Theory of Computation

Prof. Kamala Krithivasan

Department of Computer Science and Engineering

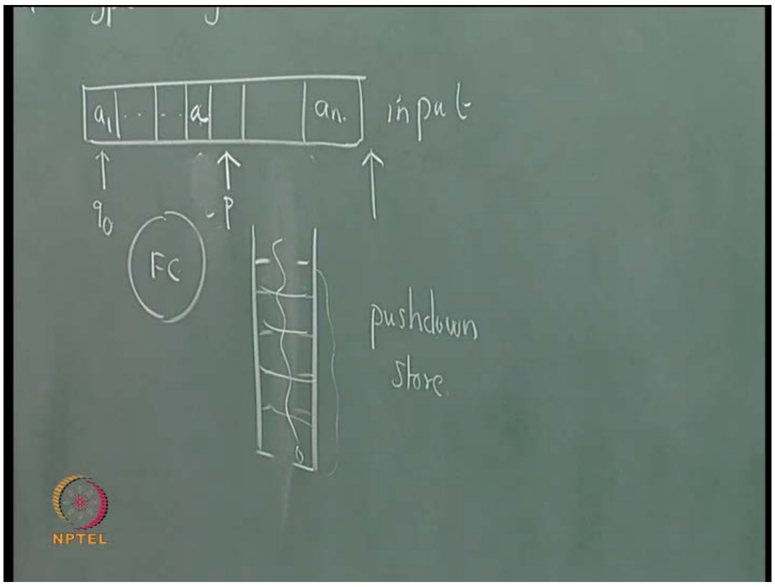
Indian Institute of Technology, Madras

Lecture No. # 20

Pushdown Automata

So, today we shall consider pushdown automata. We have seen that the smallest machine or accepting device you can think of is the finite state automaton, and it correspondent to type three grammars.

(Refer Slide Time: 00:25)



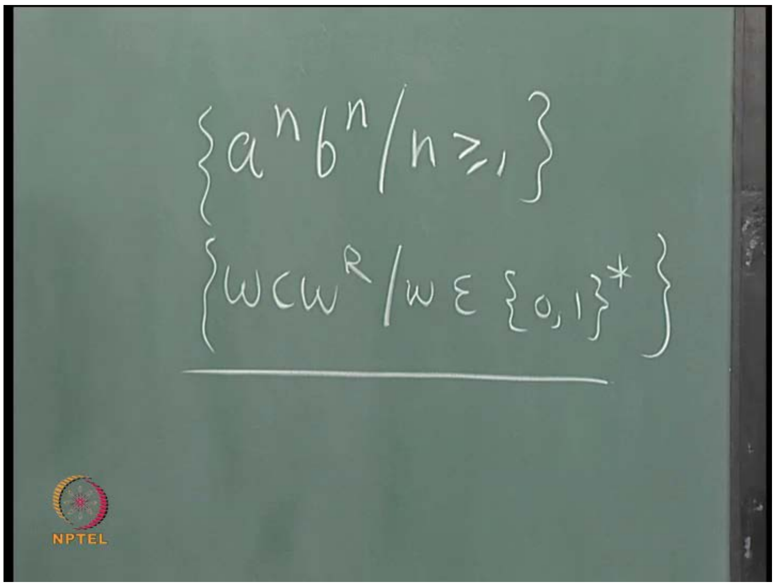
Now, what about context free grammars or type two grammars? The machine which corresponds to this or the accepting device corresponding to context free grammars is the pushdown automaton. The pushdown automaton has an input tape, it has a finite control. Finite control means state, and a stack or a pushdown store; it is a pushdown store, it is like a stack.

Initially when you start the input is given here. The stack has only one symbol z naught which is called the initial pushdown symbol, and initial state is q naught. So, the machine will be reading this first symbol leftmost symbol in state q naught, and the topmost symbol is z naught, that is only one symbol initially. After sometime you may have something on the stack say for example, it is reading this symbol in some state q. In some state q, it is reading a symbol z and at the time the stack is having some symbols; the topmost symbol is z. So, the next move depends on q a and z, depending on all the three in the case of finite state automaton, knowing q and a the next move is determined.

Now, in the case of pushdown automaton, knowing q a and z, the next state is determined. So, the mapping will be depending upon the state, the symbol read and the topmost stack symbol or the pushdown symbol. Next instance the input pointer can move, it can go to a different state and z, instead of z you can write something on the stack, a string on the stack you can even remove z. When is the input accepted by the automaton, the input is accepted when the whole of the input is read and the pushdown store is empty this is one way of acceptance. This is called acceptance by empty store.

There is another way of acceptance, which is after reading the whole input the stack remains, some portion of the stack can still remain. But the state is a final state, somehow the states will be designated as final state and the machine will reach a final state.

