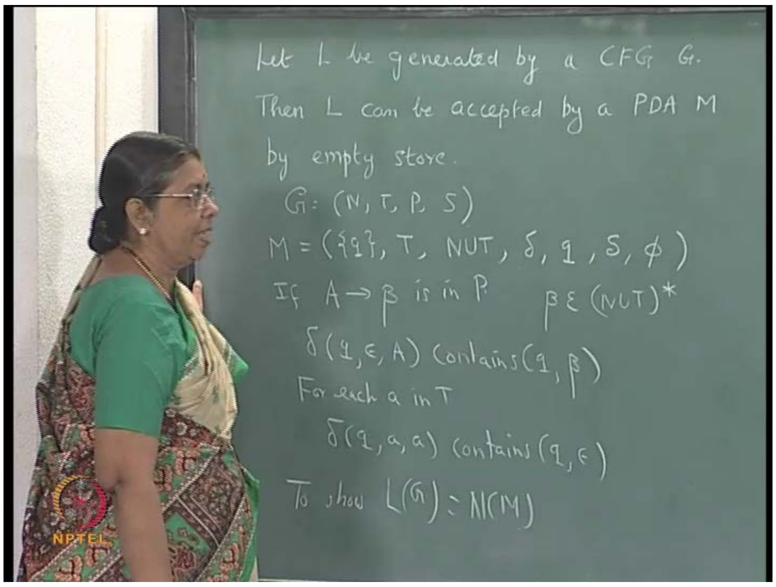
Theory of Computation

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

Lecture No. # 22
Pushdown Automata CPG to PDA

Today we shall see about the equivalence of context free grammars with pushdown automata. When we say pushdown automata, we really mean non-deterministic pushdown automata. The equivalence can be shown only between context free languages and non-deterministic pushdown automata. So, we have to prove in two directions, given a context free grammar, how you can construct an equivalent pushdown automaton and given a pushdown automaton, how you construct an equivalent context free grammar.

(Refer Slide Time: 00:43)



Let L be generated by a CFG G.
Then L can be accepted by a PDA M
by empty store.

$G = (N, T, P, S)$

$M = (\{q\}, T, N \cup T, \delta, q, S, \phi)$

If $A \rightarrow \beta$ is in P $\quad \beta \in (N \cup T)^*$

$\delta(q, \epsilon, A)$ contains (q, β)

For each a in T

$\delta(q, a, a)$ contains (q, ϵ)

To show $L(G) = N(M)$

So, first we shall consider on one way that is let, L be generated by a CFG G. Then L can be accepted by a PDA M by empty store. The equivalence we prove in this manner that is we show the equivalence between context free grammars and PDA accepting by empty store. If context grammar is given, how to construct the pushdown automata accepting by empty store and you pushdown automata accepting by empty store is given, then how to generate the context free grammar? This way we will prove. We have already proved the equivalence between PDA accepting by final state, both directions we have proved.

So, that means even CFG you can construct a PDA accepting it by final state and given a PDA by final state you can generate the context free grammar that is also possible as this is transitive, equivalence is transitive. So, you can prove that way. So, what we consider today is this, given a context free grammar, how to construct the pushdown automata which will accept the language at generate by the context free grammar by empty store. Actually two proofs are possible, in one you can assume that the context free grammar is in griebach normal form. In another proof knows such an assumption is necessary. So, we will take the proof where there is no assumption first and then how the proof is given for the case when the grammar is in griebach normal form, we will take up later.

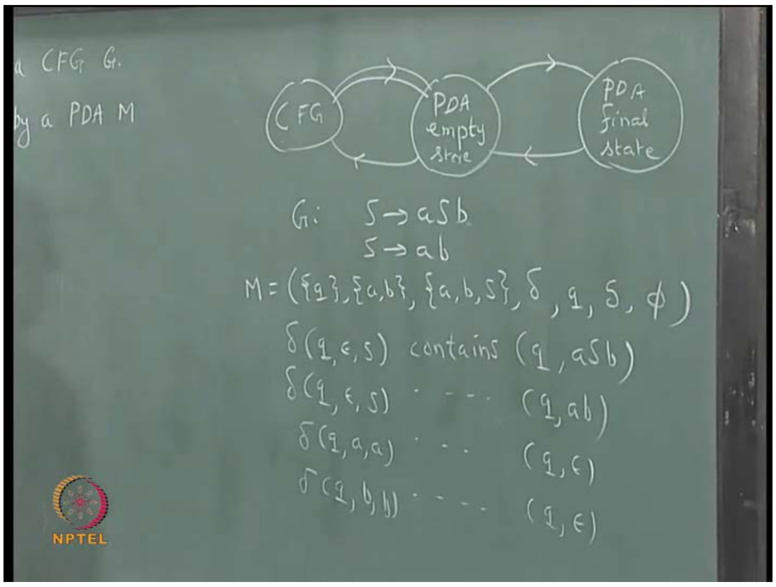
So, here you are given the context free grammar G is equal to N T P S. Now, you have to construct the pushdown automata M is equal to K sigma delta etcetera. But K is just only one state you can do it with one state K sigma is T, gamma is N union T, all symbols which are non terminals and terminals can be stack symbols, delta you have to define q naught there is only one state that has to be the initial state and the initial start symbol is the initial pushdown symbol and because it is acceptance by empty store you have phi.

So, you have only one state you can accept with one state and the terminal symbols are the input symbols, the stack symbols or the pushdown symbols are N union T, delta we have to define. There is only one state which has to be the state throughout does not make any difference, the initial pushdown symbol is the start symbol of the grammar. No final states are specified because it is acceptance by empty store.

