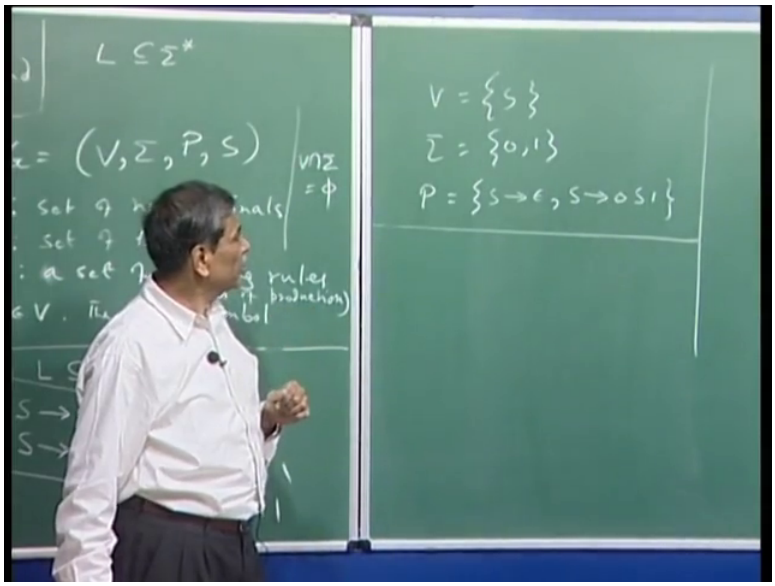
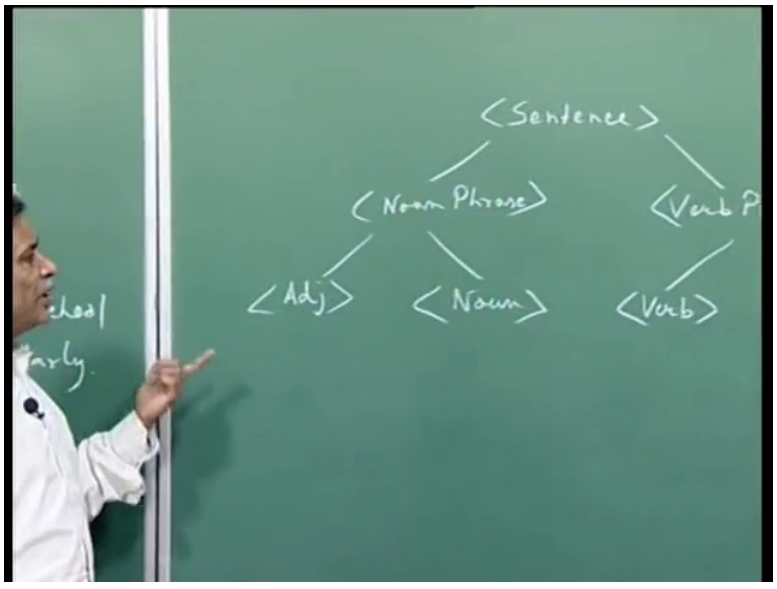
Theory of Computation.
Professor Somenath Biswas.
Department of Computer Science & Engineering.
Indian Institute of Technology, Kanpur.
Lecture-20.
Introduction to context free languages (cfls) and context free grammars (cfgs)
Derivation of strings by cfgs.

(Refer Slide Time: 0:52)





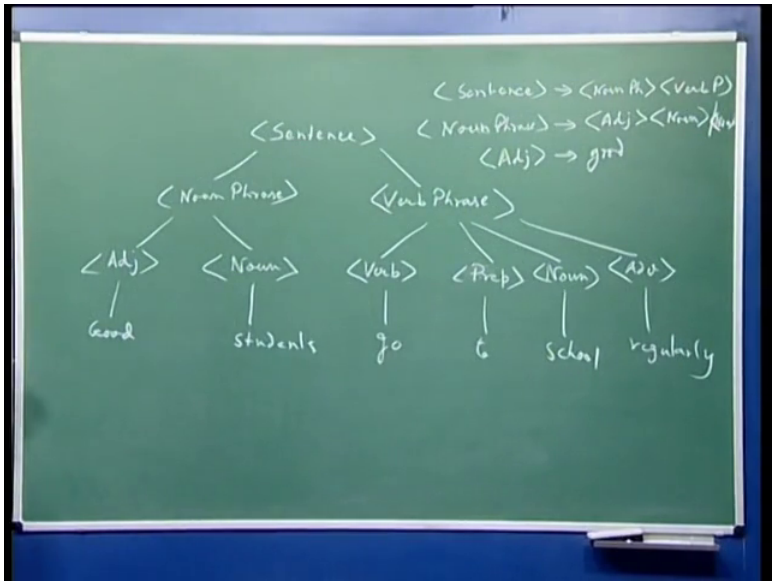
So far we have seen a class of languages called regular languages, we have studied their properties, they are very interesting but we also saw that there are many languages which this class of languages did not contain. Today we start looking at another class of languages and this class is called context free languages, the class of context free languages. And we will see

they are very natural class of languages, in a way your, will be familiar with some context or the other. And these languages are defined through context free grammar, that is the way to define these class of languages.

