammar to an equivalent unambiguous. Give an unambiguous grammar for ambiguous grammar.

But in this example, we have analyzed how the strings are formed and in the compiler, what do we do. The evaluation will be from left to right and star will have a higher priority than plus. So, we were able to give an unambiguous grammar. This any grammar means you have to look into the problem, there is no hard and first rule that you have to go like this step by step in greibach normal form, it was a step by step procedure, it is a very straightforward you do this step first, do this steps again and so on. Now, if you want to convert an ambiguous grammar into an equivalent unambiguous grammar, it is not a straightforward procedure. Each problem, you have to look in to that problem and see what can be done, convert this to that way and so on. So, is the way you can do.

(Refer Slide Time: 47:35)



There are as I mentioned inherently ambiguous languages like a power n, b power m, c power k, n is equal to m or m is equal to k. Such languages are inherently ambiguous, (No Audio From: 47:44 to 47:53) we have already noted that given a context free grammar, it is un decidable whether it is to find out, whether it is ambiguous or not. But if a language is a bounded language, what is the bounded language? There are fixed words w 1 w w 2 like that w k fixed words such that, the language will be of this form, i 1 i 2 i k they are integers having some relationship.

Now, if the language is of this form, it is called the bounded language. And given a bounded language, there are algorithms which will tell you whether it is inherently ambiguous not. If L is a bounded language, bounded C F L there exist an algorithm which tells us whether L is inherently ambiguous or not. But we will not go into the details of the algorithm it involves some other definitions, some other concepts and so on. So this is the result which will take for grant.