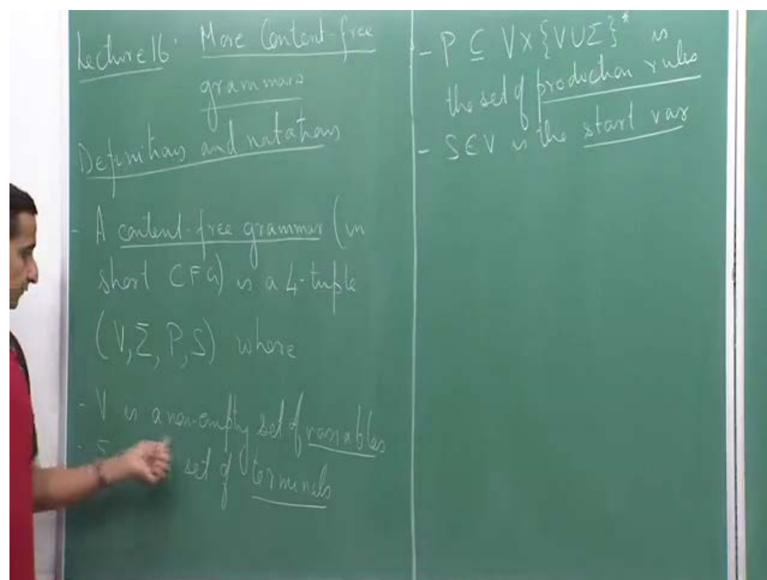
Theory of Computation

Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 16
Examples of CFGs, Reg subset of CFL

Welcome to the 16th lecture of this course. Today, we will start by giving the formal definition of a Context Free Grammar and how we formally define acceptance by a context free grammar then we look at some more examples. And finally, we will conclude by showing that the set of regular languages form a subset of the class of languages accepted by context free grammars.

(Refer Slide Time: 00:47)



So, definitions and notations; so in short CFG is a 4 tuple, write it here is a 4 tuple V, sigma, P, S, where V is a non empty set of variables; sigma is the set of terminals. So, you just underline. So, these are the formal terminologies. So, we have variable, we have terminal.

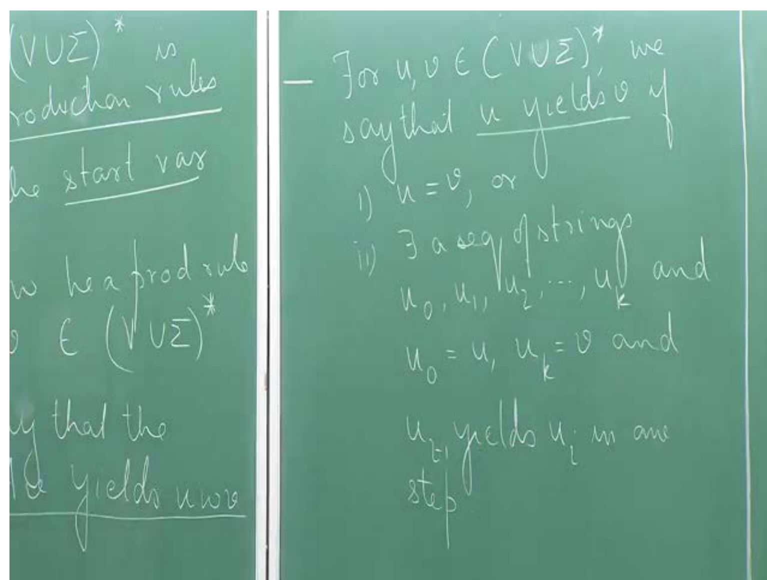
P is a subset of the relation V cross V union sigma star is the set of production rules often these are just called a rules instead of saying the whole thing production rules. And S belonging to V is the start period. So, both these sets V and sigma are finite sets we only look at finite sets. V is non-empty, because it must contain the start variable at least is has one variable which is the start variable. Sigma is the set of terminals.

As I said last time, that we usually denote the variables with capital letters and the terminals with either small letters or a numeric symbols. Now this is slightly important. So, we have the production rules. So, we denote the production rules as a subset of the relations from V to V union sigma star. So, we do not write it as functions from V to V union sigma star.

The reason for this is note that we can have multiple production rules corresponding to one variable. So, for A variable we can I mean even has 0 production rules or we can have one or more than one production rules. So, to accommodate for that fact, we have this as a relation instead of A functions it is a relation from V cross V union sigma star. So, one variable gets map to a string from which consists of symbols from V and sigma and the start variable is of course, there.