So what we are trying to say is that we are going to define a class of languages called context free languages. Recall, when we say this is a class of context free languages, that means this is a set where each set is a language or a set of strings. And if you go back when we considered regular languages, the way we defined the set of languages, the class of regular languages was that for each regular languages, regular language we had an automaton, a finite automaton. And the class of strings accepted by that automaton was the language considered. But here instead of a machine or an automaton, we are going to look at totally different way of capturing a language, describing a language and that notion is that of the grammar.

Now grammar is the notion of course we are familiar from our school days, right. For example, let me, what i am trying to do is notion of grammar and at this point this notion is informal, right. We consider a sentence like this that good students go to school regularly, right. This is of course the english language sentence and if you recall, in school we said this is a grammatical sentence, that this is an acceptable english-language sentence because we can parse it according to the grammar, grammar of english language. So the way we would parse it in school, maybe something like this that we will say that a sentence, an english sentence, okay.

(Refer Slide Time: 5:14)

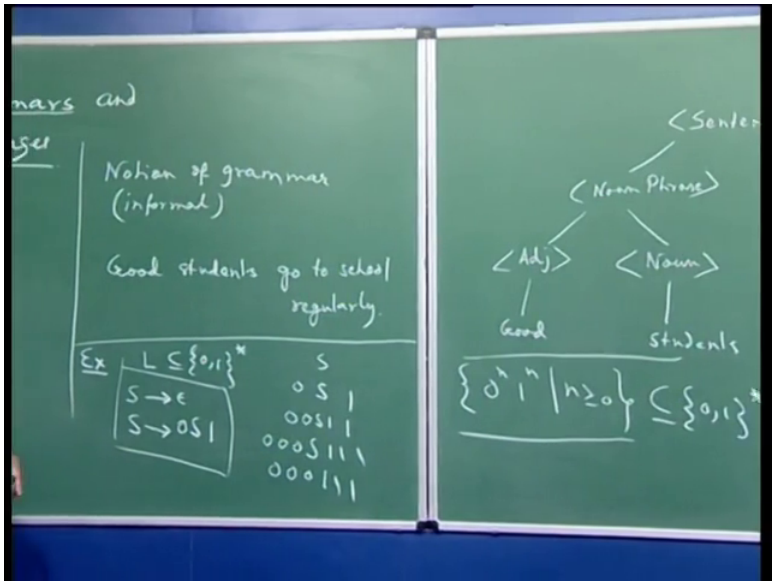


It has 2 parts, noun phrase and a verb phrase and a noun phrase may again have an adjective and a noun. And a verb phrase can have a verb and let say a noun phrase or let us say preposition followed by a noun as well as an adverb. The details are not important but i am trying to remind you that we did something like this in school and an adjective, there are many possible adjectives, in particular one adjective is good and one of many nouns, one is students and verb is go, preposition is to school and regularly. So we justify that this is an english-language sentence, good students go to school regularly by what we call in school by parsing it according to the rules of english grammar.

And what were those rules which we have used there? That any sentence or a sentence is composed of 2 parts, so we say sentence, this thing is a noun phrase followed by a verb phrase. For example, so we will have, we have rules like this that a sentence can be rewritten, this thing can be seen as a noun phrase followed by a verb phrase, right. And a noun phrase if we expand further, then we have, we say it can be adjective followed by a noun, also possibly a noun itself and an adjective is good, etc., right. So again, the detail of english-language grammar is not important, what is important here is the abstract notion of what is happening.

So firstly how do we see english-language sentences for the we say that english-language sentences are composed of words. But any juxtaposition of you know a set of words will not be a sentence, unless they follow certain rules as we derived that sentence. You know all these notions will come naturally to the definition of context free language, context free grammar, right. And the set of all grammatical english-language sentences are those which can be derived, which can be generated using the rules of the grammar which some of these rules that i have said here, okay, right.

