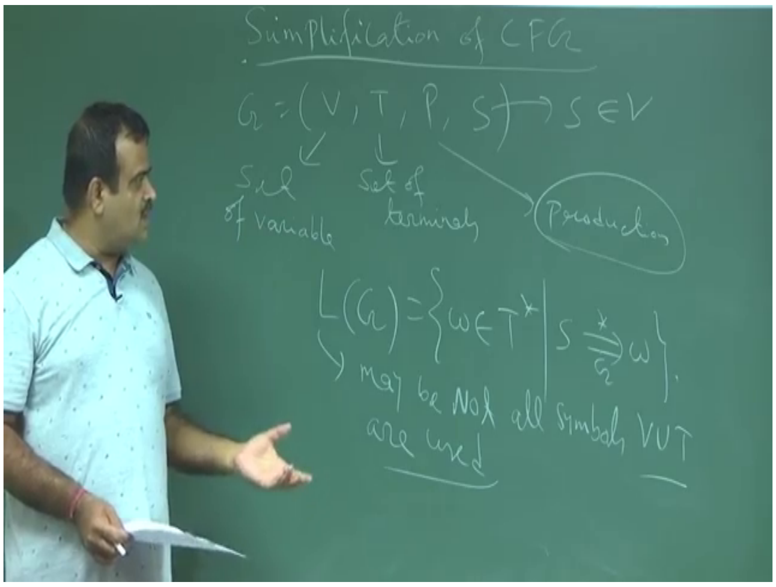
Introduction to Automata, Languages and Computation
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 43
Simplification of CFG

Ok, so, we are talking about context free grammar.

(Refer Slide Time: 00:34)



So, now we will discuss the, how to simplify the grammars like, how to minimize the variables, terminals and the productions, because ultimately, we are looking for, given a grammar G, this is a sorry, this is 4 tuple V T P S, where V is the set of vertices, finite number of vertices, I sorry, the variable set of variables, which is finite and this is the set of input alphabet or we call this terminal in the terms of CFG set of this, also finite terminals.

And P is the productions, set of productions or rules and S is a special symbol, which is a sub set of, which is belongs to V, which is a variable and a which is called starting variable. And we define the language of language generated by this g is nothing, but set of all string of terminals as that which is derived from S, I mean there is a parse tree, , which yields W ok.

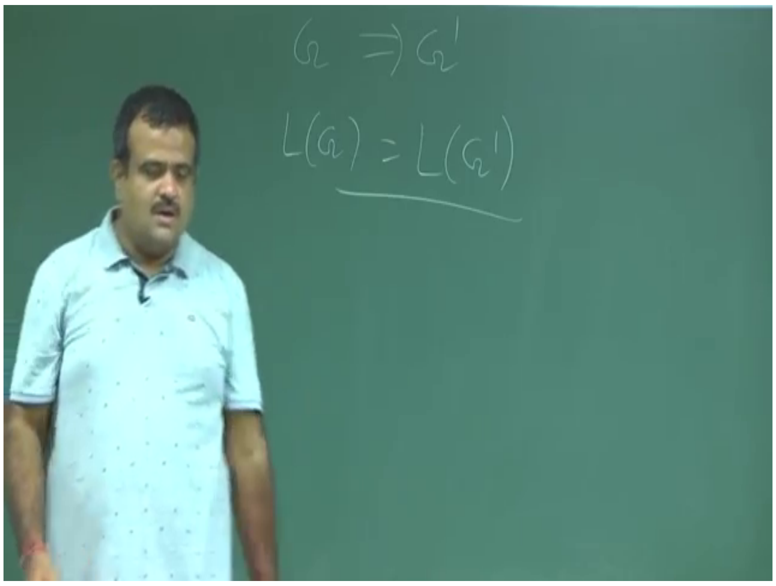
So, this is the language generated by G. Now, we are concerned about this language only, I mean we do not much bother about which, at the variable, it is using inside which of the terminal. Ultimately, we are looking for that the language, which is generating by this grammar; this is the context free language. Now, when we do this, we may observe that may not be all the variables or the terminals are used. So, when we do this we will observe these few things, first one is maybe not all symbols in V union T, I mean either it is a variable or terminals are used.

