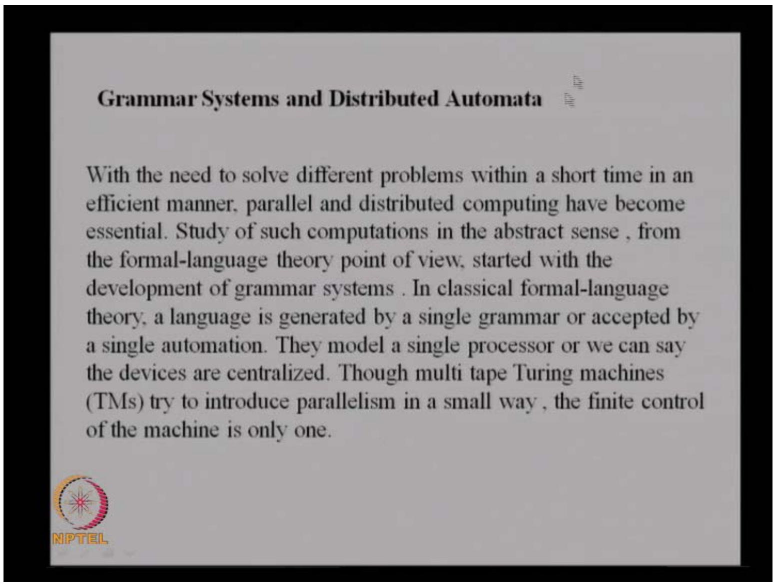
Theory of Computation

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

Lecture No. # 40
Grammar Systems

(Refer Slide Time: 00:29)

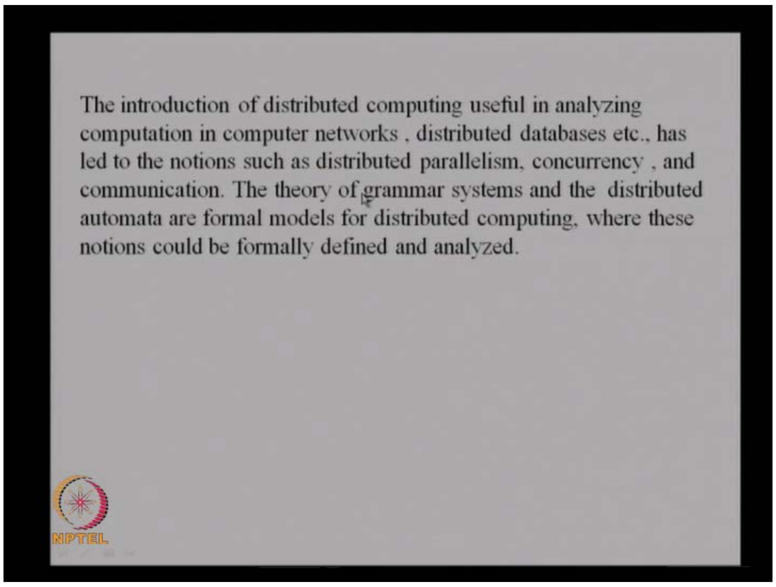


Grammar Systems and Distributed Automata

With the need to solve different problems within a short time in an efficient manner, parallel and distributed computing have become essential. Study of such computations in the abstract sense , from the formal-language theory point of view, started with the development of grammar systems . In classical formal-language theory, a language is generated by a single grammar or accepted by a single automation. They model a single processor or we can say the devices are centralized. Though multi tape Turing machines (TMs) try to introduce parallelism in a small way , the finite control of the machine is only one.

Today, we shall study about grammar systems. What is the motivation for this? Today the idea of distributed computing plays an important role. Different problems of high intensity computation time; they are process using distributed computing, and parallel computing. So, how do you abstract these models? So, from a formal language theory of point of view, this was model using the idea of a grammar system. In classical formal language theory, a language is generated by a single grammar or accepted by single automata, but in a grammar system, several grammars join together and generate a string; and in the automata several automata joined together and process a string. So, single grammar corresponds to a single processer, they are centralised; you can say them they are centralized.

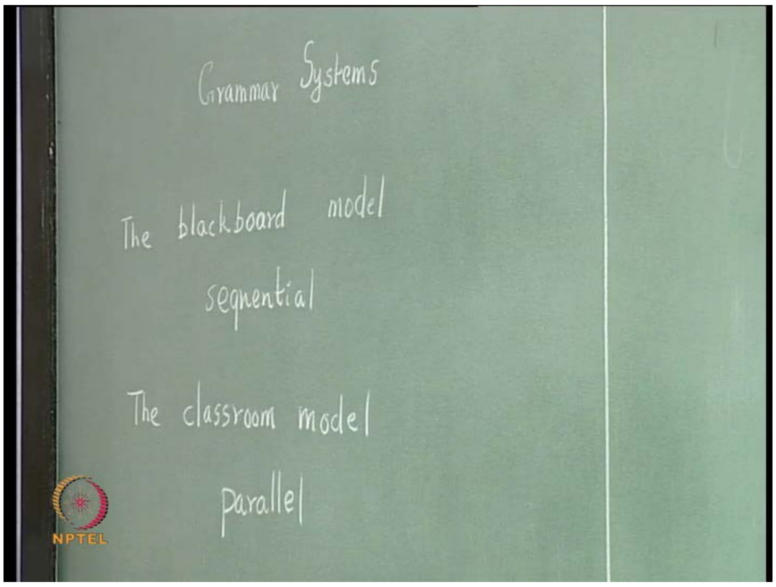
(Refer Slide Time: 01:32)



The introduction of distributed computing useful in analyzing computation in computer networks , distributed databases etc., has led to the notions such as distributed parallelism, concurrency , and communication. The theory of grammar systems and the distributed automata are formal models for distributed computing, where these notions could be formally defined and analyzed.

But in distributed computing, you have several processes and the processes are all communicating with each other. How do you capture this? With the idea of a grammar or a grammar system that is what we are going to study today.

(Refer Slide Time: 02:00)



Grammar Systems

The blackboard model
sequential

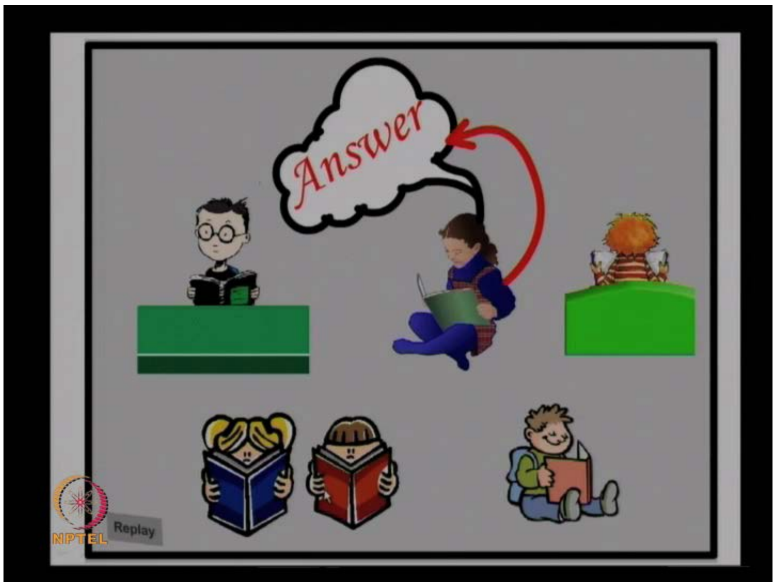
The classroom model
parallel

So, in grammar systems, we have two models; the blackboard model which is a sequential model and the classroom model, which is a parallel model. What is a blackboard model or a sequential model? You have a problem to be solved and the students are going to solve that in a class. There is a blackboard one student comes and write something on the board. He solves part of the problem, he goes back. The second student comes and continues with the solution tries to work out a few more steps. And then he goes back then the third student comes he tries to work a few more steps and so on.

This continues until the whole problem is solved. So, the idea is that of a sequential processing all the students do not solve the problem simultaneously one by one they come and solve the problem. A portion of the problem and then the next student takes over. So, that is why this is called the blackboard model. So, let us see how it works the first student goes and he write something on the blackboard.

He comes back then the second student goes he continues with the solution writes a few more steps comes back, then the third student goes continues with the solution he comes back and then it is continued with the next student. So, this type of a model is called the sequential model or the blackboard model. The grammars one by one they generate the sentential form and ultimately lead to the string generator.

(Refer Slide Time: 03:56)

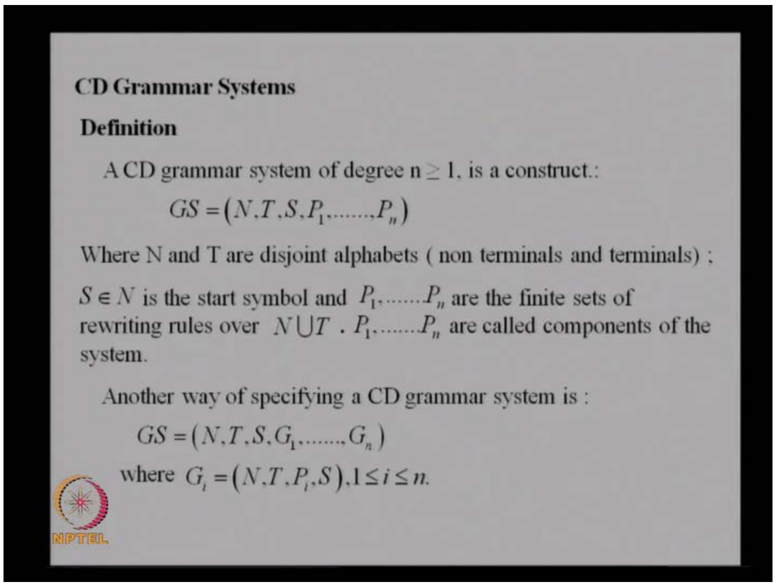


In contrast to that you have the classroom model; in classroom model what happens is it is like a program. You have a main program and in the main program you have sub routines each student is given one portion of the program. And they have to find the routine they have to write down the solution for that particular routine and the main class leader. He is the main person he is writing the main program or trying the main solution.

So, when he executes, he want some answer at some point he has to make a function call it is or a sub routine call at that stage he makes a query and the sub routine corresponds to the problems solve by another student. And she or he provides the corresponding answer to that this sort of a processing goes on until the whole problem is solved. So, let us see how this works this each student is trying to work out some portion of the problem and then this person gets a query. This answer is provided by this person after some instance this is the class leader.

He is trying with the solution and he may get another query for which the answer may come from this person it is like making sub routine calls this person. When he calls a sub routine it answer is given here then he makes another query. The answer comes from here and this person himself, he may ask for a query and then the answer may be supplied by this portion this person. So, this sort of a thing may continue and this sort of a thing is called a parallel model, because all of them are working on their sub problems simultaneously and one person has a query means he asks and another person provides the answer this is called the parallel communicating model or the classroom model.

(Refer Slide Time: 06:11)



CD Grammar Systems

Definition

A CD grammar system of degree $n \geq 1$, is a construct.:

$$GS = (N, T, S, P_1, \ldots\ldots, P_n)$$

Where N and T are disjoint alphabets (non terminals and terminals) ;

$S \in N$ is the start symbol and $P_1, \ldots\ldots P_n$ are the finite sets of rewriting rules over $N \cup T$. $P_1, \ldots\ldots P_n$ are called components of the system.

