Theory of Computation

Prof. Kamala Krithivasan

Department of Computer Science and Engineering

Indian Institute of Technology, Madras

Lecture No. # 38

Regulated Rewriting

In the next three lectures, we shall consider some advanced topics. We shall consider regulator rewriting and lindenmayer systems and grammar systems. Today, we shall consider regulator rewriting on grammars, what do we mean by regulator rewriting? The application of the rules is regulated by some external mechanism.

(Refer Slide Time: 00:37)

Regulated Re-writing

In a given grammar , re-writing can take place at a step of a derivation by the usage of any applicable rule in any desired place. That is if A is a nonterminal occurring in any sentential form say $\alpha A\beta$, the rules being

$$A \rightarrow \gamma$$

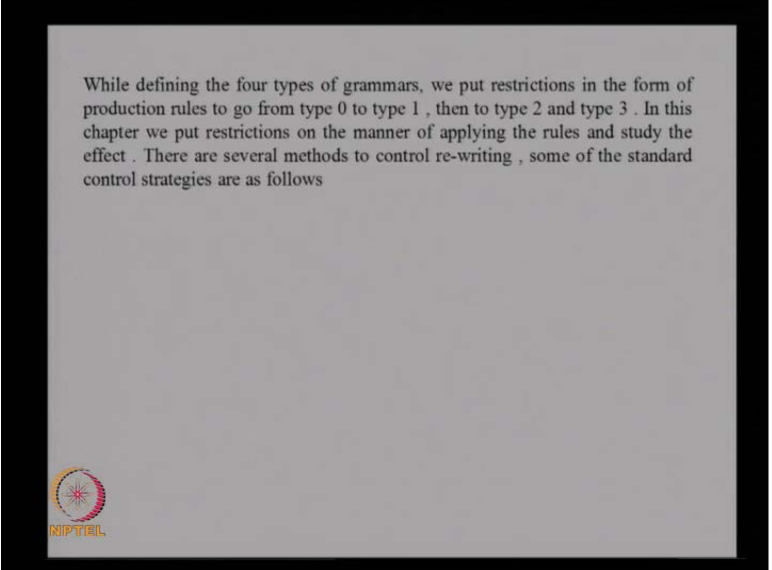
$$A \rightarrow \delta$$

then any of these two rules are applicable for the occurrence of A in $\alpha A\beta$. Hence, one encounters nondeterminism in its application. One way of naturally restricting the nondeterminism is by regulating devices, which can select only certain derivations as correct in such a way that the obtained language has certain useful properties. For example, a very simple and natural control on regular rules may yield a non regular language .

So, let us see what it means. In a given grammar the rewriting can take place at a step of the derivation like this. We have a sentential form alpha A beta, then at this stage, you can apply, suppose I have two rules A goes to gamma or A goes to delta. I can apply this rule A goes to gamma and get alpha gamma beta or I can apply the rule A goes to delta

and get alpha delta beta. There is nondeterminism, but if in some manner I try to control which rule is to be applied at that stage, then the derivation is being regulated, it is being controlled. If you do that, what happens that is what we want to study in this lecture.

(Refer Slide Time: 01:34)



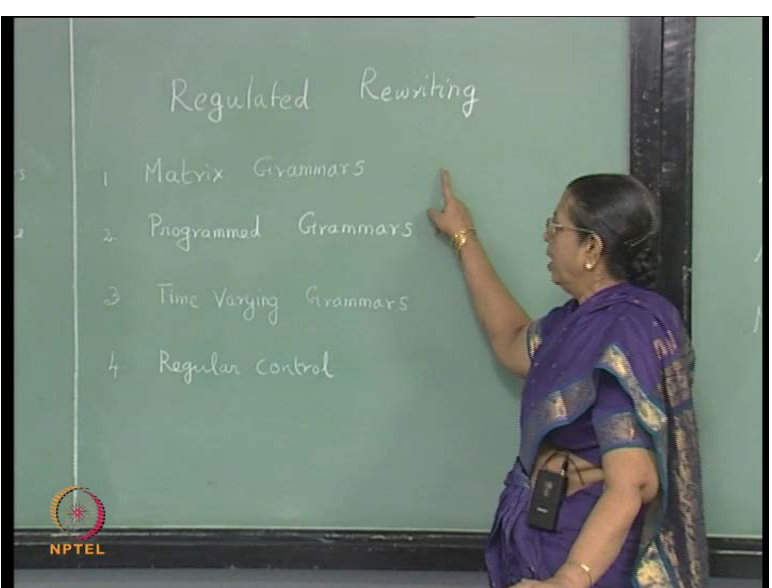
Now, while defining the four types of grammars; we have put restrictions in the form of the production rules, initially we define type 0 grammars. Then by putting the restriction that the length of the right hand side is greater than or equal to the length of the left hand side we obtained type 1. There is a slightly different definition of type 1 as well that we know. Then if we put still some more restrictions and say that on the left hand side, you can have a single non terminal and on the right hand side you can have a string. Then we have type 2 grammars or the context free grammars, which we have studied in detail. Then if you still put some more restrictions, and say that on the left hand side you have a single non terminal, and on the right hand side you have a single terminal; or a single terminal followed by a non terminal then we get type three grammars which generate the regular sets of course, we have to accommodate for the lambda rules also.

We find that by putting restrictions on the form of the production rules, we get a lesser class. In type 0 grammar, we have put some restrictions and obtained type 1 grammar and the class of type 1 languages is included in the class of type 0 languages, we have

obtained a smaller family. Similarly, by putting some more restriction, we have got type 2 grammars. And type 2 languages are still a smaller family by putting some more restrictions. We have obtained type 3 grammar and type 3 languages are regular set is the smallest family in the Chomsky and hierarchy.

So, we find that by putting restrictions on the form of the production rules, we are getting lesser and lesser, or smaller and smaller families. Now, we are going to put restriction on the manner of applying the rules not on the form, but on the manner of applying the rules. If you put some restrictions on the manner of applying the rules, what do we get? We find that the power is increased, that is you may be having just type three rules or the context free rules. Where the on the left hand side you have a single non terminal and on the right hand side you have a string of terminals and non terminals. But if you regulate the rewriting in some manner, you may even be able to go up to type zero. That is the power will increase, the generative power will be increased by putting restrictions on the manner of applying the rules.

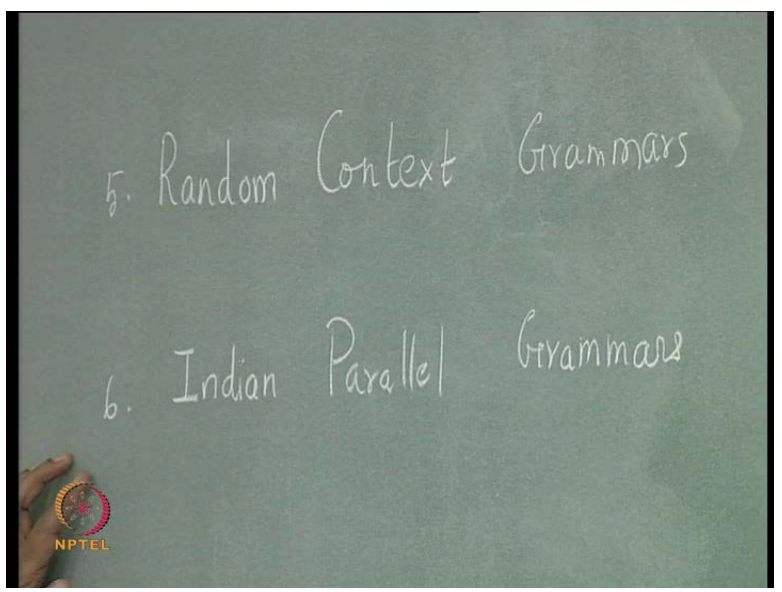
(Refer Slide Time: 04:50)



There are several ways in which you can put restrictions on the manner of applying the rules. We shall consider a few of them, we shall mainly consider the definitions and some examples, we are not going to prove any results though I may be stating some results. So,

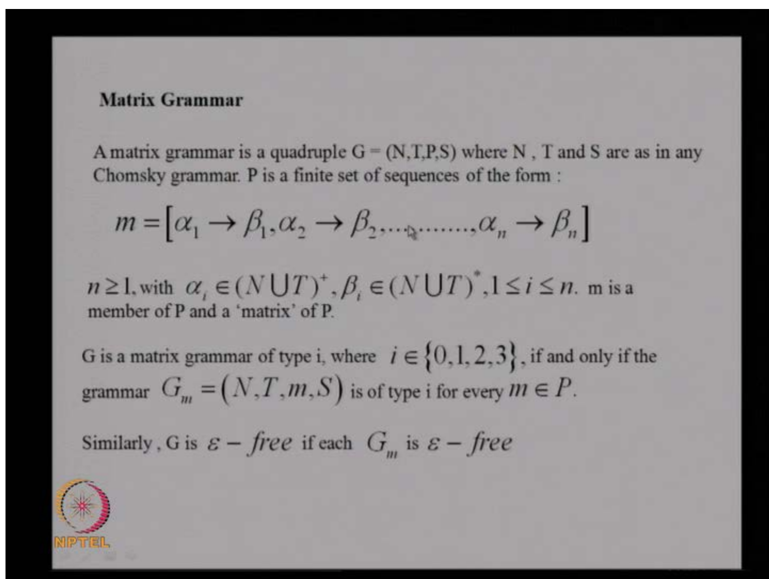
the classes we are going to study are matrix grammars and programmed grammars, time varying grammars and regular control. These four classes have something in common which we will see in the end. Apart from that we shall also study the definition of random context grammars, Indian parallel grammars.