(Refer Slide Time: 03:59)

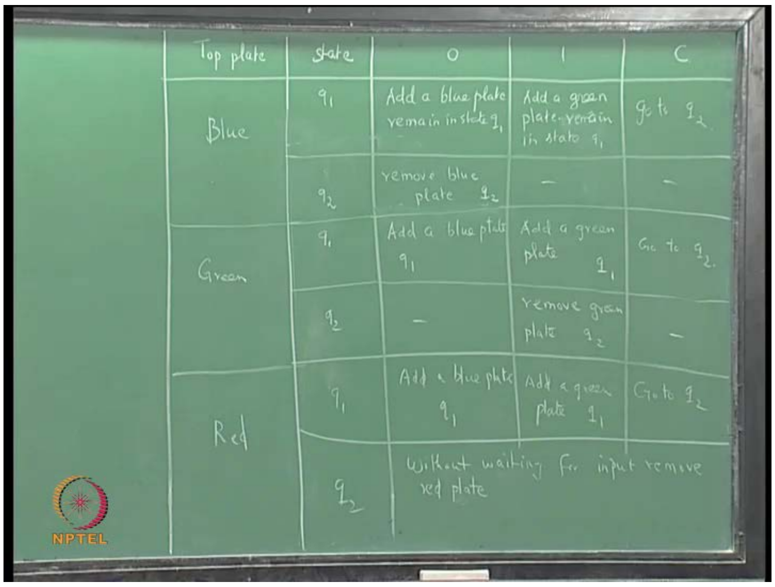


$$\{a^n b^n / n \geq 1\}$$
$$\{w c w^R / w \in \{0,1\}^*\}$$

We have already seen that, languages like a power n b power n, n greater than or equal to 1, w c w R. (No Audio Time: 04:10 to 04:18) They are context free languages, which are not accepted by finite state automaton. Why finite state automaton cannot accept it? You can use either Myhill Nerode theorem or pumping lemma to prove that but intuition is finite state automaton has a finite amount of memory. Please, do not say that finite state automaton does not have memory, it has memory but the memory is finite. Finite state automaton has finite amount of memory and within that it is not able to accept that.

Now, let us see how this language is accepted by the pushdown automata. Actually, we can look at this, as a some sort of a spring over which the plates are all placed in a cafeteria and you can only see the top plate. We can put more plates on that or remove the top plate and so on. The pushdown store, it is the stack and it acts like that, usually when you consider parsing, you call it as a stack here, we use the more common word is pushdown store. It means the stack pushdown store really means a stack only.

(Refer Slide Time: 05:41)

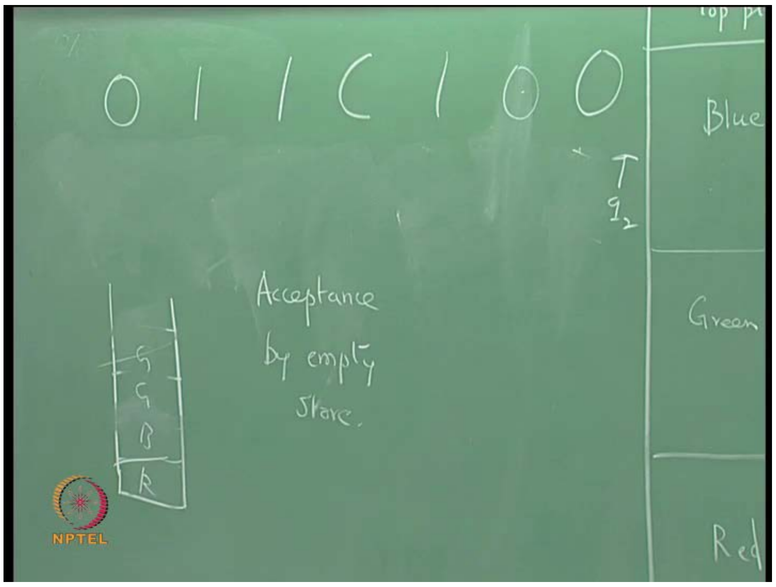


Top plate	state	0	1	C
Blue	q_1	Add a blue plate remain in state q_1	Add a green plate remain in state q_1	go to q_2
	q_2	remove blue plate q_2	–	–
Green	q_1	Add a blue plate q_1	Add a green plate q_1	Go to q_2.
	q_2	–	remove green plate q_2	–
Red	q_1	Add a blue plate q_1	Add a green plate q_1	Go to q_2
	q_2	without waiting for input remove red plate		

So, the mappings can be defined in this manner, the automaton has two states q 1 and q 2, the input symbols can be 0 1 and C. And you can have three symbols on the pushdown store can have b or a g or a red, blue plate, red plate and green plate. So, and the action is when the top plate is blue and the state is q 1, what is the action if you see a 0? Add a blue plate remains in state q 1, when the top plate is blue and a state is q 1 and it is 1. Add a green plate remain in state q 1, when the top plate is blue and you get a q 2 you are in state q 2 and you get a 0, remove the blue plate and remain in q 2, that is what I have written q 2. And if you are in state q 1 top plate is blue and you get the input C, you go to q 2, state q 2 without manipulating the stack.