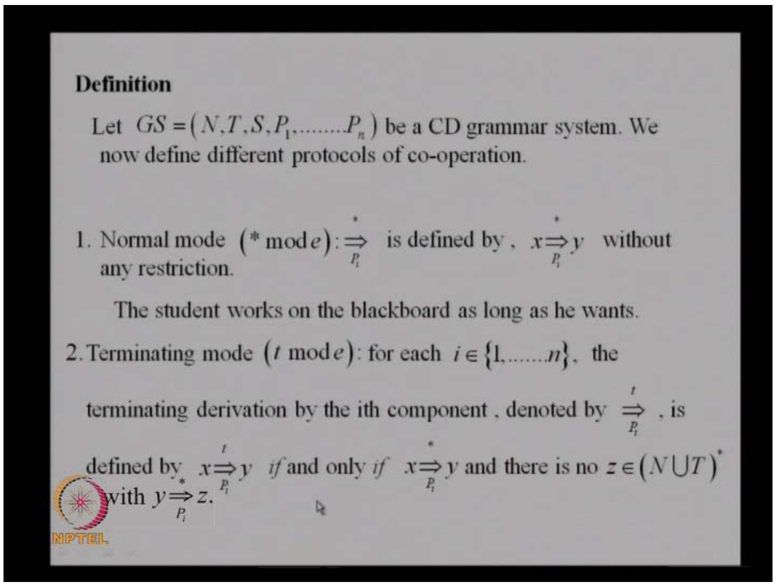
Another way of specifying a CD grammar system is :

$$GS = (N, T, S, G_1, \ldots\ldots, G_n)$$

where $G_i = (N, T, P_i, S), 1 \leq i \leq n.$

So, let us see the formal definition now. So, the first model is called CD grammar systems or co operative distributed grammar system. The definition is like this A CD grammar system of degree n is a construct G S N T S P 1 P 2 P n where n is the set of non terminals, t is the set of terminals S is the start symbol and P 1, P 2, P n are n sets of productions they are finite sets of rewriting rules over N union T. You can have them as type 0, type 1, type 2 or type 3 or you can also specify the grammar system like this N T S G 1, G 2, G N where each grammar G i is specified like this. You have the set of non terminals, you have the set of terminals for each grammar and you have the start symbol S and the set of productions are given by P i for G i either way it can be specified. It is easier to specify this way. We can specify this way also does not matter.

(Refer Slide Time: 07:29)



Definition

Let $GS = (N, T, S, P_1, \ldots\ldots P_n)$ be a CD grammar system. We now define different protocols of co-operation.

1. Normal mode $(*\ \mathrm{mode})$: $\underset{P_i}{\overset{*}{\Rightarrow}}$ is defined by, $x \underset{P_i}{\overset{*}{\Rightarrow}} y$ without any restriction.

   The student works on the blackboard as long as he wants.

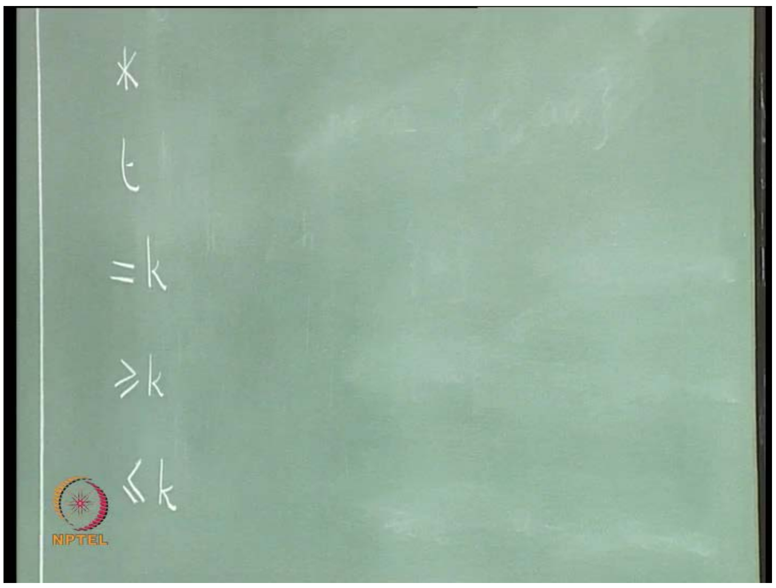
2. Terminating mode $(t\ \mathrm{mode})$: for each $i \in \{1, \ldots\ldots n\}$, the terminating derivation by the ith component, denoted by $\underset{P_i}{\overset{t}{\Rightarrow}}$, is defined by $x \underset{P_i}{\overset{t}{\Rightarrow}} y$ *if* and only *if* $x \underset{P_i}{\overset{*}{\Rightarrow}} y$ and there is no $z \in (N \cup T)^*$ with $y \underset{P_i}{\overset{*}{\Rightarrow}} z$.

Let G of S is equal to N T S P 1 P 2 P n be A CD grammar CD stands for cooperative distributed grammar system. We now, define different protocols of cooperation what do you mean by different protocols of cooperation. Now, we have seen that CD grammar corresponds to blackboard model one student works out a few steps and then he goes back. Then the next student comes. Now the question is at what time does the first student go back and the second student comes. And at what time does the second student go back and the third student comes.

There can be several protocols for that one is any time the student can go back and the next student can come in that is called the star mode in the formal definition. Another thing is a student proceeds as much as possible that is he is trying to work out a solution and he proceeds till at a certain stage he is not able to proceed further then at that stage. He goes back and the next student who is capable of proceeding further comes and takes over and this is called the terminating mode. Because the first student spends time on the board as long as possible and goes back only when it is not possible for him to proceed and the next student comes in formal model this is called the t mode or the terminating mode.

There are other modes let k be a finite integer and when that finite integer is there the first student comes and he works out for k steps in the solution. He continues for k steps then he goes back whether he is able to proceed or whether he is not able to proceed. He just goes back and then the second student comes and continues with the next k steps. So, each student spends exactly k steps on the board that is he writes just k steps on the board and then he goes back this is called the equal to k mode for k is equal to 1 2 3 etcetera then you have what is known as greater to or equal to k mode less than or equal to k mode.

(Refer Slide Time: 10:15)

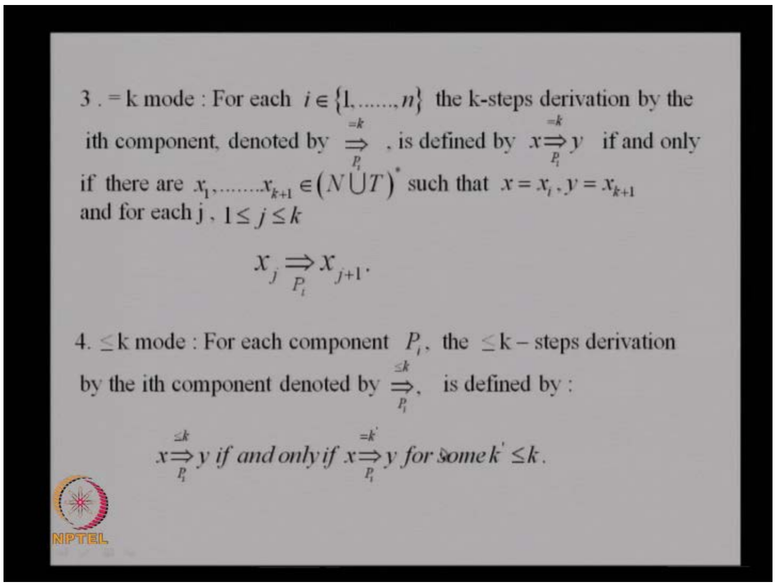


So, let me write down the modes. We have seen what a star mode is any time the student can go back t mode the student goes back when he is not able to proceed equal to k mode. These are all modes each student writes k steps and greater than or equal to k mode, each student spends k or more steps on the board. That is he comes and works for k steps he may continue for some more step, but minimum is k. So, minimum k steps, but it can be more also then he goes back the next person comes and so on. Similarly, you also have less than or equal to k mode. That is each student spends k or les number of steps on the board he can start 1 step or 2 steps up to k steps he can go, but he cannot go beyond that.

So, you have 5 modes of cooperation in this grammar then star mode is called the normal mode and the definition of derivation is like this normal mode the i'th component. It is defined by from x you can derive y without any restriction any number of steps. You can spend on the i'th component or the i'th grammar you can have a derivation any number of steps in a (( )) manner. When you say the student works on the blackboard as long as he wants terminating mode for each i belonging to 1 to n the terminating derivation by the i'th component that is denoted by this symbol double arrow.

This denotes the component in which the processing is done t denotes the terminating mode that is defined like this. That is you are able to derive from x the string y and from y you cannot proceed in the same component that is there is no z such that from y you can derive is z then this is called the terminating mode.

(Refer Slide Time: 12:42)



3 . = k mode : For each $i \in \{1,\dots\dots,n\}$ the k-steps derivation by the ith component, denoted by $\overset{=k}{\underset{P_i}{\Rightarrow}}$, is defined by $x \overset{=k}{\underset{P_i}{\Rightarrow}} y$ if and only if there are $x_1,\dots\dots x_{k+1} \in (N \cup T)^*$ such that $x = x_i , y = x_{k+1}$ and for each j , $1 \le j \le k$

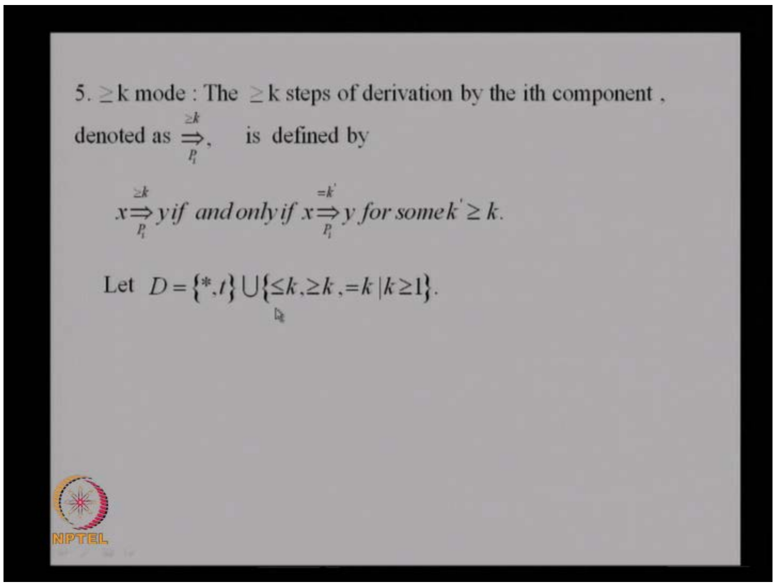
$$x_j \underset{P_i}{\Rightarrow} x_{j+1} .$$

4. $\le$k mode : For each component P_i, the $\le$k – steps derivation by the ith component denoted by $\overset{\le k}{\underset{P_i}{\Rightarrow}}$, is defined by :

$$x \overset{\le k}{\underset{P_i}{\Rightarrow}} y \textit{ if and only if } x \overset{=k'}{\underset{P_i}{\Rightarrow}} y \textit{ for some } k' \le k .$$

Then you have the equal to mode equal to k mode k is an integer i denotes the component number the k steps derivation by the i'th component is denoted by this from x. You can derive y in this if, you have x 1 x 2 x k plus 1 where x is equal to x 1, y is equal to x k plus 1 and from x 1 you can derive x 2 in 1 step, x 2 from x 2 you can derive x 3 and so on. In general from x j you can derive x j plus 1. So, you have k steps deriving x 2 from x 1, x 3 from x 2, x 4 from x 3 and so on. So, in k steps you derive x k plus 1 from x 1 and all of them take place in the same component P i and less than or equal to k mode you have k or less than or number of steps that is x less than or equal to k y if and only if it is done in k dash number of steps where k dash is less than or equal to k.

(Refer Slide Time: 13:56)



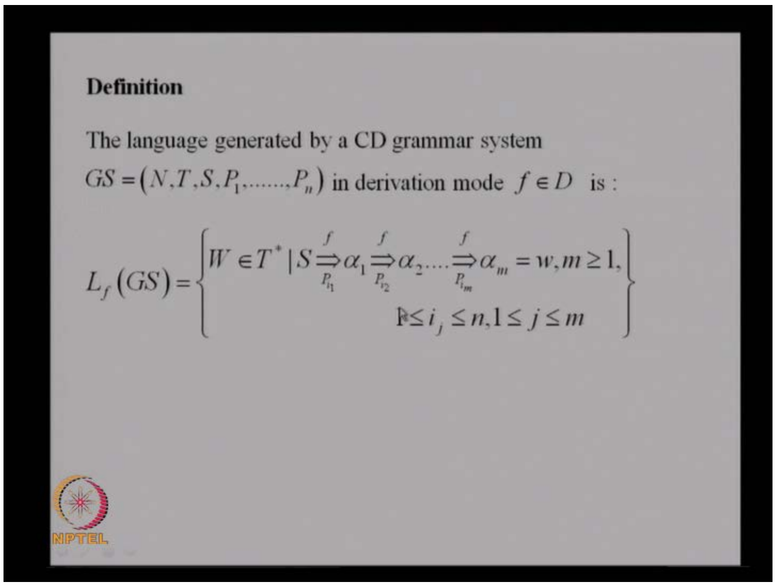
5. $\geq$k mode : The $\geq$k steps of derivation by the ith component , denoted as $\overset{\geq k}{\underset{P_i}{\Rightarrow}}$, is defined by

$$x \overset{\geq k}{\underset{P_i}{\Rightarrow}} y \; if \; and \; only \; if \; x \overset{=k'}{\underset{P_i}{\Rightarrow}} y \; for \; some \; k' \geq k.$$

Let $D = \{*, t\} \cup \{\leq k, \geq k, =k \mid k \geq 1\}$.