Now, we have to define only delta. Delta is defined like this, if a goes to beta is in P the context free rules are of the form the left hand side you have a single non terminal, on the right hand side, you have a string. What is P? Beta belongs to N union T star, it is a string of non terminals and terminals on the right hand side. If this is a rule then you have delta of q epsilon A contains q beta. You, make an epsilon move, you do not read any input but, on the pushdown stack you replace a by beta. Please remember that when we write like this the leftmost symbol of beta becomes the top of the stack. Then for each a in T, each terminal symbol you have delta of q a a contains q epsilon; this is all the mapping, delta mapping you have.

Now, what we have to show is L G is equal to N of M; this is what we have to show. Construction is this, grammar is given. You are constructing the pushdown automata like this, then you have to show this. This proof is of course, by induction we will go to that in a moment before that let us illustrate the construction by an example.

(Refer Slide Time: 07:04)



a CFG G.

by a PDA M

$$\begin{tikzcd}\text{CFG} \arrow[r, bend left] & \text{PDA empty state} \arrow[l, bend left] \arrow[r, bend left] & \text{PDA final state} \arrow[l, bend left]\end{tikzcd}$$

G: $S \to aSb$

$S \to ab$

$M = (\{q\}, \{a,b\}, \{a,b,S\}, \delta, q, S, \phi)$

$\delta(q,\epsilon,S)$ contains (q, aSb)

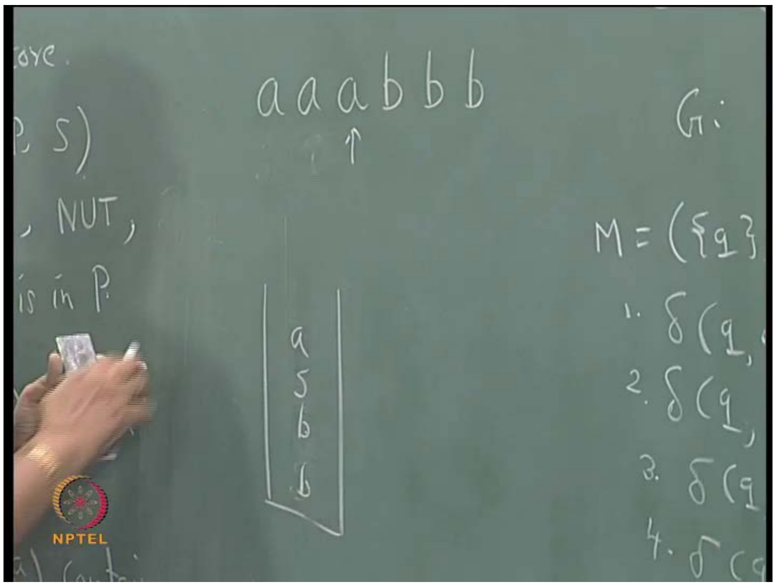
$\delta(q,\epsilon,S)$ ---- (q, ab)

$\delta(q,a,a)$ --- (q,ϵ)

$\delta(q,b,b)$ ---- (q,ϵ)

Let us take the familiar example you have, G consist of two rules; then you have the pushdown automaton M is equal to q, input symbols are a, b, pushdown symbols are a, b and S, initial state is delta we have to specify, initial state is q, initial pushdown symbol is S, no final states. Now, we have to define delta and how do you define delta? Delta of q, epsilon, S contains for the first rule q, aSb; q epsilon S contains q, a S b; then delta of second rule q epsilon S contains q, ab. So, we can immediately say that it is non deterministic in nature. You have two moves for this q epsilon is either you can replace by a S b or a b, it is not deterministic. Then a and b are the two terminal symbols so delta of q a a contains q epsilon, delta of q b b contains q epsilon.

(Refer Slide Time: 09:09)

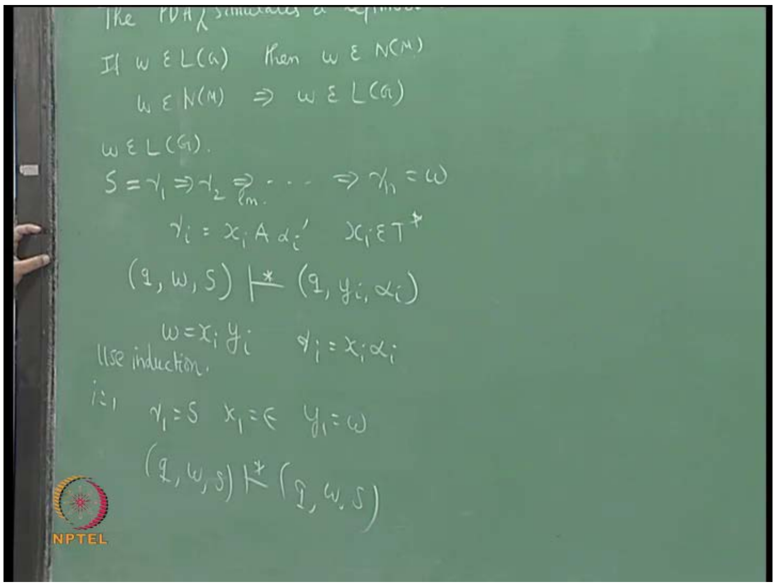


Now, let me take an input and see how it works. You know that the grammar generate equal number of a’s and b’s or equal number of a’s followed by an equal number of b’s, that should be the way a a b b b. How this will be accepted. This pushdown automaton should accept that, so you start from here with initial state is throughout q so need not mark that, initially you have S then when you have a non terminal on the stack use and epsilon move. When you have a non terminal on the top there stack symbols are terminals and non terminals. So, when you have a non terminal on the top of the stack you use an epsilon move. When you have a terminal on the top of the stack you use a pop move.