(Refer Slide Time: 05:09)



So, one by one we will take it up first, we shall take matrix grammars. What is the definition of a matrix grammar?

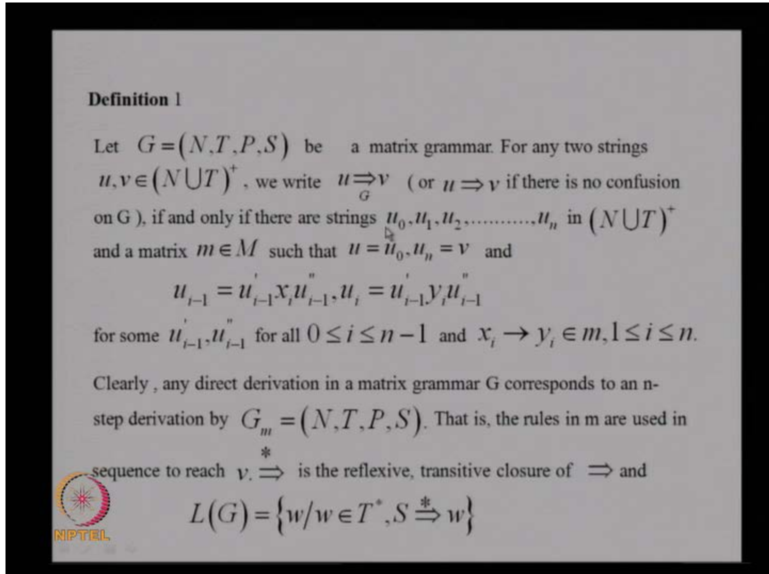
(Refer Slide Time: 05:26)



A matrix grammar is a quadruple G N T P S where N and T and S are as in any grammar. But the productions are a finite sequences of the form. It consists of matrices of the form m, matrices means it is sequence of rules, really that is rules are of the form alpha 1 goes to beta 1, alpha 2 goes to beta 2, alpha n goes to beta n, with alpha i belonging to N union T plus, that is the rules could be of type 0 type 1 or type 2 or type 3. Actually, it is not of much interest to study type 1 or type 0 or type 3, but type 2 is of interest, when the rules are context free, the power suddenly goes up to type 0, we shall see that.

So, matrix grammar is of the form N T m S, the rules are not like P, but they are sort of matrices sequences of rules. And the rules can be context free, they can be epsilon free depending upon that the language will be defined. The understanding is that when you have a sentential form and start applying this rule, next you have to apply this rule, next you have to apply the next rule and so on.

(Refer Slide Time: 06:57)

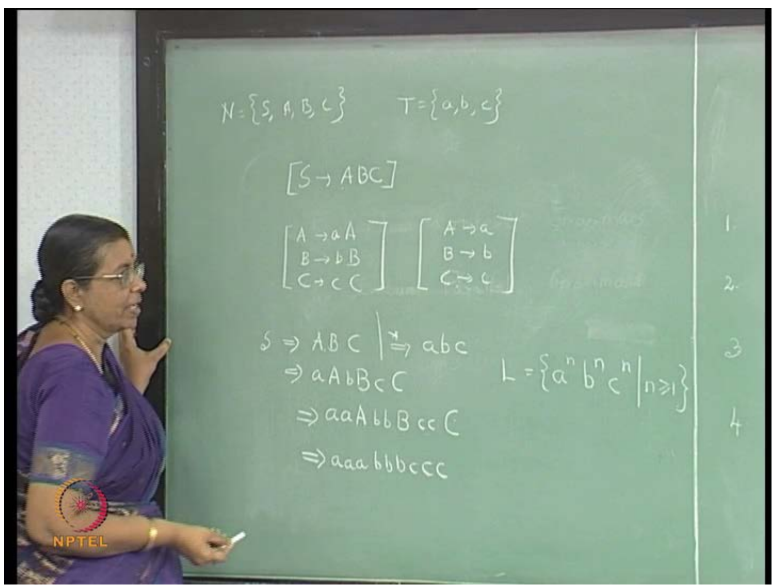


The matrix grammar the derivation from u to v is got like this, it is a sequence really u 0 u 1 u 2 u n, where you start with u 0 and then that is u is u 0 and v is u n in between you got u 1 u 2 u n minus 1. How do you get one from the other? From u i minus 1 you get u i. If u i minus 1 is of this form, it is of the form u i minus 1 prime x i u i minus 1 double dash and the next sentential form will be of the form u i minus 1 dash y i u i minus 1 double dash. That is, you have this sentential form and from this you apply the rule x i goes to y I, if it is the i’ th rule in the sequence. So, from x i you have obtained y I, this is the rule applied and from u i minus 1 you get u i.

Now, you have n rules in a say matrix and then by applying one rule you get from u naught to u 1, then applying the second rule you get to u 2, then applying the third rule and so on, you have to obtain the rules in the sequence. And finally, you have to end up with the terminal string. So, the language clearly any direct derivation corresponds to n step derivation in the; If you look at them as single rules, but you know, that the rules have to be applied in a particular sequence. That is the rules in m r used in sequence to reach v from u. Now, if you denote this as double arrow that is from u which is a sentential form, you get a sentential form b by applying a matrix and the matrix has n rules x 1 goes to y 1, x 2 goes to y 2 and so on.

So, by applying one by one in n steps, you go from u to v and that is denoted by double arrow. Double arrow star is the reflexive transitive closure of double arrow, that is you can apply the matrices in some order, some anyway you want. But when you apply a one matrix, the rules in the matrix have to be applied in that particular sequence. The language generated will consist of strings of the form w, where w is a terminal string. And starting from the start symbol, you are able to derive this terminal string by the application of the matrices.

(Refer Slide Time: 10:32)

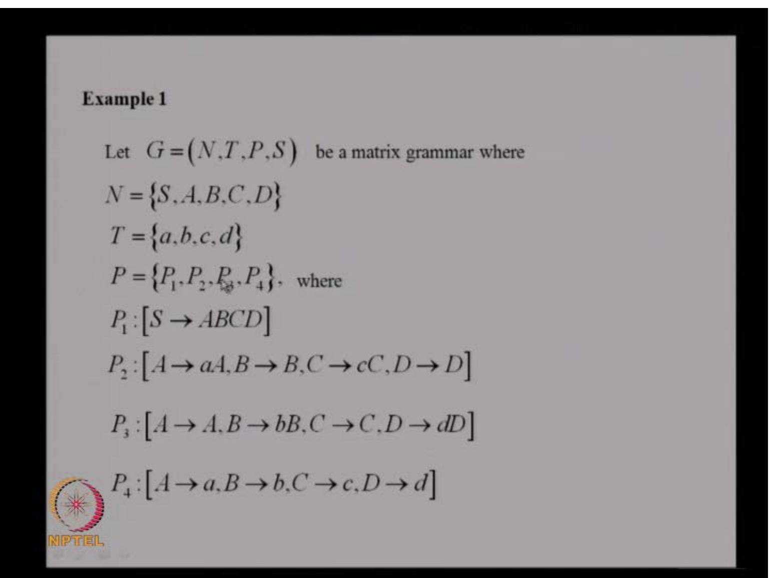


Let us consider a very a simple example, I will use this I have a matrix which has got this rule, one matrix which has got a single rule and then I have rules of the form A goes to a A, B goes to b B, C goes to c C. Then, I have the rule of the form A goes to a, B goes to b, C goes to c, this is one matrix. The grammar has non terminals S A B C and the terminals are small a small b small c and these are the matrices. Now, starting from this, if we apply the first matrix, I can only apply the first one, I get this sentential form. And I can apply this matrix or this matrix, suppose I apply this matrix for A I have to apply small a, for B I have to small b, for C I have to apply small c so, I will get a b c. Now, instead for A if I start applying this matrix, I have to apply the rule a A. See, I have to if I apply this rule, I have to apply this rule, I have to apply this rule.

So, for B I apply b B, for C I apply c C then, I can again use this matrix, where I can get a a A b b B c c C. Then, if I apply this matrix I will get a a a b b b c c c, it is not difficult to see that if I apply this matrix one a one b one c is generated, if I apply this matrix which is the terminating matrix then also one a one b and one c will be generated. It is easy to see that the language generated is a power n b power n c power n, n greater than or equal to 1.We know that, this is the context sensitive language and it is not a context free language. Plus we know that the rules are all context free, note that the rules in this in matrix are context free, in fact they are regular. Whereas this rule is context free, but anyway if look into the rules they are all context free.

But, by putting some restriction on the manner of applying the rules, we are able to get a context sensitive language. So, the power is really increased, so by putting some restrictions on the manner of applying the rules, we are able to get a higher a language belonging to a higher class.