So, those symbols we can just remove, I mean remove in such a way that it should, it should give us the same language. Ultimately, we are looking for this language, you want to get this language, but if there are I mean in the process of the productions, I mean derivation. If we can, find out the some of the variables or some of the terminal even are not, you are not at all use in that derivation.

So, in any of the derivation; so we can just remove those and also maybe not all the productions are used, not all the productions are used to generate these variables. I mean generate this grammar, generate this language ok. So, then we can remove those productions. So, we try to eliminate those symbol, those production, so that way we will do that, we will get the simplified version of this grammar and that will give us the same, that will generate the same language. So, let us take an example, then it will be more clear.

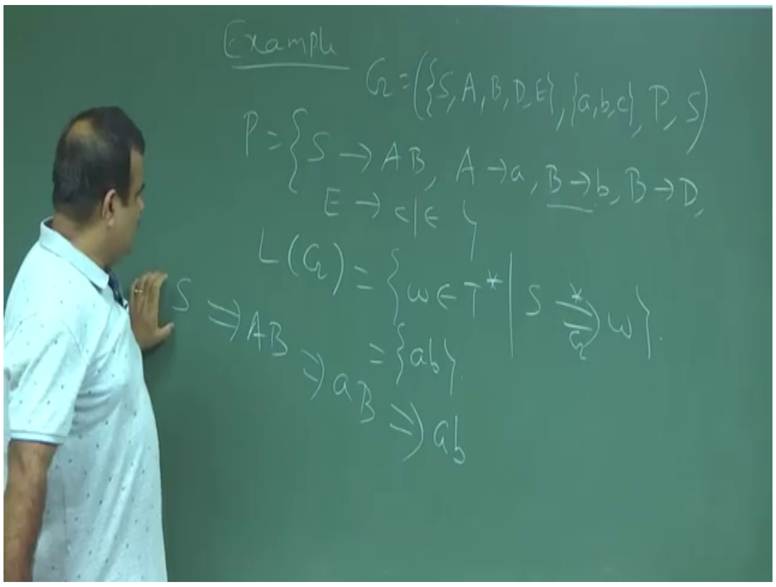
So, ultimately we are looking for the grammar, the language generated by this grammar. So, if we can reduce these, by reducing the some of the variable, some of the terminal, some of the productions, then it is good.

(Refer Slide Time: 04:14)



Ultimately, it will we have a grammar G, if we can reduce this to G prime by reducing some of the variable some of the terminals and if we can count this to are generating the same grammar, by same language, that is good, that is called simplification; that is what we are looking for in this lecture.

(Refer Slide Time: 04:38)