Similarly for green, if you are in state q 1 that is the top plate is green, if you are in state q 1 and see a 0, add a blue plate if you see a 1 add a green plate but you remain in the same state q 1 and if you see a C go to state q 2. If the top plate is green and if you are in state q 2, you are not suppose to get a 0, if you get a 1 remove the green plate and remain in state q 2. If you have a red plate on the top, if you are in q 1, if you see a 0 add a blue plate, if you see a 1 add a green plate, if you see a C go to state q 2.

(Refer Slide Time: 08:04)



So, the action is given in this table. Now, let us see how it works on a input, on a particular input 0 1 1 C 1 1 0. This should be accepted by the machine, the initial state is q 1. And we are going to consider acceptance by empty store, the formal definition I will give in a moment.

Now suppose, I start with, I will mark the state just to below this, you are reading this in q 1 and initial tree in the pushdown store you are having a red plate. This is what with you start with initial pushdown symbol is red, you are having a red plate on the stack. Now, what is the action when you are in state q 1 and the top plate is red and you see here 0, what is the action? Add a blue plate remain in state q 1. So, you add a blue plate, you are remaining in state q 1 but you are moving the input pointer, you are reading this means you have read a 0. So, input pointer will be moved and a state is still q 1.

Now, in q 1 you see a 1 and the top plate is blue, state is q 1 you are seeing a 1. What is the action taken? Add a green plate remain in state q 1. So, you add a green plate you read this. So, you move the input pointer to the next 1 and this will be the situation. Now, the same situation you have green plate on the top, you have q 1 on this state, then 1 is being red. Green 1 q 1, what is the action taken? Add a green plate remain in q 1. So, you add a green plate input is red you move here.

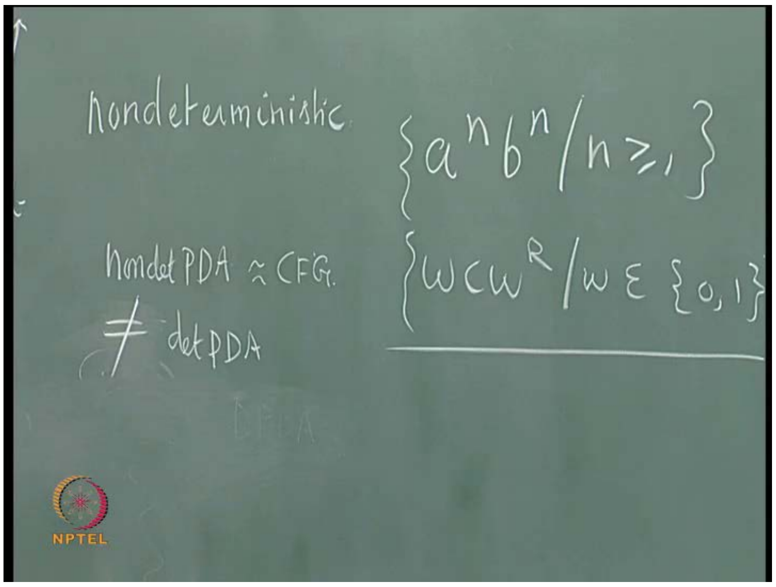
Now, in q 1 when you read a C, the action is go to q 2 you are not manipulating the stack but just change the state to q 2. So, you go here in state q 2, in q 2 either you can have a g and a 1 combination, green and a 1 combination or a blue and a 0 combination. The other way combination you should not have a blue and a 1 combination, green and the 0 combination. If you it gets such a combination it will halt, machine will halt. So, when you have a blue and a 1 combination, I am sorry blue and a 0 combination you remove the blue plate. If you have a green and a 1 combination remove the green plate.

So, what you do is, when you have a g and a 1 combination remove the green plate, go here in state q 2. Again you are having a green and a 1 combination in state q 2. So, you remove it, remove the green plate. Now, you are having a blue and a 0 combination remove the blue plate, this is removed. Now, the whole input has been red now and you are in state q 2. There is one symbol remaining on the stack that is red symbol. Now, the action is red and q 2 without waiting for a, the input remove the red plate.

So, you need not have to read anything, just without waiting for the input you remove this. So, at the end of reading the input, the stack has been empty. So, this string will be accepted by the machine. Now, we are defining acceptance by empty store, this is acceptance by empty store. Suppose, instead of this I had say this, then I would have added after having z blue green green. This green would have been removed but then there will be a 0 and a green combination, the machine will halt without reading the input. That green and blue green and 0 mapping is not defined a blue and 1 the mapping is not defined.