So, now use an epsilon move. This epsilon S will be you can use the first move, delta of q epsilon S contains q a S b. So, this S will be replaced by a S b, leftmost symbol becomes a top of the stack. Now you have a terminal symbol on the top of the stack so you use a you have to use a pop move. So, this a will be read and this will be removed from the stack now and again the top of the stack is a non terminal then you use an epsilon move you can again use the first move, so it will be a S b on the pushdown store. Now, the top of the stack is a terminal symbol, so you use a pop move. So, this will be read and this will be removed. Now, again the top of the pushdown symbol, top of the stack is a non terminal so you use the second move use an epsilon move replace it by a b.

Now, the top of the pushdown stack is a terminal so you read this, remove this, read this now again the same thing will happen read and remove this, read this and remove this, read this and remove this at the end the stack is empty. You are simulating a leftmost derivation of the grammar on the stack but, in the case of a linear grammar this is a linear grammar does not make a difference between leftmost rightmost and so on. So, happens that this only one non terminal generally a leftmost derivation of the grammar is simulated by the pushdown automata.

(Refer Slide Time: 12:29)



The PDA M simulates a leftmost

If $w \in L(G)$ then $w \in N(M)$

$w \in N(M) \Rightarrow w \in L(G)$

$w \in L(G)$.

$S = \gamma_1 \Rightarrow \gamma_2 \underset{lm}{\Rightarrow} \cdots \Rightarrow \gamma_n = w$

$\gamma_i = x_i A \alpha_i' \quad x_i \in T^*$

$(q, w, S) \vdash^* (q, y_i, \alpha_i)$

$w = x_i y_i \quad \gamma_i = x_i \alpha_i$

Use induction.

$i = 1 \quad \gamma_1 = S \quad x_1 = \epsilon \quad y_1 = w$

$(q, w, S) \vdash^* (q, w, S)$

So coming to the proof of it the PDA simulates a leftmost derivation in G, the PDA M we have defined it as M, simulates the leftmost derivation in G. This is what we have to prove.

So, given a grammar we have constructed the pushdown automata in the proof, what we have to show that if a string is generated by the grammar, it will be accepted by the pushdown automata and if a see that is in the proof what you have to prove is, if w belongs to L G then w belongs to N of M. One way you have to prove, in another way you have to prove that w belongs to N of M implies w belongs to L of G, please note that construction is one way and proof you have to do this. Then similarly, when we construct the grammar from the pushdown automaton we have to prove. So, first we have to prove this. So, suppose w belongs to L G then there is a derivation S is equal to gamma 1. Gamma this is a leftmost derivation, gamma n equal to w. This is a leftmost derivation.

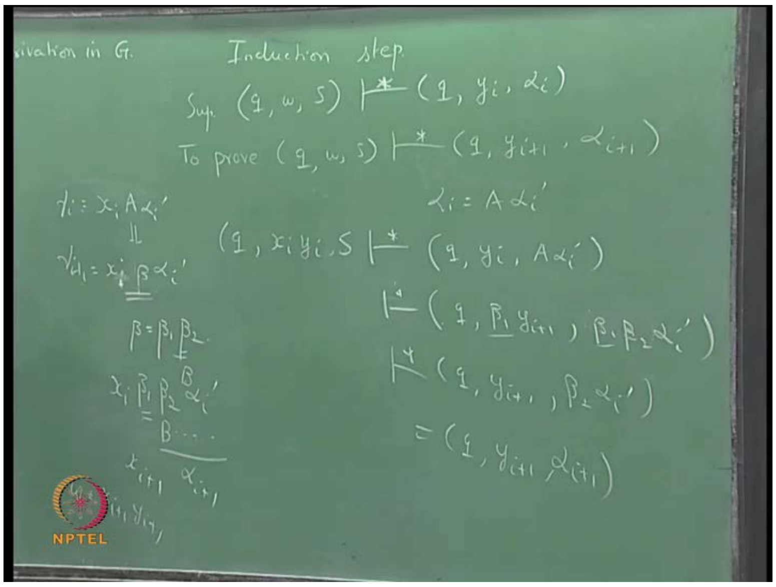
So, what can you say about each gamma i? Each gamma i will be of the form some x i a some alpha i dash. This x i will be a terminal portion, it is a string of terminals. Leftmost non terminal is always replaced so if each one is a sentential form. Each from S which is gamma 1 you derive by one step gamma 2 by one step you could derive gamma 3 and so on. So, it is a leftmost derivation.

So, what can you say if a is the leftmost non terminal, this portion is a string of terminals x i belongs to T star. And this alpha i dash can be anything at the next step this a has to be replaced. Next step a will be replaced. So, what you prove is if in this derivation you have gamma i like this, from q w S. This is the initial configuration isn't it. There is only one state. State does not change throughout, w is the whole input to be read and s is the initially the stack contains only S and ID of the pushdown automata, we have earlier seen that the ID consist of three components q some x alpha or something like that. Three components q is the current state, x is the portion of the input still to be read and alpha is the content of the stack and ID is defined like this.

So, what you prove is q w S, you can go to q y i alpha i, for all i you have to prove by induction this step. Where what is y i? What is alpha i? w can be written in the form x i y i, w is x i y i and from going this ID to this ID this x i has been read, pushdown automaton has consumed this input. It has read this portion. This portion which is still to be read is y i and what is alpha i? Now, gamma i is x i alpha i, gamma i here is x i alpha i. That is this terminal portion has been derived and this is still some portion which is to be derived and please remember that the leftmost symbol of alpha i will be a non terminal. Now, use induction. What do you get? First, when i is equal to 1 what is gamma 1? It is S itself, gamma 1 is S itself so what can you say, gamma 1 is S, so x 1 is epsilon, y 1 is w the whole input has to be read now.