(Refer Slide Time: 9:00)



So as an example of a former language which is in the same style, which can be described by means of a grammar, let us take this example which is a very simple example. Okay, i am considering a particular language l over 0 and 1, right. And very similar to what we had done there, let me write in this fashion that an s can be either an epsilon or an s can be 0s1, okay. Now these are the rules of this grammar, i will describe these things more carefully, define these things a little more carefully but let us understand this example, very simple example, the way it is. What we are saying that look the thing that you can do is to generate strings using these rules, why? Because for example you say i start with s and then apply this rule, right, s you can, this arrow will let us read it as s can be rewritten as 0s1, the way, same way we can show that or you know we can think of a sentence, this, whatever the entity is, this can be written as 2 things noun phrase and verb phrase.

Same way i am saying, this rule is basically saying that this symbol s can be rewritten as 0s1, so let me do that, all right. Why stop once applying this rule, it may apply once more, right now we have 0 s1. Why not apply this same rule once more. So that would mean 0, now this s i am rewriting as 0 s1 and this old 1 is there. Maybe i want to rewrite this s again using this same rule, so then what will happen, i will get 000s111 and then now i finally decide that i will rewrite s using the 1st rule which is s goes to epsilon. Epsilon because epsilon is an empty string. So when i apply that, i will get 000111.

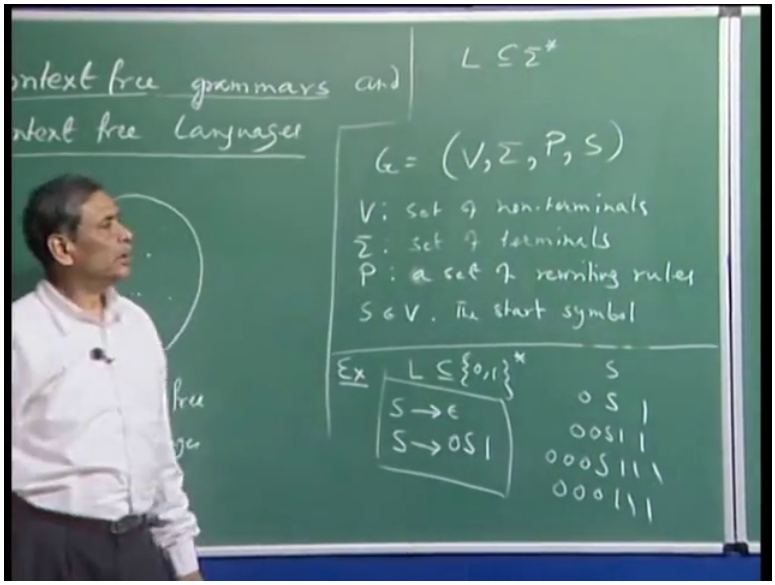
Can i apply any more of these rules? No because the only way we could have applied these rules, if i had an occurrence of this symbol s. Alright. So it is very clear that just by using these 2 roles, starting from the symbol and the rules basically, the rules of rewriting, what i

have done, i have come to a string and that string is over 0 and 1. Is this the only string i can generate, of course not, it is very easy to see i can generate all strings in this using these rules i can generate all strings of this kind, right, 0n, 1n where n is greater than or equal to 0. So this set of strings over the binary alphabet 0, 1 can be generated simply by using these rules, right.

Now what have i got, i have got a set of strings, right, over 0, 1, in other words this is of course is a subset of set of all finite strings over 0, 1 and therefore this is a language in our terminology, it is a language, to the formal language, this thing is a former language over 0, 1 and i have described it, very simply, i can describe this set very simply by saying that an element of this set, an element is the string over 0 and 1, is a string which can be obtained by repeatedly applying these rules of rewriting the symbol s starting from just one occurrence of a symbol s, okay.

So let us now formalise it, formalise this idea a little more carefully. We, starting from this very simple example, we will try to obtain the notion of a context free grammar. Again let us look at this example, we see 2 kinds of things here immediately, 2 kinds of symbols. Ultimately we are generating a language over 0 and 1, remember the language was 0n, 1n where n is greater than or equal to 0. So it is a language over this alphabet 0, 1 but other than that were also using a symbol in this case which is this capital s. Okay. So right away we see there are 2 classes of symbol, one is of course sigma, right, sigma is the alphabet of the strings which will constitute the language that we are going to define.