So, if such a thing happens the machine will halt abruptly, it can accept only after retreats the whole of the input. In between it can read epsilon also, that is also allowed in the case of the pushdown automaton. That is it can manipulate the stack change the state or something like that without reading a symbol. Without moving the input pointer it can also manipulate on the stack that is allowed. And basically the pushdown automaton is nondeterministic in nature. (No Audio Time: 14:29 to 14:37)

(Refer Slide Time: 14:26)


nondeterministic

$\{a^n b^n / n \geq 1\}$

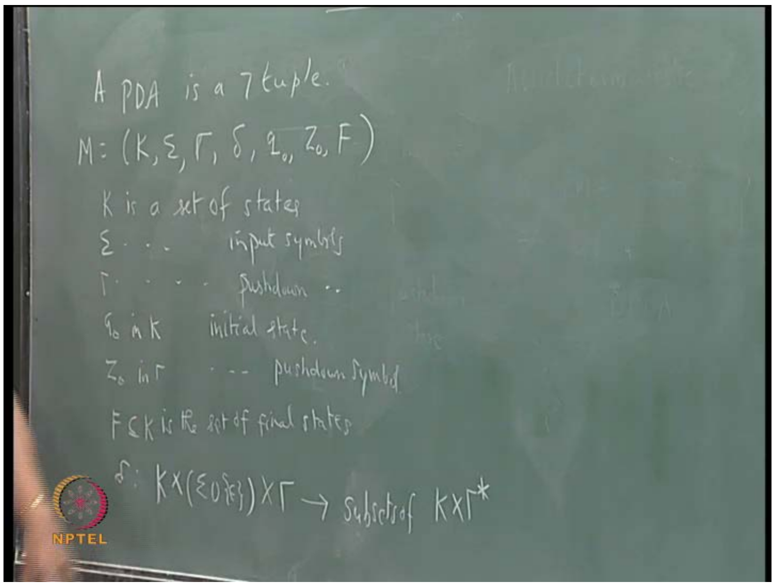
nondet PDA $\approx$ CFG.

$\neq$ det PDA

$\{wcw^R / w \in \{0,1\}\}$

The next move is not uniquely defined, nondeterministic pushdown automata (No Audio Time: 14:44 to 14:52) or equivalent to CFG or nondeterministic pushdown automata is the machine characterization for context free languages. And deterministic pushdown automata or you generally write it as d PDA, these are not equal, they are not equal, nondeterministic PDA is not equivalent to deterministic PDA. Deterministic PDA form a sub class. So, we shall that in a moment.

(Refer Slide Time: 15:56)



A PDA is a 7 tuple.

$M: (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

K is a set of states

Σ . . . input symbols

Γ . . . . pushdown . .

q_0 in K initial state.

Z_0 in Γ . . . pushdown symbol

$F \subseteq K$ is the set of final states

$\delta: K \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow$ subsets of $K \times \Gamma^*$

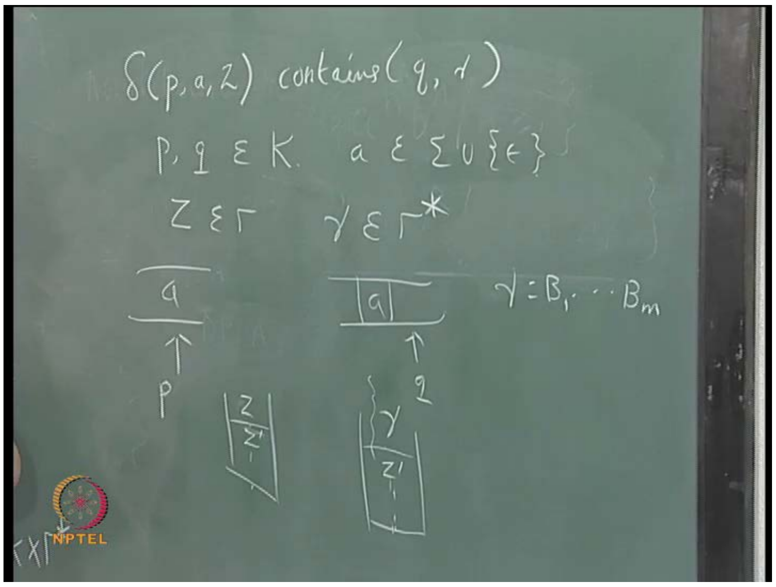
First let us define formally, what is a pushdown automata? (No Audio Time: 15:43 to 15:56) A pushdown automaton or a PDA is a 7 tuple, M is equal to K sigma gamma delta q naught z naught F, the 7 tuple like this.

Where K is a set of states, sigma is a set of input symbols, gamma is the set of stack symbols or pushdown symbols, q naught in k is the initial state, z naught in gamma is the initial pushdown symbol. F contained in K is the set of final states, when you define acceptance by empty stores F can be empty, you need not have to bother about final state. If you define acceptance by final state, you have to specify what is F.