So, q w S reflexive transitive closure q w S. x i y i is w and gamma i is x i alpha i what is alpha i? Alpha i is this, alpha 1 assessment.

(Refer Slide Time: 19:07)



rivation in G.

Induction step

Sup. $(q, w, S) \vdash^* (q, y_i, \alpha_i)$

To prove $(q, w, S) \vdash^* (q, y_{i+1}, \alpha_{i+1})$

$\gamma_i = x_i A \alpha_i'$

$\Downarrow$

$\gamma_{i+1} = x_i \beta \alpha_i'$

$\alpha_i = A \alpha_i'$

$(q, x_i y_i, S \vdash^* (q, y_i, A\alpha_i')$

$\vdash (q, \underline{\beta_1} y_{i+1}, \beta_1 \beta_2 \alpha_i')$

$\vdash^* (q, y_{i+1}, \beta_2 \alpha_i')$

$= (q, y_{i+1} \alpha_{i+1})$

$\beta = \beta_1 \beta_2$

$x_i \beta_1 \beta_2 \alpha_i'$

$B \ldots$

$x_{i+1} \quad \alpha_{i+1}$

$x_{i+1} y_{i+1}$

So, bases is ok. Induction step, suppose q w S you can derive. q y i alpha i where you must remember that alpha i the leftmost symbol is a non terminal. Then to prove q w S derives q y i plus 1 alpha i plus 1. Now, alpha i I can write as some a alpha i dash leftmost symbol is a non terminal.

So, from this q this we can write as x i y i, x i actually it is like this x i y i w is the after reading this portion the stack contains alpha i, that is the situation. Alpha i is a alpha i dash I am sorry I am sorry this is S so in you from this you go to q y i, a alpha i dash. Alpha i means a alpha i dash. So, in this derivation in the grammar you have gamma i is x i a alpha i dash. In the next step gamma i plus 1 you have x i a will be replaced by beta, a will be replaced by beta then this is there.

Now, beta can be fully terminal or portion of it can be non terminal, portion of it can be terminal and so on. Whatever it is suppose I assume when this is only one case beta can be full of terminals also, does not matter. So the thing is, next step, a is replaced by beta. Then after you replace it you get a gamma i plus 1. Now, this you can write as x i plus 1, y i plus one this you can split.

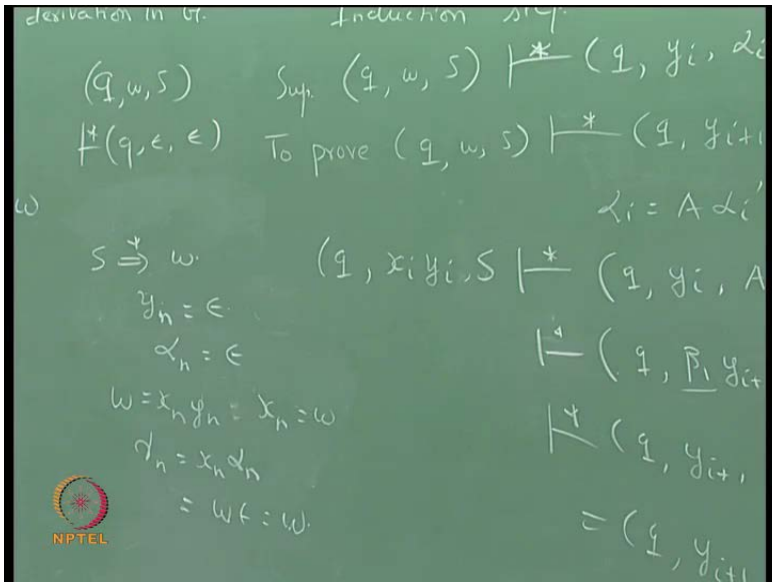
So that the first portion is entirely terminal y i plus 1, this may contain some non terminal and so on. So, in this step from gamma i you go to gamma i plus one in the left most derivation. What you do is a is replaced by beta so you get x i, x i beta alpha i dash. This you can write as x i plus 1, y i plus 1. Where x i plus 1 includes x i when prefix of x i plus 1 is x i and all the terminal portion is there, the first non terminal is b and whatever is occurring after that is beta. I am sorry I am sorry this is y naught y i plus 1, beta alpha i dash this you can write as x i plus 1, b alpha i plus 1 dash this is the way, it is not y i. I made a mistake, this is alpha i plus 1, x i plus 1, alpha i plus 1. This can contain non terminals. w only can be written in the form x i plus 1 y i plus 1. Now what is this y i plus 1? What is this y i plus 1? I will take a simple case where beta has a non terminal, that is only one case. Other case will be similar.

So, suppose beta has a non terminal, beta is beta 1 beta 2. The first symbol of beta 2 is a non terminal, beta 1 is terminal. So when beta, you get x i beta 1, beta 2, alpha i dash, write the first symbol of this is b and you have something. This you call as alpha i plus 1. This is fully terminal, beta 1 is fully terminal and so that will be x i plus 1.

So, here what happens is after this step, this y i the first portion will be beta 1 and so on. It will be like this, the y i see the portion this beta 1 you will consume and so this is x i plus 1 means w is x i plus 1, y i plus 1. What is x i plus 1 x i with beta 1. So, y i is beta 1 y i plus 1, this is equal to this. y i is this, this equal to this and a alpha i dash you have.