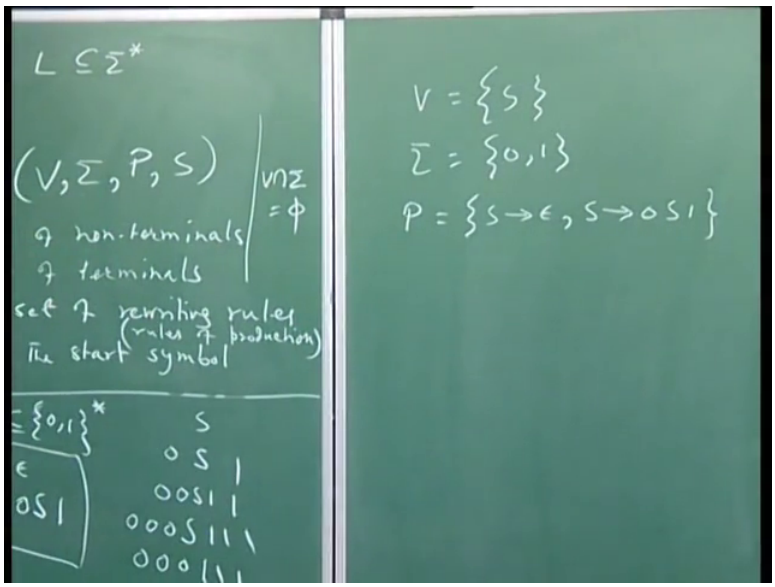
(Refer Slide Time: 15:46)



So let us look at it once more clearly. Suppose i want to define a language over sigma, some alphabet, so of course i will use that alphabet. But besides that alphabet, i am also using here one symbol which is s. So let me use a set v to say this is the set of non-terminals and this sigma is of course, i, let me say it as set of terminals. We will see immediately, a little later, why one we are calling, we are using this terminology, terminal and non-terminal. Besides these 2, of course these 2 these 2 are set of symbols, we also have these, what we call you know some rules of rewriting. So let me say this p to be a set of rewriting rules, right.

So this example, i have already taken care of s, i have already taken care of the binary alphabet 0, 1, as well as i have taken care of these rules. Is there anything else in this example which we need to you know generalise for the notion of grammar, yes. And that is i have to say that look we have how do we start this process of rewriting. Right. We start, we said that the strings we will derive, we will start getting, starting from just writing the symbol s. Now here i have a set of these non-terminals, i must say which of these will be the start symbol. Let me write this s, this s is going to be an element of v which we will say the start symbol which of course is an element of the set of non-terminals. Okay.

(Refer Slide Time: 19:02)



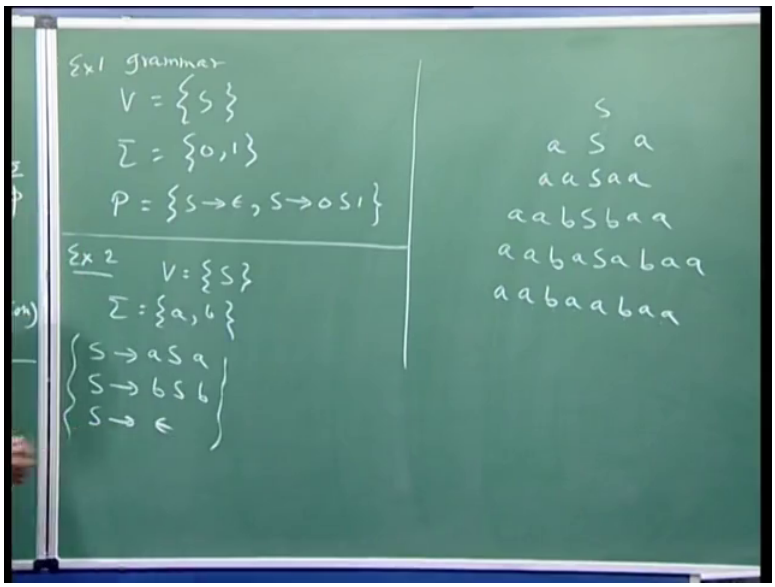
These 4 things together, let me call the grammar g. In particular, when the rules are like this, we call them context free grammar. All right. So in this example let us just write this that v was the set s, sigma was the alphabet 0, 1, p, the rules of production, incidentally these are also called rewriting rules, also called rules of production alternatively. P was just these 2 rules, s goes to epsilon and s goes to 0s1, all right. And s, there was only v consists of only one particular symbol, one terminal and the start symbol has to be an element of v. And

therefore s is by default the start symbol in this example. Now we will also, we should say that this v and sigma, they are disjoint.

In other words there is, their intersection is empty. Right. So they serve 2 different purposes as we shall see little more elaborately now. And now what we are doing, exactly the same thing i will now say in terms of our terminologies that what we are doing is derive or to get such terminal strings, we start with the start symbol s and then use a production whose left-hand side is s. Right, in this case both the productions have the same left-hand side, so i, instead of this s i replace the this s with right-hand side of a production whose left-hand side is s. Right.

