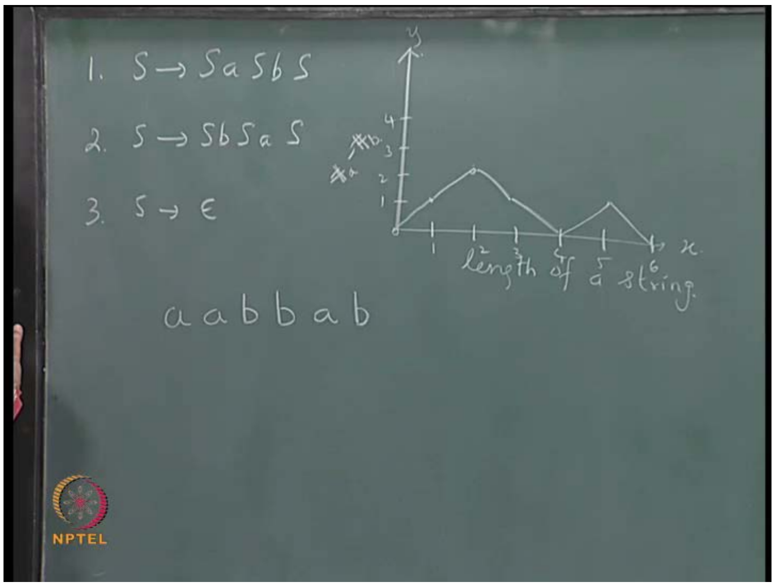
Theory of Computation

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

Lecture No. # 04

Ambiguity IN CFG

(Refer Slide Time: 00:23)



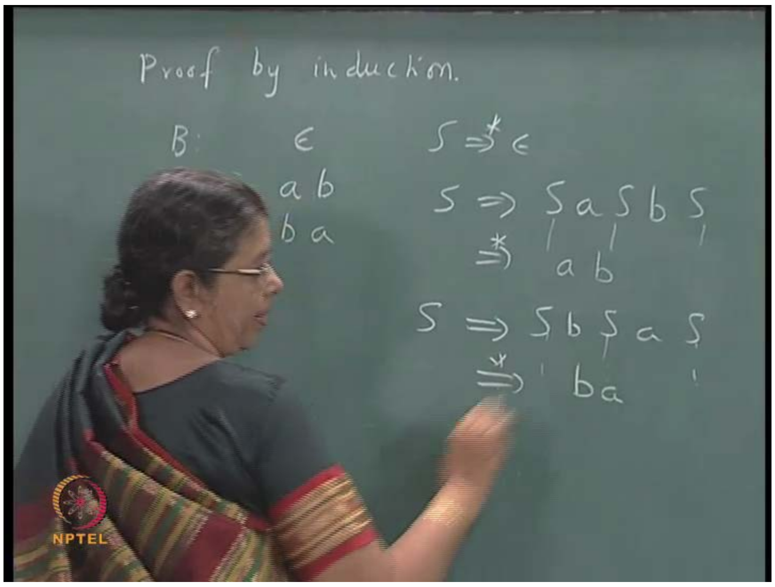
We were considering some examples of languages, the grammars generating them etcetera. So, this particular example we consider in the last lecture itself, the grammar has only one non terminal S, and two terminal symbols a and b. There are three rules; S goes to S a S b S, S goes to S b S a S, and S goes epsilon. It generates strings having equal number of a(s) and b(s). In one way it is very easy, whatever string is generated by the grammar it has it has got equal number of a(s) and b(s). This is very easy to realize, because whenever you apply this rule 1 a and 1 b will be generated, and whenever you use this rule 1 b, 1 a will be generated, and when you use this rule, no a(s) or no a b(s) are generated.

So, ultimately whatever string you generate it will have equal number of a(s) and b(s), but you should also show that any string having equal number of a(s) and b(s) will be generated by this grammar. So, let us see how to go about showing that, you draw a graph with x axis denoting x denoting the length of a string length of a string and the right hands and the y axis it denotes the number of a (s) minus the number of b (s). Suppose, I take a string like this a a b b a b, the length of the string is 6; 1, 2, 3, 4, 5, 6, and as you go along the string first you have you read 1 a, it start from here, this a 1, 2, 3, 4, 1, 2, 3, 4, 5, 6, 1.

When you read the first portion; first symbol alone, the number of a(s) is 1 more than the number of b(s). So, the graph will be like this, then the first two a(s) when you read, the number of a(s) is 2, number of a(s) minus number of b(s) will be 2 minus 0 2, so it will be like this. Then when you read this b up to this point the length of the string is 3, and the number of a(s) minus the number of b(s) is 1, so it will taper down like this. Then when you read the first four symbols number of a(s) minus number of b will be 0, so the graph will be like this. Then when you read the first four symbols, number of a(s) minus number of b(s) will be 3 minus 2 1, and when you read the whole string the number of a(s) minus number of b(s) is 0. So, you can draw a graph like this. So, any string if you take, if it has got equal number of a(s) and b(s), it will start here and ultimately end on the x axis, but it may go down also, sometimes the number of b(s) may be more. So, it may go below the graph can be something like this, or it can be like this, or it can be like this.

But, it starts at the origin and then ends on the x axis, if it has got equal number of a(s) and b(s).