Now, a will be replaced by beta. So, the next step what happens is this beta 1, beta 2 alpha i dash. Now, beta 1 consist of terminals alone so you can move use a moves of the form q a a contains q epsilon and pop the symbols. So, this will be read and this will be removed. So, you will end up with q y i plus 1, beta 2 alpha i dash. But, what is beta 2 alpha i dash that is alpha i plus 1. So, that is equal to q y i plus 1 alpha plus 1. This is a case when beta contains a non terminal, beta can be full of terminals also in that case you will read symbols up to the first non terminal in alpha i dash. You will be reading symbols terminal symbols and popping of symbols from the stack.

(Refer Slide Time: 27:35)



derivation in G.

(q, w, S)

$\vdash^* (q, \epsilon, \epsilon)$

w

$S \overset{*}{\Rightarrow} w.$

$y_n = \epsilon$

$\alpha_n = \epsilon$

$w = x_n y_n = x_n = w$

$\gamma_n = x_n \alpha_n$

$= w\epsilon = w.$

Induction step.

Sup. $(q, w, S) \vdash^* (q, y_i, \alpha_i$

To prove $(q, w, S) \vdash^* (q, y_{i+1}$

$\alpha_i = A \alpha_i'$

$(q, x_i y_i, S \vdash^* (q, y_i, A$

$\vdash^* (q, \underline{\beta_1} y_{i+}$

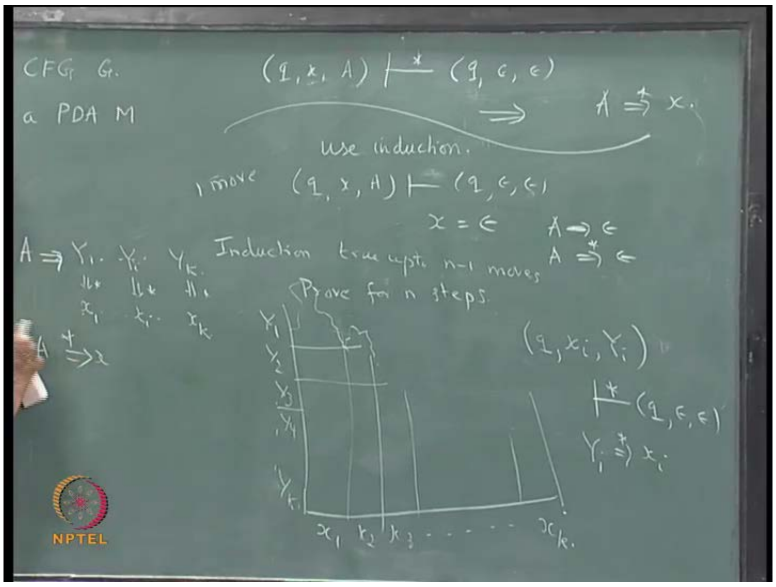
$\vdash^* (q, y_{i+},$

$= (q, y_{i+1}$

So, whatever it is if you have this you can have this by induction from this step you can go to this step and generally in the derivation from S you go to w that means the nth step y n is epsilon, alpha n is also epsilon. w is x n y n, so x n is w y n is epsilon and you have gamma n is x n alpha n, x n is w alpha n is epsilon, so w.

So, x n y n is epsilon, alpha n is epsilon, so from q w S you go to q epsilon. What does that mean? You are accepting w by empty store. So, if a string w is generated by the grammar it is accepted by the machine by empty store. Now, we have to prove the converse, not the converse I mean this portion, if w belongs to N of M then w belongs to L of q, how do you prove this?

(Refer Slide Time: 29:34)


CFG G.
a PDA M
$(q, x, A) \vdash^{*} (q, \epsilon, \epsilon) \Rightarrow A \overset{*}{\Rightarrow} x.$
use induction.
1 move $(q, x, A) \vdash (q, \epsilon, \epsilon)$
$x = \epsilon$ $A \to \epsilon$
$A \overset{*}{\Rightarrow} \epsilon$
$A \Rightarrow Y_1 \ldots Y_i \ldots Y_k$
Induction true upto n-1 moves
Prove for n steps.
$x_1 \ x_i \ x_k$
$A \overset{*}{\Rightarrow} x$
Y_1 Y_2 Y_3 Y_4 Y_k
$x_1 \ x_2 \ x_3 \ldots \ldots \ x_k.$
$(q, x_i, Y_i) \vdash^{*} (q, \epsilon, \epsilon)$
$Y_i \overset{*}{\Rightarrow} x_i$

Now we have to prove something more general. If you have some q x A from this state, if you go to q epsilon epsilon; you are starting in state q and you are consuming the portion x on the input tape, you are initially starting with a on the stack, and then ending up the consuming the whole input and emptying the stack. In this case, you show that A derives x, from A you can derive x. This implies A derives x, this portion you have to use induction; to show this you have to use induction. There the induction was on the number of steps in the derivation of the grammar; here the induction is on the number of steps of the moves of the PDA. Number of moves of the PDA so for one step one move, basis class will be one move.

So, from q x A you get q epsilon epsilon in one move, not more steps. In one move you get. What does that mean? See terminal symbols can be consumed, input can be consumed only when you have a terminal symbol on the stack by our construction the way we have constructed, when there is a non terminal on the top of the stack you can use only the epsilon move. When there is a terminal on the top of the stack you can use a true input move and consume in input. So, when there is a non terminal on the top you can use only epsilon move. That means x is equal to epsilon and you have this must, you we should have written because of the rule a goes to epsilon, this one is possible only if you have a rule like a goes to epsilon is it not? Then only you will remove the stack otherwise a is replaced by beta, beta will be here is it not? If a is replaced by beta you will have beta here. You are having epsilon that means a goes to epsilon is a rule, that is how you have written this mapping and come to this move. So, that means from here you are able to derive epsilon, this is the basis portion, basis class.