And you know we did all these and in the process finally, at some point of time when i have a string in which there is no occurrence of any symbol from v, so that means the strings that i have consists solely of terminal strings or what we are, our alphabet sigma, then we say that is, one string that we have generated. So using the rules as and when as we wish, let me give a couple more examples before we proceed with the formalism. So this particular grammar gave me a set of strings as we have seen here, which was the set of all strings in which the zeros occur before and ones occur later and number of zeros is same as number of ones. Right, we will come to more about the language generated.

(Refer Slide Time: 23:04)



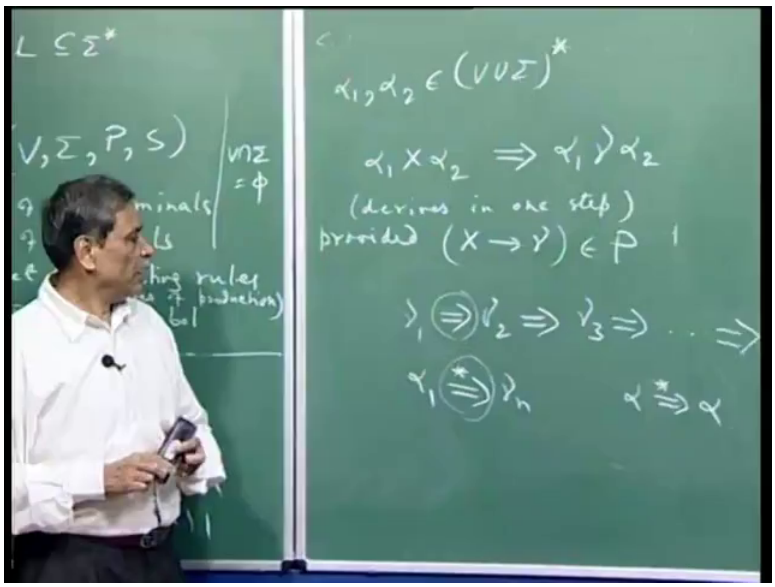
But let me tweak this a little bit, same example to get something like this. Let me again, so this was a example 1 grammar and what is another example, which is also very simple, again i will have only one non-terminal s, let me use a different alphabet a, b of terminal strings,

terminal symbols and for production i will have this s goes to a s a, s can also go to b s b and let say s can again go to epsilon. So this consists, these 3 rules will constitute capital p, the production rule. What kind of strings do i generate from using these rules? Again we start with s, so let me, let me choose to use these particular rules, so i will get a s a, then why not do it once more use the same role once more so a a s a a, now let me use this rule. Right.

So a a b s b a a and once more let me use the 1st rule a a b, so this particular s is being rewritten as a s a, these things, these 3 things will follow after that. And now i use the final, last rule to get a purely terminal, the string of made solely of these terminal strings, terminal symbols, so a a b a a b a a. What can you say about this thing, that this thing reads the same from left to right as well as from right to left, is not it. A a , a a b a a, so a a b a a, right. So whether you read it from the left or you read it from the right, it you get the same string. Such a string is called a palindrome as you know, we have seen in other contexts. And does it generate all palindromes, not really, i mean palindromes which are, whose lengths are even. Right.

This is another example. Now in both these cases what is happening, we will have you know these v, sigma, p, s, we start with s and if rewriting using p, you have seen examples of rewriting, that whenever. What is rewriting, that i have a string in which, presumably there will be some terminal symbols and non-terminal symbols. But non-terminal symbol in such a string can be replaced by the right-hand side of the rules whose left-hand side is that particular non-terminal. For example what i am doing, this particular non-terminal i am replacing by this because s goes, s arrow bsb, so s is the left-hand side and bsb is the right-hand side of the rule. This is what we have been doing all through. Okay.

(Refer Slide Time: 28:00)



Now let me describe this process a little more formally. So what i am doing is that suppose i have a string which is of this kind of alpha 1, then x, alpha 2, right and x arrow gamma is the production rule. Then, from, then it is basically what i am saying is if i add this and using this rule i can get, so starting, not starting from this string, using this we get we take we get alpha 1 gamma alpha 1 gamma alpha 2, right, basically i have just replaced this x by the right-hand side of this production. Which, now in general x may have many productions, there are many productions whose left-hand side is x. I will choose any and just replace, whatever, whichever production i choose, the right-hand side of that is used to replace this particular, right.