And similarly, greater than or equal to k mode that is you derive y from x in greater than or equal to k mode in the component P i if you derive y from x in k dash number of steps where k dash is greater than or equal to k. So, these are the modes and so, the mode you can denote as D is equal to star t star you can have the star mode, you can have the t mode, you can have the less than or equal to k greater than or equal to k equal to k mode where k can be an integer it can be 1 2 3 etcetera.

(Refer Slide Time: 14:39)



Definition

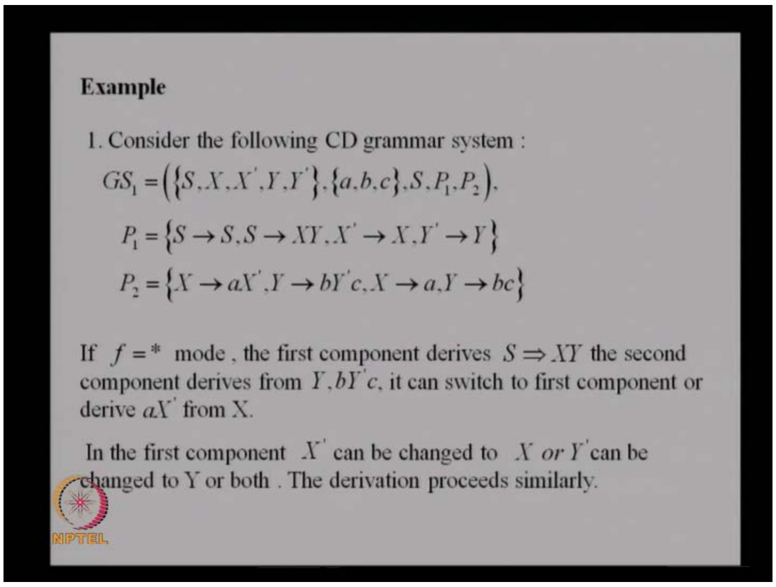
The language generated by a CD grammar system $GS = (N, T, S, P_1, \ldots\ldots, P_n)$ in derivation mode $f \in D$ is :

$$L_f(GS) = \left\{ W \in T^* \mid S \underset{P_{i_1}}{\overset{f}{\Rightarrow}} \alpha_1 \underset{P_{i_2}}{\overset{f}{\Rightarrow}} \alpha_2 \ldots \underset{P_{i_m}}{\overset{f}{\Rightarrow}} \alpha_m = w, m \geq 1, \; 1 \leq i_j \leq n, 1 \leq j \leq m \right\}$$

The language generated by the grammar system with N components in the mode f f belongs to D. We have seen what is D earlier it is L of f G of f is equal to a terminal string, which belongs to T star such that from S you derive the sentential form alpha 1 in component i 1 in the f mode and from alpha 1 you derive the sentential form alpha 2 in component i 2 in the same mode proceeding like this. You derive alpha m from alpha m minus 1 in component i m using the same mode. So, m is greater than or equal to 1 and these are all one of the n components.

So, each i j is within 1 and n it is one of the components after k if it is equal to k mode after k steps it has to switch component similarly, if it is a T mode when it is not able to proceed further it switches component.

(Refer Slide Time: 15:56)



Example

1. Consider the following CD grammar system :

$GS_1 = (\{S, X, X', Y, Y'\}, \{a, b, c\}, S, P_1, P_2)$,

$P_1 = \{S \to S, S \to XY, X' \to X, Y' \to Y\}$

$P_2 = \{X \to aX', Y \to bY'c, X \to a, Y \to bc\}$

If $f = *$ mode , the first component derives $S \Rightarrow XY$ the second component derives from $Y, bY'c$, it can switch to first component or derive aX' from X.

In the first component X' can be changed to X *or* Y' can be changed to Y or both . The derivation proceeds similarly.

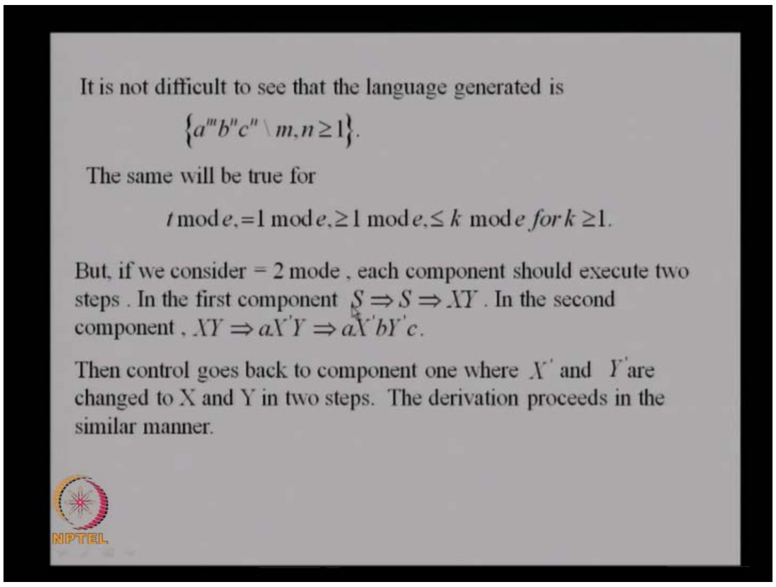
Let us consider some example consider the following CD grammar system, where you have these are the non terminals S, X, X dash Y; Y dash terminals are a b c S is the start symbol. You have two components, in one component you have these set of rule the other component you have these sets. So, these are the rules in the first component in the second component you have the following set.

Now, what is the language generated and their different modes suppose you consider the star mode any time you can switch from one component to another. You have to start the derivation which is S so, S goes to S you can use any number of times you want then you have to go from S to X, Y. Now, when you go to X, Y you cannot proceed in this component, because there is no rule with X on the left hand side or y on the left hand side here, and you have to use either this or this.

Now, if we use this the star mode after once step again you can come back here, if you use this again, you can come back here or you can use both and come back here, then X dash and Y dash can be turned into X and Y. You can just turn one of them and again go back or turn both of them and go back all possibilities exist, because of that you find that this rule along with X dash goes to X will generate number of a’s. So, one a will be generated again you can repeat this process any number of times. So, as many a as you want you can generate similarly, you can use this rule and again change Y dash to Y and then use this rule any number of times you want.

So, with this sort of a thing any number equal number of a's and b's will be b's equal number of b and c will be generated. And that can be done any number of times finally, you end up the derivation with this rule or with this rule both X and Y have to become terminals as long as there is one non terminal the derivation continues. So, in the sentential form you will have either X dash or Y dash or both or X or Y or both you can have X dash and Y are X and Y dash and so on. So, it is possible to derive any number of a's and any numbers of b's and c's, but the number of b's and c's will be equal, because this is the linear rule.

(Refer Slide Time: 18:44)



It is not difficult to see that the language generated is

$$\{a^m b^n c^n \setminus m,n \geq 1\}.$$

The same will be true for

$$t \text{ mod} e, =1 \text{ mod} e, \geq 1 \text{ mod} e, \leq k \text{ mod} e \textit{ for } k \geq 1.$$

But, if we consider $= 2$ mode , each component should execute two steps . In the first component $S \Rightarrow S \Rightarrow XY$. In the second component , $XY \Rightarrow aX'Y \Rightarrow aX'bY'c$.

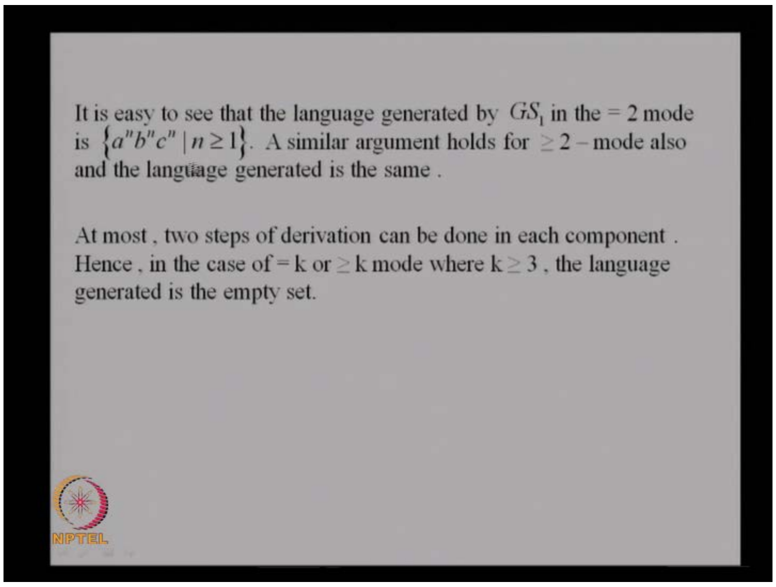
Then control goes back to component one where X' and Y' are changed to X and Y in two steps. The derivation proceeds in the similar manner.

So, the language generated will be equal number of b's and c's and any number of a's of course, there is no connection between m and n except the they are greater than or equal to 1. So, the same idea you can also consider for t mode e equal to 1 mode greater than or equal to mode less than or equal to k mode etcetera, but if you consider the equal to 2 mode there is a difference. So, if I consider equal to 2 mode (No audio from 19:13 to 19:21) in each component there should be 2 steps taking place.

So, what you have is you start here 1 step second step. So, X and Y you have and now, you switch to the next component and the next component you have this rule and this rule. So, when you have this rule 1 a is generated when you apply this rule 2 steps have to be performed in this component and the only way you can do is this and this. So, 1 a will be generated 1 b will be generated 1 c will be generated now, after applying 2 steps you go back to the first component change the X dash to X, Y dash to Y.

So, 2 steps are over here come back again generate 1 a 1 b and 1 c go back and. So, on this you can continue and every time you come to the second component 1 a, 1 b and 1 c will be generated and when you go to the first component X dash and X, Y dash will be change into X and Y finally, you have to terminate the derivation with this rule and with this rule if you use just one of them and go back there will be problem. What is the problem you are spending only one rule here and it is not possible you have to use two rules. So, when you want to use this rule and then if suppose you use this rule and go back here you can use only one rule and that is not allowed you have to use two rules in each component.

(Refer Slide Time: 21:04)

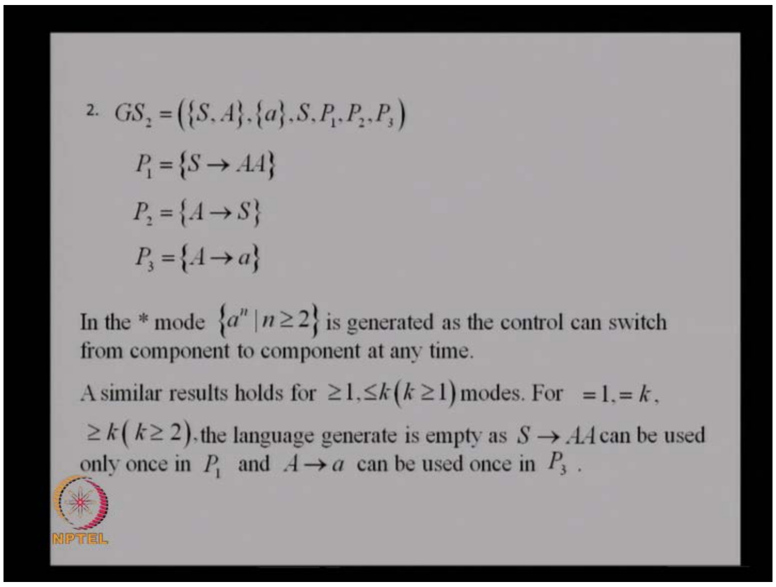

It is easy to see that the language generated by GS_1 in the = 2 mode is $\{a^n b^n c^n \mid n \geq 1\}$. A similar argument holds for ≥ 2 – mode also and the language generated is the same .