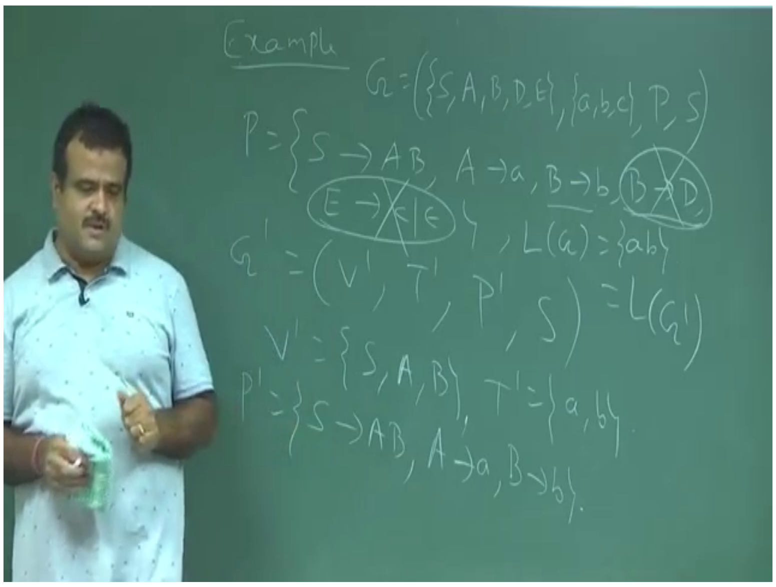

Example
G = ({S, A, B, D, E}, {a, b, c}, P, S)
P = {S → AB, A → a, B → b, B → D, E → c | ε}
L(G) = {w ∈ T* | S ⇒*G w}
= {ab}
S ⇒ AB ⇒ aB ⇒ ab

So, we will take some example to see some of the, so let us take an example. So, suppose we have given this grammar G, which is 4 tuple again. These are the variables S A B D E and the terminals are small abc and they have a productions rule and S is the starting variable. So, let me write the P. So, P is in consists of all these rules. S is going to A B this is wrong rule. A is going to a, B is going to b, and also B is going to D, and E is going to small c or epsilon.

So, these are the productions, this set is the P set, the production set or the rule set. Now, we want to see what are the what is the L of G? L of G is nothing, but we start with S and which terminally L of G is nothing, but all the strings, which can be derived from S using the production on in G. So, this is nothing, but only a b, if we see a b string, because S is going to we can take A B or you one, rule then, if we reach to D by using B; if we can use the left most derivation, if we use the leftmost derivation. So, A is going to small a and b. Now, B is b you have to option on a small, B and on this D. So, if you go to D then we are stuck, we are no had to go.

So, then we use this one ab; ultimately, a b is the only string, it is going, it is it is yielding by the parse tree or it is derived from the yes, so this is only a b. So that means, in this grammar, we can observe, there are many variable, which are of no use.

(Refer Slide Time: 07:15)


Example
$G = (\{S, A, B, D, E\}, \{a, b, c\}, P, S)$
$P = \{S \to AB,\ A \to a,\ B \to b,\ \cancel{B \to D},\ \cancel{E \to c \mid \epsilon}\}$
$L(G) = \{ab\}$
$G' = (V', T', P', S)$
$= L(G')$
$V' = \{S, A, B\},\ T' = \{a, b\}$
$P' = \{S \to AB,\ A \to a,\ B \to b\}$.

So, L of G is just a b. For example, this is, this c is not use anywhere, in deriving this, this language, even this rule, this D is not use. So, we can reduce this grammar by G prime, where V prime, T prime, P prime and S will be same, because S is the starting variable, which will be same in both the cases.

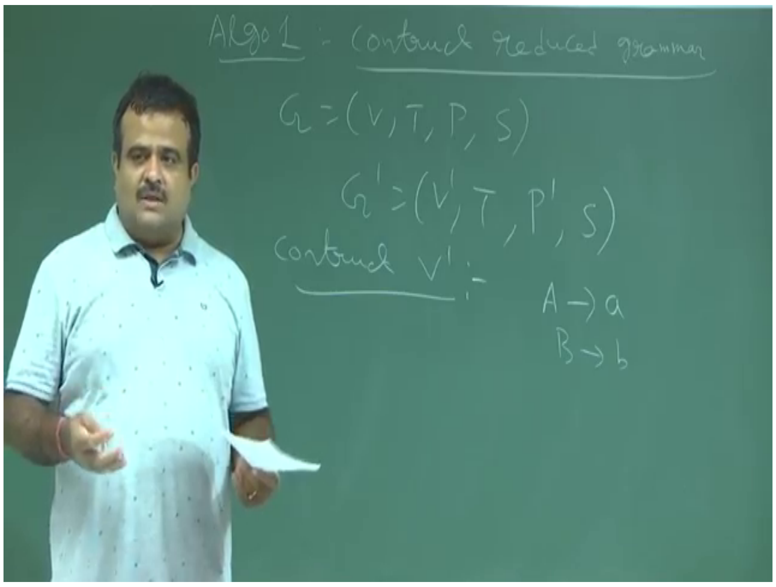
So, V prime will be what? V prime will be, we can just remove all those variables, which are not at all used to generate A B. So, which are not at all use like D is not use E is not used. So, we can just put S A B, these are all only use. So, what is T prime? T prime is we can similarly, move all the terminals which are not used. So, only a b are use, small a small b and we can remove some production, which are not participating in the deriving the language.