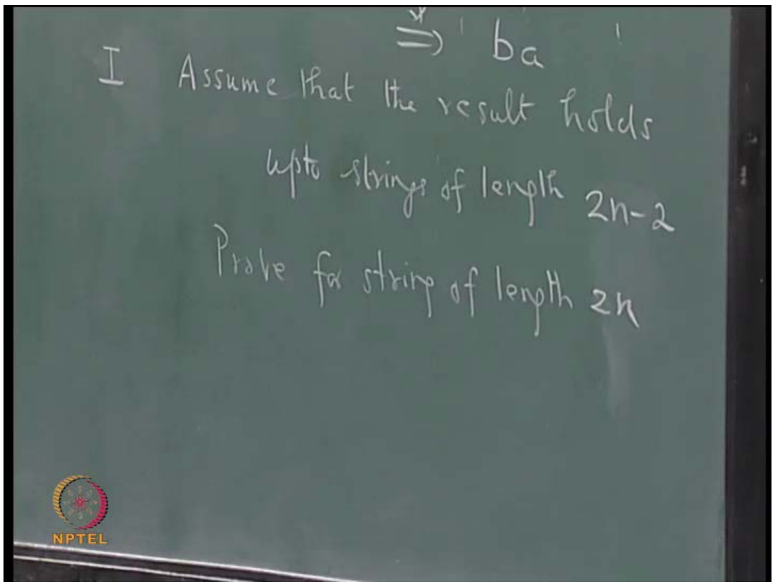
(Refer Slide Time: 04:18)



So, take w again proof by induction proof by induction base is epsilon if you take it has got 0 number of a(s) and 0 number of b(s), and it is derivable from S, you can derive epsilon from S. Then if it takes strings of length a to a b and b a are the strings of length two having equal number of a(s) and b(s), we can see that you apply the first rule S goes to S a S b S, then make all of the S(s) go to epsilon you will get a b, make all the S(s) go to epsilon you will get a b right.

So, a b can be generated by the grammar, then S goes to S b S a S, then make all the S(s) go to epsilon, you will get b a, so b a also can be generated. So, the bases portion you have proved, that is string’s epsilon a b b a can be generated.

(Refer Slide Time: 05:40)



$\Rightarrow$ ba

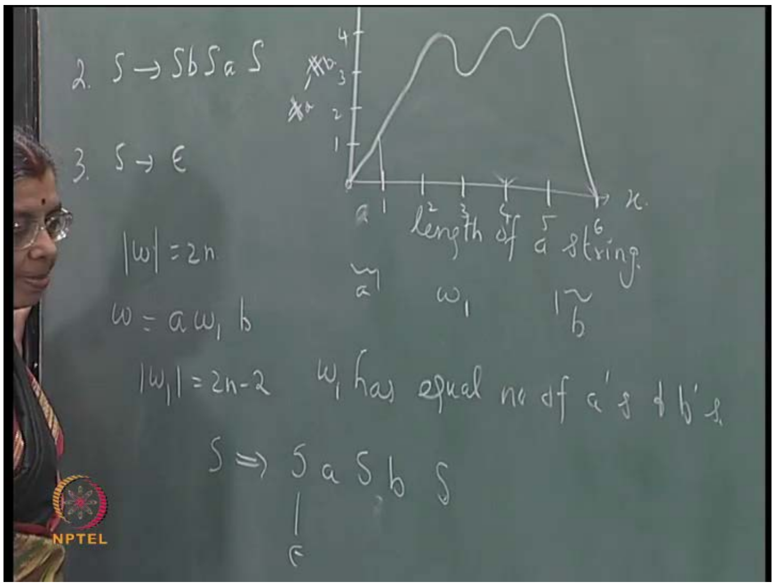
I Assume that the result holds upto strings of length 2n-2

Prove for string of length 2n

So, the induction portion, assume that the result holds up to strings of length n minus 1. Prove for string of length n. Actually, you will be reading only even length strings, when you have equal number of strings you need even length strings.

So, I can say that, assume that the result holds up to strings for of length 2n minus 2, then you prove for strings of length 2n.

(Refer Slide Time: 07:00)



2. $S \to SbSaS$

3. $S \to \epsilon$

$|w| = 2n$

$w = a w_1 b$

$|w_1| = 2n-2$ w_1 has equal no of a's & b's

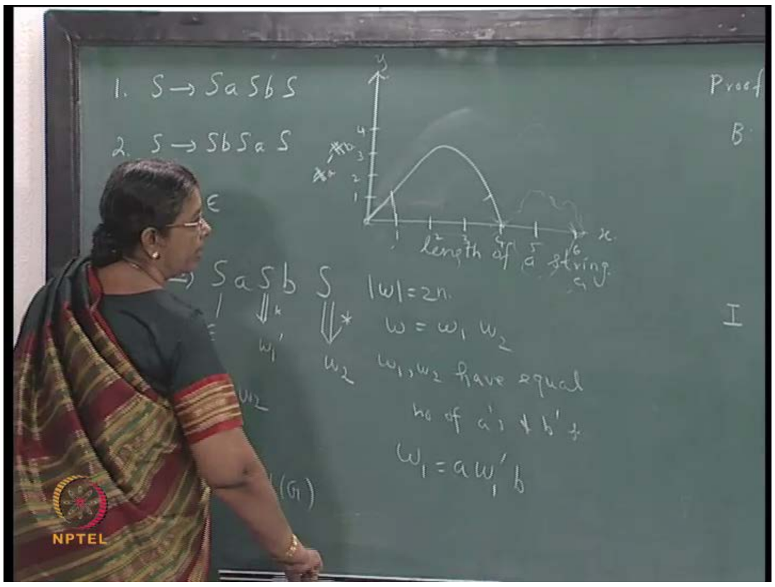
$S \Rightarrow SaSbS$

ϵ

Now, if you have a string w if you have a string w and the length of w is 2n, then draw a graph like this pointing like this, we will consider three possibilities. Actually one possibility, two possibilities we will consider and elaborate, the other one will be other will be similar. First is... It does not... It can be like this and then it does not cross the x axis in between, but towards the end it touches the x axis, the graph can be something like this. What does that mean? The first symbol is a, the first symbol has to be a, the last symbol has to be b. If the graph is like this, the first symbol has to be a, the last symbol has to be b. And in between you have a string w 1, and what can you say about the length of w 1? In this case w is of the form a w 1 b, and the length of w 1 is 2 n minus 2, right and w 1 has because w has equal number of a(s) and b(s), 1 a you have taken out, 1 b you have taken out.