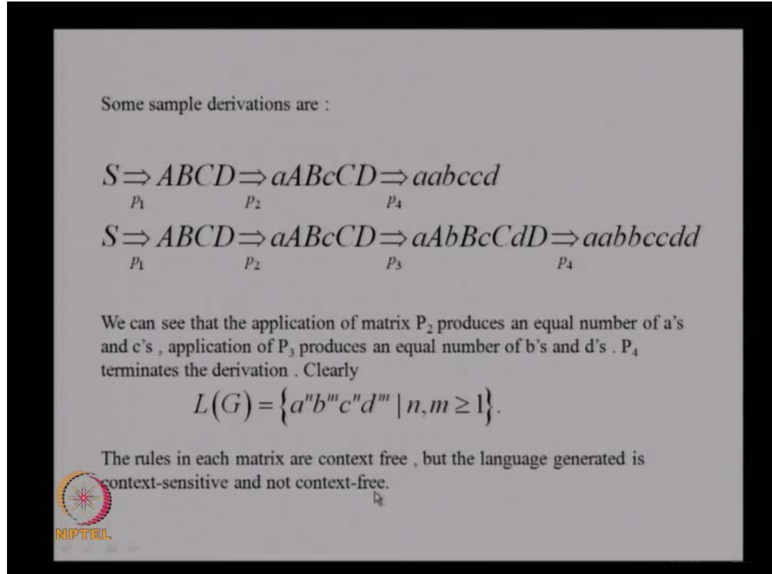
(Refer Slide Time: 13:56)



Let us see one more example, now, let us see this, the grammar has non terminals S A B C D, terminals are a b c d. There are four matrices, first one is like this, second one if you apply this, one A will be generated and one C will be generated where as nothing will happen to B and D. If you apply this matrix one B will be generated one D will be

generated, but A and C remain as they are, this is the terminating matrix where you have one a one b one c one d generated.

(Refer Slide Time: 14:49)



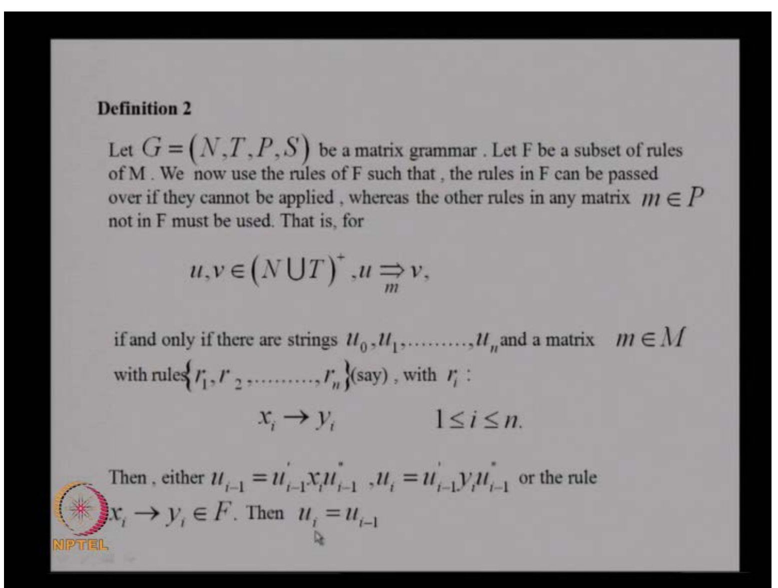
What is the language generated here, you can see that S goes to A B C D and by applying the matrix p 2 one a is generated one c is generated. Then by applying another matrix the last matrix A B C D are converted to a b c d. So, you find that the number of a's and number of c's is equal, they are equal. Similarly, the number of b's and the number of d's will be equal, another derivation is starting from S you apply the first matrix to get A B C D. Then by applying matrix to one a and one c are generated and applying matrix three one b and one d are generated. Then, you can generate one a one b one c one d using p 4, which is the terminating derivation.

You find that two a's, two b’s, two c's and two d's are generated, but in general you can see that whenever you apply P 2 one a and one c will be generated. So equal number of a's and c's will be generated. But when you apply P 3 one b and one c will be generated one d will be generated. So, equal number of b's and d's will be generated, but terminating matrix is this, that is at least one a one b one c one d will be generated. So, you can easily see that by applying the matrices in different orders, you can generate equal number of a's and c's any number you want greater than or equal to 1. Similarly, by

applying the rule P 3 any number of times, you can generate b power m, d power m equal number of b's and d's and finally, you have to apply the terminating rules so, at least one a one b one c one d will there.

So, the language generated will be of this form, it will consist of strings of the form a power n b power m c power n d power m, where the number of a's and c's are equal and number of b's and d's are equal. This we know is a context sensitive language, but note that in the matrices these are all regular rules of course, we are having unit productions also, this is a context free rule, type 2 rule. So, all rules are context free, but we are getting a language which is context sensitive.

(Refer Slide Time: 17:57)



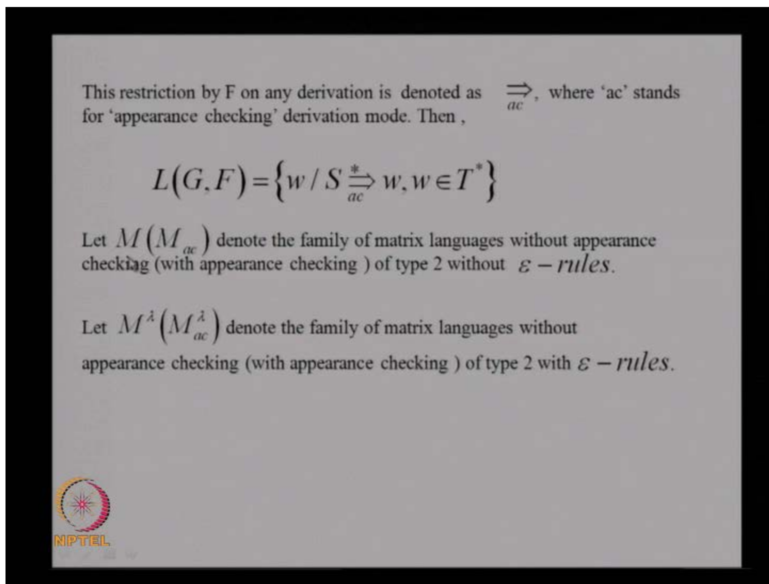
Now, let us see one another slightly different aspect of it. That is, let us see what we mean by an appearance checking? Generally you have the grammar with matrix like this. Now, some of the P is the set of rules and we have a subset, a subset of the rules, we denote as F, let F be a subset of the rules of M, M is the matrices and P consists of all rules in the matrices we can label them also and F is a subset of that. Now, the rules in F can be passed over if they are not applicable or if they cannot be applied. That is, you reach a stage where you have to apply a rule, but you are not able to apply that rule. Then, you have to just check whether it belongs to F, if it belongs to F we leave it go to

the next rule.

But rules not in F they have to be applied, other rules in the matrix which are not in F must be used. That is for u and v belonging to N union star, you say that u derives v by the application of matrix, if you have rules r 1 r 2 r n in m, the rules are in this sequence. And you have sentential forms u 0 u 1 to u n from u 0 by applying r 1 you should get u 1 by applying r 2 you should get u 3 and so on, so finally, v is u n. Now, if you have this appearance checking, this is called appearance checking mode, what happens is the earlier? What we had is from u i minus 1 which is denoted like this and from u i which is denoted like this. You apply the rule x i goes to y i and get that is from u i minus 1, you get u i by the application of the rule x i goes to y i replacing this sub string x i by the sub string y i.

Now, in the appearance checking mode, what happens is, u i is the same as u i minus 1. You find that u i minus 1 is you are getting this sentential form in the middle and at that stage, you have to apply the rule r i, r i is x i goes to y I, but x i is not a sub word of u i minus 1. Suppose x i is not a sub word of u i minus 1, you are not able to apply the rule. If you are not able to apply the rule, then check whether such a belongs to F. If such a rule belongs to f, then you can use that rule in the appearance checking mode. That is you just skip that rule and u i becomes u i minus 1, is the same as u i minus 1, even though you are suppose to apply the rule x i goes to y i at that stage. Because x i is not a sub string of u i minus 1, you cannot apply that then, you have to just check whether such a rule is in f and keep u i as u i minus 1, you skip that rule and proceed with the derivation.

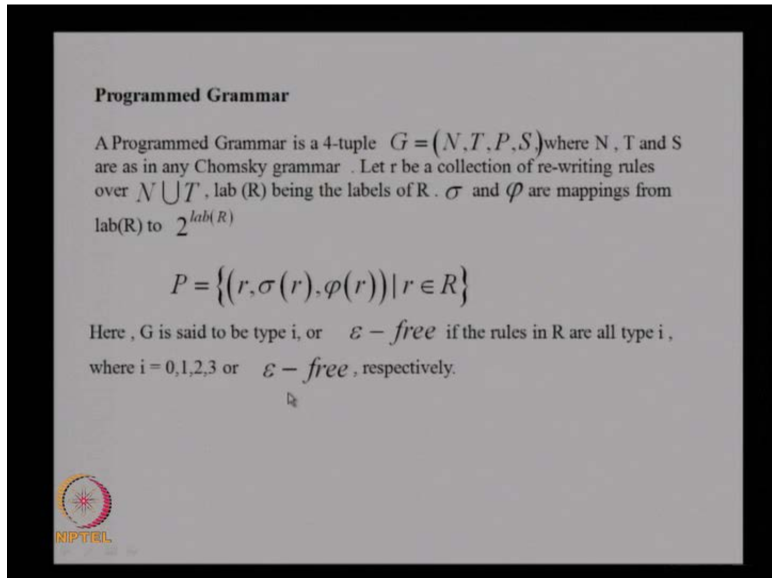
(Refer Slide Time: 21:45)



The restriction by F on the derivation denoted by this rule that is double arrow with ac, ac means appearance checking. The language generated with this mode is denoted as L G F that is along with the four components of G, you also specify F which is a subset of P that is the set of rules which can be used in the appearance checking mode. And S derives w of course, the double arrow star is the reflexive transitive closure. In the appearance checking mode you are using and the string generated should belong to the terminal set it is a string of terminals.