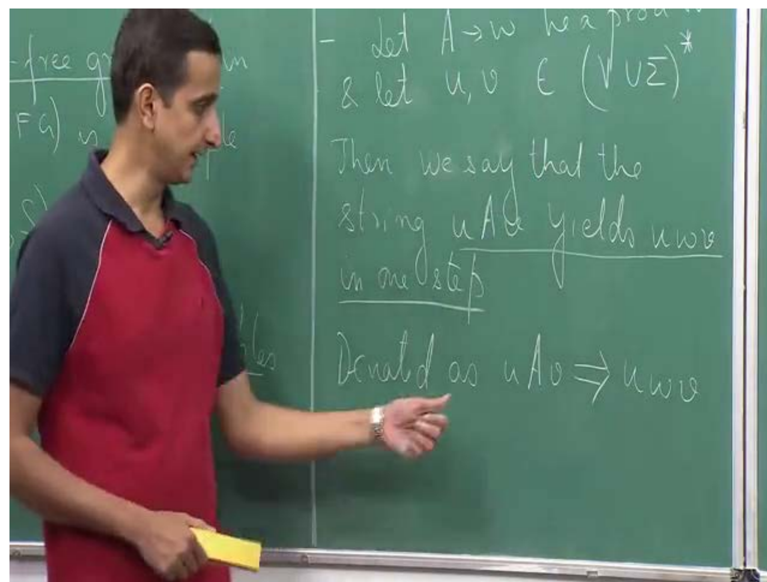
Let A going to w be a production rule, and let u and v be strings in, so I will just write it in A short hand notation. So, u and v are strings over V union. So, once again I should not write this is a set this is so V union sigma itself is a set. So, I will just write it with round bracket. So, A to w is a production rule and we have u and v which are strings over variables and terminal symbols. Then we say that the string u v yields u w v in one-step. Essentially, what this means is that if I have a variable in some string, let say u A v replacing that variable with one of its production rules gives me a certain string, so that is what means to say that some string yields some other string in one-step.

(Refer Slide Time: 08:12)



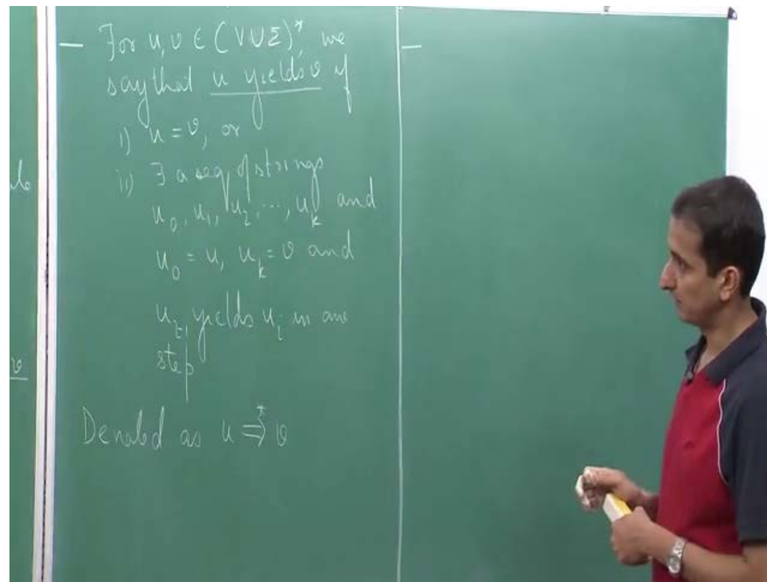
Now we can generalize this definition. So, for u v in V union sigma star, we say that u yields v, if either they are the same strings or there exist a sequence of strings u 0, u, 1 u 2 and so on till u k. And u 0 equals u, u k equals v; and u i yields or u i minus 1 yields u i in one-step. So, essentially this is generalizing this two multiple steps. So, we say that a string u yields a string v, if basically I can go from u to v in 0 or more steps. So, if u and v are the same then it means going in 0 steps. Otherwise, I can just have A sequence of strings u 0 equal to u, u k equal to v - which are not the same. And I go from u 0 to u 1 in one-step, u 1 to u 2 in one-step, u 2 to u 3 and so on till u k each in one-step.