W 1 has equal number of a(s) and b(s). So, what you can do is, from S you can derive S a S b S, and you can make this go to epsilon, make this S go to epsilon, but from this S you can derive w 1 by induction hypothesis. By induction hypothesis from this S you can derive w 1, so w equal to a w 1 b belongs to the language. And exactly similar proof you can give,

(Refer Slide Time: 09:24)



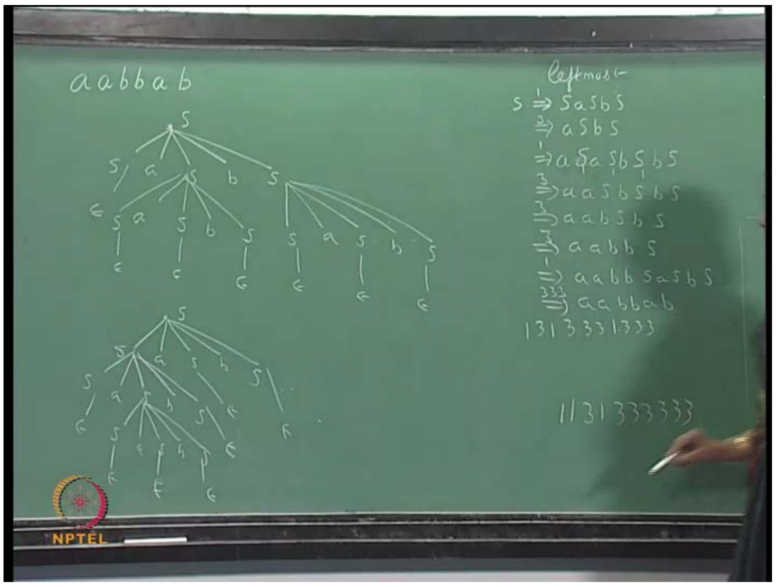
if the graph is of this form. It does not cut the x axis, but it is below it lies below the x axis, what does that mean? It means that the first symbol has to be a b, the last symbol has to be a a right. So, in this case, w you can write as b w 2 a, and the length of w 2 is 2 n minus 2, and w 2 has equal number of w 2 also has equal number of a(s) and b(s). So, what you can do is, you can start with S derive S b S a S, from this, and from this S derive epsilon, from this S derive epsilon, but from the middle S you derive w 2.

So, w equal to b w 2 a belongs to l of g. So, two cases we have considered, some two more cases we will consider. Suppose, it cuts the x axis, it touches the x axis at some point, and then it can be like this or it can go below this does not matter. It goes above the x axis and touches the x axis in the middle, then there is some other portion it may lie above or below does not matter. In this case, we have taken w to be of length 2n, in this case, w can be written in the form w 1 w 2, this corresponds to w 1, this portion corresponds to w 1, this corresponds to w 2. And what can you say about w 1 and w 2? w 1 and w 2 both have equal number of a(s) and b(s). w 1 w 2 have equal number of a(s) and b(s) . And the first portion is like this, so the it begins with the and here it ends with a b, w 1 starts with a a and ends with a b, is not it?

So, use this rule, S goes to S a S b S, and make this S go to epsilon, and by induction hypothesis from this S, you can derive w 2. And w 1 itself, you can write as a w 1 dash b, because this portion is like this, so it has to start with a and end with a b. So, from this S you can derive w 1 dash, so ultimately the string derived is a w 1 dash b w 2, which is nothing but w 1 w 2 or just w, and that belongs to the language. On the other hand in a similar manner you can prove, if the graph starts below the axis at some point it reaches here, then there is another portion which is either above or below the x axis that does not matter, you can give a similar argument.

But, the first rule you will be using is rule number two instead of rule number one, because in this portion the first symbol will be a b, and the last symbol will be a. So, this way you can show that, any string which is having equal number of a(s) and b(s) is derivable in this grammar.

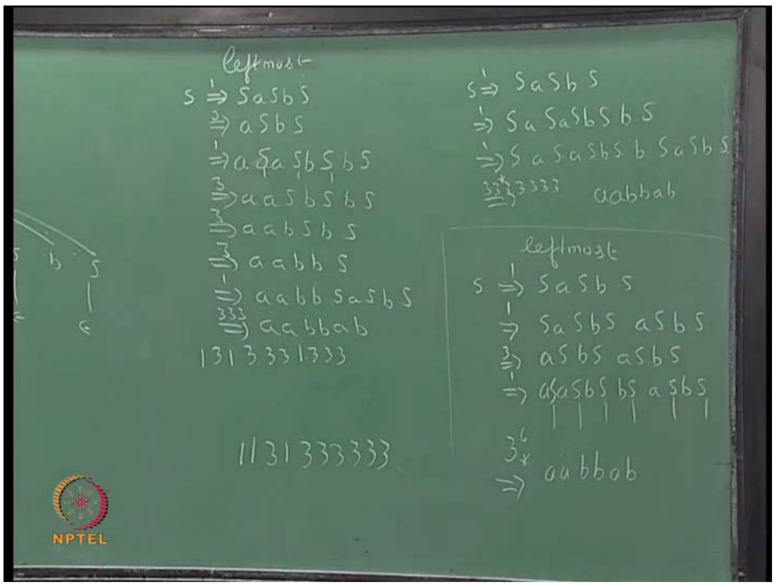
(Refer Slide Time: 13:55)



Now, let me take a particular string and derive it in different ways, let me take this string a a b b a b.