Now, delta is the mapping, what is delta? Delta is the transition mapping. What does it depend on? It depends on the state the symbol red and the top of the pushdown store. So, this K into sigma union epsilon because epsilon moves are allowed you can also not read any input that is also allowed. So, K into sigma into gamma state, input symbol are nothing red and the topmost pushdown symbol this into, it is basically nondeterministic 2. I will write 2 power subsets of or I can say subsets of k into gamma star.

If it reads a input symbol, input pointer will be moved, if it reads epsilon input pointer will not be moved, from one state it will go to another state. And rewrite the topmost pushdown symbol by a string.

(Refer Slide Time: 19:09)



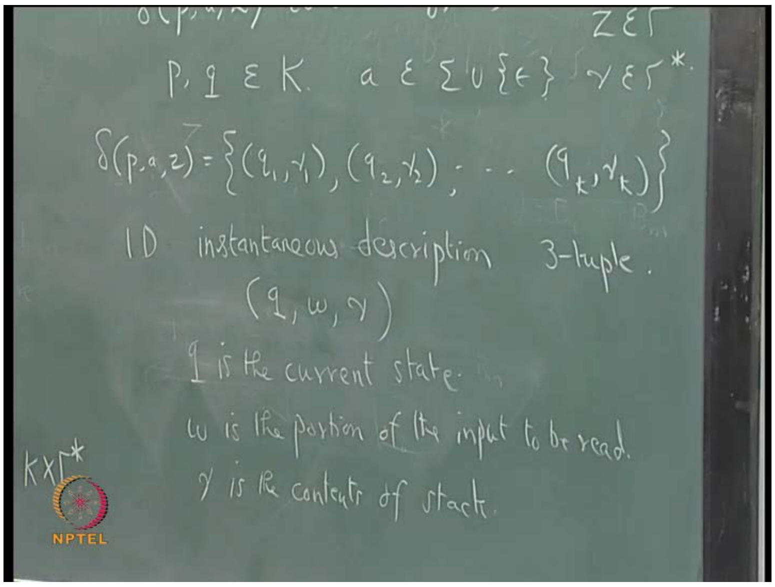
$\delta(p, a, Z)$ contains (q, γ)
$p, q \in K$ $a \in \Sigma \cup \{\epsilon\}$
$Z \in \Gamma$ $\gamma \in \Gamma^*$
$\gamma : B_1 \cdots B_m$

So, the mappings are like this delta of p a z contains q gamma, where p and q belong to K, a is a belongs to sigma union epsilon. And z belongs to gamma, gamma belongs to gamma star. Please, note that we write it as delta p a is that contains give gamma not equal to. If it is deterministic PDA you can write is as equal to, what it means is when you are reading a in state p. And the topmost pushdown symbol is z one possibility is next instance you can go to q, move the input pointer and some I will write it as z dash up to z dash it will be there, z will be replaced by gamma, gamma is a string, it is not a single symbol it is a string really.

Now, notation wise you have to be very careful, gamma is a string it can be something like gamma is B 1 B 2 B m say, then when you write like this is B m the top or B 1 in the top, different books followed different notation, if you take Hopcroft Ullman. He takes B 1 as the top, it will be written as B 1 B 2 B m leftmost symbol will become the top of the stack. If you take Michael Harrison's book (( )) organs book, it will be the other way around they would have taken B m as the. Generally the leftmost symbol is the top of the stack, that is the general convention but of course, some authors follow a different notation. So, if you take a new book and read, you must be careful which notation they are following.

(No Audio Time: 21:30 to 21:41)

(Refer Slide Time: 21:44)



$Z \in \Gamma$

$p, q \in K \quad a \in \Sigma \cup \{\epsilon\} \quad \gamma \in \Gamma^*$.

$\delta(p, a, z) = \{(q_1, \gamma_1), (q_2, \gamma_2), \ldots (q_k, \gamma_k)\}$

ID instantaneous description 3-tuple.

(q, w, γ)

q is the current state.

w is the portion of the input to be read.

γ is the contents of stack.