(Refer Slide Time: 10:38)



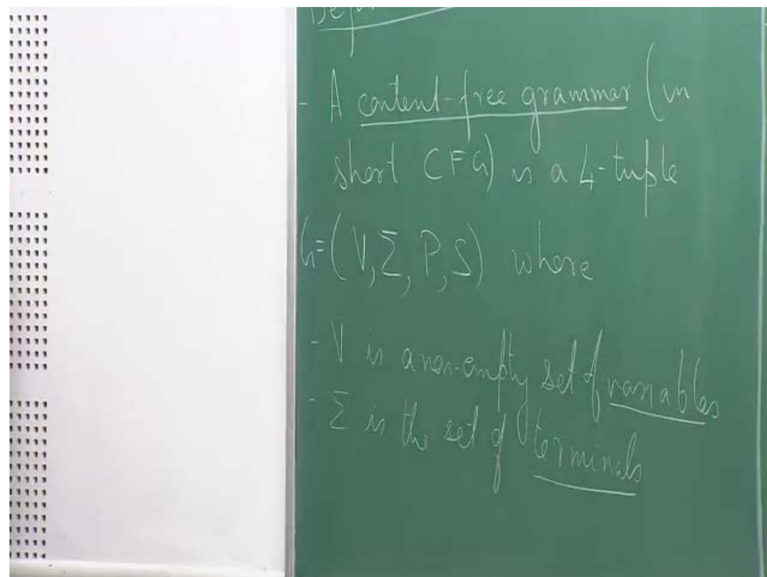
So, here we denote this as so this is denoted as u A V with a double arrow going to u w v. So, we have yield in one-step, it is in indicated with the double arrow.

(Refer Slide Time: 11:00)



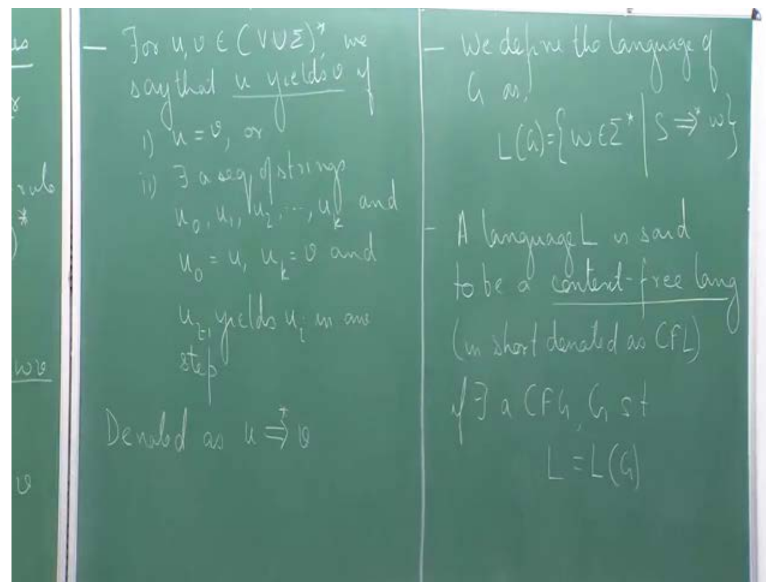
And when we have yield in multiple steps, so this is denoted as u with A star on top with A star over here going to v. So, this means that from u, I can go to v in 0 or more steps. Now that we have a grammar; let me do one thing, let me just go to the definition of this grammar and give this a name.

(Refer Slide Time: 11:40)



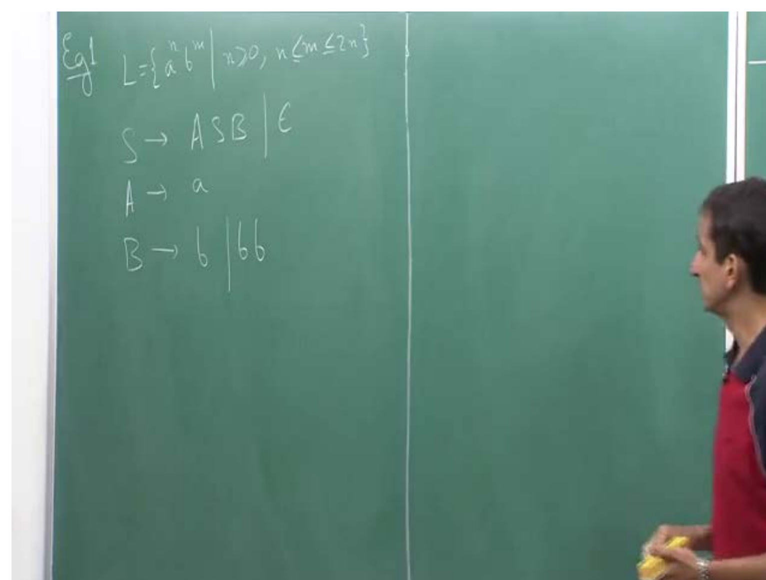
Let us call this grammar G. So, we have this grammar V, sigma, P, S.