So, a derivation tree you can have like this for this, S goes to S a S b S, and this can go to epsilon, this again can go to a S a S b S, this goes to epsilon, this goes to epsilon, this goes to epsilon, this can go to S a S b S, and this will go to epsilon, this goes to epsilon, this goes to. This is the generation tree or a derivation tree for this. Now, I will write a derivation, which is leftmost and another one which is not leftmost.

(Refer Slide Time: 15:06)



So, a leftmost derivation is this S goes to S a S b S, and then the first S goes... you are using rule number one then rule three, so that this S goes to epsilon, so you get a S b S, then again you apply rule one, this one a S a S b S, then b S, then you apply rule three for repeatedly 3, 3, 3, three times. So, this will go to epsilon, this will go to epsilon, this will go to epsilon, one by one. The leftmost one has to (( )) I will write step by step a a S, this S has gone to epsilon.

Now, this S is again expanded using rule one, one by one the S(s) are erased. So, three times applying that the string is, (( )) this is the leftmost derivation. So, in this step, first this is erased using rule S goes to epsilon, then this S is removed using S goes to epsilon, then this S is removed. You can have another derivation which is not leftmost for example, you can have S a S b S, then this S is expanded as S a S b S b S, then you are using rule 1 S a S a S b S b then this is expanded as S a S b S . So, then 1, 2, 3, 4, 5, 6, 7 S(s) 1 by 1 any order you can remove using rule three removed and then you will get this is star these S(s) can be removed in any order.

So, you find that for this derivation tree, this is the leftmost derivation and you can have many derivations which are not leftmost or any order you can replace the S(s) and at ultimately to get this string a a b b a b . So, the correspondence between derivation trees and derivations is not 1 to 1 correspondence. For 1 derivation tree you can have a leftmost derivation you can have a rightmost derivation you can have several derivations which are neither leftmost nor rightmost.

So, for one derivation tree you can have several derivations, but what is the connection between one derivation tree and one leftmost derivation. For one derivation tree there will be only one leftmost derivation and for one leftmost derivation there will be only one derivation tree. This is the bijection the correspondence between derivation trees and leftmost derivations is a bijection.

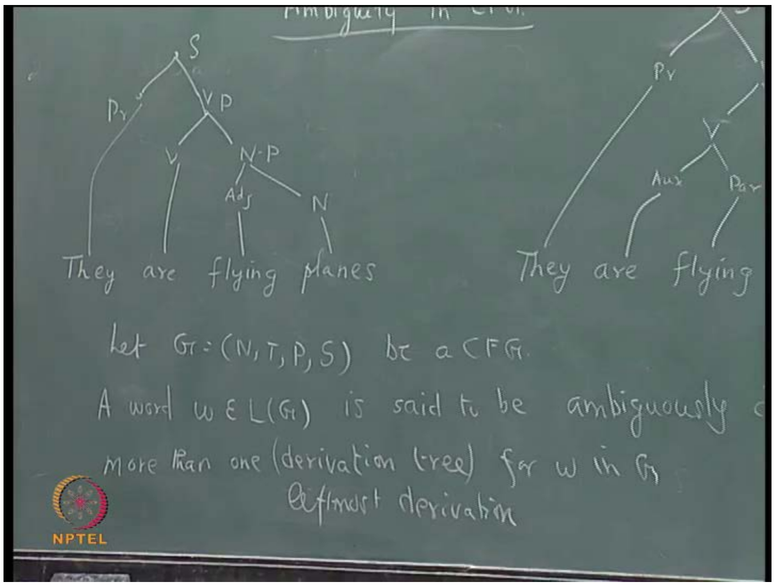
But, the correspondence between derivation trees and all derivations it is not bijection. Now, let me draw another derivation tree. (No audio 19:30 to 20:22) Consider this derivation tree what is the result of this derivation tree what is the string derived here you must read the leaves from left to right epsilon you can omit. So, it will be a a b b a b what is the string the same string a a b b is derived right, but that derivation trees are different here in this leftmost derivation what is the sequence of rules applied 1 3 1 3 3 3 1 3 3 3 see this is sequence in which you have applied the rules.

Now, let me write the leftmost derivation for the other one for this tree leftmost derivation how do you apply S derives S a S b S that is rule one, then the first S is expanded is not it? So, again you use rule 1 S a S b S a S b S right, then this S is removed. So, you get a S b S a S b S then this S is expanded. So, use rule 1 a a S b a S a S b S b S a S b S.

Now, all this S(s) have to be removed 1 by 1 the leftmost first remove this then remove this then remove this then remove this then remove this and so on. So, six times you can apply three and get a a b b a b. So, the sequence of rules you apply here is 1 1 3 1 3 3 3 3 3 3 this is the leftmost sequence in which you have applied the rules. These two derivation trees they generate the same string a a b b a b. Again for this you may have 1 leftmost derivation one rightmost derivations several derivations which are neither leftmost nor rightmost and so on, but the leftmost derivation corresponding to this is the sequence in which you apply rules like this right. The leftmost derivation for this is this and for this sequence of rules for this leftmost derivation, this is the derivation tree. For this leftmost derivation this cannot be the derivation tree. So, for this leftmost derivation this is the derivation tree and for this derivation tree this is the leftmost derivation.

So, it’s a bijection and similarly for this derivation tree this is the leftmost derivation and for this leftmost derivation that is the derivation tree in a bijection. So, having basically this idea we well go on to consider what is ambiguity in context free grammars.

(Refer Slide Time: 24:14)



Ambiguity in CFG

let $G = (N, T, P, S)$ be a CFG.