So, if we use type 2 grammars, we may include the epsilon rule or we may exclude the epsilon rule. If we exclude the lambda productions, the set the class of language generated it is denoted by M. And if we use type 2 grammars without epsilon rules in the appearance checking mode the class generated is denoted by M ac. If we use type 2 grammars including the lambda rule, but not appearance checking the class is denoted by M lambda. If we use lambda productions in type 2 grammars and also we use in the appearance checking mode, then it is denoted as M lambda ac. Actually, you; this is the smaller class and you find that this has got only semi linear languages, that the languages whose spheric mappings are only semi linear.

(Refer Slide Time: 23:51)

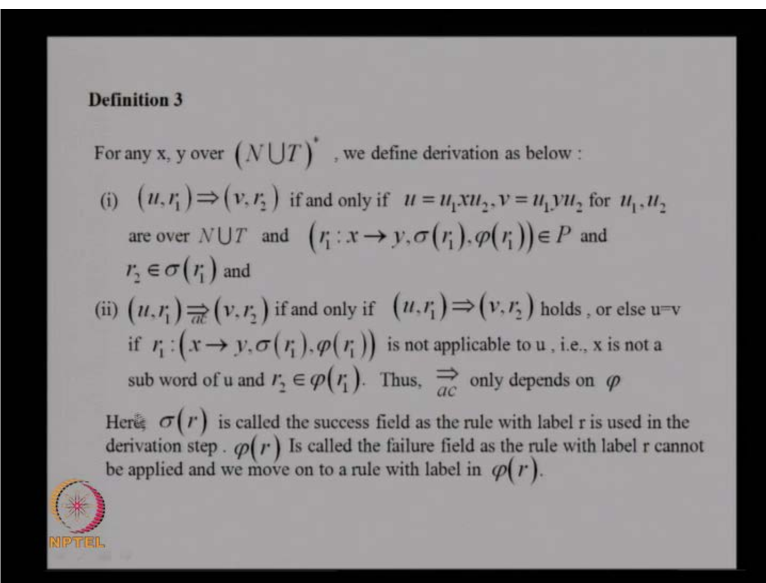


Let us go to the next definition, what is meant by a programmed grammar, this is another type of a definition. The programmed grammar consists of non terminals, it has got terminals, the set of productions, and the productions are labeled, you call the rules as r 1 r 2 r 3 etcetera and S is the start symbol and label R is the less set of labels. You have two functions sigma and phi, they are mappings from label of R 2 power set of label of R, instead of too much. We looking into that, let us take an example and see, but before that the you must remember that rules are of this form that is for each rule r is a rule, it has got label r.

And, then it will be a rule in the usual sense u goes to v, as along with that two subsets of the whole set of rules will be associated sigma r and phi r, sigma is called the success field and phi is called the failure field. (No audio from 25:11 to 25:20) The rules are of this form, you have a rule which is of the form u goes to v, then along with that you have two components, this is the subset of rules, this is also a subset of rules, this is called the success field and this is called the failure rule, failure set. Now, at a particular step if rule r is applicable you apply the rule, the next rule should be applied from this set success field. Now, if you try to use it in the appearance checking mode at a particular step or may not be applicable, if r is not applicable then the next rule should be applied from the failure field.

The rules can be epsilon free, you can include they can be type 0 1 2 3, but of interest or type 2 rules including epsilon rules or excluding epsilon rules. You find that, when you use these grammars with type 2 rules and epsilon rules including the epsilon rules in the appearance checking mode, you get the power of a type 0 grammar or that of turning machines.

(Refer Slide Time: 26:51)

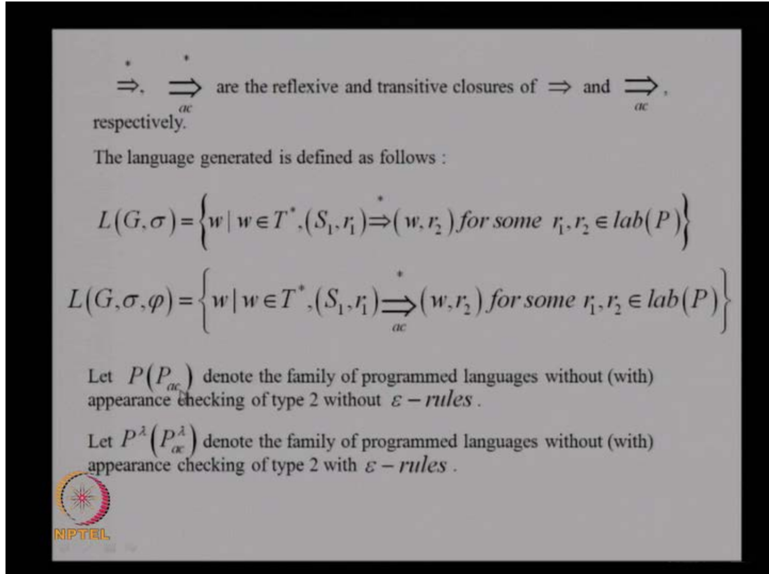


Now, formally defining the derivations, let x and y be two strings. Now, from u you derive a string v, if you are able to apply a r 1, what is the rule r 1? r 1 is the rule x goes to y. So, you are having a sentential form u 1 x u 2 and you apply the rule x goes to y and, you get the sentential form u 1 y u 2, that is you are applying the rule r, r 1. You have been successful in applying r 1 so, the next rule should be from this field, success field that is next rule is r 2. So, r 2 should belong to this set it can be any one of them, if this has the finite rule, set of rules, r 2 can be any one of them, but if you are applying r 1 to u and are successful then, the next rule should be applied from the success field.

In the appearance checking mode, you are trying to apply x goes to y for u and x is not a sub word so, you are not able to apply the rule and so v also remains the same as u. In that case, the next rule applied r 2 should be taken from the failure field so, r 2 belongs to phi of r 1, this is called appearance checking. Appearance checking only depends on the

failure field, if you do not have failure field or if you do not have any element in the failure field, you are applying in the sense, where you do not use appearance checking.

(Refer Slide Time: 29:12)



So, the language if you do not use the appearance checking is denoted as L G sigma. Where starting from the start symbol you are able to get a terminal string by the application of the rules and the application of the rules, we have explained earlier. If you use it in the appearance checking, then you also use the failure field. Then starting from the start symbol you get the string w, which is a terminal string by the application of the rules. And we have seen how the rules have to be applied, but here we are using appearance checking also, that is the failure field is also used.

Now, you can use type 2 grammars, the rules can be of type 2, it can include the epsilon rules or it need not include the epsilon rules. If it includes the lambda rules, if we do not use appearance checking you get the class P lambda. If lambda rules are included and you use with appearance checking the class is called a P lambda ac, this becomes equal to the class of type zero languages. If you use the programmed grammars with type 2 rules, no lambda rules, then you denote it as P. If you use type 2 rules, but no lambda rules, but you use them in the appearance checking mode, it is denoted as P ac.

(Refer Slide Time: 30:49)

Example 3

Let $G = (N, T, P, S)$ be a programmed grammar with

$N = \{S, A, B, C, D\}$

$T = \{a, b, c, d\}$

P:

	r	$\sigma(r)$	$\varphi(r)$
1.	$S \longrightarrow ABCD$	2,3,6	ϕ
2.	$A \longrightarrow aA$	4	ϕ
3.	$B \longrightarrow bB$	5	ϕ
	$C \longrightarrow cC$	2,3,6	ϕ

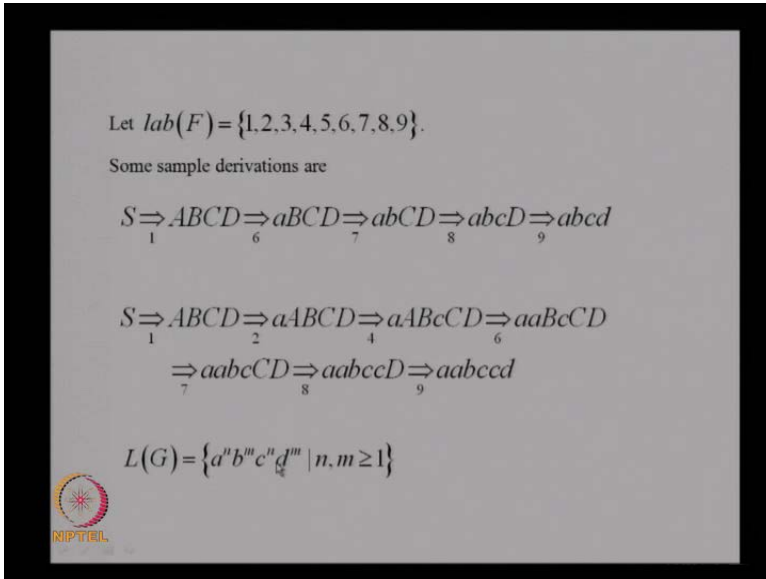
Take as an example this, you have the following rules, the non terminals are A B C D, terminals are small a b c d and these are the rules, rule number one is this. And when it is successful, you apply the next set from 2 3 or 6, there is no failure field it is not used in the appearance checking mode. The second rule is A goes to A and if you are successful, you must apply 4. The third rule is this and if you are successful, you must apply the next rule as 5.

(Refer Slide Time: 31:30)

	r	$\sigma(r)$	$\varphi(r)$
5.	D ⟶ dD	2,3,6	ϕ
6.	A ⟶ a	7	ϕ
7.	B ⟶ b	8	ϕ
8.	C ⟶ c	9	ϕ
9.	D ⟶ d	ϕ	ϕ

Then, if you are successful in applying 5, you can go to 2 3 or 6 then 6 7 8 9 are like this. If you are able to apply 6, you must use next 7 then if you are successful you must use 8 then you must use 9 then 9 terminates the derivation.