So, like this production, this is not at all participating, there this production which is including the epsilon null production. We can eliminate null production, later on formally we will see how we can eliminate the null production and this is also called unique production. B is going to D unit production, then we can replace B by D and we can remove this production from this B ok.

So, we can eliminate this production, we can eliminate this production. So, our P prime is nothing, but S is going to AB, A is going to a and B is going to b. This is our P prime and S, capital S is the same, because that is the, that is the starting variable for each of this and this is our G prime and this G prime will give us the same language.

So, this is the language generated by G prime also ok. Now, we will formally do this verification, will use few algorithm or few theorem to do this verification. I to do the simplification sorry; so let us try to simplify formally.

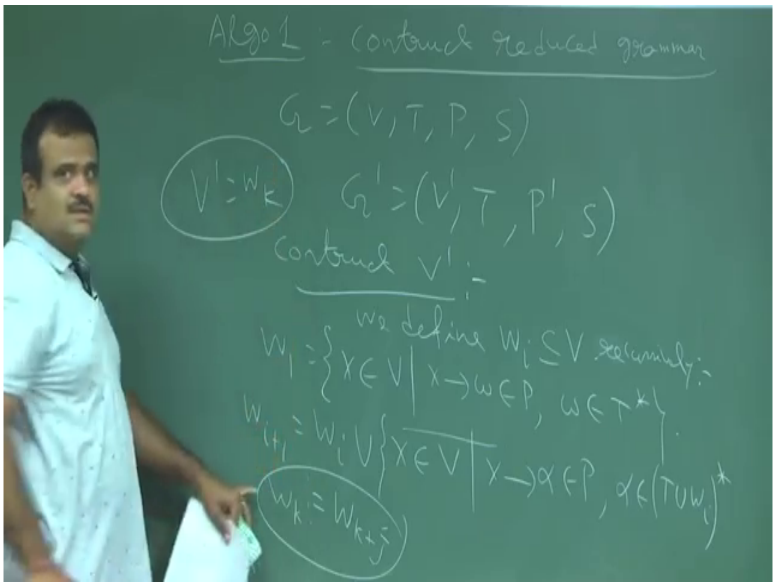
(Refer Slide Time: 10:14)



So, to get the reduced grammar; so this is the first algorithm, Algo 1 we can say or theorem 1, Algo 1 to reduce construct, reduce grammar ok. So, idea is, so we have given a grammar G, we want to reduce this grammar, we want to simplify this grammar, G prime, S is same and to construct this V prime, we can just so, here we are not reducing the T, here we are reducing the only the V prime and the production and how to construct this will recursively construct the V prime.

So, what we do? We just look at from the all the production, which is going to the terminal and we look at those variables. Suppose, we have a say production A is going to say, it sometime terminal A and say B is going to some terminals. So, we only look at all the production, which is whose the right hand side are only the terminals, then we will include the left hand side variable to this V prime, then that is our w 1 and then we will take the all the previous level line, the level wise. Let us define that.