A word $w \in L(G)$ is said to be ambiguously

more than one (derivation tree) for w in G

leftmost derivation

So, next we go on to ambiguity in context free grammars or type two grammars. We saw how to parse English sentences let us take one English sentence and parse it. They are flying planes how can you parse this sentence.

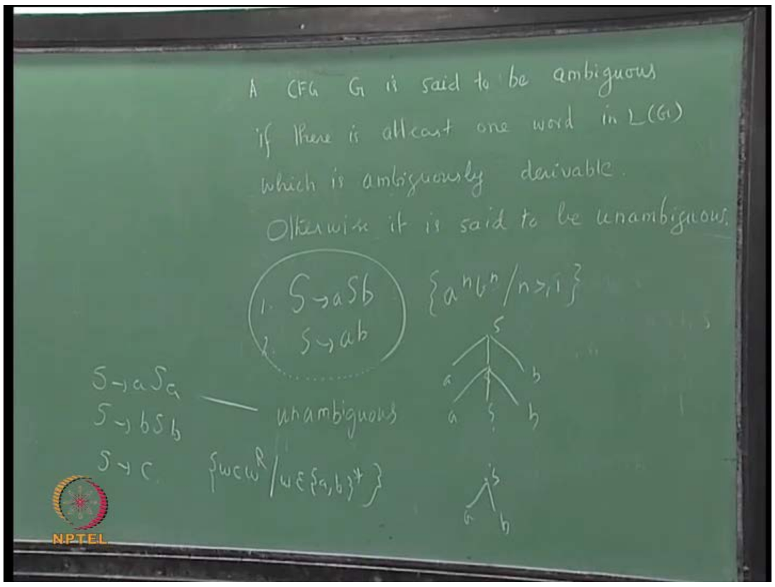
Actually, this sentence can be parsed in two different ways. In one you have like this sentence goes to pronoun verb phrase pronoun goes to the verb phrase goes to verb and noun phrase; verb goes to are noun phrase goes to adjective and noun; noun goes to plane adjective goes to flying, you can parse it in this way. In another way also you can parse this sentence and that is start with S pronoun verb phrase pronoun is replaced by then, but verb is replaced by verb and noun; noun goes to planes verb itself auxiliary and participle and it goes to are and flying . So, this sentence can be parsed in two different ways and you can have two different derivation trees and that gives you two different meanings. In this one they refers to what they refers to the planes. In this sentence if you parse it in this way they refers to the people in the plane person sitting in the plane they are flying planes.

So, the meaning differs and the difference in the meaning comes because you are able to parse the sentence in two different ways. This is called ambiguity and the ambiguity arises because you are able to have two different derivation trees. In general, when you write a grammar for a programming language we, will consider some expressions and so on, you would like to avoid ambiguity ambiguity has to be avoided because when you generate the code if it is ambiguous two types of codes can be generated and you do not know which is correct and so on.

Sometimes that leftmost evaluation left to right evaluation will take error even though the grammar is ambiguous in any expression you know that it has to be evaluated from left to right you put the restrict. Such restrictions will make the code unambiguous even though the grammar is ambiguous we will come to that in a moment.

So, the ambiguity or the meaning is different because you are having two different derivation trees and because of the correspondence between leftmost derivation and derivation trees ambiguity can be derived in terms of derivation trees or in terms of leftmost derivation both is equivalent. So, you define like this. Let G is equal to N, T, P, S you define a grammar with four components like this be a CFG. A word or a string a word w belonging to L(G) is said to be ambiguously derivable derivable if there exist more than one derivation tree for w in, this is the definition of ambiguity. A word w belonging to L(G) is said to be ambiguously derivable if there exist more than one derivation tree for w in G. You could equally define it as instead of derivation tree more than one leftmost derivation for w in G either way it can be defined both are equivalent.

(Refer Slide Time: 30:43)



A CFG G is said to be ambiguous if there is atleast one word in L(G) which is ambiguously derivable. Otherwise it is said to be unambiguous.

1. $S \to aSb$
2. $S \to ab$

$\{a^n b^n / n \geq 1\}$

$S \to aSa$
$S \to bSb$
$S \to c$

unambiguous

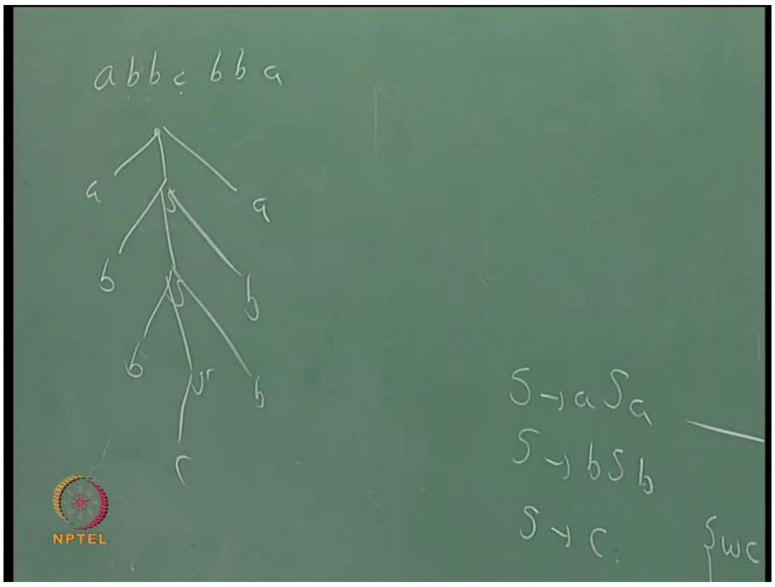
$\{wcw^R / w \in \{a,b\}^*\}$

Now, having defined this you can very easily see that this grammar in this grammar a a b b is ambiguously derivable because it is having two derivation trees two or more. When do you say that a context free grammar is ambiguous? A CFG G is said to be ambiguous if there is atleast one word in L(G) which is ambiguously derivable. A CFG G is said to be ambiguous if there is atleast one word w it can be ambiguously derivable in that grammar; otherwise it is unambiguous. Otherwise, it is said to be unambiguous. The grammar is said to be unambiguous.