(Refer Slide Time: 31:52)

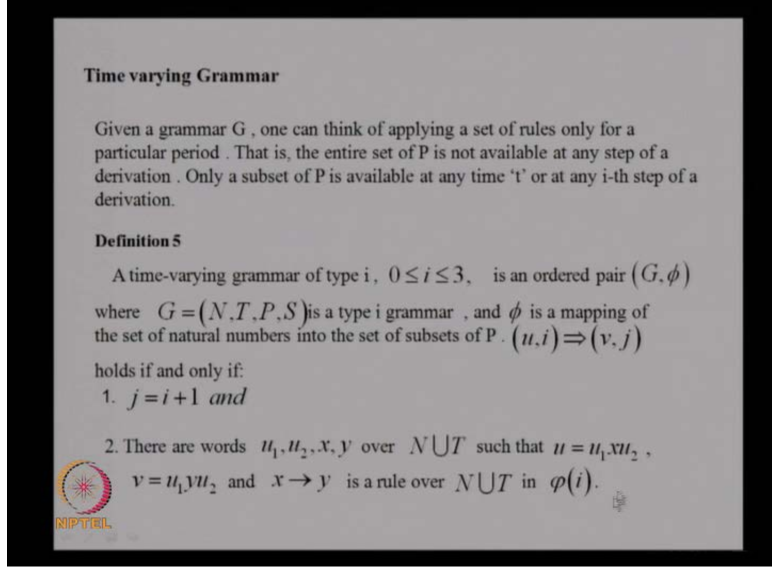


So, let us see how we generate this, it is the same languages before. You see that starting

from S you apply rule 1 and you are successful in applying the rule. So, the next rule should be applied from the set 2 3 or 6, you can apply 2 or 3 or 6. Now, you apply 6 then you get a, A is replaced by a 6 is this rule. Now, you see that you are successful in applying this so the next rule should be from 7 then the next rule should be 8 and the next rule should be 9. So, applying in that sequence you get the string a b c d. Now, the other way around after applying one you can also apply 2, so when you apply 2, you get this. Then you see that when you apply 2, you must apply a 4 next after 4 again you have a choice of using 2 3 or 6.

So, after applying 2, you apply 4 this makes sure that you are generating equal number of a c's, a's and c's, one a is generated means you must generate one c also. Then again using 6 7 8 and 9 in succession, you will get a power n b; you will get a a B c C D. In general whenever you apply 2 it should be followed by 4, whenever you apply 3 it should be followed by 5, that make sure that equal number of a's and c's are generated, also equal number of b's and d's are generated, because you have to apply 6 7 8 9 in that order, you find that one a one b one c one d will be generated. So, the language generated is a power n b power m c power n d power m and this is a context sensitive language. But we have used only context free rules, note that all these rules are regular in fact and this is the only rule which is context free, but anyway all rules are context free.

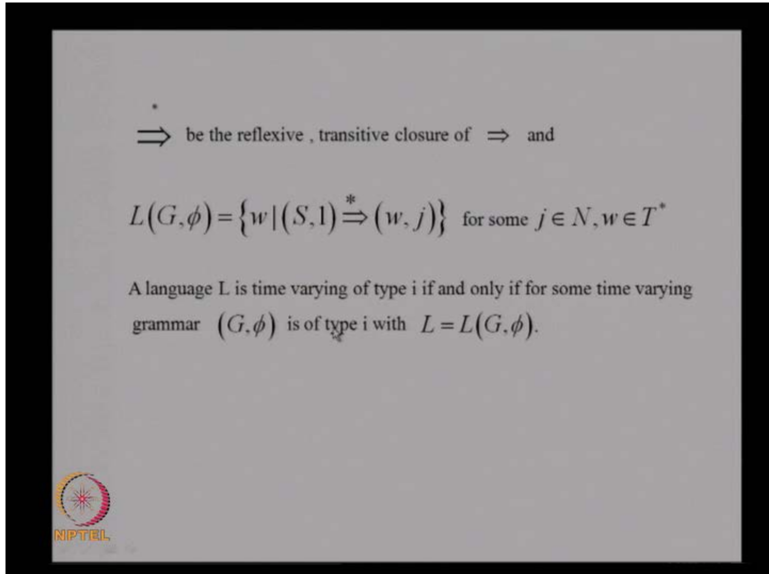
(Refer Slide Time: 34:12)



So, next let us see, what is the time varying grammar? In the time varying grammar is you put another type of restriction on the manner of applying the rules. That is we find that at the instance of time i , we have to use only a subset of rule probably at odd instances. You can use only a subset and even instances, you can use only a subset something like that you can have. If you have put that restriction, what sort of language will be generated. So, again we have the four components, the set of rules is P and the derivation is defined like this, from u i you go to v j then j will be i plus 1 and you are applying a rule x goes to y, which can be use at the i th instance of time.

There is a function phi which maps the rules, it is a subset of rules that is phi i denotes a subset and at the i th instance, you only use rules from that subset. So, you are having a sentential form u, which is u 1 x u 2 and this subset of rules can be used at that i th instance. You can use any rule from this set, suppose this rule x goes to y belongs to that subset, then you replace x by y. And, in the next instance you have to use rule from the next set that is phi i plus 1.So, j is the next instance and it is i plus 1 and the next instance you have to use rules from phi i plus 1, that is the without appearance checking mode.

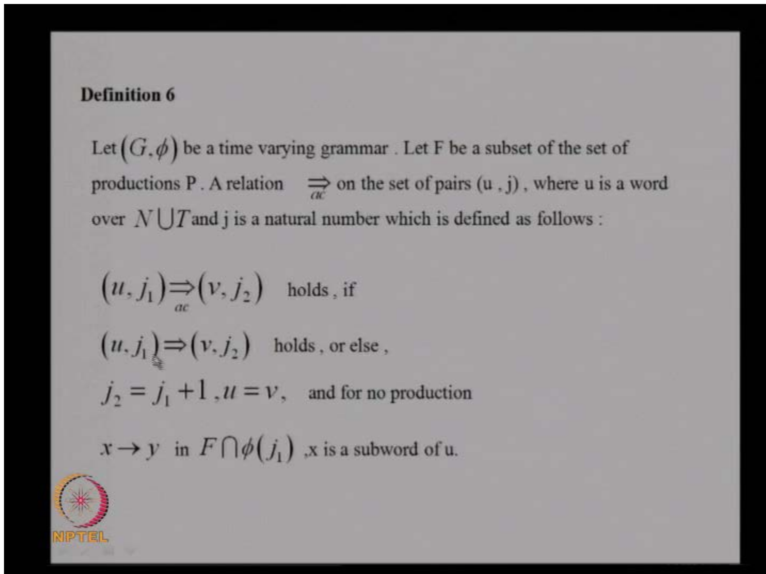
(Refer Slide Time: 36:10)



So, the language generated is starting from S the first instance, you use rule from phi one and so on, until you get a terminal string. So, each time one will go to two and two will

go to three and so on, the second component will denote the step number. So finally, you must end up with the terminal string then this is denoted like this. The rules can be type 1 type 2 type 3 type 4 etcetera, I mean there is no type 4 type 0 type 1 type 2 type 3.

(Refer Slide Time: 36:53)



Now, in the appearance checking mode, how do you define this? You denote subset, if you want to apply a rule in the appearance checking mode then that rule should be present in F, you can skip it. If it is not applicable, you can skip that that is what? It is meant by appearance checking. So, from u j 1 you go to v j 2 if it holds in the ordinary sense or from u j 1 you go to v j 2, but j 2 is j 1 plus 1 that is it is incremented without applying any rule, u goes to v and none of the rules in the set phi j 1 is applicable at that time. So, when you have a sentential form u at the j 1 th instance of time, you are suppose to apply rule from this set phi j 1.

There will be a few rules in them and none of the rules is applicable say and then all the rules are also in F that is the set of rules in the appearance checking mode. Then you will just keep u as it is and j 1 is incremented b 1, this is using in the appearance checking mode.

(Refer Slide Time: 38:22)

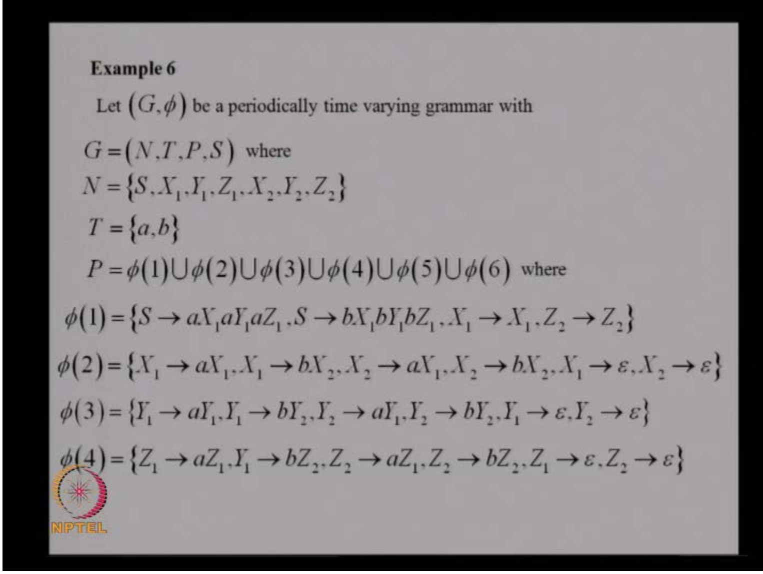
$\overset{*}{\underset{ac}{\Rightarrow}}$ is the reflexive , transitive closure of $\underset{ac}{\Rightarrow}$. Then , the language generated by (G,ϕ) with appearance checking for productions in F is defined as :

$$L_{ac}(G,\phi,F)=\left\{w \mid w\in T^{*} \mid (S,1)\overset{*}{\underset{ac}{\Rightarrow}}(w,j)\ for\ some\ j\right\}$$

The family of languages of this form without appearance checking when the rules are context free (context-free and $\varepsilon-free$) and ϕ is a periodic function are denoted as τ^{λ} and τ respectively. With appearance checking, they are denoted as τ_{ac}^{λ} and τ_{ac} , respectively.