At most , two steps of derivation can be done in each component . Hence , in the case of = k or $\geq$ k mode where k $\geq$ 3 , the language generated is the empty set.

So, when you have equal to 2 mode the language generated will be a power n, b power n c power n, which is a context sensitive language. Now, the rules in each component can be type 0, type 1, type 2 or type 3, but of interest is type 2, because when you have context free rules the power is increased.

You are able to generate context sensitive languages with context free rules in a grammar system whereas, if you use just type 3 grammar regular rules then the power will not really be increased you will be able to get only regular sets. Now, suppose k is greater than or equal to k mode or greater than or equal to mode you consider where k will be 3, 4 or 5 then look at the rules here, I can have only 2 steps I cannot have more than 2 steps. So, if it is equal to k or greater than or equal to k mode where k is 3, 4 or 5 or etcetera no derivation is possible the language generated is empty.

(Refer Slide Time: 22:15)



2. $GS_2 = (\{S, A\}, \{a\}, S, P_1, P_2, P_3)$

$P_1 = \{S \to AA\}$

$P_2 = \{A \to S\}$

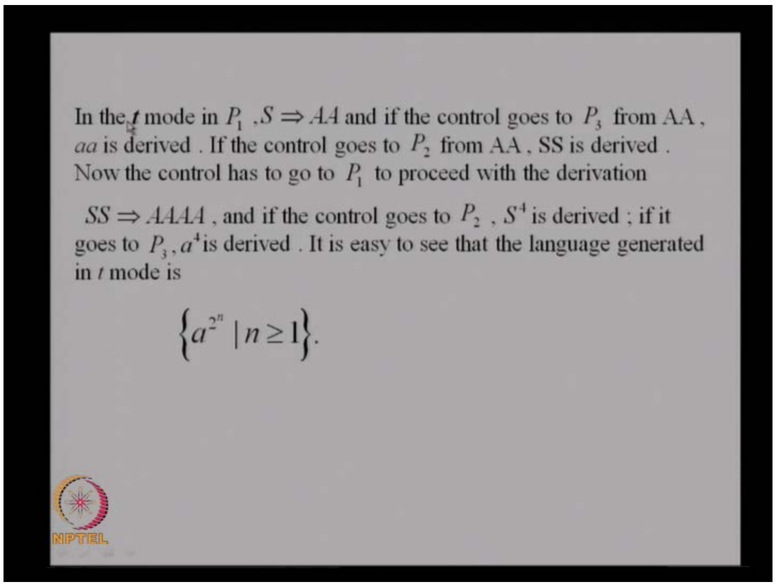
$P_3 = \{A \to a\}$

In the * mode $\{a^n \mid n \geq 2\}$ is generated as the control can switch from component to component at any time.

A similar results holds for $\geq 1, \leq k\,(k \geq 1)$ modes. For $= 1, = k$, $\geq k\,(k \geq 2)$, the language generate is empty as $S \to AA$ can be used only once in P_1 and $A \to a$ can be used once in P_3.

Another grammar let us look into this. So, this is a grammar with three components there are two non terminals, one terminal and three components the rules are like this. In the star mode we can use this rule and then again you can make it go into this you can generate A, A, A anytime you can convert the A into small a or convert it into S and generate 2 more A S and you can see that you can generate 2 A S, because first time you have to use this rule 1 A it is not possible to generate it is possible to generate 2 A S or 3 A S and so on. So, the language generated will be this a power n, n greater than or equal to 2 which is a regular set now, similar a cell holds for equal to 1 great equal to k and. So on.

(Refer Slide Time: 23:13)



In the t mode in P_1 , $S \Rightarrow AA$ and if the control goes to P_3 from AA , aa is derived . If the control goes to P_2 from AA , SS is derived . Now the control has to go to P_1 to proceed with the derivation

$SS \Rightarrow AAAA$, and if the control goes to P_2 , S^4 is derived ; if it goes to P_3 , a^4 is derived . It is easy to see that the language generated in t mode is

$$\left\{a^{2^n} \mid n \geq 1\right\}.$$

What is the language generated in the t mode, if you have greater than or equal to k, k greater than or equal to 2, then the language generated is empty as this can be used only once in P 1 and a goes to a can be used only once in P 3 for 3 equal to 1 mode. Now, in the t mode what is the language generated go back S goes to A A then suppose I start using this rule then I have to convert both the a into S S then I go back here then both the S I have to convert into A A.

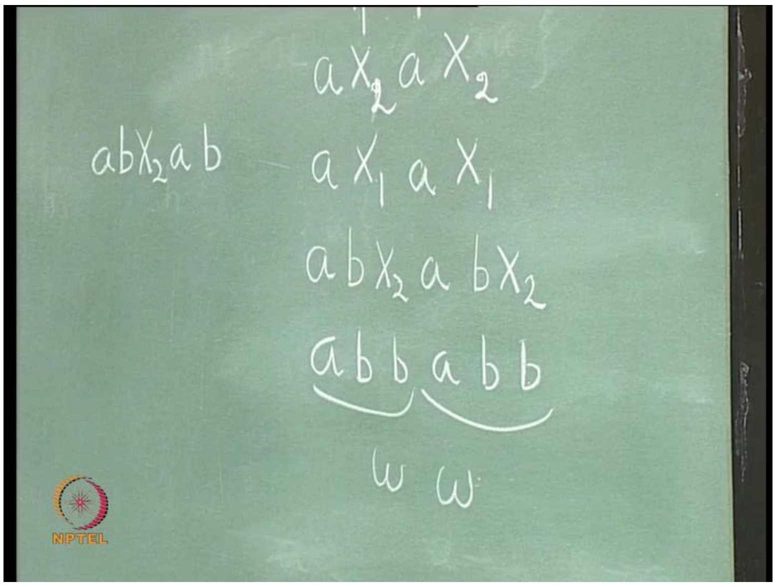
So, I will get 4 a. So, the derivation would be like this either I will get 2 a and the a can be converted into small a. Where I will get a square or they will get converted to S S and again, I use the rule S goes to A A it is a terminating mode I should proceed in the same component as long as possible. So, again this S also will be converted into 2 a. Now, I have a chance to go to component 3 where all the 4 a will be converted into small a. It is not possible to just convert portion of it and leave the other, because it is a terminating mode as long as it is possible to apply the rule a goes to a I will have to use it then if I use the rule a goes to S all the 4 a will be converted into 4 S S.

And so, 4 S S will be there they will be converted into 8 a using the first component and again I have the choice of making them into small a or proceeding further. So, you will find that you can generate a power 4. We can generate a squared you can generate a power 8 and so on. So, the language generated in the t mode is a power 2 power n another grammar let us consider this also has three components the first component has rules like this the second component has rules like this and so on. There are three non terminals, two terminals and so, what happens when you have star mode equal to 1 mode less than or equal to k mode.

So, you have to apply this rule to proceed further then I can just proceed with this X 1 in the star mode and generate something I want turning into X 2 and then again going back to X 1 going back here and so on, anything I can derive. So, from X 1 I can derive some string of a and b I can do the same thing with this X 1 have derive some string from the second X 1. So, I can clearly derive two different strings from the 2 X 1's, but minimum is one symbol. So, the length of the string will be generated the length of the string generated will be greater than or equal to 2 minimum length is 2 and actually w will be of the form w 1, w 2 where w 1 and w 2 are any strings of a's and b's.

So, that is why you just have any string of a's and b's generated where the length of the string will be minimum 2 it can be anything, but what happens when you use equal to 2 mode. So, at each component you have to use 2 steps the derivation in each component consists of 2 steps in that case what happens you use 2 steps S goes to S, S goes to X 1 X 1. Now, from this X 1 if I use this rule and from the second X 1 if I use this rule I can derive 1 a and 1 a but if I go here X 2 can be converted into X 1, but here I have to spend 2 steps right that is not possible. So, if I use this rule for this X 1 I have to use the same rule for this X 1.

(Refer Slide Time: 28:01)



abX_2ab

aX_2aX_2

aX_1aX_1

abX_2abX_2

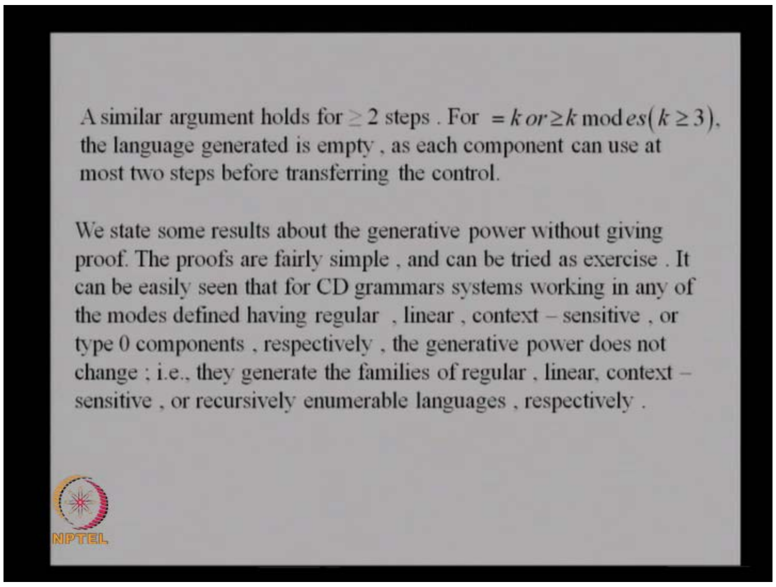
$\underbrace{abb}_{w}\underbrace{abb}_{w}$

So, the derivation will be like this in the first component starting with S I get S in 1 step actually the rule S goes to S is just added. So, that you are able to get 2 steps in the first component then from this you get X 1, X 1 this is done in 2 steps then you have to go to another component. If you go to P 2 and apply the rule you will get a X 1 you have to use the same rule here a X 2, a X 2, a X 2 then you go back to the first component and convert this into a X 2 into X 1, X 2 into X 1.

So, from here to here there are 2 steps this X 1 derives this X 1 derives this and here again 2 steps X 2 goes to X 1, X 2 goes to X 1 then I can proceed using the rule by going to the third component b X 2 a then using the same rule I can get b X 2 and so on, then I can terminate the derivation using something. So, I can get something like a b b again note that you are using two rules in each component suppose at this stage I tried to use b X 2 here, but just b here what happens when I go to the first component to convert X 2 to X 1 I can use only 1 step and that is not allowed I have to use 2 steps.

So, when I use the rule it has to be a repeated twice and I get strings of the form w w. We know that this is a context sensitivity language, but note that each component has only context free rules. So, with context free rules you are able to generate context sensitive languages. In fact, the 3 languages a power n, b power n, c power n, w w and a power n, b power m, a power n, b power m they are called they have some features, they are useful for defining what is known as mildly context sensitive languages. And usually when you consider a context sensitive language means usually consider these three languages and show that they can be generated of course, a power n square does not come under this. (No audio from 30:51 to 31:05)

(Refer Slide Time: 30:51)

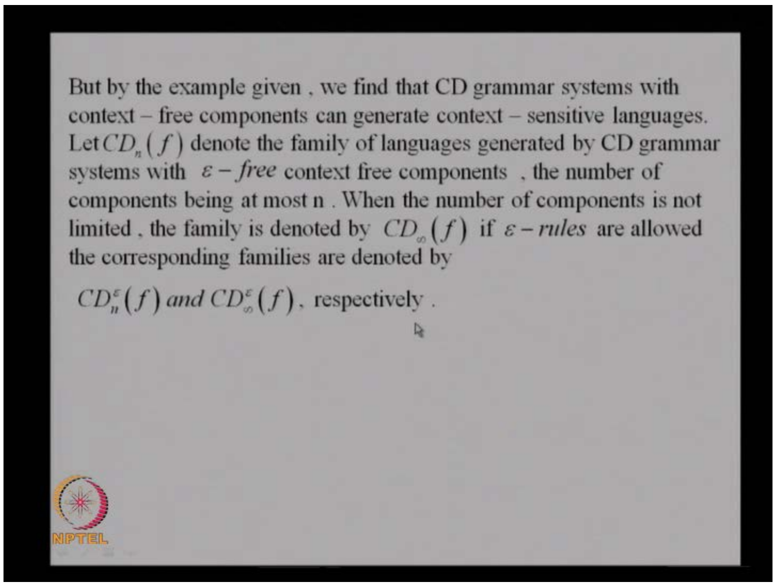

A similar argument holds for ≥ 2 steps . For $= k\, or \geq k$ modes$(k \geq 3)$, the language generated is empty , as each component can use at most two steps before transferring the control.