Obviously, the grammar which we considered just now is ambiguous and look at this grammar S goes to a S b; S goes to a b what is the language generated by this? a power n b power n n greater than r equal to one. If you take any word the derivation tree will be like this S goes to a S b a S b like that and the last one will be a b; n a(s) and then n b(s). Any tree will be of this form there is only one if you take a power n b power n there will be only 1 tree for that.

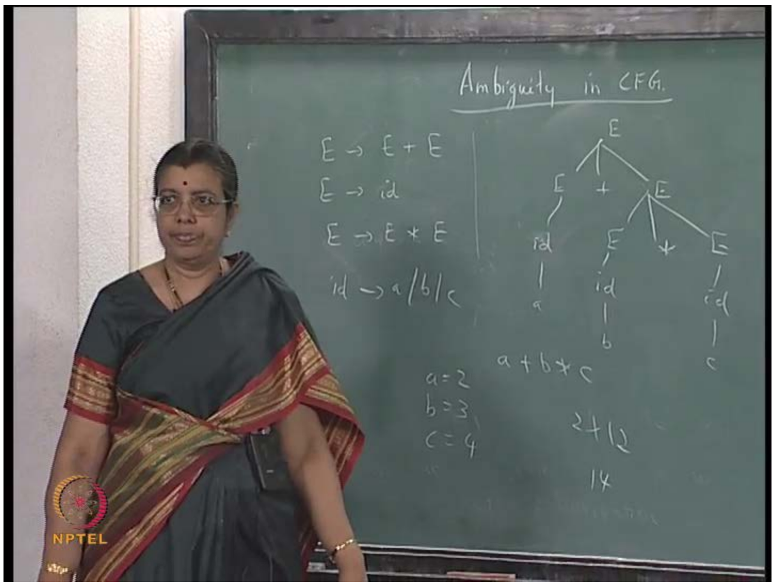
So, this grammar is unambiguous and another example is this one which we considered earlier S goes to a S a; S goes to b S b; S goes to c what is the language generated? The language generated consists of strings of the form w c w r where w belongs to a b star.

(Refer Slide Time: 34:10)



If you take any string in this take for example, a b b c b b a w c then the reversal of this. Here the first symbol is a you have to apply a S a; second symbol is b. So, you have to apply b S b; third symbol is b. So, you have to apply b S b, there will be only one derivation tree for each word. So, this is another example of a grammar which is unambiguous.

(Refer Slide Time: 35:21)



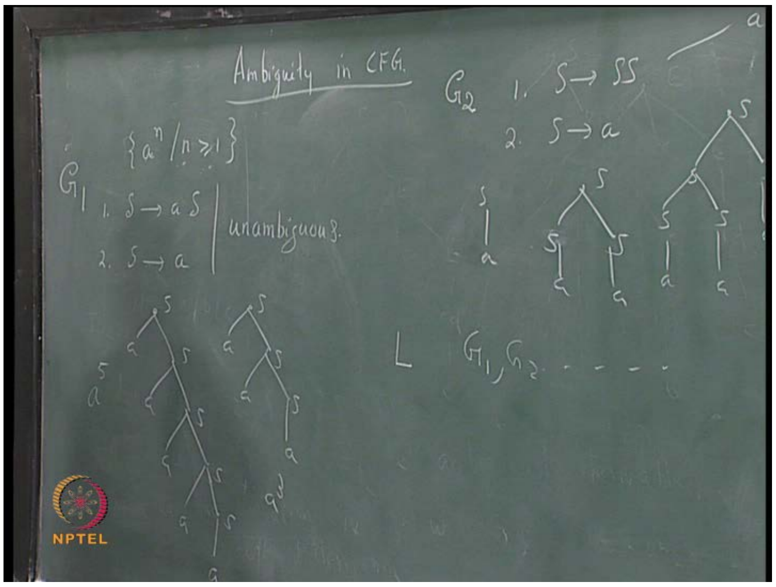
So, the problem with arithmetic expressions compilers it will come like this. Suppose, I have E goes to E plus E; E goes to a and this if you consider a plus a plus a you or I will add one more rule E goes to E star E, then you can have E goes to E plus E E star E and each one of them going to a an identifier a or b or c or whatever it is, Id and then Id can go to a b c may be I can write like this or you can also have it like this. In both cases the string derived is a plus b star c this is the string derived.

Now, if we evaluate this suppose I say a is equal 2, b is equal to 3, or c is equal to 4; a is equal to 2, b is equal to 3, c is equal to 4, this will be first the star operation will be performed. So, b into c 12 then added to 2 it will give 14 whereas, in this case a is 2, b is 3 they will be added first that will be 5, 2 plus 5; 5 into c is 4, 5 into 4 20.

So, the evaluation will be like this star and 4 plus 2 3. So, 5 multiplied by 4 and this way it will give 20. So, it is very essential you cannot write grammar like this it is very essential that the grammar should be unambiguous this is because the grammar is ambiguous and for the same string a plus b star c you are having two different derivation trees right, but you can have this grammar and usually we what while evaluating you say that star has higher priority than plus and then the evaluation will be from left to right. You put some restrict some more restrictions this can be used.

But, without those restrictions these two represent different trees and the evaluation will be even though they as a string string generated is this a plus b star c, two types of evaluations becomes possible that is two types of codes can be generated which has to be avoided. So, we saw when a word is ambiguously derivable and when a grammar is said to be ambiguous.