(Refer Slide Time: 12:01)

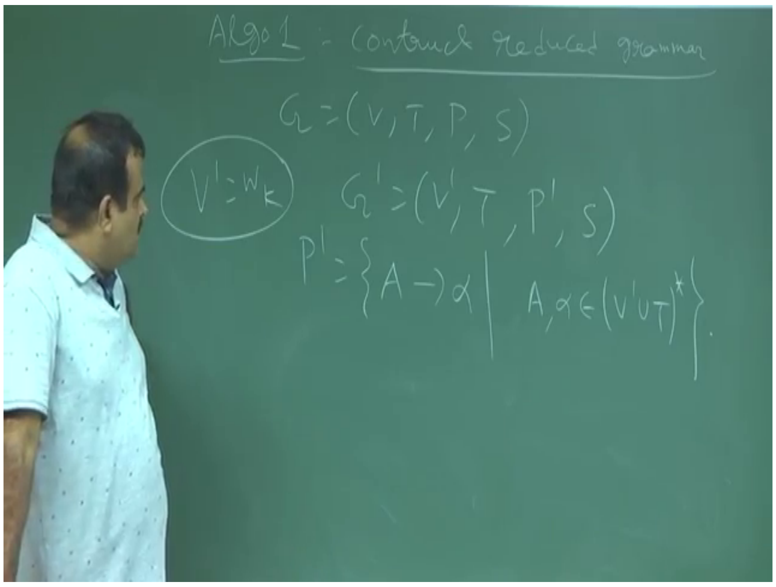

Algo 1 :- Construct reduced grammar
$G = (V, T, P, S)$
$V' = W_k$
$G' = (V', T, P', S)$
Construct V' :-
we define $W_i \subseteq V$ recursively :-
$W_1 = \{X \in V \mid X \rightarrow w \in P, w \in T^*\}$
$W_{i+1} = W_i \cup \{X \in V \mid X \rightarrow \alpha \in P, \alpha \in (T \cup W_i)^*\}$
$W_k = W_{k+j}$

So, we define this w i which is a subset of V, recursively how. Now w 1 is nothing, but all the variables we add in w 1 which is having, which is having a production form w. From this X to alpha, alpha is a string of terminals. So, X is going to, this is a product where w is T star. So, we capture all the variables which are directly going to the that, which are directly going to the terminal strings. So, these we capture. And once we get this one, once w 1 is formed then we can have w 2 w 3. So, in general wi plus 1 is equal to w i union off

So, like i is equal to 1, then we have already from w 1 then we are going to get the w 2. So, w 2 will be; so all the X. Now we have a X is going to alpha in P where this alpha belongs to this w 1 union T star. So, T union wi star; that means, this is a string of terminals and the variables which are already added in the earlier w. So, those strings we are capturing. So, you are starting from the, this is sort of we are considering all the variables which is directly going to the terminal. Once we captured those then we are capturing the next level variable which are going to those variables or mixture of terminals and those variables

So, that is our w 2, w 3 like that. So, if we continue this, at some point of time we will see it will try, it will converge. I mean it will it will stop. So, say w k. So, w k is equal to w k plus J, it is not further update then we can say our V prime is w k. Once it is stopped we will take an example and then what will be P prime, once we get the V prime we have the T. So, P prime will be the all the rules which are involves only these symbols.

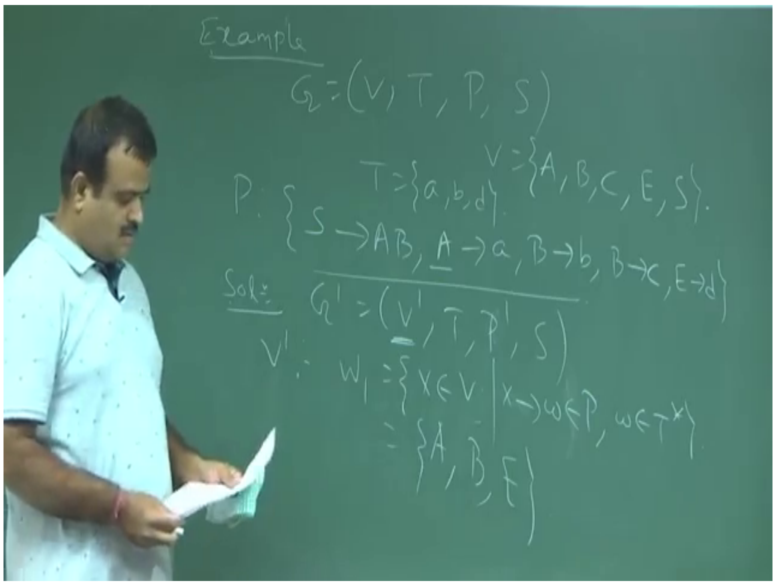
(Refer Slide Time: 14:52)