We state some results about the generative power without giving proof. The proofs are fairly simple , and can be tried as exercise . It can be easily seen that for CD grammars systems working in any of the modes defined having regular , linear , context – sensitive , or type 0 components , respectively , the generative power does not change ; i.e., they generate the families of regular , linear, context – sensitive , or recursively enumerable languages , respectively .

Now, what can you say about the generative power. It is interesting that context free rule with context free rules you are able to generate context sensitive languages. But what happens if you use type 0 or type 1 rules actually type 0 or type 1 it can be seen that the CD grammar systems working in any of the modes. It can be star mode, t mode or anything, but if you have regular rules or linear rules or context sensitive rules or type 0 rules then the generative power does not change it does not increase whatever you can do with n components you can just do with one component.

We can re-write the rules in such a way that you achieve the same thing with one component they if generate the families of regular linear context sensitive and recursively enumerable languages. The power is not increased where as when you use context free rules the power is increased.

(Refer Slide Time: 32:07)

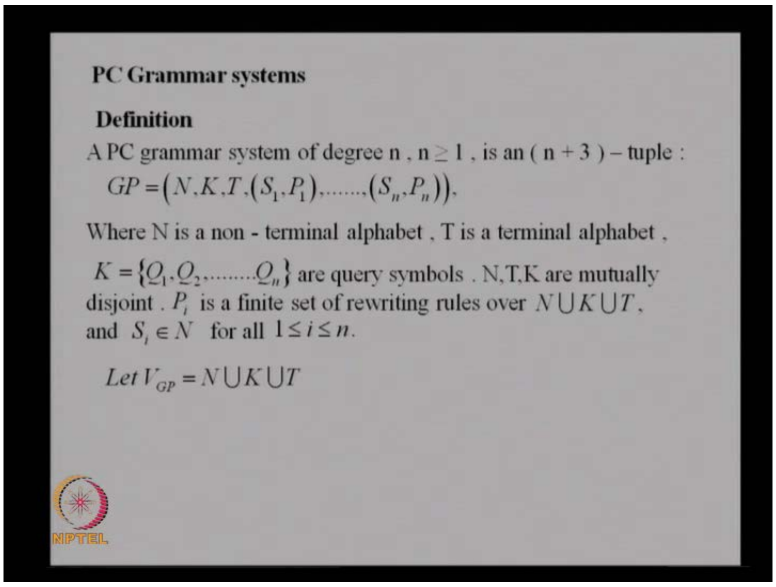


But by the example given, we find that CD grammar systems with context – free components can generate context – sensitive languages. Let $CD_n(f)$ denote the family of languages generated by CD grammar systems with $\varepsilon - free$ context free components, the number of components being at most n. When the number of components is not limited, the family is denoted by $CD_\infty(f)$ if $\varepsilon - rules$ are allowed the corresponding families are denoted by

$CD_n^\varepsilon(f)$ and $CD_\infty^\varepsilon(f)$, respectively.

So, this is seen by the examples which you have just now, seen by the examples given you find that the CD grammar systems with context free components can generate context sensitive languages. If, you denote by CD n (f) the family of languages generate by CD grammar systems, which epsilon free context free rules and n maximum n components then you also denote why CD infinity (f). If you do not put that restriction on the number of components and if you are allowing epsilon rules you also use this and this and there are several results related to their power. We will not go into the details of them.

(Refer Slide Time: 32:56)



PC Grammar systems

Definition

A PC grammar system of degree n, n ≥ 1, is an (n + 3) – tuple :

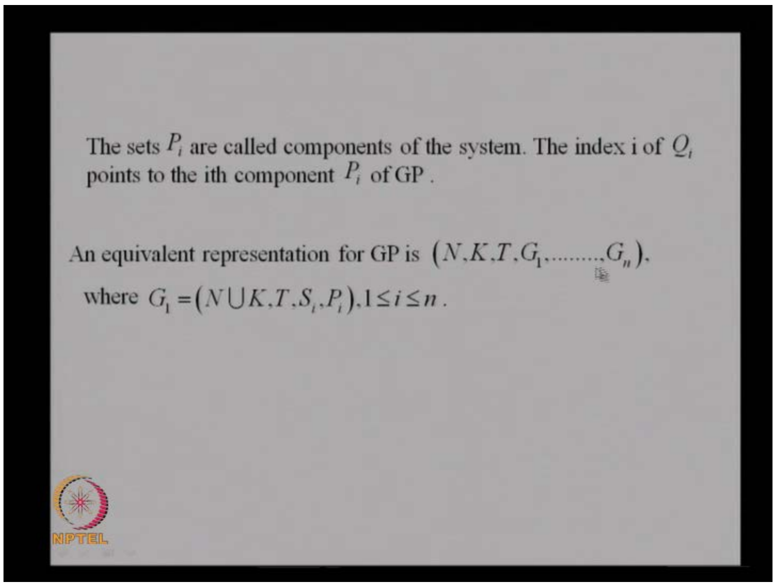
$GP = (N, K, T, (S_1, P_1), \ldots\ldots, (S_n, P_n))$.

Where N is a non - terminal alphabet, T is a terminal alphabet, $K = \{Q_1, Q_2, \ldots\ldots\ldots Q_n\}$ are query symbols. N,T,K are mutually disjoint. P_i is a finite set of rewriting rules over $N \cup K \cup T$, and $S_i \in N$ for all $1 \le i \le n$.

$Let\ V_{GP} = N \cup K \cup T$

Now, let us come back to the other model PC grammar systems or this is called parallel communicating grammar systems or the classroom model. So, what happens here is you have a set of non terminals the formal definition is this. You have a set of non terminals, you have a set of terminals and you have another set which is called the set of query symbols. If, there are n components you have n query symbols and corresponding to each component you have a start symbol, you have a set of productions, start symbol a set of productions and so on. Now, in this you have n components. So, there are n start symbols and n production sets now, without loss of generality we assume N T K are all mutually disjoined and the total alphabet is denoted by N union K union T and we denote it as V GP is N union K union T.

(Refer Slide Time: 34:11)

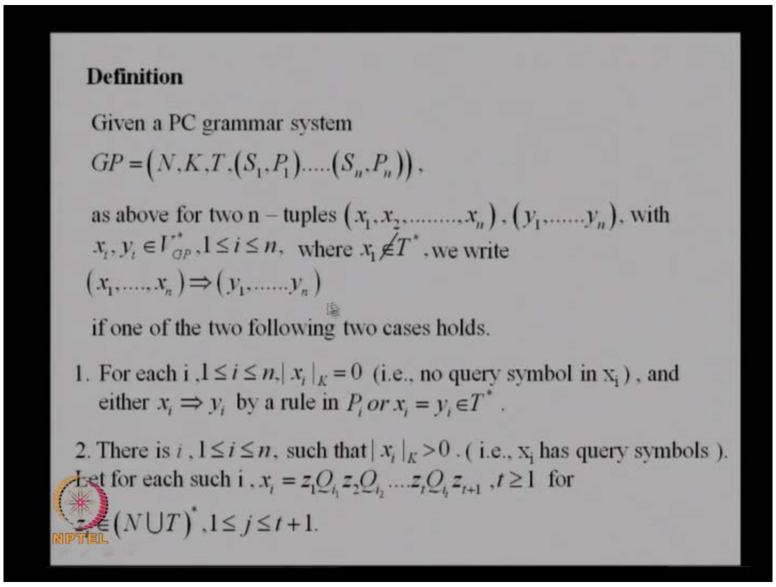


The sets P_i are called components of the system. The index i of Q_i points to the ith component P_i of GP .

An equivalent representation for GP is $(N, K, T, G_1, \ldots\ldots\ldots, G_n)$, where $G_1 = (N \cup K, T, S_i, P_i), 1 \le i \le n$.

The sets P i are called the components of the system the index i for Q i Q a is the query symbol for the i’th component. So, when Q i appears the answer has to be provided by the i’th component the equivalent representation is also like this instead of P 1, P 2, P n, S 1, P 1, S 2, P 2, S n, P n you can also have G 1, G 2, G n where G i is G i is N union K T S i P i.

(Refer Slide Time: 34:50)



Definition

Given a PC grammar system

$GP = (N, K, T, (S_1, P_1) \ldots\ldots (S_n, P_n))$,

as above for two n – tuples $(x_1, x_2, \ldots\ldots\ldots, x_n)$, $(y_1, \ldots\ldots y_n)$, with $x_i, y_i \in V_{GP}^{*}, 1 \le i \le n$, where $x_1 \notin T^*$, we write

$(x_1, \ldots\ldots x_n) \Rightarrow (y_1, \ldots\ldots y_n)$

if one of the two following two cases holds.

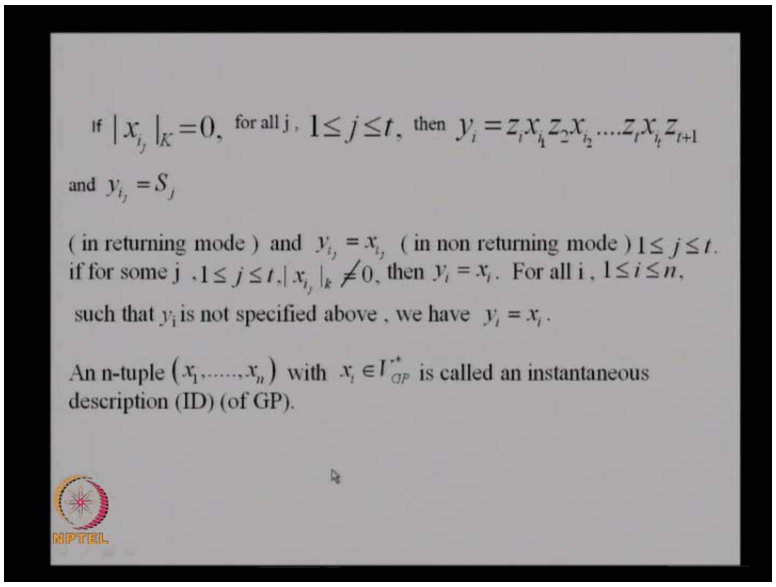
1. For each i $, 1 \le i \le n, |x_i|_K = 0$ (i.e., no query symbol in x_i), and either $x_i \Rightarrow y_i$ by a rule in P_i or $x_i = y_i \in T^*$.
2. There is $i, 1 \le i \le n$, such that $|x_i|_K > 0$. (i.e., x_i has query symbols). Let for each such i, $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}, t \ge 1$ for $z_j \in (N \cup T)^*, 1 \le j \le t+1$.

Given a PC grammar system like this how do you define derivation in this case? So, you have n strings and from this is an ID instantaneous description it consists of n strings from this the next ID is y 1 y 2 y n. How do you get this there are 2 steps two different types of steps? you can use one is the actual derivation another is the communicating step.

Now, there are two possibilities here, none of the exercise contain the query symbol this tells you that X i does not contain any query symbol no query symbol in X i and then there is a non terminal. if there is a non terminal you can use a derivation step and get y i from X i. If, there is no non terminal it is only the terminal string. You have to keep it as it is the other one is the communication step that is X i has a query symbol. This denotes that this sort of notation denotes that X i has a query symbol.

In that case suppose X i is like this Z 1 Q i 1, Z 2 Q i 2, Z t Q i t, Z t Q i plus 1 where Z 1, Z 2 they do not contain any query symbols the query symbols appearing in X i are Q i 1, Q i 2, Q i r then what happens is the next stage X i is replaced by putting the value of the corresponding string in the i 1’th component here, the corresponding string in the i 2’th component here, corresponding string in the i t’th component here and you get y i.