Now do you see that if i do this, then i say that alpha 1 x alpha 2 derives in one step alpha 1 gamma alpha 2, okay. Is this clear? So i have come again let us let us look at this. I have a string, what, by the way what are alpha 1 and alpha 2, alpha 1 and alpha 2 in general are strings over v and sigma. So let me write it more clearly here. Alpha 1 alpha 2 are strings over v union sigma star, that means they are strings over the alphabet v and sigma together. So basically such a string can have occurrences from v as well as from terminal alphabet. And i have this string alpha 1 x alpha 2 and i have another string alpha 1 gamma alpha 2.

I say that these 2 strings are related by this relation which i read it as that alpha 1 x alpha 2 derives in one step alpha 1 gamma alpha 2. So let me write it, provided x goes to come, this is a rule inyou know my production. Now what we are saying, i am saying that look 2 strings over v union sigma, i can relate them by saying that one string derives in one step the other string, provided, what i do is i rewrite one non-terminal in that 1st string by the right-hand

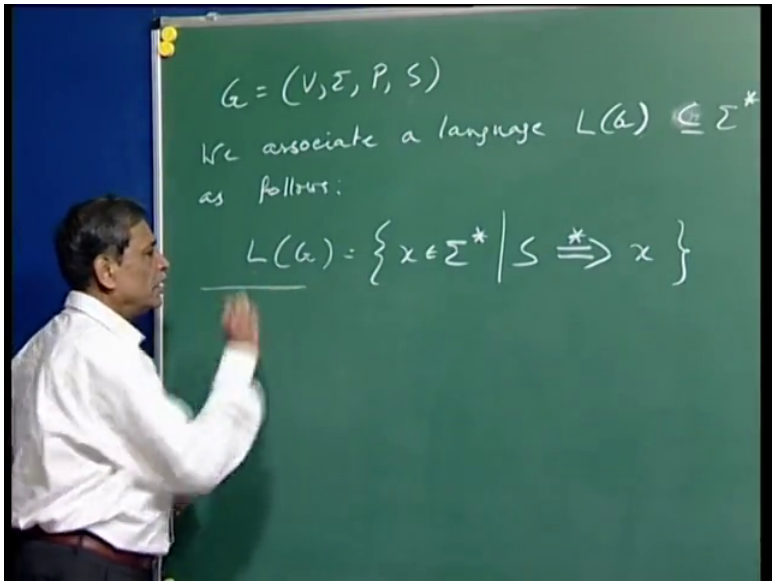
side of the production, gamma is the right-hand side of the production whose left-hand side is the non-terminal which i am replacing.

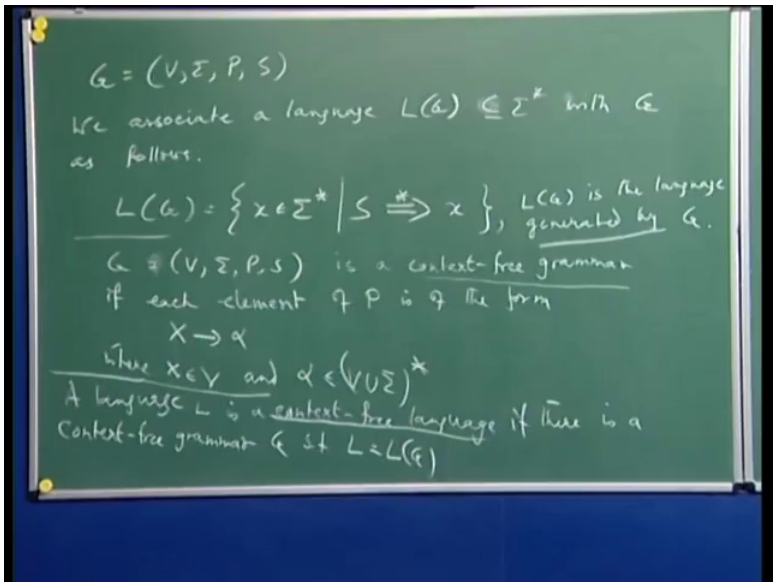
This is very simple, i had an occurrence of x, that is what i am replacing by gamma and that is why i am saying that this particular string is obtained from this particular string in one single step. So now you know a step here is basically an application of the production rule to rewrite one non-terminal by the right-hand side of that non-terminal of a production. Alright. So this is clear, if this notion is clear, what does it mean to say a string derives in one step another string.

So now you can see that, you know i may have some string gamma 1 and i keep on doing this from gamma 1 in one step i get gamma 2, from gamma 2 in one step i get gamma 3 and so on so forth, finally let say i get some gamma n. And in each i am doing the same thing, like in gamma 1 there was some non-terminal which i am replacing by the right-hand side of the production whose left-hand side is that non-terminal and so on, from gamma 2 also i am replacing one terminal by the right-hand side of a production whose left-hand side was that non-terminal which am replacing. So now i will say that alpha-1 derives alpha n, simply derives and let me write it as like this.