Algo 1 : construct reduced grammar
$G = (V, T, P, S)$
$V' = w_k$
$G' = (V', T, P', S)$
$P' = \{A \rightarrow \alpha \mid A, \alpha \in (V' \cup T)^*\}.$

So, P prime is all the rules, A is going to alpha where A and alpha are belongs to V prime each time. So, will not consider all the other variables which are not involving V prime or T prime, i mean or T and S is remain same.

So, let us take an example. So, this is we are saying, this is the Algo 1 to reduce the grammar.

(Refer Slide Time: 15:42)

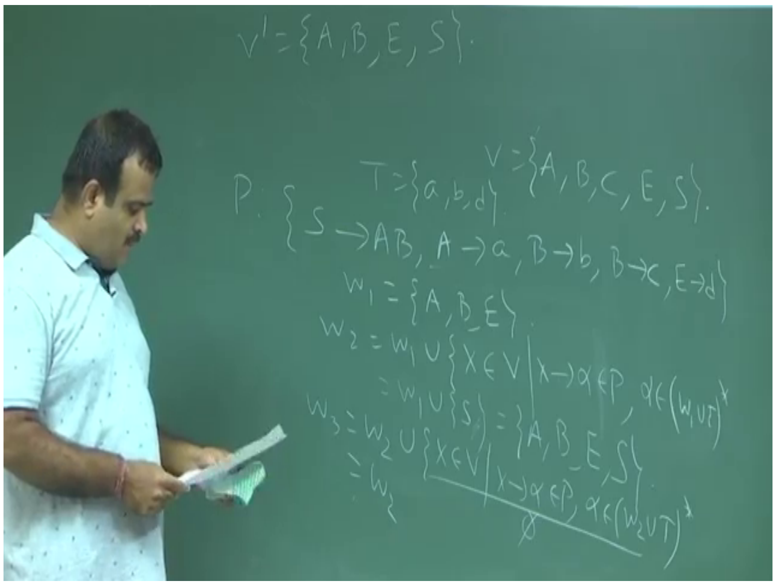

Example
$G=(V, T, P, S)$
$V=\{A, B, C, E, S\}$.
$T=\{a, b, d\}$.
$P: \{S \to AB, \underline{A} \to a, B \to b, B \to c, E \to d\}$
Sol:
$G'=(\underline{V'}, T, P', S)$
V':
$W_1=\{X \in V \mid X \to w \in P, w \in T^*\}$.
$=\{A, B, E\}$

So, let us take an example. So, ok, so let G be the grammar V T P S where V is the set of variables A B C E S and T is the terminal set which is say a b d. So, all are small and P consists of this rules, S is going to A B A is going to small a, B is going to small b, B going to c and E is going to d, E is going to d.

So, this is given, this grammar is given. From there we are we are going to reduce this grammar using the process. We have seen the Algo 1; that means, we want to see that we are looking for the those variables which are directly going to the string of terminals; that is the first level. Then the next level we are just adding those variables which are going to the first level, including mix string of terminals and the first level variable. Then we go for second level which are including the first level. So, these are all subsets ok. So, this is way we construct; so, let us construct.