The language obtained in the appearance checking mode is denoted by L G phi of course, this is the reflexive transitive closure of this. And, the language generated is denoted as starting from s and the step one at the j th step you get a terminal string w and you use the rules in the appearance checking mode.

(Refer Slide Time: 38:55)



Take this example, so what you get is you have the; it is a periodically time varying grammar that is step 1. You can use this step 2 step 3 step 4 then step 5 and step 6, it is a period 6 step 7 again you have to use from this set, step 8 you have to use rules from this set. So, the; it is a periodically time varying grammar, the rules are like this.

(Refer Slide Time: 39:25)

$$\phi(5) = \{X_2 \rightarrow X_2, Y_1 \rightarrow Y_1\}$$
$$\phi(6) = \{Y_2 \rightarrow Y_2, Z_1 \rightarrow Z_1\}$$

Some sample derivations are a

$$(S,1) \Rightarrow (aX_1aY_1aZ_1,2) \Rightarrow (aaY_1aZ_1,3) \Rightarrow (aaaZ_1,4) \Rightarrow (aaa,5)$$
$$(S,1) \Rightarrow (bX_1bY_1bZ_1,2) \Rightarrow (baX_1bY_1bZ_1,3) \Rightarrow (baX_1baY_2bZ_1,4)$$
$$\Rightarrow (baX_1baY_1baZ_1,5) \Rightarrow (baX_1baY_1baZ_1,6)$$
$$\Rightarrow (baX_1baY_1baZ_1,7) \Rightarrow (baX_1baY_1baZ_1,8)$$
$$\Rightarrow (babaY_1baZ_1,9) \Rightarrow (bababaZ_1,10)$$
$$\Rightarrow (bababa,11)$$

$$L(G,\phi) = \{www \mid w \in \{a,b\}^+\}$$

Let us see one derivation; first instance you are applying from the first set s is replaced by a X 1 a Y 1 a Z 1, note that this is incremented by one. Then if you use X 1 goes to epsilon, the derivation is getting terminated second step, third step you use Y 1 goes to epsilon, then the fourth step you use Z 1 goes to epsilon. The language generated will be of the form w w w, strings of the form w w w three copies, where w can be any string of a's and b's. Let us again (( )) first rule you are applying here, another derivation is like this, first step you are applying this, second step you are applying this, third step you are applying this, 4th step you are applying this, then again 5th step 6th step 7th step, you use X 1 goes to X 1, Y 1 goes to Y 1, Z 1 goes to Z 1 and so on.

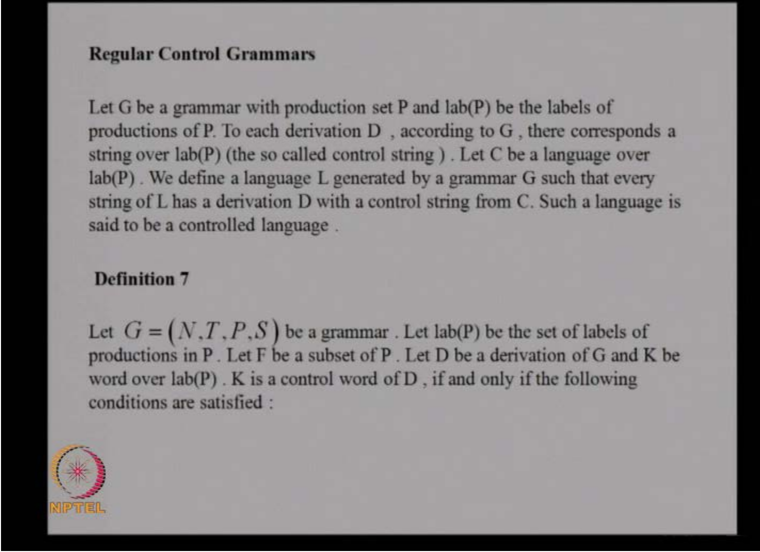
See 5th step you use Y 1 goes to Y 1, 6th step you use Z 1 goes to Z 1, then 7th step X 1 goes to X 1 and you have the same thing, then the 8 step again terminate, use rules a goes to a and so on. So finally, you will end up with this derivation in the 11th step.

(Refer Slide Time: 41:24)

$$\phi(3) = \{B \to B_1, B \to bB_2, B \to \varepsilon\}$$
$$\phi(4) = \{C \to cC_1, C \to C_2, C \to \varepsilon\}$$
$$\phi(5) = \{D \to D_1, D \to dD_2, D \to \varepsilon\}$$
$$\phi(6) = \{A_1 \to A, B_2 \to B\}$$
$$\phi(7) = \{B_1 \to B, C_2 \to C\}$$
$$\phi(8) = \{C_1 \to C, D_2 \to D\}$$
$$L(G,\phi) = \{a^n b^m c^n d^m \mid n,m \geq 1\}.$$

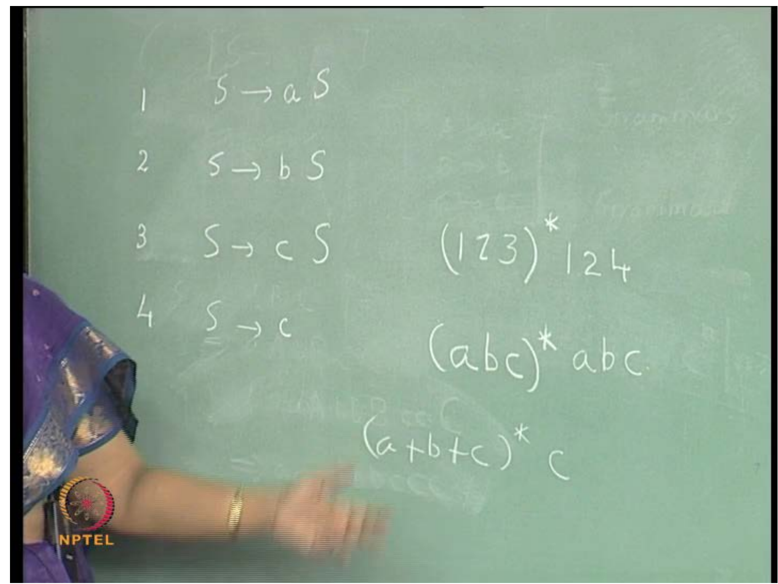
Similarly, you can find that, we can also generate a power n b power m c power n d power m, where you have equal number of a's and c's and equal number of b's and d's. Note that these are all context sensitive languages, but the rules we have used are all only context free, in fact only one rule will be context free rest of them are all type three rules.

(Refer Slide Time: 41:50)



The next we looked into that is regular control, what do you mean by regular control, it is very simple idea, let me use the board now.

(Refer Slide Time: 42:17)



(No audio from 42:05 to 42:15) Suppose I have a rule.

(No audio from 42:20 to 42:40)

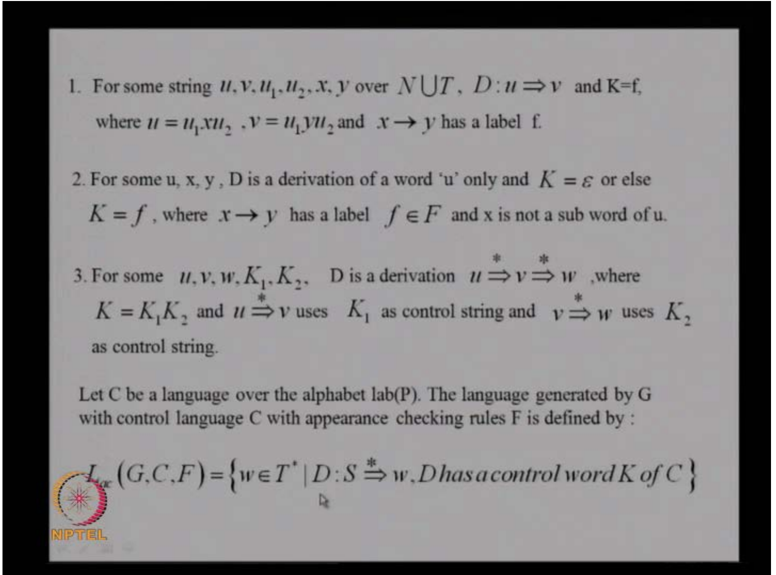
I have 4 rules like this. Now, you will see that the language generated, we can; is any string of a's and b's, but it will end with the c. So, the language generated, you can use them in any order you want and then finally, you have to terminate the derivation with this rule, so the string will end up with the c. But if I put the restriction, that the rules have to be applied in this order, that is 1 2 3, in this order only it has to be applied several times. Then I will generate a, I will generate b, I will generate c, a b c I will generate again and again, then finally, I have to use rules 1 2 and 4, I will end up with a b c. If I put the restriction that is the rules have to be applied in this order, then the language generated is a b c star a b c, where as if I do not put, it will be a plus b plus c star c, the language generated will be this.