(Refer Slide Time: 36:55)



If $|x_{i_j}|_K = 0$, for all j, $1 \le j \le t$, then $y_i = z_i x_{i_1} z_2 x_{i_2} \dots z_t x_{i_t} z_{t+1}$

and $y_{i_j} = S_j$

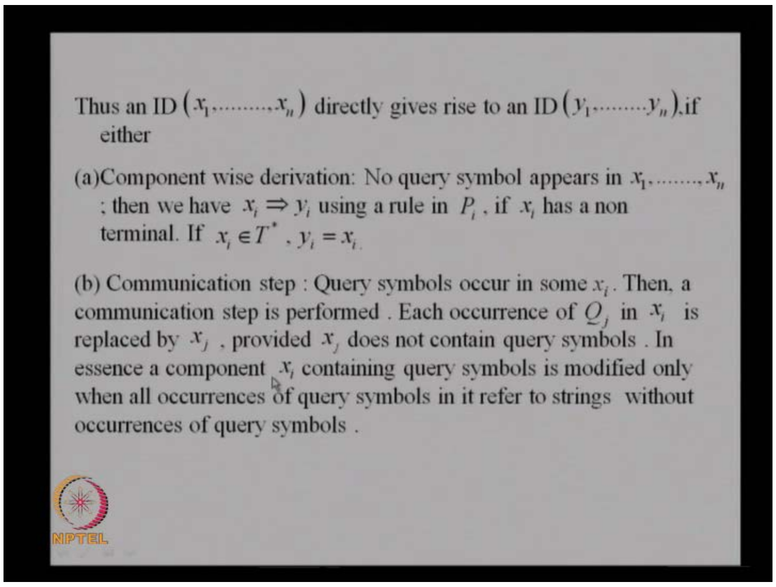
(in returning mode) and $y_{i_j} = x_{i_j}$ (in non returning mode) $1 \le j \le t$.
if for some j $, 1 \le j \le t, |x_{i_j}|_k \ne 0$, then $y_i = x_i$. For all i, $1 \le i \le n$, such that y_i is not specified above, we have $y_i = x_i$.

An n-tuple $(x_1, \dots\dots, x_n)$ with $x_i \in \Gamma_{GP}^{*}$ is called an instantaneous description (ID) (of GP).

So, the y i will be Z i, X i 1, Z 2 etcetera Z 1, Z 2, Z t, Z t plus 1 remain as they are, but the query symbols are replaced by the strings provided by the corresponding components. Now, after this is done from X i you have got y i in the i’th component what happens for the i j’th component. The i 1’th component provides this string to the i’th component the i 2’th component provides the string for the i’th component, but what happens to them what happens to the i j’th component it starts again once again it start from S j that is called the returning mode. So, it returns the start symbol in the non returning mode it keeps the corresponding string X i j as it is and proceeds further.

Let us see one example and this is achievable only if these strings do not contain query symbols if, they have query symbols there will be problem you cannot do this. So, you have an instantaneous description like this.

(Refer Slide Time: 38:12)



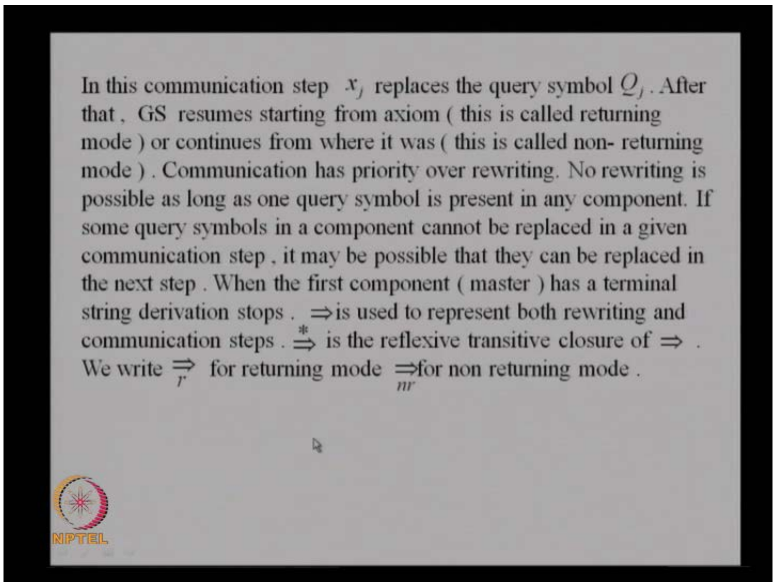
Thus an ID $(x_1 \ldots\ldots\ldots x_n)$ directly gives rise to an ID $(y_1 \ldots\ldots\ldots y_n)$,if either

(a)Component wise derivation: No query symbol appears in $x_1, \ldots\ldots, x_n$; then we have $x_i \Rightarrow y_i$ using a rule in P_i , if x_i has a non terminal. If $x_i \in T^*$, $y_i = x_i$

(b) Communication step : Query symbols occur in some x_i . Then, a communication step is performed . Each occurrence of Q_j in x_i is replaced by x_j , provided x_j does not contain query symbols . In essence a component x_i containing query symbols is modified only when all occurrences of query symbols in it refer to strings without occurrences of query symbols .

So, an ID is a n tuple like this each is a string over the total alphabet and it can give rise to the next ID y 1, y 2, y n if you can have component wise derivation that is no query symbol is there from X 1 you derive y 1 from X 2 you derive y 2 and so on. Each X i has a non terminal if there is no non terminal it is a terminal string you keep it as it is. So, you either use this rule or you keep it as it is then you have the communication step in the communication step. You have query symbols suppose Q j appears in X j then Q j will be replaced by X j provided X j does not contain any query symbol.

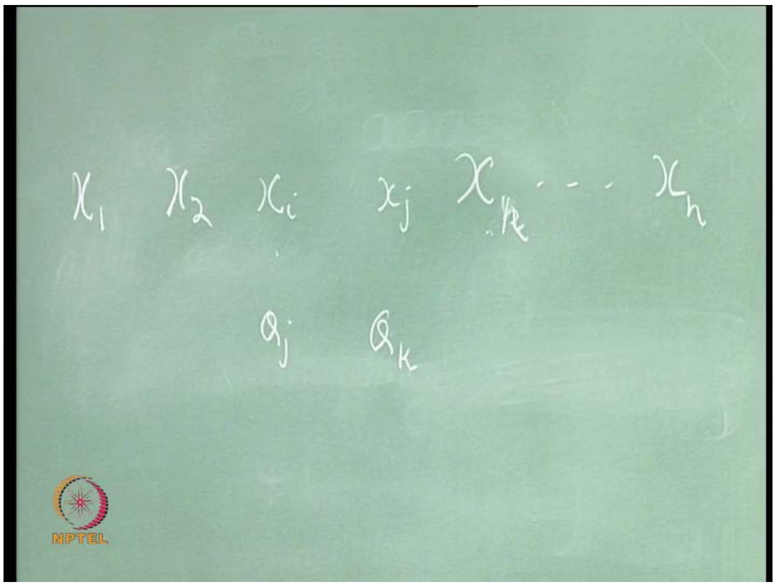
In essence the component X i contains query symbols is modified only when all occurrences of the query symbols in it refer to strings without the occurrence of the query symbols. The communicating step X j replaces the query symbol Q j after that the system assume starting from the x m that is the i'th, j'th component goes back to the start symbol means it is called the returning mode or if it continues from where it was it is called a non returning mode.



In this communication step x_j replaces the query symbol Q_j. After that, GS resumes starting from axiom (this is called returning mode) or continues from where it was (this is called non- returning mode). Communication has priority over rewriting. No rewriting is possible as long as one query symbol is present in any component. If some query symbols in a component cannot be replaced in a given communication step, it may be possible that they can be replaced in the next step. When the first component (master) has a terminal string derivation stops. $\Rightarrow$ is used to represent both rewriting and communication steps. $\overset{*}{\Rightarrow}$ is the reflexive transitive closure of $\Rightarrow$. We write $\underset{r}{\Rightarrow}$ for returning mode $\underset{nr}{\Rightarrow}$ for non returning mode.

The communicating step X j replaces the query symbol Q j after that the system assume starting from the x m that is the i'th, j'th component goes back to the start symbol means it is called the returning mode or if it continues from where it was it is called a non returning mode. And always communication has priority over rewriting only when there is no more communication possible you will use the derivation step, but there should not be any circular question, query symbols. We will see what a circular query symbol.

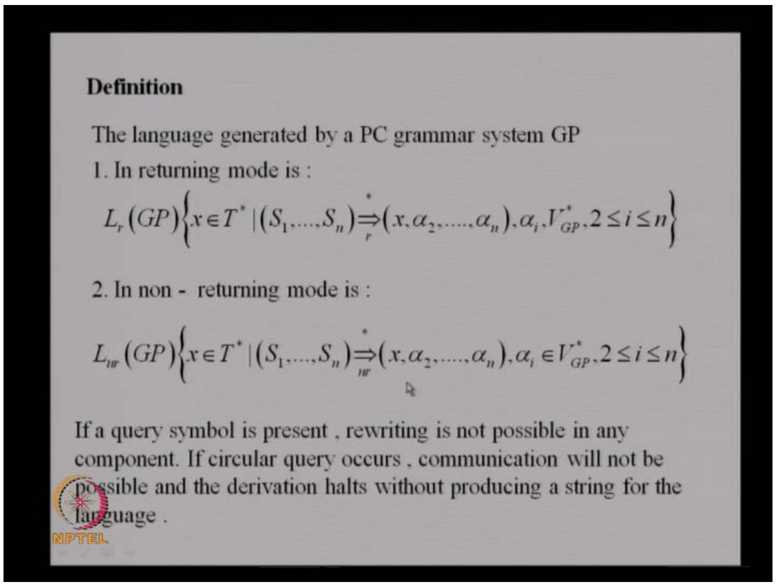
(Refer Slide Time: 40:24)



Now, you may not be able to replace X j by the corresponding string suppose at some stage I have X 1, X 2, X n and I also have say this has Q j, but X j has this symbol Q k say then I cannot replace it like this, but the k'th component suppose this is k component this is the k’th component this may not contain any symbol. So, first replace this with this then you use this to replace X i, Q j in X i. So, this can be done when it is not possible to do something like that it is called circular query. And the derivation stops at that stage if some query symbols in a component cannot be replaced in a given communication step.

It may be possible that they can be replaced in the next step that is what we can do when the first component first component is called the master of the system or the central centralised system. It has the terminal string when the first component derives the terminal string no more derivation takes place the derivation stops the double arrow is used to denote both the rewriting step and the communication steps, as usual double arrow star is the reflexive transitive closure of double arrow. When you use returning mode you use R below that when you use non returning mode you use n R below that.

(Refer Slide Time: 42:08)



Definition

The language generated by a PC grammar system GP

1. In returning mode is :

$$L_r(GP)\left\{x \in T^* \mid (S_1,\ldots,S_n) \overset{*}{\underset{r}{\Rightarrow}} (x,\alpha_2,\ldots,\alpha_n), \alpha_i, V_{GP}^*, 2 \le i \le n\right\}$$

2. In non - returning mode is :

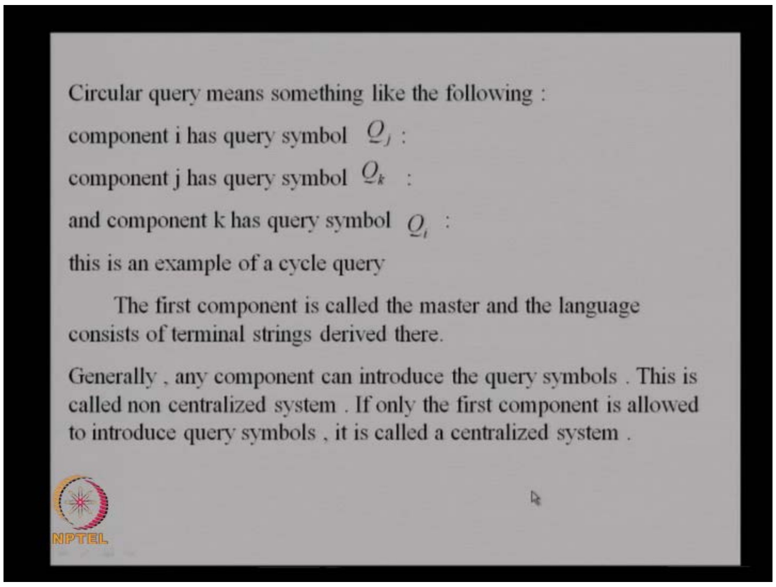
$$L_{nr}(GP)\left\{x \in T^* \mid (S_1,\ldots,S_n) \overset{*}{\underset{nr}{\Rightarrow}} (x,\alpha_2,\ldots,\alpha_n), \alpha_i \in V_{GP}^*, 2 \le i \le n\right\}$$

If a query symbol is present , rewriting is not possible in any component. If circular query occurs , communication will not be possible and the derivation halts without producing a string for the language .