(Refer Slide Time: 11:52)



We define the language of G as so it is denoted as L of G. And it is the set of all strings w in sigma star consisting only of terminals such that S yields w. So, all those strings over terminals that we can generate starting from the start symbol. And finally, a language L is said to be a context free language if there exist a, so this is in short denoted as CFL. So, language L is said to be a context free language, if there exist a context free grammar G such that L equals L of G. So, all those languages for which we have a context free grammar are known as context free languages.

(Refer Slide Time: 14:13)



Let us look at some examples. So, the first example that we will see today is let us look at the language L which has a sequence of a(s) and followed by a sequence of b(s) such that n is greater than or equal to 0, and m is greater than or equal to n, but less than or equal to 2 n. So, the number of b(s) is at most twice the number of a(s), but more than the number of a(s), so it lies between n, and 2 to the power n. How do we create A language for this or how do we create a grammar for this language?

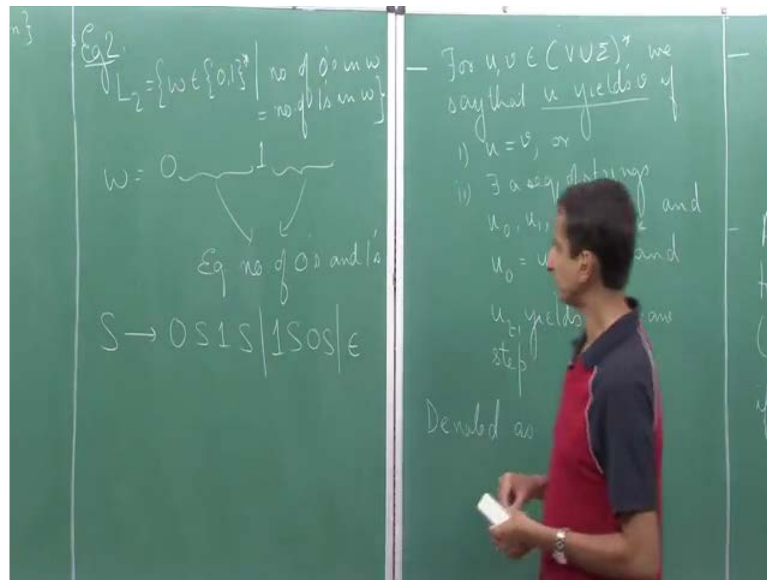
So, what is the intuition? So, essentially we want to have, we want to form sequence of a(s), and sequence of b(s), so we will use the start variable, and we will have some variable to form sequence of a(s) and b(s). But when we are forming our sequence of b(s), we will use the fact that from one variable, I can have multiple production rules to add either a single b or 2 b(s) for every a, that I add.

Let us try to see what we mean. Let us say we have a production rule going from A to A S B. So, this capital A will get replaced by a small a this means that I have equal number of capital A(s) and equal number of capital B(s), or it can go to epsilon as well. And let say that capital A goes to small a. Now where do I send capital B to, so if I send capital B to let say small bb, what I get is for every a, I have two b(s). Which means if I have n number of a(s) I will get 2 n number of b(s).

Instead of 2 b(s), if I just keep a single b, then what I get is for every a, I have a single b. So, if I have n number of a(s), I will get n number of b(s) only, but what I want is something in between n and 2 to the power as 2 n. So, what I will do is I will have add 2 rules b and bb. So, whenever if I have any number between n and 2 n, I will use the second production rule to add twice the number of b(s), and then when the number of a(s) becomes equal to the number of b(s), I will use the first production.

So, this ensures as the number of b(s) cannot be less than the number of a(s), and it cannot be more than twice the number of a(s). So, this is I mean a very important example where we use the fact that I can have multiple production rules for one variable to accept languages which have this kind of a form. So, of course, this is our start variable and small a and small b are the terminals and capital A, B and S are the variables.