And i will also just for the sake of completeness, let me also say that a string also derives itself. Those of you who are, who love long words, what this particular relation is the reflexive transitive closure of this relation. But very simply what we are saying that 0 or more applications of this derives in one step, if true that i can go from alpha-1 alpha-1 to i should have said gamma 1 to gamma 1 to gamma n in one step, right. So here i have used so many times 1, 2, 3, so many times i have made use of, n -1 derives in one step i have used to go from here to there, all right. So now using this notion i should be able to describe the language generated by grammar.

(Refer Slide Time: 35:50)





Let g be a grammar, remember grammar for us has a set of non-terminals, set of terminals, set of production rules and start symbol s, okay. And now with this grammar i associate a language, we associate a language lg which is a subset of sigma star, that means this is the language , consisting of some set of finite strings over the alphabet sigma in the following way. The grammar g is as follows, we say lg is the set of all terminal strings, such that s derives, remember the notion of derives as opposed to essentially derives in one step, derives in one step means, just by one application of the production rule. Derives, when we say derives, that means i can allow many steps, so this s derives x, okay.

So what we are saying, that look, the language generated by the grammar g, that is how we read it, language generated by the grammar g is a set of all terminal strings x, which can be

obtained, which can be derived from the start symbol of the grammar. Now do you see every of these notions, everyone of these notions are given. In this of course to define which all strings s can derive, there i am using the set of production rule, i am using the start symbol in a very essential manner because a language of the grammar, language of this grammar g, not only has to be a terminal string but it has to be also something which is derived from the start symbol. Okay.

Right. So i am using p in this, what, when i am saying that x, s derives x, i am using the, using s of course here and i am using sigma and p is of course defined to v, v, you know v can occur, so in p and therefore all the closure there are being used. Okay. Now let us be a little more clearer, for you know we have been very informal so far. So we say g, v, sigma, p, s, by the way the usual restrictions apply, that is v and sigma, they are (())(39:59) and so on, is a context free grammar if each element of p is of the form x goes to alpha where x is an element of v and alpha is a string over v union sigma star.

You remember our notations that basically means that alpha is a string using the symbols of v and sigma. Now these are the only kinds of v have (())(41:16), they need not be, there can be you know left-hand side of this, by the way this is called the left-hand side and this is right-hand side, i had been using that, now you know also. The left-hand side can be more complex, something which could have been a string, right-hand side also could be a string but in context free language, left-hand side is always this one non-terminal. And notice that what you are rewriting is always a non-terminal. So you have some non-terminal, you replace that non-terminal in the class of derivation.

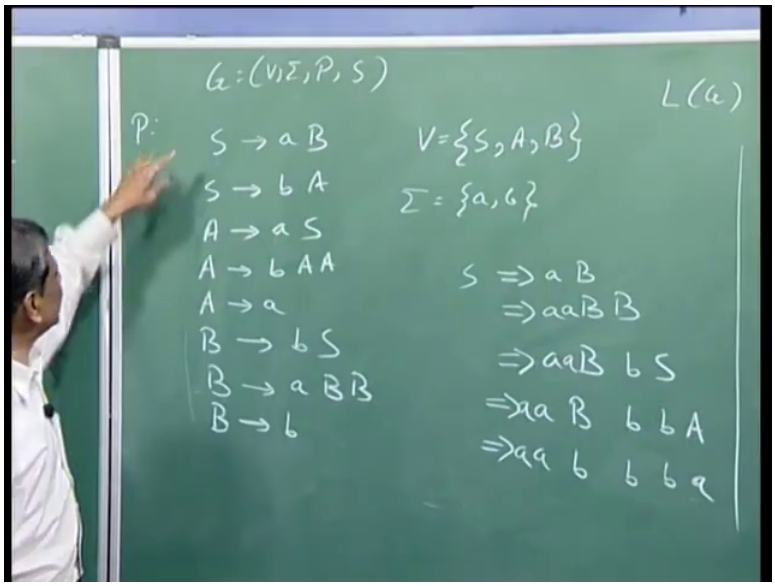
So now we know what is a context free grammar and language l is context free, so let me now give the official definition of context free language, we been, we have seen some examples. A language l is context free is a context free language if there is a context free grammar g such that this l is the language derived or associated with the grammar g. We you know the meaning of given g, what is the meaning of lg. Lg is the set of all terminal strings you can derive from the start symbol of the grammar. And what you are saying is that a language is context free, a language is a context free languages you can find a context free grammar g such that this language is the associated language with that grammar.