(Refer Slide Time: 39:13)

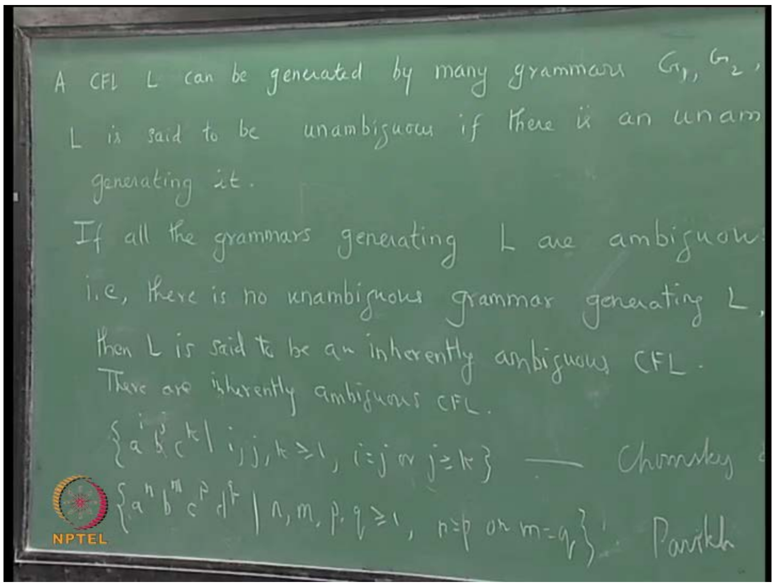


Now, let me consider this language a power n n greater than r equal to a sequence of a(s) a a a a like that. Look at this grammar which is of type 3 S goes to a. There are two rules S goes to a S and S goes to a, and this is a type tree grammar and it generates this language. Suppose, I want to have the derivation tree for a power 5 say it will be like this a S a S a S a S a this generates a power 5. For a cubed S goes to a S a S this generates a cube. So, any string a power n if you take there will be only one derivation tree.

So, what can you say about this grammar? It is unambiguous. Now, if you consider this grammar what is the language generated by this grammar? S goes S(s); S goes to a the language generated is the same, but this is not a type 3 grammar this is type 2 grammar. The language generated by this grammar is the same a power n n greater than or equal to 1. Now, let us consider some derivations S goes to a, a will be generated like this a squared will be generated like this. Now, if you come to a cubed you can have a derivation tree like this or we can have a derivation tree like this. For a cubed you can have two different derivation trees.

So, this grammar is ambiguous, but the language generated is this power n n greater than or equal to 1 for which you are having a grammar G 1 which is unambiguous another grammar G 2 which is ambiguous. So, if you take a language L there may be several grammars for that G 1, G 2, etcetera. Many grammars may generate the same language L. Some of them may be ambiguous some of them may be unambiguous.

(Refer Slide Time: 43:14)



A CFL L can be generated by many grammars G_1, G_2,

L is said to be unambiguous if there is an unam

generating it.

If all the grammars generating L are ambiguous

i.e, there is no unambiguous grammar generating L,

then L is said to be an inherently ambiguous CFL.

There are inherently ambiguous CFL.

$\{a^i b^j c^k \mid i, j, k \geq 1,\ i = j \text{ or } j = k\}$ — Chomsky

$\{a^n b^m c^p d^q \mid n, m, p, q \geq 1,\ n = p \text{ or } m = q\}$ Parikh

If atleast one grammar is unambiguous, the language is unambiguous. If all the grammars are ambiguous L is said to be inherently ambiguous. A CFL L can be generated by many grammars G 1, G 2, G 3. L is said to be…

See, after those grammars some of them may be ambiguous some of them unambiguous. So, L is said to be unambiguous if there is an unambiguous grammar. There is atleast one unambiguous grammar of course, unambiguous grammar. By grammar I mean type 2 type 2 or CFG generating it. If all the grammars generating L are ambiguous that is in other words you can say that there is no unambiguous grammar generating L, then L is said to be an inherently ambiguous an inherently ambiguous context free language.

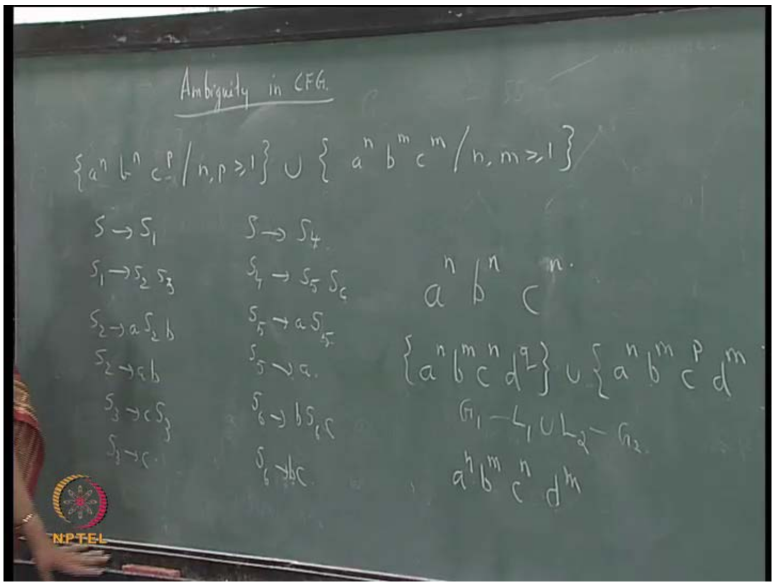
So, if there are several grammars generating L if there is at least one which is unambiguous the language is unambiguous if all the grammars are ambiguous the language is said to be inherently ambiguous. There are this language you take there is one grammar which is unambiguous, there is one grammar which is ambiguous. So, it is unambiguous language language is unambiguous. There are inherently ambiguous context free languages. There are inherently ambiguous CFL example is a power I; b power j; c power k; i j k greater than or equal to 1; i is equal to j or j is equal to k this.