(Refer Slide Time: 18:04)



Now, let us look at another example. So, this is an example of a non-regular language that we had proven earlier. So, we take let us call this L. Let us call this one L 1. So, we will call this L 2; L 2 consists of strings over 0 1 such that w has or number of 0s in w is equal to the number of 1s in w. So, it has equal number of 0s and 1s. So, before proceeding to create a grammar for this language, let us try to understand the intuition. So, when I have when I look at a string which has an equal number of 0s and 1s, firstly, I can break up this string I mean firstly, I can divide this into two cases, either the string starts with the 0 or the strings start with the 1. In either case, suppose the string starts with the 0; if the string starts with a 0, then this starting 0 is matched by someone somewhere down in the string.

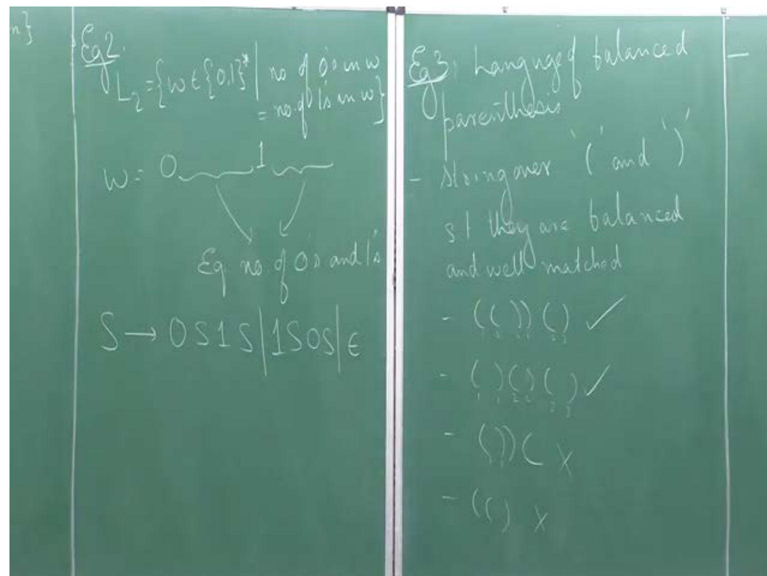
For example, let say that I have a string w which starts with the 0, then because this string has equal number of 0s and 1s, I have this 0 then maybe I have some other string in between, and there is a 1 which gets match to it. By matched I mean that this inner portion again has equal number of 0s and 1s, and then again there can be something after this also may be there is a string after this. So, when I can have some string, but the property is that so both for both these pieces this one as well as this one, so both these parts have an equal number of 0s and 1s.

So, this is the case when the strings start with the 0; even if the strings start with the 1, I have a similar sort of analysis. Now I want to use this fact to construct a context free

grammar. So, I take my start variable and I replace S with 0 followed by S 1 S. So, this means that I have 0 and 1 which are matching each other. So, this 1 is basically matching the first 0; and between them I can have any string, so basically S can get replaced with again some other string which is generating a an equal number of 0s and 1s.

And again, after the 1, I can have a string which is which has an equal number of 0s and 1s which S is generating. So, I have this I also have the case when the strings starts with a 1. So, I will have 1 S 0 S. And I have the empty string epsilon, because by definition this string does not have any 0 or 1 which means that it has 0 number of 0s and 0 number of 1s which are equal. So, this is essentially the grammar. So, this grammar has only one variable S; it has two terminals and it has three production rules.

(Refer Slide Time: 22:23)



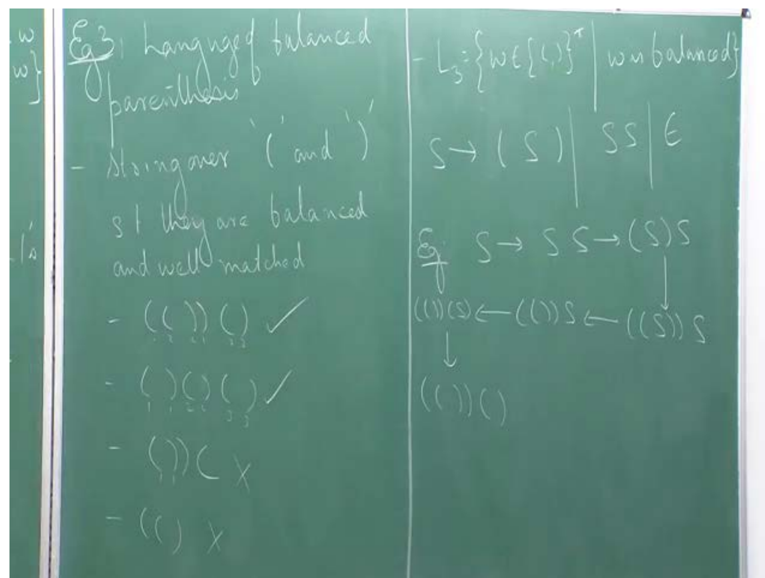
Now we will see our third example, which is kind of similar. So, this is known as the language of balanced parenthesis. So, basically balanced parenthesis means is so balanced parenthesis is a string over, so it has two symbols, the left parenthesis and the right parenthesis.