So, solution, let us construct the G prime. So, G prime is G prime T is same here and also P prime and S. So, what is G prime. So, first you have to construct the V prime. So, to construct V prime you have to construct this w. So, w 1, w 1 is the set of all variables X which is going to w. This is a rule, all the rules or w is a. So, this is a. Sorry, I am sorry this is all the variables, all the variables in all the variables in V such that X is going to w in P; that is a rule where w belongs to T star. So, if you do that then who are the variables are directly going to the terminal strings or terminals. So, this A B and E, so A B E, this is our w 1. Now we have to get w 2 ok

(Refer Slide Time: 18:50)

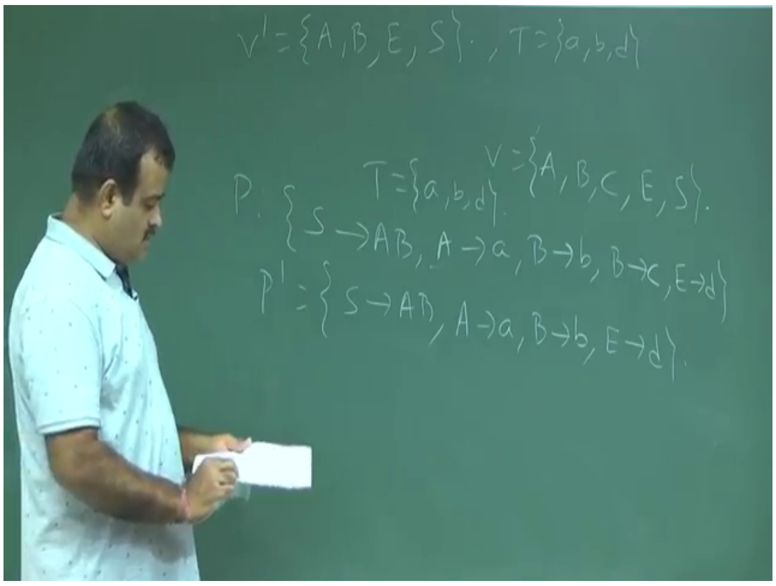

$V' = \{A, B, E, S\}$.
$T = \{a, b, d\}$ $V = \{A, B, C, E, S\}$.
$P: \{S \to AB, A \to a, B \to b, B \to c, E \to d\}$
$W_1 = \{A, B, E\}$.
$W_2 = W_1 \cup \{X \in V \mid X \to \alpha \in P, \alpha \in (W_1 \cup T)^*\}$
$= W_1 \cup \{S\} = \{A, B, E, S\}$
$W_3 = W_2 \cup \{X \in V \mid X \to \alpha \in P, \alpha \in (W_2 \cup T)^*\}$
$= W_2$
$\emptyset$

So, A B E is our w 1. So, let us, so w 1 is nothing, but A B E. Now what is w 2? w 2 is w 1 union of those X which are going to X is alpha in P, which is a rule where alpha is belongs to. Now w 1 union T star; that means, X is going to the string which involves either the terminals, but terminals we have already captured. So, it will be a mixture of the, that the variable we have already captured and the, and the terminals. So, who are going there; so S is going there

So, this will be w 1 union, S is going there. Any other B is going to c, E is going to d, E is already captured, so S is only going there ok. So, this is basically, w 2 is basically A B E S A B E S. Now what is w 3. w 3 is nothing, but w 2 union of same thing X belongs to V; such that X is going to alpha is a rule where alpha is now belongs to w 2. Alpha is a string combining the w 2 variables w 2, variables for w 2 and the terminals.

So, this is the ao this is empty. You can easily check this is empty. So, this will be w 2. So, it is converging to w 2. So, what is w 2? w 2 is this, so that is our V prime. V prime is nothing, but A B E S. So; that means, C is not coming into the picture capital C ok. Now we have to get the rules; we have to get the rules. Let me rub this. So, now, we have to get the P primes.