We considered this earlier the languages a power I, b power j, c power k, i j k greater than or equal to 1 either i is equal to j or j is equal to k. Another example is this a power n, b power m, c power p, d power q; n, m, p, q greater than or equal to 1; n is equal to p or m is equal to q. To show some context free language is inherently ambiguous is not easy the proof is quite involved, you have to show that any grammar which generates has to be ambiguous that is not a very easy thing to prove. You can intuitively see you can intuitively see that this is the what happens.

But intuition is different from rigorously proving the proves are quite involved. In fact, this was the first language which was shown to be inherently ambiguous by Parikh and then this was next it was this was shown to be inherently ambiguous by Chomsky and and spelling may be slightly Chomsky and Schutzenberger hope this spelling is correct.

Now, let us see why they are inherently ambiguous. Please intuitively rigorous proof we will not go into because it is quite involved.

(Refer Slide Time: 49:24)



Ambiguity in CFG.

$\{a^n b^n c^p / n, p \geq 1\} \cup \{a^n b^m c^m / n, m \geq 1\}$

$S \to S_1$ | $S \to S_4$
$S_1 \to S_2 S_3$ | $S_4 \to S_5 S_6$
$S_2 \to a S_2 b$ | $S_5 \to a S_5$
$S_2 \to ab$ | $S_5 \to a$
$S_3 \to c S_3$ | $S_6 \to b S_6 c$
$S_3 \to c$ | $S_6 \to bc$

$a^n b^n c^n$

$\{a^n b^m c^n d^q\} \cup \{a^n b^m c^p d^m$

$G_1 - L_1 \cup L_2 - G_2$

$a^n b^m c^n d^m$

But, let us see intuitively why it is inherently ambiguous. We have considered this in the last class a power n, b power n, c power p, n p greater than or equal to 1, this is actually the first one you can write like this a power n, b power m, c power m, n m greater than or equal to 1. So, either this is equal to this this is equal to this or this is equal to this. So, the grammar will be S goes to S 1; S goes to S 2 and then S 1 goes to S 2, S 3 which generates this portion S 2 goes to a S 2 b; S 2 goes to a b; S 3 goes to c S 3; S 3 goes to c and here I am sorry S 2 we have used. So, this let me use S 4 here S 4; S 4 goes to S 5, S 6; S 5 generates many a(s), a S 5; S 5 goes to a; S 6 goes to b S 6 c; S 6 goes to b c.

So, this portion generates this sort of strings and this portion generates this. Now if you take a string of the form a power n, b power n, c power n where the number of a(s) is equal to the number of b(s) is equal to the number of c(s), then there will be one derivation for that this and there will be one derivation in this. So, there will be two different derivations for strings of the form the derivation structure of that (( )) tree structure will be different, but if they are see if I have something like this where p is not equal to n then there will be only one derivation.

If I have something like this, there will be only one derivation, but when you have equal number of a(s), equal number of b(s) and equal number of c(s) in any grammar generating this is I am giving particular grammar and showing that this is generates (( )), but in general see that is not enough if I show if I give a one grammar and show that it is ambiguous that is not showing that it is inherently ambiguous. It involves much more some results also.

But, the intuitive idea is when the language is of this form there will be two derivations possible for such things one in this and one in this. Others type of strings will have only one derivation. Similarly, the other one when you said a power n, b power m, c power n, d power q this is one portion where a(s) and c(s) are equal the other portion is a power n, b power m, c power p, d power m, b(s) and d(s) are equal. This is the other language due to Parikh.

Now, if you look at this language you can write it as L 1 union L 2 and try to give a grammar for L 1 and try to give a grammar for L 2. So, L 1 will generate where you have equal number of a(s) and equal number of c(s). Number of b(s) and d(s) can be different. L 2 will generate equal number of b(s) and equal number of d(s), but a and can be a(s) and c(s) can be anything, but if you have a string of the form a power n, b power m, c power n, d power m, where the number of a(s) and c(s) are equal also number of b(s) and d(s) are equal, then that will have one derivation in this grammar L 1 is generated by G 1, L 2 is generated by G 2 say then some portion G 1, G 2 together will form a grammar generating this language.

So, there will be one derivation which make sure that the number of a(s) and c(s) are equal and there will be one derivation where it make sure that the number of b(s) and number of d(s) are equal. So, two different derivations you will get for a string of the form. Other type of strings will have only one derivations and any grammar generating such a language will have this feature. Actually, you a have to make sure that the number of a(s) and c(s) are equal in some some strings that should be done by one part and another part will make sure that the number of b(s) and number d(s) are equal that is another part. So, any grammar generating this will have this feature and so, this language is inherently. This is only intuitive argument it is not a proper argument, proper argument is very lengthy and involved.

Now, the last question is given a grammar how do you find out whether it is ambiguous or not. Suppose, the grammar has 100 rules or 50 rules how will you find out? How will you write an algorithm for this? Given a grammar how do you find out whether there it is a ambiguous or not. The answer is it is not possible to write an algorithm, finding out whether a given grammar is ambiguous or not is an undesirable property.

So, the last result we study are this we will prove later on toward the end of this course. It is undividable to find whether a CFG is ambiguous or not. So, one of the results due to turing turing halting problem and decidability can be reduced to this or actually it can be reduced to what is known as post correspondence problem and the post correspondence problem can be reduced to this, so this undividable property. So, with this we have learnt little bit about ambiguity. In the next class we shall see some reductions.