So, when I put some control and if this control is of the form, it is a regular set what sort

of a language will be generated, this is what we want to explore here in the regular control. Let us see the formal definition, the formal definition will be like this, let G be a grammar with production set P and you consider the productions with labels, label P is the labels of the productions. To each derivation D according to D there corresponds a string over lab (P), label of P it is called the control string. Let C be a control language over label of P, in a sense it can be anything, it can be context sensitive, it can be context free, but we would like to consider regular control. Every string in L has a derivation D with a control string from C then such a language is said to be a controlled language.

Let G be a grammar, then the set of labels is denoted by label P. Label F is subset again this is the appearance checking mode, D is the derivation and K is string over lab P, K is a control word if the following conditions are satisfied.

(Refer Slide Time: 45:48)

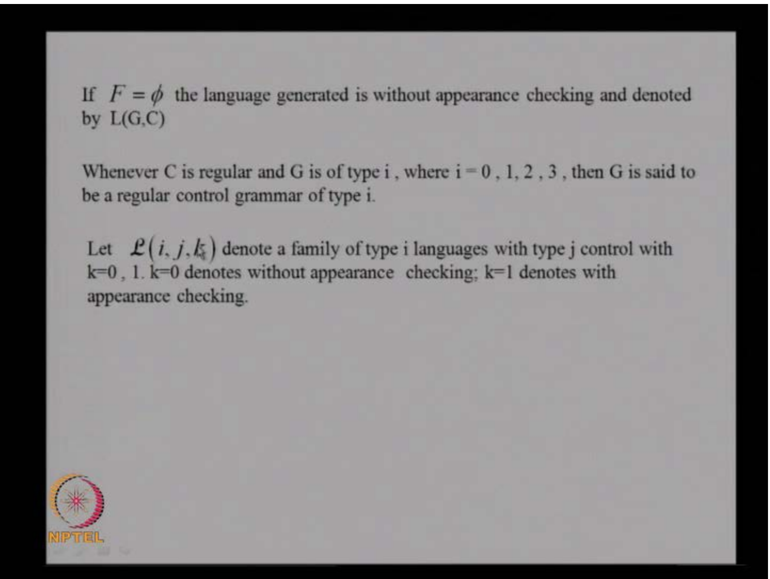


That is from u to you go to v, if u is of the form like this, v is of the form and x goes to y has a label f, that single label. Another thing is, you derive a word u, if k is just f or epsilon and k is; see, suppose I have a sentential form u and I want to apply the rule x goes to y with control K, if K is epsilon means I cannot apply any rules so, v will be the same as u. But if x is not a sub word of u and K is f, control word is f then I have to check whether it belongs to the subset F and if it belongs to that subset F, then I can use

it in the appearance checking mode and v will be the same as u.

Now, from u you get v using the control word K 1, from v you get w using the control word K 2 then using the control word K 1 followed by K 2, that is K is equal to K 1 K 2. That is first you apply control K 1 then control K 2, from u you will get v then from v you will get w. So, from u you will get w, the language generated, it is denoted like this grammar control set and, if you are using in the appearance checking mode, the subset of the set of rules. So, from S you derive w using the control word K which is a string of C and you have to end up with a terminal string.

(Refer Slide Time: 47:52)



Let us consider an example, if you do not have F, F is empty we are using the thing with without appearance checking. Again the rules can be 0 1 or 2 type 0 type 1 type 2 or type 3, we denote the family by alpha i j k, where i denotes the type of rule, j denotes the type of control you have, and k is 0 means without appearance checking, k is 1 means with appearance checking.

(Refer Slide Time: 48:27)

Example 8

Let $G=(N,T,P,S)$ be a regular control grammar where

$N=\{A,B,C,D,S\}$

$T=\{a,b,c,d\}$

$P:$

1. $S \rightarrow ABC$
2. $A \rightarrow aA$
3. $B \rightarrow bB$
4. $C \rightarrow cC$
5. $D \rightarrow dD$

Let us consider this; you are having a grammar like this. The non terminals are A B C D and S, terminals are a b c d these are the production rules rule number 1 is S goes to A B C, this is context free rest of them you see, they are all regular rules 2 is this 3 is this 4 5 6 7 8 9.

(Refer Slide Time: 48:53)

6. $A \rightarrow a$
7. $B \rightarrow b$
8. $C \rightarrow c$
9. $D \rightarrow d$

Then, $lab(P)=\{1,2,3,4,5,6,7,8,9\}$

Let, $K=1(24)^*(35)^*6789$. Clearly, K is regular. Then

$L(G,K)=\{a^n b^m c^n d^m \mid n,m \geq 1\}$

Some sample derivations are :

for $u=124356789 \in K$,

The labels are denoted as 1 2 3 4 5 6 7 8 9 and the control is 1, 2 4 star 3 5 star 6 7 8 9, note that this is a regular language, this is a regular set. Now, if you use a string from this and control the derivation 2 4 star would mean that whenever you apply rule 2, it should be followed by rule 4. What is rule 2? You are generating one A and rule 4 is you are generating one C. Similarly, you can use 3 5 together.

(Refer Slide Time: 49:39)

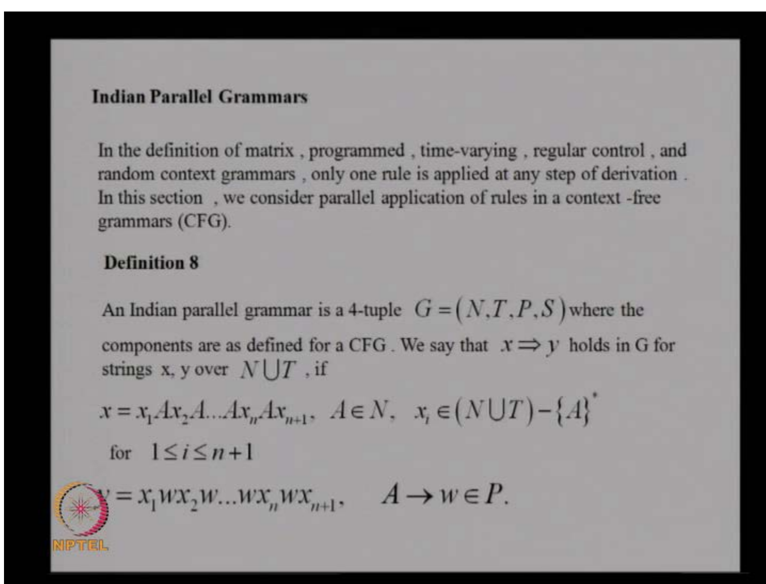
$$S \underset{1}{\Rightarrow} ABCD \underset{2}{\Rightarrow} aABCD \underset{4}{\Rightarrow} aABcCD \underset{3}{\Rightarrow} aAbBcCD$$
$$\underset{5}{\Rightarrow} aAbBcCdD \underset{6}{\Rightarrow} aabBcCdD \underset{7}{\Rightarrow} aabbcCdD$$
$$\underset{8}{\Rightarrow} aabbccdD \underset{9}{\Rightarrow} aabbccdd$$

If $u = 124246789 \in K$

$$S \underset{1}{\Rightarrow} ABCD \underset{2}{\Rightarrow} aABCD \underset{4}{\Rightarrow} aABcCD \underset{2}{\Rightarrow} aaABcCD$$
$$\underset{4}{\Rightarrow} aaABccCD \underset{6}{\Rightarrow} aaaBccCD \underset{7}{\Rightarrow} aaabccCD$$
$$\underset{8}{\Rightarrow} aaabcccD \underset{9}{\Rightarrow} aaabcccd$$

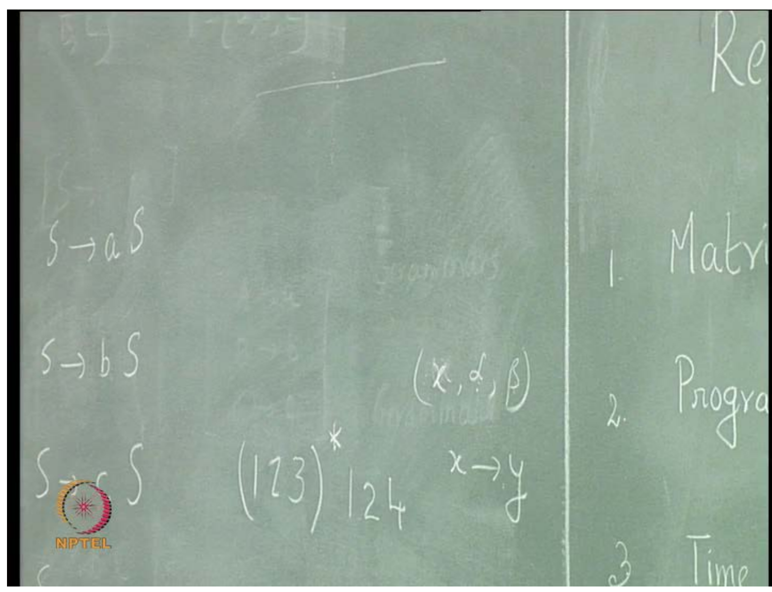
So, whenever you apply 3, you should also apply rule 5 that is you have a derivation like this, 1 then apply 2 4 3 5 then you end up with 6 7 8 9, you get this string. So, the control word is 1 2 4; 1 2 4 3 5 6 7 8 9 if you use 1 2 4 2 4 6 7 8 9 so 1 then 2 then 4 then again 2 and 4 then 6 7 8 9, you get 3 a's, 3 c's, but one b and one d, here you get two a's two b's two c's and d. Anyway, you find that the number of a's will be equal to the number of c's and the number of b's will be equal to the number of d's. So, the language generated will be of the form a power n b power m c power n d power m, this is the context sensitive language or the type one language. And you can see that the rules are all context free only, but then in fact only one rule is context free, rest of them are type three, where the language generated becomes a context sensitive language.