(Refer Slide Time: 21:20)

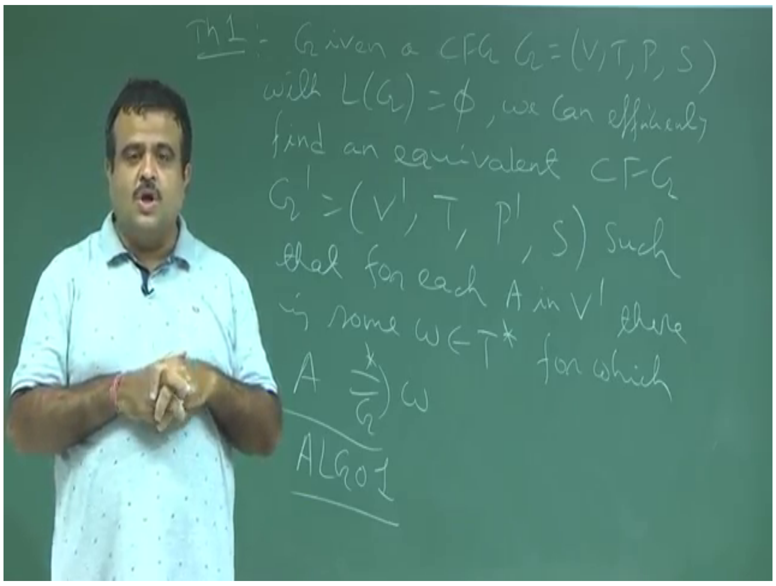

V' = {A, B, E, S}. , T = {a, b, d}
T = {a, b, d}. V = {A, B, C, E, S}.
P. {S → AB, A → a, B → b, B → c, E → d}
P' = {S → AB, A → a, B → b, E → d}.

So, what is P primes? P primes will be the, it is a subset of P, basically the transition production such that which involves only the variables and the terminals T is same, T is same. So, which are the productions are going to this. So, S is going to AB, it will be there and A is going to a, B is going to b and is there

So, E is going to d, E is going to d that is all and terminally still ab abd we are the, but although this is not used, but we have, since E is coming into this. So, in the next algorithm we will reduce this. So, next algorithm we will see, so this is sort of from whichever is, from the terminals we are getting this. Now the next algorithm we will see from the S which are the variables or the set string of terminals and variable we can reach from S.

So, if we apply those then these rules will go. This rules E will go, d will go and this rules will go, because this is only accepting the string A B ok. So, this is our P and. So, P is this, T is remain same and d is, S is remain same. So, this is our reduce grammar for this given grammar by the, by this simplification process which is, which we are referring to Algo 1 ok. So, now, we will write this in a theorem form. Let me write this in a theorem form yeah. So, this is the theorem, this is the construction.

(Refer Slide Time: 23:50)



Th1:- Given a CFG $G = (V, T, P, S)$ with $L(G) = \phi$, we can efficiently find an equivalent CFG $G' = (V', T, P', S)$ such that for each A in V' there is some $w \in T^*$ for which $A \xRightarrow[G]{*} w$

ALGO 1

We can write this in a, yeah we can write this in the theorem. Form this is we referred as theorem 1. So, this is telling given a grammar, given a context free grammar G V T P S with l of G is not empty. If it is empty; that means, the w 1 is, there is no string which is in w 1, nobody is going there. So, that case we are not considering. We can efficiently find an equivalent C F G which is G prime V prime E same S; such that for each A in V prime, there is some w string of terminal, for which, A is going to w

So, for each variable of w is for each variable in G prime derives a terminal string. So, that is the idea, that is the idea. So, this is basically our Algo 1, this is basically our Algo 1. And we can prove this if we construct this by Algo 1; the way we did it just now. Then we can formally prove that this will give us the, this is true; that means, all thus have been given any string of terminal for each variable. There is some terminals, because each variable we are considering, only those variable we are reaching to the terminals, terminal strings ok.

So, maybe not all the, all the variables, all the variables are reachable form S. So, that, but we are not considering here, so that will be the Algo 2, where we will only look at that those variables or those terminals. I mean string of variable and string of terminals which are reachable from S only. So, that we will discuss in the next algorithm; that is algorithm 2.

Thank you.