Now, true for i, true for n, true up to n minus 1; so, induction portion true up to n minus 1 moves here strong induction we use up to n minus 1 moves we assume. Strong induction up to n minus 1 set the result is true. From this you go to this in, if you go to this i steps where i is less than or equal to n minus 1 the result rules then you prove for n steps.

Now, if there is a non terminal on the stack. Top of the stack then what can be the first move? First move has to be an epsilon move, no input will be consumed but, a will be replaced by something is it not? So in the grammar if a is replaced by say Y 1 Y 2 Y k some set some string some of them maybe non terminal, some of them maybe terminal. Then you have a mapping delta of q epsilon a contains q Y 1 Y 2 Y k isn’t it, this is the way we wrote down the mapping.

So, a will be replaced by Y 1 Y 2 Y k on the stack. That is the first step, you would have used an epsilon move no input has been consumed but, the top in the stack you had a now you have replaced a by say Y 1 Y 2 Y k stack. You have done this. Some of them maybe non terminals, some of them maybe terminal symbols, the stacks pushdown symbols consisted of both N and T symbols. Now, what is a input to be read? Input to be read is x you are reading x, after reading x this stack is empty. That is what it means after reading x starting with a this stack is emptied.

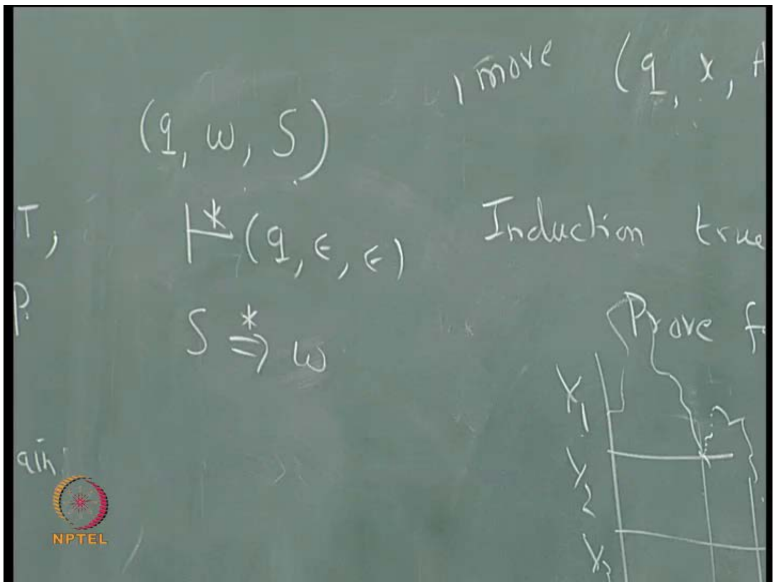
So, after reading this suppose x is this, some portion after reading the stack is emptied. Now, x is split as x 1 x 2 x k, x 1 x 2 x 3 x k. Now, during reading this Y 1 the stack may grow and the first time Y 2 is exposed as a top of the stack, you would have read some portion and that is x 1, some portion of the input you would have consumed and that is x 1. Now Y 2 is exposed as the top of the stack.

Now, you read some more portion and when Y 3 becomes the, in between the stack may grow and come back but, the first time y 2 is exposed as the top of the stack you would have read x 1 and x 2 both this portion you would have read. Similarly, when Y 4 is Y 3 is consumed and Y 4 becomes the top of the stack, x y is the portion of the input you have read and finally, x k is read. What does this mean it means that starting with q when you read the x i portion with y i on the top of the stack at the end in between it may grow and come back but, at the end y i is removed, the next symbol becomes the top of the stack but, here we are starting with just y i.

So, this q the input x i has been consumed and how many steps it would have taken. This whole thing takes n minus 1 steps, first step is to replace a by this, after that it has taken n minus 1 steps to reach this but, this itself you would have taken some much less than n minus 1 isn't it.

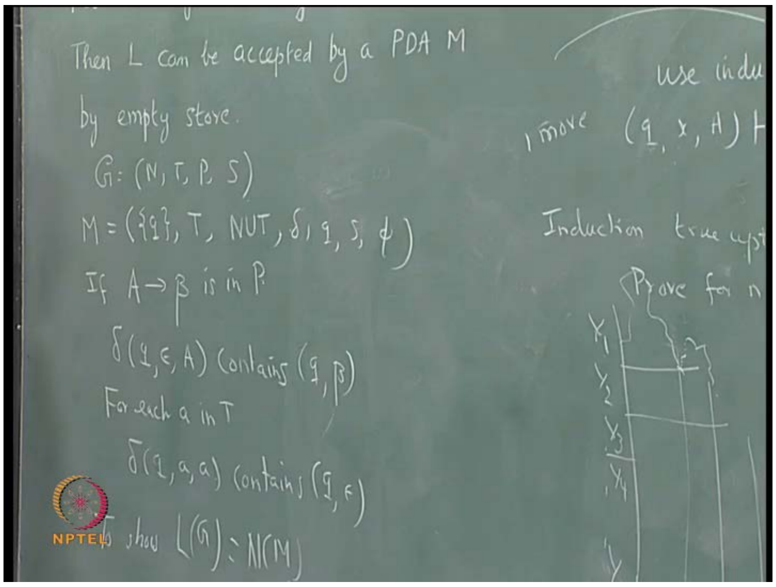
So by the induction hypothesis you have y i derives x i, so the first step you have replaced a with Y 1 Y 2 Y k, that could have happened only if you have a rule like this. Only, if you had a rule like this you can replace the stack by Y 1 Y 2 Y k. So, that is from a you are able to derive Y 1 Y 2 Y k, from Y i you can derive x i. So, for each one of this it happens x 1 x 2 x i x k and this is nothing but, from Y 1 you can derive x 1 from Y i you can derive x I, from Y k you can derive x k and so on. So, that is from a you can derive x.