The language generated by the PC grammar system in the returning mode is L r (GP) the first component derives a terminal string. So, you start with a n tuple S 1, S 2, S n and you arrive at x alpha 2, alpha 3, alpha n these may be non terminals or terminals does not matter, but the first component derives the terminal string in the non returning mode you use the non returning derivation steps. So, starting from the n tuple S 1, S 2, S n the first component derives the terminal string and that is the string belonging to the language if a query symbol is present rewriting is not possible.

So, you have to finish all the query symbols before further rewriting takes place. If a circular query occurs communication will not be possible and the derivation halts without producing a string for the language.

(Refer Slide Time: 43:07)



Circular query means something like the following :

component i has query symbol Q_j :

component j has query symbol Q_k :

and component k has query symbol Q_i :

this is an example of a cycle query

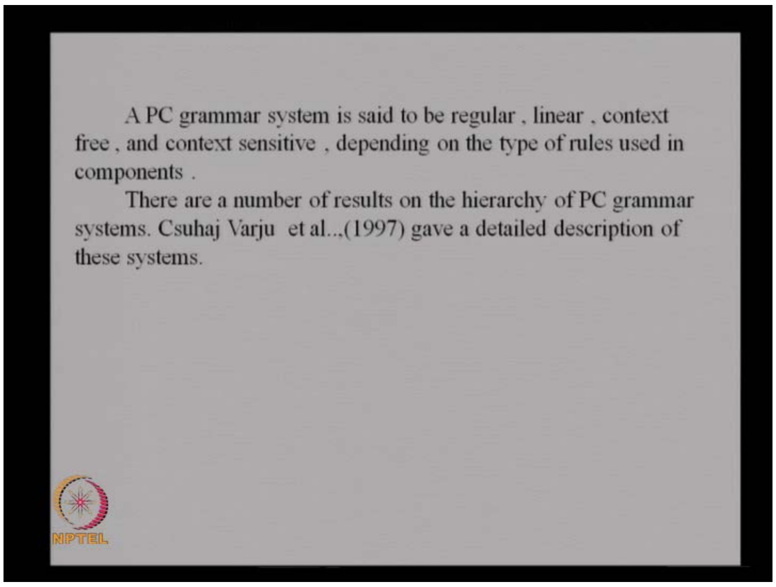
The first component is called the master and the language consists of terminal strings derived there.

Generally , any component can introduce the query symbols . This is called non centralized system . If only the first component is allowed to introduce query symbols , it is called a centralized system .

What do you mean by circular query component I has Q j and j has Q k and k component k has Q i. So, when it is asking for Q j, X j it has Q k and when it is asking for the i'th component it will have Q i. So, this way you will never be able to get out of all the query symbols the first component is called the master of the system. And the language consists of terminal strings derived there generally any component can introduce query symbols. So, you do not put any restriction any component can generate a query symbol and the corresponding component can give the answer. We saw the several students working in a classroom any student can ask a question if he does not get have the answer for that.

And the student who has the answer for that should produce the answer this is the non centralised version. The centralised version only class leader is allowed to ask questions. He is proceeding with the main program and main problem or the main program and when he gets some queries or when he wants a answer for something he ask the corresponding student who is working that particular portion of the problem. And that student has to provide the answer only the reader has the capability of asking questions or only he is permitted to ask questions or in the grammar system only the first component can produce query symbols, such a system is called the centralised system. So, you have centralised, non-centralised returning, non-returning. So, 4 types of thing you have 2 into 2.

(Refer Slide Time: 45:03)

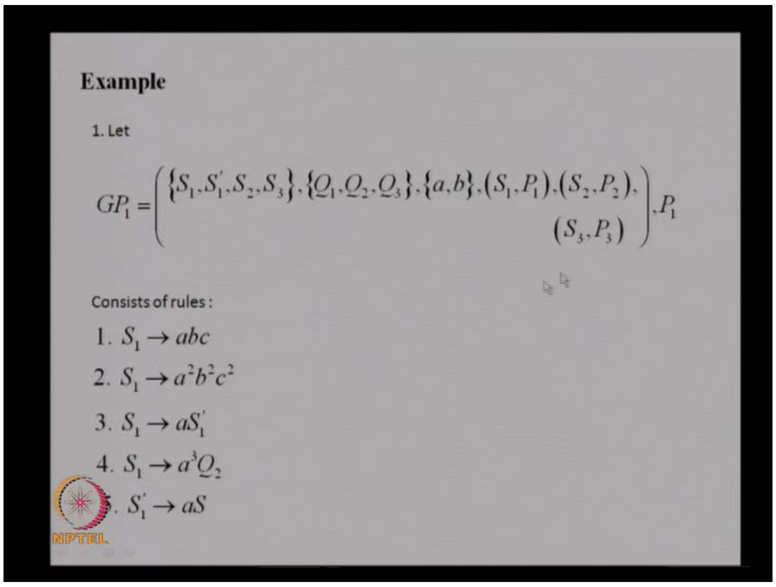


A PC grammar system is said to be regular, linear, context free, and context sensitive, depending on the type of rules used in components.

There are a number of results on the hierarchy of PC grammar systems. Csuhaj Varju et al..(1997) gave a detailed description of these systems.

And what is the language generated under different things that is been studied what can the rules be it can be regular rules, linear rules context free context sensitive. What is the class of language generated? What are the properties of them these are been studied (( )) book there are books chapters on this.

(Refer Slide Time: 45:21)



Example

1. Let

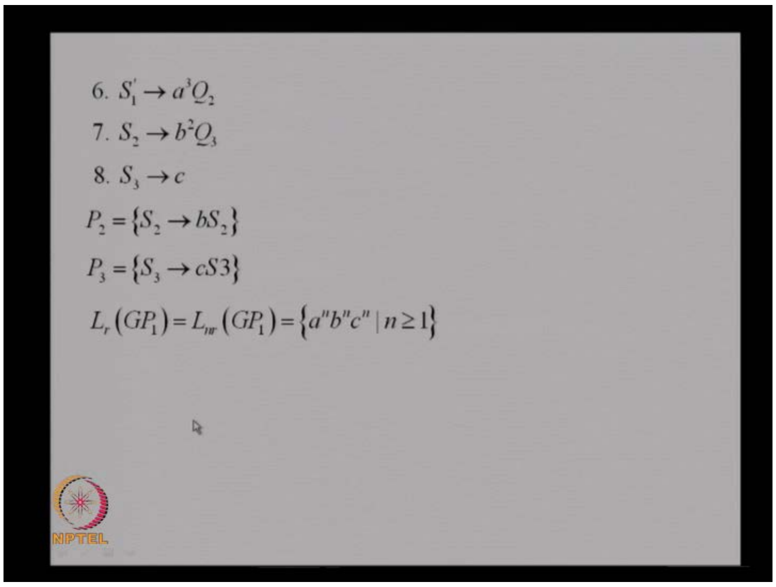
$$GP_1 = \left(\begin{array}{r} \{S_1, S_1', S_2, S_3\}, \{Q_1, Q_2, Q_3\}, \{a, b\}, (S_1, P_1), (S_2, P_2), \\ (S_3, P_3) \end{array} \right), P_1$$

Consists of rules:

1. $S_1 \rightarrow abc$
2. $S_1 \rightarrow a^2b^2c^2$
3. $S_1 \rightarrow aS_1'$
4. $S_1 \rightarrow a^3Q_2$
5. $S_1' \rightarrow aS$

So, let us just consider some examples. Now, consider this example you have three components P 1 has these rules you have S 1, P 1, S 2, P 2, S 3, P 3, three components.

(Refer Slide Time: 45:47)



6. $S_1' \rightarrow a^3 Q_2$

7. $S_2 \rightarrow b^2 Q_3$

8. $S_3 \rightarrow c$

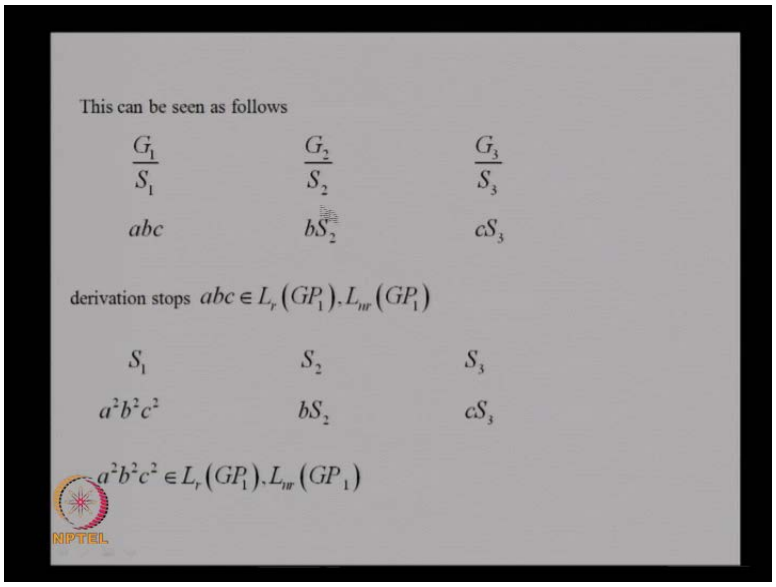
$P_2 = \{S_2 \rightarrow bS_2\}$

$P_3 = \{S_3 \rightarrow cS3\}$

$L_r(GP_1) = L_{nr}(GP_1) = \{a^n b^n c^n \mid n \geq 1\}$

P 1 has a number of rules like this 1, 2, 3, 4, 5 8 rules P 2 has just 1 rule P 3 has 1 rule what is the language generated in the returning and the non returning mode. We will find that the language generated is a power n, b power n and c power n.

(Refer Slide Time: 46:03)



This can be seen as follows

$\underline{G_1}$	$\underline{G_2}$	$\underline{G_3}$
S_1	S_2	S_3
abc	bS_2	cS_3

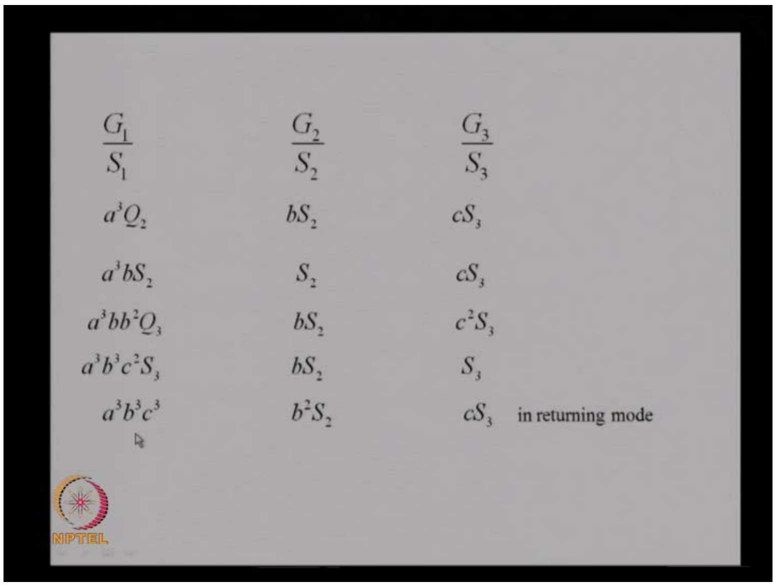
derivation stops $abc \in L_r(GP_1).L_{nr}(GP_1)$

S_1	S_2	S_3
$a^2b^2c^2$	bS_2	cS_3

$a^2b^2c^2 \in L_r(GP_1).L_{nr}(GP_1)$

You can see this S 1, S 2, S 3 there is only 1 rule. S 2 goes to b, S 2 S 3 goes to c S 3. In these two components you have to apply that only, but in component one you have 8 rule, you can use any one of them if you use the first rule, you can use the second rule, you can use any one of them. So, c if you use the first rule you get this is the terminal string no further derivation is allowed. So, this belongs to language a, b, c belongs to the language. It can be either returning mode or non returning mode does not matter similarly, you use the first rule you will get a square, b square, c square in the first component. It is both in the returning mode and the non returning mode it will be generated.