(Refer Slide Time: 50:56)



So, similarly you can also define what is known as a random context grammar that is in a sentential form, it denotes some symbols as permitting context and some symbols as forbidding context.

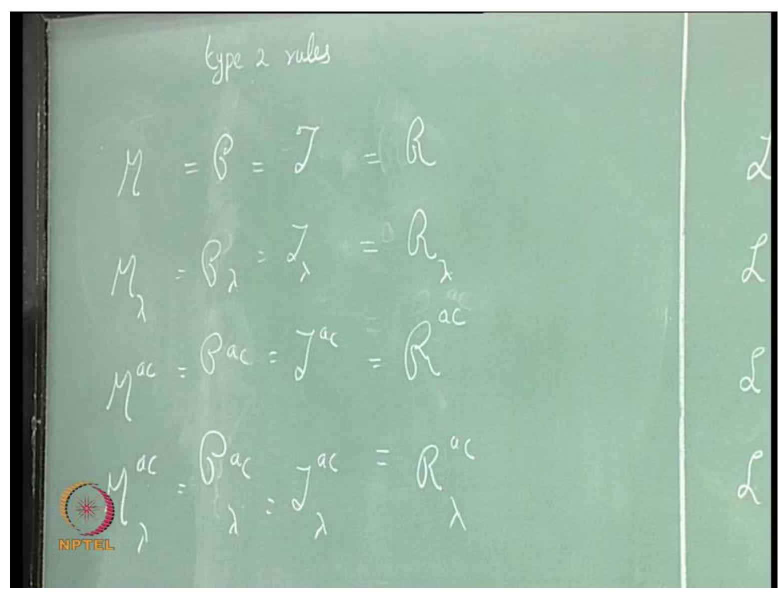
(Refer Slide Time: 51:13)



So, when you have a sentential form like this, you can apply the next rule, the rules will

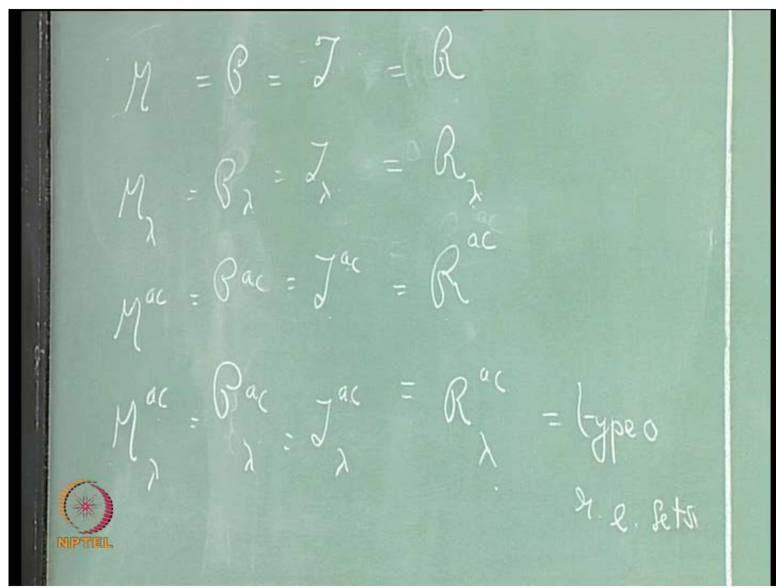
have some sets alpha beta, this is the symbol some left hand side. Now, if you want to apply this rule, that is the rule is of the form x goes to y. Now, we want to apply this rule, all symbols in alpha must be present and none of the symbols in beta must be present in the sentential form. Then only you can apply x y such is called a permitting context and forbidding context and the grammar is called a random context grammar. We will not go in to that in detail.

(Refer Slide Time: 51:58)



But before that we have seen matrix grammars with epsilon production type 2. Type 2 rules only, type 2 rules with epsilon productions, without epsilon productions, with appearance checking, without appearance checking. Similarly, programmed grammars type 2 rules only with lambda productions, with appearance checking or without appearance checking. And periodically time varying grammars with type 2 rules, with lambda rules, without lambda rules, with appearance checking, without appearance checking similarly, the families with regular control.

(Refer Slide Time: 52:44)



That is type 2 rules without lambda rules or with lambda rules, control is regular. So, the control language is type 3 and you can have appearance checking or you need not have appearance, 0 denotes no appearance checking, 1 denotes appearance checking. The families of languages generated are denoted by R, R lambda R a c, R lambda a c that is R denotes; we are using type two rules only, it is without appearance checking, without lambda rules. Here lambda rules are used, but no appearance checking, here lambda rules are not allowed, but we are having appearance checking, here lambda rules are allowed, and we are using appearance checking.

And in that case, we find these results; we are not going to prove these results, but these rules results hold that is these families are equal. These four families are equal, again these four families are equal, and these four families are equal and they are equivalent to type 0 languages or recursively enumerable sets accepted by Turing machine. This is the highest class in the Chomsky and hierarchy.

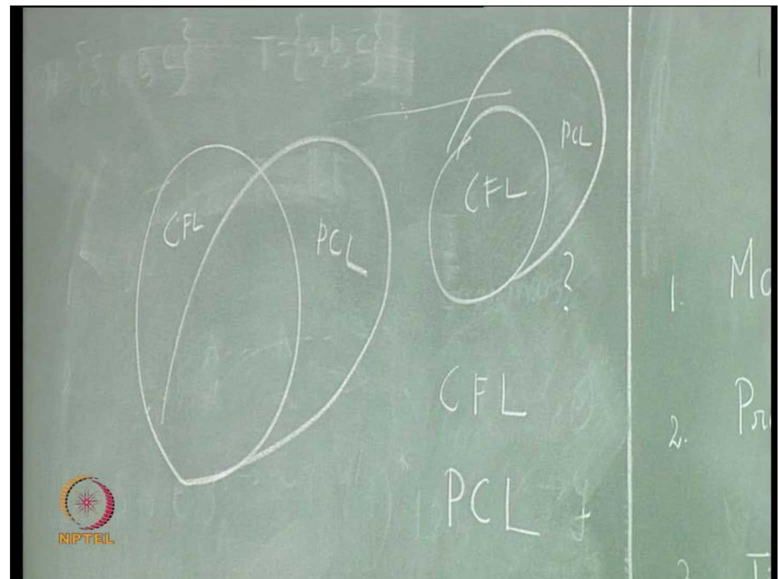
(Refer Slide Time: 54:36)

So, this is a brief introduction to regulator rewriting, how the regulator rewriting can increase the power of a grammar, we have I mentioned one more thing which is known as Indian parallel grammar. Here again, (No audio from 54:38 to 54:44) what is it? Suppose I have a sentential form in which case three A's appear, say and I have two rules for A, A goes to alpha, A goes to beta of course, I consider only context free rules here. Now, if I start applying the rule A goes to alpha here, I have to replace A by alpha I simultaneously do that for all the A's in the sentential form. That is I do not do it in sequential, but I do it in parallel, replace all the A's by the same rule, for one A I cannot use A goes alpha another A I cannot use beta, that is not possible. If I do that, do I get something higher than what we have, suppose I have the grammar with two rules, S goes to S S, S goes to a.

Now, starting from S if I apply this rule, I get a alone, but starting from S if I apply this rule I get S S, then I have to apply the same rule for this S S, both the S S. So, I will get a, a starting from S, I get S S and then for both the S S I use the first rule, I will get. If I use the second rule, I will get a power 4, but if I do not have that restriction you know that the language generate in the ordinary sense, it is a power n, n greater than or equal to 1. But if I put the restriction that at any stage I have to use the same rule, then you see that the language generated will be a power 2 power n, n greater than or equal to 1 which

is not context free, it is the context sensitive language.

(Refer Slide Time: 57:20)



So, by putting this restriction, I am able to get a language which is not context free, the power is increased the question. The question; one of the question which has open in 1974 was this, if I denote the class of context free languages as CFL and the class of languages generated parallelly, as parallel context free languages or PCL. We know that, there is a language in this, which is not here that, is a power two power n is a parallel context free language, but it is not a context free language. The other way around, can every context free language be generated by the parallel context free mechanism or that is CFL, is CFL included in PCL or they are like this.

(Refer Slide Time: 58:28)

Example 10

We consider the Indian parallel grammar:

$$G = (\{S\}, \{a\}, \{S \to SS, S \to a\}, S).$$

Some sample derivations are

$S \Rightarrow a$

$S \Rightarrow SS \Rightarrow aa,$

$S \Rightarrow SS \Rightarrow SSSS \Rightarrow aaaa$ and

$L(G) = \{a^{2^n} / n \geq 0\}.$

It is clear from this example that some non-context free languages can be generated by Indian parallel grammars.

The other way round , the question is : can all context free languages (CFL) be generated by Indian parallel grammars ? Since the first attempt to solve this was made in (Siromoney and Krithivasan . 1974) , this type of grammar is called an Indian parallel grammar.

So, this was open problem in 1974 and we attempted to solve this problem. In our paper in information and control 1974.This the problem was, can all context free languages be generated by Indian parallel grammars. Now, we first attempted this problem and if the result was published in this paper, we proved that it the situation is not like this, if it is like this. And the example of a context free language which is not a parallel context free language is the duck set. Duck set are the well formed strings of parenthesis, it is not a language of finite index and this cannot be generated by a parallel context free grammar. So, these are some of the attempts about regulator rewriting and in the next lectures we shall see some more advanced topics.