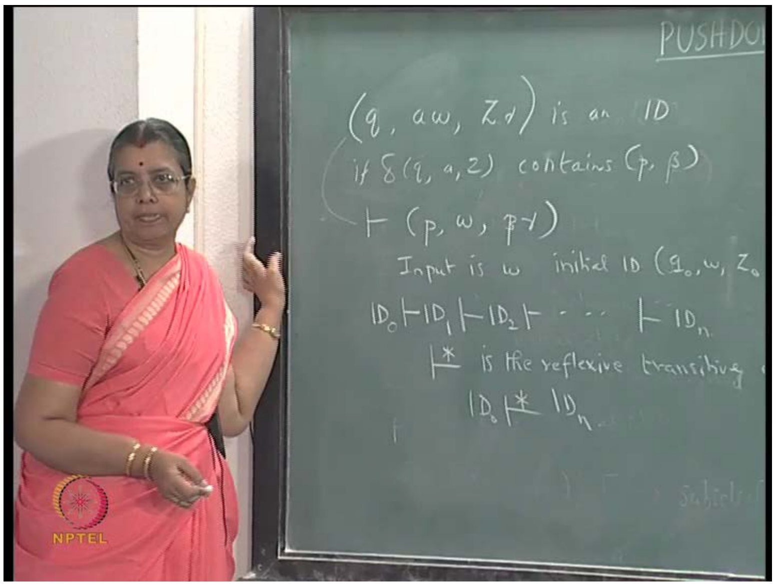
$K \times \Gamma^*$

So, z belongs to gamma and gamma belongs to gamma star. So, generally you write like this delta of p a z is equal to q 1 gamma 1, q 2 gamma 2 some q K gamma K. That is when you are in state p reading the symbol a and the topmost pushdown symbol is z, you have a choice of going to q 1 read a. And rewrite z by gamma 1 or you have the choice of going to q 2 and rewrite z by gamma 2 etcetera, you have a finite number of choices. The basic machine is nondeterministic in nature but the example which we consider just now, it is deterministic in nature.

Now, how do you define a move of the machine in a formal manner? An ID or instantaneous description, it is a 3 tuple (No Audio Time 23:10 to 23:18) it is a 3 tuple q w gamma, where q is the current state, w is the portion of the input to be read, remaining portion of the input gamma is the contains of stack or pushdown store. Then convention is, the leftmost symbol of gamma is the top of the stack that is what we will follow.

(No Audio Time: 24:25 to 24:41)

(Refer Slide Time: 24:44)

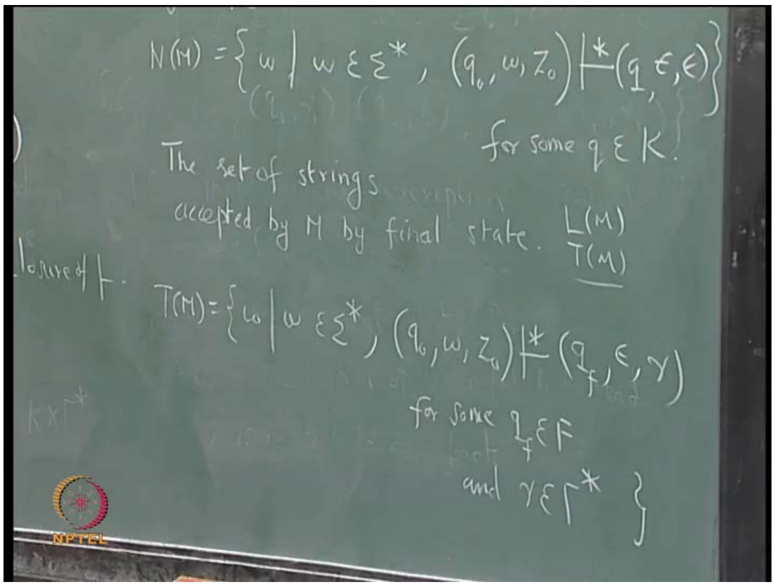

PUSHDO
$(q, aw, Z\gamma)$ is an ID
if $\delta(q, a, Z)$ contains (p, β)
$\vdash (p, w, \beta\gamma)$
Input is w initial ID $(q_0, w, Z_0$
$ID_0 \vdash ID_1 \vdash ID_2 \vdash \cdots \vdash ID_n$
$\vdash^*$ is the reflexive transitive
$ID_0 \vdash^* ID_n$
NPTEL

So, suppose q aw Z gamma is an ID, that is the machine is in state q and a portion of the input remaining is aw Z is the top of the pushdown store under that you have gamma. And if delta of q a z contains p beta say, there is a mapping like this, then from this ID what ID you will go to? You write this symbol, if you go from this ID to the next ID you use this symbol. So, state will change to p, a will be read. So, w will be remaining, z is replaced by beta. So, you will have beta gamma, the leftmost symbol of beta is the top of the stack.

So, from one ID in one move you can go to this ID if this is the mapping, define in the transition function. So, state what will be the initial ID? If the input is w say, then what will be the initial ID? Initial ID will be q naught w z naught, you are in the initial state, the whole of the input has to be read and z naught is the only symbol on the stack. Now, when do you say that a string is accepted, starting from the initial configuration, you have to read the whole input and the stack can be empty that is one way of acceptance. Another way of acceptance is start from the initial ID whole input has to be read and you have to reach a final state.

Now, before that, what does that this represent? You are an ID naught is a initial ID, then one move you can go to ID 1 ID 2 and so on finally ID n you can go to. So, what does this represent, this symbol? It is a relation between ID's, this represents a relation between ID's is not it. So, generally if this is the relation, what does this represent? Reflexive transitive closure off, so you say ID from ID in many moves if you go from an one move you write this. If you go from this to the other one in many moves you write like this.