(Refer Slide Time: 38:43)



So, we have proved this portion using induction. Now, in the end what is initial ID? q w S, this is the initial ID isn't it. This is the initial ID and what is the final ID? When w is accepted this is the final ID, isn't it. Acceptance by m test so use that if this is possible that means S derives w in the grammar, isn't it. a derives x here S derives w. So, we have proved both this portion, given a grammar we have constructed a pushdown automata the construction is like this, given a grammar like this.

(Refer Slide Time: 39:45)


Then L can be accepted by a PDA M
by empty store.

$G: (N, T, P, S)$

$M = (\{q\}, T, N \cup T, \delta, q, S, \phi)$

If $A \to \beta$ is in P

$\delta(q, \epsilon, A)$ contains (q, β)

For each a in T

$\delta(q, a, a)$ contains (q, ϵ)

To show $L(G) = N(M)$

use indu

1 move $(q, x, A) \vdash$

Induction true upt

Prove for n

X_1

X_2

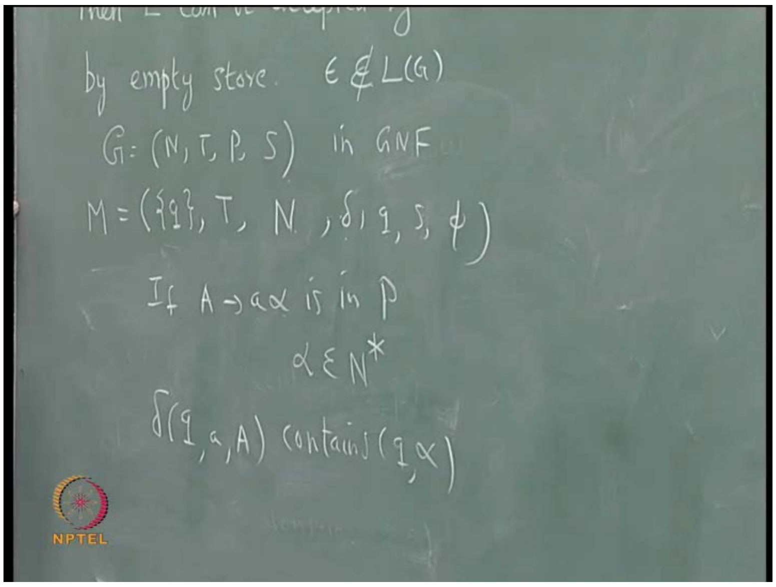
X_3

X_4

You construct a pushdown automaton, the other components I have rubbed off, delta q S phi, you have constructed an automaton. The rules are, the mappings are given like this if a goes to beta is a rule in p then you have delta q epsilon a contains q beta and for each a in t.

So, terminal symbol you have delta of q a a contains q epsilon, this is the construction. And for this construction you have to show L G is equal to N M and that you show like this, you show that the PDA M simulates the leftmost derivation in the grammar and in two steps you prove that given a w in L in L G you can show that it can be accepted by M by empty store, here this portion you prove by induction on the number of steps in the derivation then you also show that if w is accepted by empty store by the pushdown automaton then w is generated by the grammar. Here the proof is by induction on the number of moves of the PDA.

(Refer Slide Time: 41:18)



by empty store. $\epsilon \notin L(G)$

$G = (N, T, P, S)$ in GNF

$M = (\{q\}, T, N, \delta, q, S, \phi)$

If $A \rightarrow a\alpha$ is in P

$\alpha \in N^*$

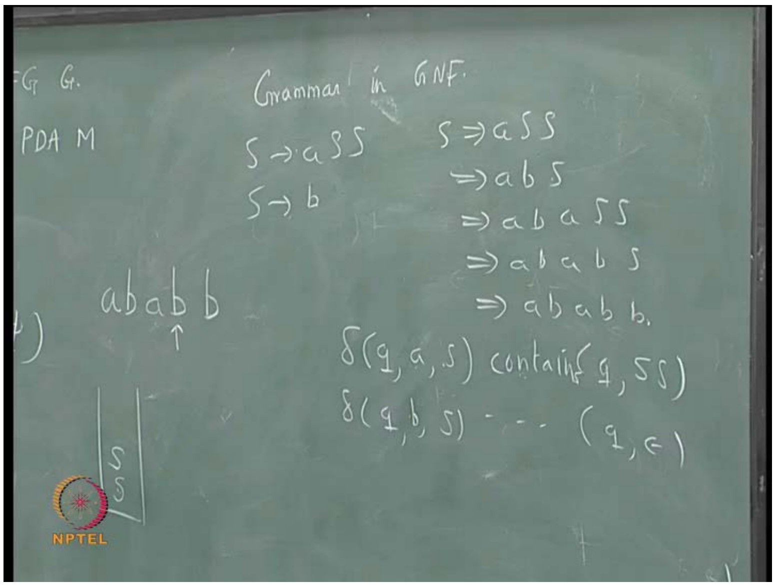
$\delta(q, a, A)$ contains (q, α)

So, this is one proof, there is another proof where the grammar is in griebach normal form, grammar in griebach normal form. So, G is equal to N T P S in GNF, now without loss of the del T you assume that epsilon does not belong to L G because anything without epsilon can be generated by a grammar in GNF, if epsilon is there you have to make some adjustment you have to add S goes to epsilon make sure that s does not occur as the symbol on the right hand side and so on. So, without loss of generality you can assume that S epsilon is not in L G, you have to make small modifications, if epsilon belongs to L G.