(Refer Slide Time: 46:53)

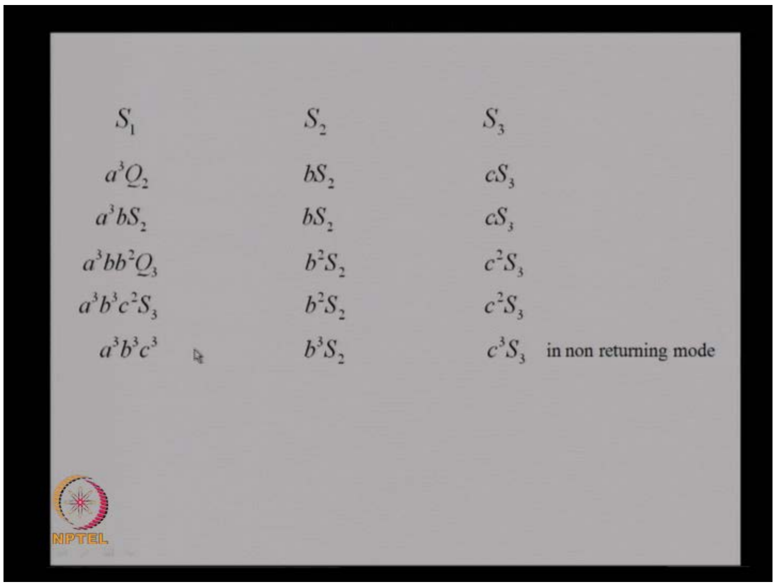


$\frac{G_1}{S_1}$	$\frac{G_2}{S_2}$	$\frac{G_3}{S_3}$	
a^3Q_2	bS_2	cS_3	
a^3bS_2	S_2	cS_3	
$a^3bb^2Q_3$	bS_2	c^2S_3	
$a^3b^3c^2S_3$	bS_2	S_3	
$a^3b^3c^3$	b^2S_2	cS_3	in returning mode

Now, what happens when you use the other rules suppose the fourth rule I use here. So, if I use the fourth rule in the first component a cube Q 2 is generated here. I can use only 1 rule be S 2 goes to b, S 2 here, I can use only 1 rule S 3 goes to c S 3. So, what I have is this now, a communication step takes place Q 2 is there the query symbol. So, the answer has to be provided from here. So, this is shifted here and the returning mode this becomes S 2 this is continuing as c S 3.

Now what is the rule for S 2, S 2 goes to b squared Q 3. So, if you use that S 2 goes to b square Q 3 this S 2 goes to b S 2, S 3 goes to c S 3. If you use you get this now, at this stage again the query symbol Q 3 occurs. So, this will be transferred here. So, when you transfer this you get a cubed c square S 3 this remains as it is you go back to S 3 here. That is returning mode in the returning mode you get this so, a cubed b cubed then you use the rule S 3 goes to c and you get this string.

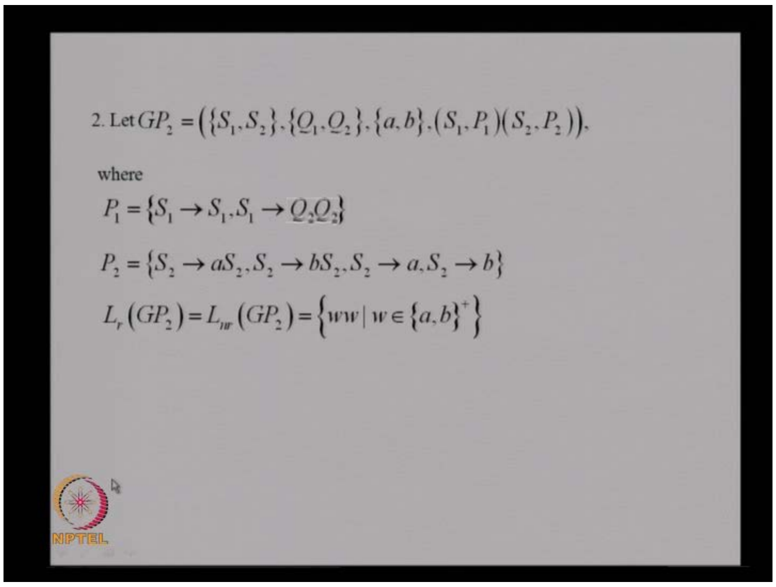
(Refer Slide Time: 48:33)



S_1	S_2	S_3	
a^3Q_2	bS_2	cS_3	
a^3bS_2	bS_2	cS_3	
$a^3bb^2Q_3$	b^2S_2	c^2S_3	
$a^3b^3c^2S_3$	b^2S_2	c^2S_3	
$a^3b^3c^3$	b^3S_2	c^3S_3	in non returning mode

Similarly, you can find that in the non returning mode also the same type of a derivation takes place, but here the components may be having something else. So, whatever it is you can find that a power 4, b power 4, c power 4 also can be generated in the returning mode and in the non returning mode.

(Refer Slide Time: 48:56)



2. Let $GP_2 = (\{S_1, S_2\}, \{Q_1, Q_2\}, \{a, b\}, (S_1, P_1)(S_2, P_2))$,

where

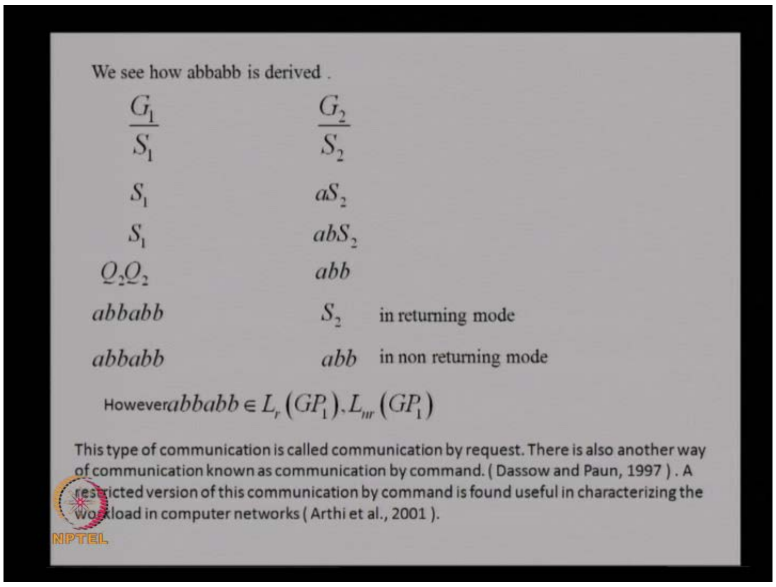
$P_1 = \{S_1 \rightarrow S_1, S_1 \rightarrow Q_2Q_2\}$

$P_2 = \{S_2 \rightarrow aS_2, S_2 \rightarrow bS_2, S_2 \rightarrow a, S_2 \rightarrow b\}$

$L_r(GP_2) = L_{nr}(GP_2) = \{ww \mid w \in \{a, b\}^+\}$

So, the language generated is a power n, b power n, c power n consider this grammar this is the PC grammar there are two components. This is one component, this is another component the rules are given by this the language generated is of this form w w.

(Refer Slide Time: 49:23)



We see how abbabb is derived .

$\frac{G_1}{S_1}$	$\frac{G_2}{S_2}$	
S_1	aS_2	
S_1	abS_2	
Q_2Q_2	abb	
$abbabb$	S_2	in returning mode
$abbabb$	abb	in non returning mode

However $abbabb \in L_r(GP_1).L_{nr}(GP_1)$

This type of communication is called communication by request. There is also another way of communication known as communication by command. (Dassow and Paun, 1997) . A restricted version of this communication by command is found useful in characterizing the workload in computer networks (Arthi et al., 2001).

Because in the first component always use the rule S 1 goes to S 1, S 1 goes to S 1, S 1 goes to S 1 and in the second component I can use the rule a goes to a S 2, b goes to b S 2 and so on, any string I can derive here. Now, once you terminate the derivation here, as long as you can proceed here you can proceed as much as you want 1 step, 2 step, 3 step any number of steps once you get a terminal string. You cannot derive anything further here and from S 1 if, you use the other rule the other rule here is S 1 goes to Q 1, Q 2 it should be Q 2, Q 2 it should be Q 2 Q 2.

So, when you generate Q 2, Q 2 here there are 2 query symbols appearing both of them refer to the second component. So, for this Q 2 a, b, b is supplied for this Q 2 again a, b, b is supplied. So, you get strings of the form w w here. And because you can use any order of derivation here any w can be generated. So, the generated string will be of the form w w in the returning mode this becomes S 2 the non returning mode. It becomes the same thing I mean not disturbed does not matter.

We are only interested in the terminal string generated in the first component. So, the string generated will be of the form w w. So, we have seen how context sensitive languages can be generated using the context free components in parallel communicating model also. This type of a communication is called communication by request there is also, something known as communication by command. What is communication by request? Because the query is generated query symbol is generated Q i is generated then you ask the i’th component to provide the answer for that.

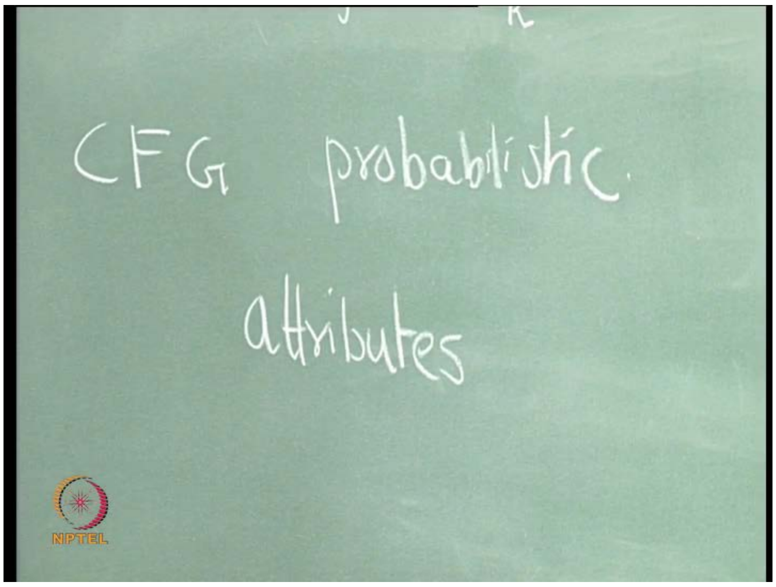
So, the request is made request is made and answer is provided. So, this sort of a communication is called communication by request; whereas, there is something called communication by command, what is communication by command? So, each component is deriving some string; and when a particular type of a string occurs, with each component, you also attach a set a regular set; and when that regular set appears when the string generated belongs to that regular set, something is communicated to other component.

This component at a particular stage when the string belongs to the set decides to communicate something from here to another component. So, it is giving a command. So, when something occurs a command is generated and some string is transferred from this component to another component. Earlier what we have considered is query symbol is generated as a request and something is provided for the query symbol. Here a some command is generated here and when that command is executed this will be transferred to some other component.

And this sort of a communication by command is also very useful in computer networks you want to study about the design of the system you may have (( )) you may have servers and you may want to talk about congestion how the system can be uniform distribution can be given among the servers. So, that one is not over loaded and so on. When you want to study this first of all it is very essential to study the work load on each server. What type of a work load is generated here?

One server may just handle one set of simple sort of a thing one client may be just using email and another client may not communicate with the server at all some text processing may just be going on there. So, it is very essential to characterise a work load with each server and then arrive at scenario where the load will be distributed properly. So, at one stage a few years back this has been important study in computer networks. How to characterise the work load and for that a parallel communicating grammar describing the behaviour of the system there it is a client or a server you can model it. What is known as a parallel grammar or you can have a context free grammar with probabilities and attribute.

(Refer Slide Time: 54:36)


CFG probabilistic

attributes

So, you can have a CFG. So, probabilistic (No audio from 54:40 to 54:47) and also it has an attributes each client or a server may have such a system, but then you know the whole computer networks describes a distributed environment. So, instead of modelling each one by a separate single context free probabilistic and attributed system, if you have a parallel communicating system it gives a very good result. And also the communication method used for that is better you use communication by command rather than communication by request.

So, these models have lot of practical applications the cooperative distributed model, which is a sequential model and the parallel model is parallel communicating systems again they have a lot of applications, practical applications where their use is very well utilised. So, this is another model another advanced topic, which is of current interest today.