(Refer Slide Time: 28:37)



$N(M) = \{ w \mid w \in \Sigma^*, (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)$ for some $q \in K \}$

losure of $\vdash$.

The set of strings accepted by M by final state. $L(M)$ $\underline{T(M)}$

$T(M) = \{ w \mid w \in \Sigma^*, (q_0, w, Z_0) \vdash^* (q_f, \epsilon, \gamma)$ for some $q_f \in F$ and $\gamma \in \Gamma^* \}$

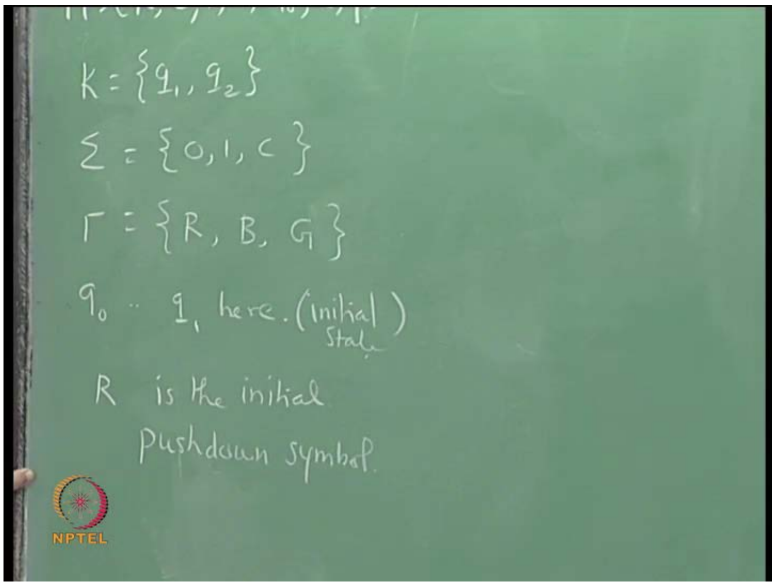
$K \times \Gamma^*$

Now, how do you define acceptance? The set of strings accepted by M by empty store, that is denoted as N of M null M or you sometimes it is written as null of M. N of M it is the strings of the form w, where w belongs to sigma star, it is a input string, what is the initial ID? They such that q naught w z naught this is the initial ID. From this you have to go to an ID, where the whole input has been read and the pushdown stores has been emptied. That is from this you have to go to an ID, it can be in any state, when you define acceptance by empty store, you are not bothered about the final state any state, can be in any state. But the whole input has to be read and whole pushdown store has to be emptied. For some q belonging to K this is epsilon.

You also define acceptance by final state. The set of strings (No Audio Time 30:36 to 30:43) accepted by M by final state sometimes it is denoted as L of M or T of M I will use the language M or T M, T of M w, w belongs to sigma star and from the initial ID, you have to go to an ID, you have to go to a configuration, where you must reach a final state. And whole of the input has been read is not it has to be read epsilon but stack can contain something now need not be emptied. For some q f belonging to F and gamma belonging to gamma star. (No Audio Time: 32:11 to 32:17)

This is how you define acceptance by final state and acceptance by empty store, when you specify acceptance by empty store, the last component of the 7 tuple can be phi empty need not have to specify the final state. Even though we define two modes of acceptance, we will show the equivalence of them that is acceptance by empty store is equivalent to acceptance by final state; we will show the equivalence in the next class.

(Refer Slide Time: 33:06)



$K = \{q_1, q_2\}$

$\Sigma = \{0, 1, c\}$

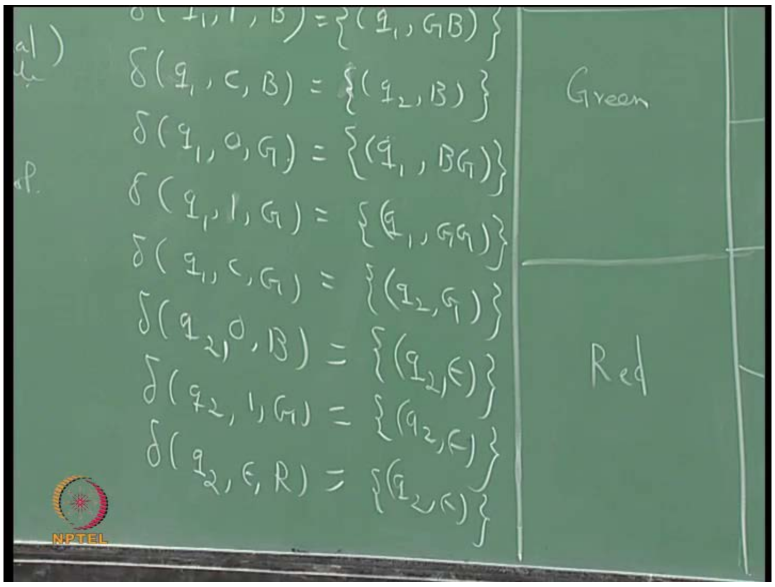
$\Gamma = \{R, B, G\}$

q_0 : q_1 here. (initial state)

R is the initial pushdown symbol.