So, it has two symbols. Such that they are balanced and well matched; I mean it should not be that, so balance means that I should have equal number of left and right; and well matched means that I should have a left before I have its corresponding right; it should have these thing. For example, so this is example of a balance parenthesis. So, this one gets matched with this one, this one gets matched with this one, and this one gets

matched with this one. Similarly, I can have something like, so this one gets matched with this, this one gets matched with this, and this one gets matched with this.

On the other hand, if I have something like let say this so if you look at this, this has equal number of left and right. So, the number of left is two number of right is also two, but this is not balanced, because I do not have a left parenthesis before my right parenthesis. So, this one gets matched with this one that is fine, but now I have a right before a left. So, this is not balanced. Similarly, if I have something like of course, this is also not balanced, because I have two left and only one right.

(Refer Slide Time: 25:11)

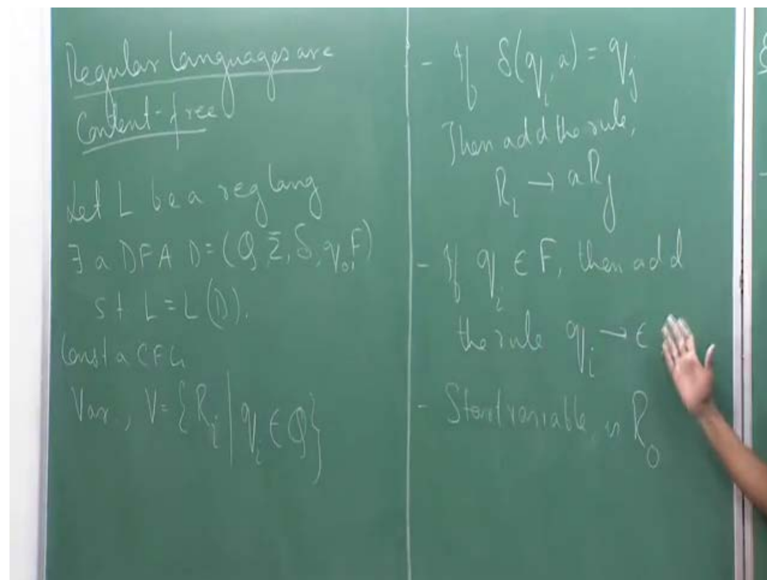


So, formally the language is let us call it L 3 is the set of all strings over these parenthesis such that w is balanced. So, how do I write a grammar for this? Firstly, I should have that for every left I should have a right and the string in between is again balanced. So, basically a left gets matched with the right and it can be next another balanced parenthesis inside it which is a smaller length, or it can be of the following form. So, I have some balanced parenthesis on the left, and I have some on the right. So, if you look at this example, so here basically how do I generate the first string?

So, if I want to generate the first string, so let us see this example. So, first I will replace S with and S S, because I want to have this part and this part. Now I replace the first S with left S right and I have the second right say second S then I replace again the first S with left left S, right right. Now I replace the first S with epsilon. So, I need to add

epsilon as well. So, I get left left, right right S, then I replace the the only S with again left S right, and now I replace this with epsilon. So, I get left left, right right, left right ok. And you can show that any other balanced parenthesis can also be obtained from this grammar. So, this is the grammar.

(Refer Slide Time: 28:01)



Now I will end by showing that regular languages are also context free. Let L be a regular language, which means that there is a DFA D, which is Q, sigma, delta, q naught, F that such that L is equal to L of D. So, from this DFA, I will construct A grammar construct a CFG whose variables are, so let call it V equal to R 0 for every or let call it R i for every q i belonging to Q. So, for every state q i, I have a variable R i the terminals are the same as the terminals or the alphabet of D. So, it is a same set.

The production rules I define as follows. If delta q i a equals q j then add the rule R i going to a R j. And if q i is a accept state, and then add the rule q i going to epsilon as well, and the start variable is the variable corresponding to the start state which is R 0. So, this context free grammar accepts the same language as this DFA, which shows that regular languages are substring. I will not again go into the proof of why this accept it correctly, but I will just stop with the construction.

Thank you.