We also say that this the this lg is the language invented by the grammar g. Let me use that term, lg is the language generated by g. So you have to be careful or you know we should appreciate one fact that while talking of regular languages, right, what did we do, we

typically free, what would it was that we defined an automatic, a finite automaton and then we said the language accepted by that automaton, right, and here what we are saying, the language generated by the grammar g. You can see these are 2 different, 1, we use there is this notion of acceptance by an automatic, so there we said you know consider the set of all strings and now for each string check whether my machine accepts it or not. If it accepts, it is in the language, if it is not, it is not in the language.

Here what we are saying that again consider the set of all strings over sigma. Now if a string is generated in that grammar through, is derived from the start symbol you know like this by the grammar, then i say that string in the language, otherwise it is not inside it, that is what you are saying, right. So here, there when talking about automaton, we have this notion of acceptance or string being recognised. Here we are saying, we are generating that string, okay. That is why these grammars are called generative devices. Now let me take our our example so that these ideas become a little clearer.

(Refer Slide Time: 46:23)



Consider another grammar, this time we will have not just one non-terminal but a number of non-terminals and it goes like this. Let me 1st write down the production, i have the start symbol, i again use s, so s goes to a b and s goes to b a, then a goes to a s and a can go to b aa, a can also go to a, b goes to, i will give the production whose right and left-hand side is b, so b goes to b s, b goes to a bb, b goes to b. While reading these rules you can say s goes to this or you can say s can be written as this or whatever, whichever is you know easy to you or more natural to you. So now these are set of productions, now you can see what are my set of non-terminals.

V is s, a, b and sigma here is the set of non-terminals is a, b and this is your set of productions and s is your start symbol. So my grammar g is v, sigma p, s where s is the start symbol. So let let us try to derive a couple of strings from here. Now 1st of all do you agree that this is a context free grammar, then this grammar that i have described there is a context free grammar? Why, because of course i have this v, sigma, p, s and my each of these productions is of the type that i have a non-terminal to be and the right-hand side is some string over terminals and all. So this is indeed the context free grammar and therefore the language generated by this is going to be a context free language.

And let me just derive a few strings of this language. We start with s and maybe i will use the 1st production. Now you see i have a choice, i can use any one of these, so why not let me use this more complex looking one, let me use the 2nd one. So this b, therefore i rewrite as a bb. Now what, i can again apply one of the b rule and notice now that i have a choice, that i put either, we rewrite this rule, this particular non-terminal of this occurrence of b or this occurrence of b. We are not saying that which one you should do. So maybe let me do one at the end, so this b i keep as such and that b i write it as b s, okay.

So again i have string, i have from s i have derived this string in which you know i have these 2 non-terminals. You know, my, you see it is in my hand, i am doing the derivation, i could either use a production whose left-hand side is b or i can use a production whose left-hand side is s. Let say that we use this one, okay and maybe this time let me use this particular one s goes to b a, so this s goes to b a and now what i will do is, where did i stop, s goes to b a and now which when i will replace, so let me replace this b by this one and then i am going to replace, there is nothing to replace here, this is b, this is b and here let me replace this by and this a by an a.

You know i made one mistake in this derivation, can you spot where that mistake in the derivation is, that will replace this b, this capital b there, what is placed this b with left-hand side of, right-hand side of this particular rule which is a bb, which i wrote. But then i forgot to write this particular, the 1st a. So i should have an a all the time before this particular day, right, in the beginning. So let me do this now. And i have got a string which is a a b b b a , right.

Just one more derivation of the from the same grammar, the point at least i tried to, which i tried to illustrate here was that at any given time you may have a choice, choices, you have 2 kinds of choices that there are several non-terminals, which one you should replace, you have

the choice there. Also given, making, having made the decision that i am going to replace one particular non-terminal by the right-hand side of a production which, which production. You know like b, when i chose here that i will replace this, i mean here i said i will replace this s, so i had a choice, i could have either used this production or that production.

So nobody is going to tell us while we are deriving which one i should do, that is fine, you know, any one of these productions whose left-hand side matches the non-terminal which i want to replace is equally welcome. And in the process of doing this finally i have come to a terminal string which cannot be, can you do anything further, no because the only thing you can do from a string as in the process of derivation to get another string is to replace one of the non-terminals either by the by using one of the production rules. But here there are, there is no non-terminal at all, so the production, the derivation will stop, right, the derivation stops once you have a string which is all terminal string.