Now, how do you construct the pushdown automaton which accepts by empty store, the construction is lightly different. You, have only one state the terminal symbols are the input symbols but, now the pushdown symbols are only the non terminals. terminals will not occur as pushdown symbols, delta we have to define q is a only state that that will be there initial state, S is the initial pushdown symbol all those things are same.

Now, the delta mappings are similar. If a goes to a I am sorry a alpha is in P, griebach normal form you remember that left hand side you have a non terminal right hand side you have a terminal symbol, single terminal symbol followed by a string of non terminals. So, alpha belongs to n star. It can be epsilon also, if it is epsilon the rule is of the form a goes to a. So, if this is the rule you have delta of q a a contains q alpha. Here, again we can prove by induction and so on.

(Refer Slide Time: 43:34)



I will illustrate with an example, consider this grammar s goes to a s s, s goes to b. This is also, this will simulate a leftmost derivation so you have s goes to a s s, a b s, a b this s is replaced by a s s, a b a this is replaced by b, a b a b b.

Now, how this will be simulated by the automaton, what will be the mapping here? The mappings will be delta of q a s contains q s s delta of q for this you the mapping is this, for this the mapping is delta of q b s contains q epsilon, so let us see how this will be accepted a b a b b. So all state is always q and there are no epsilon moves, only true input moves are there see here always an input is consumed and stack contains only non terminals. So, you have s on the stack so when you see a s with s on, when you see a a with s on the top what happens? s is replaced by s s the input pointer will now be moved here. Now, you see a b with a s on the top, so what do you do? Pop it, read the symbol, pop it. Now, we see a s on the top and you are reading a a, so what happens it will be replaced by s s and this input will be consumed.

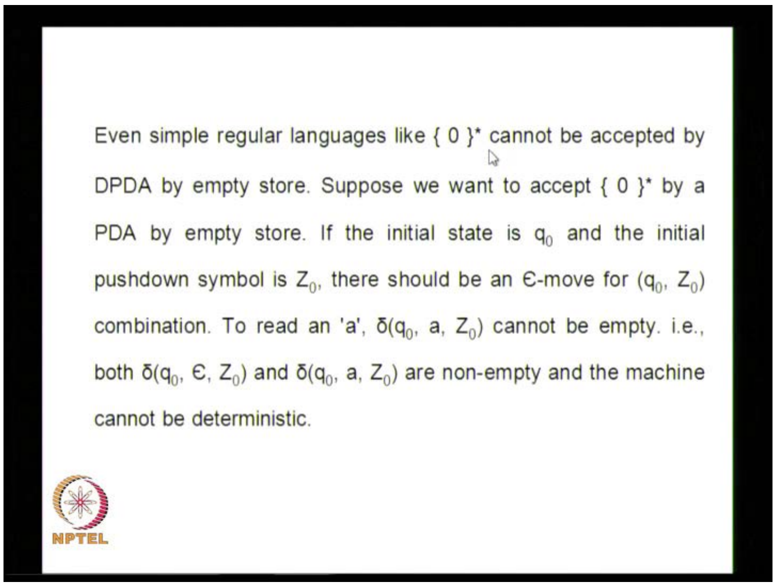
Now, you have a s and b combination. So, you remove this n s and that b combination, you remove this at the end the stack is emptied and so on. This is a d PDA isn’t it, this one is a deterministic automaton. So, you see that this proof also you have to, formal proof you have to use induction on the number of steps so of the in the derivation. Here again a leftmost derivation is simulated.

Now, I have one question you can think over this. We have showed the equivalence between acceptance by empty store and acceptance by final state, that construction was very clear if from initial configuration of one you go to the initial configuration and either simulate and finally, either erase or go to a final state. Remember, the proof? Can you do that for d PDA? We have construct non deterministic pushdown automaton and prove that. For deterministic pushdown automata is acceptance by empty store and acceptance by final state equivalent, think over this and in the next class you tell me. I will give you a hint, can any regular set be accepted by a d PDA.

(( )).

Because, you have to have the storage you do not manipulate the pushdown store at all but, that is by defining a final state. Can you accept a star or a power n, n greater than or equal to 0 or a power n, n greater ,a star regular expression a star is a power n, n greater than or equal to 0. Can you accept a star by a deterministic pushdown automata by empty store. We accept it by a deterministic automata pushdown automata by final state definitely, any regular set you can accept by a d PDA with final state.

(Refer Slide Time: 49:10)


Even simple regular languages like { 0 }* cannot be accepted by DPDA by empty store. Suppose we want to accept { 0 }* by a PDA by empty store. If the initial state is q_0 and the initial pushdown symbol is Z_0, there should be an ϵ-move for (q_0, Z_0) combination. To read an 'a', $\delta(q_0, a, Z_0)$ cannot be empty. i.e., both $\delta(q_0, \epsilon, Z_0)$ and $\delta(q_0, a, Z_0)$ are non-empty and the machine cannot be deterministic.

So, think over this even simple regular language is like 0 star cannot be accepted by deterministic pushdown automata by empty store. Suppose, we want to accept 0 star by a pushdown automaton by empty store, if the initial state is q naught and the initial pushdown symbol is Z naught then to accept epsilon, there must be an epsilon move for the q naught, Z naught combination. Now, to read a a you must have a move for q naught a Z naught, so this cannot be empty you will have a move for q naught a Z naught and to accept epsilon you have a move for q naught epsilon Z naught also.

So, both of them are not empty. So, q naught epsilon Z naught and q naught a naught Z naught both are non empty and if this happens the mission is non deterministic. So, we cannot accept 0 star by a deterministic pushdown automaton by empty store.