But, let us come back to this example. (No Audio Time: 32:54 to 33:04) Now, if you denote M as k sigma gamma delta q naught z naught F in this case it is acceptance by empty store. So, this you can write as phi what is k? In the example which you considered there are two states q 1 and q 2 and the input symbols are 0 1 C. The pushdown symbols are I will write them as R B G, we consider red plate, blue plate, green plate so on. And q naught is the initial state that is q 1 here, initial state, this is the initial state, R is the initial pushdown symbol. (No Audio Time: 34:10 to 34:20)

(Refer Slide Time: 34:30)



$\delta(q_1, 1, B) = \{(q_1, GB)\}$

$\delta(q_1, c, B) = \{(q_2, B)\}$

$\delta(q_1, 0, G) = \{(q_1, BG)\}$

$\delta(q_1, 1, G) = \{(q_1, GG)\}$

$\delta(q_1, c, G) = \{(q_2, G)\}$

$\delta(q_2, 0, B) = \{(q_2, \epsilon)\}$

$\delta(q_2, 1, G) = \{(q_2, \epsilon)\}$

$\delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\}$

Now, you have to write the mappings, delta mappings, how do you write? Delta of q 1 first with a red we will write, q 1 0 R, when you see a 0 with a red plate on the stack what do you do? Add a blue plate and remain in q 1. So, this will be q 1, add a blue plate means what will you have B or BR? BR because R instead of R you must write the thing. So, you are adding means that R is also there, q 1 BR instead if you see a 1, q 1 1 R you will be in q 1 but you are adding a green plate GR.

Now, note that because it is deterministic I have written like that, actually it is a set. So, I should write with, write it as a set this example is deterministic. So, there is only one possible next move but generally it will not be like that. And delta of q 1 C R is if you see a C go to q 2 is not it. So, it will be q 2 but the stack remains as it is. So, it is q 2 R.

Now, take the blue plate delta of q 1 0 B, when you are having a blue plate and you are in q 1 and you are seeing a 0 what do you do? Add a blue plate. So, what will be the mapping you will be in q 1 only, when you are adding a blue plate what you have to write B is already there. So, B or B B tell me, what you should write? B B, B B right, when you are having a blue plate. But you are reading a 1; you should add a green plate that is q 1 GB. Note that this is left side is the top of the stack, when you are adding g means, you are writing it as a GB not as BG.

And delta of q 1 C B is you go to q 2, this is when blue plate is on the top, when green plate is on the top also you have to write like that, q 1 0 G when you see a 0 add a, see whenever you see a 0 you are adding a blue plate. Whenever you see a 1, you are adding a green plate that is why at the when you go to state q 2, you are removing the when the reverse order that is why you get W C W R.

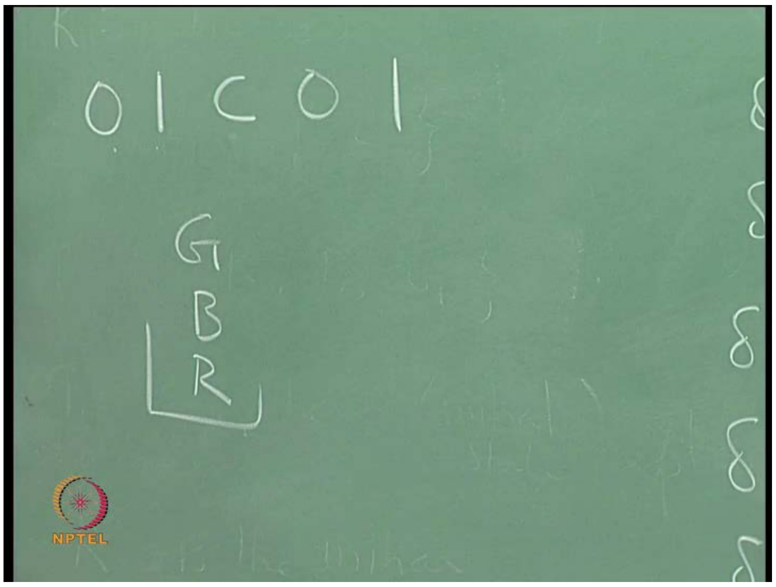
If you have a 0 and 1 a string whenever you see a 0, you are adding a blue plate, if you see a 1, you are adding a green plate. But when you remove them in the, you are removing in the other order. So, that is why the string which follows, see it is in the reverse of the string which is occurring before C, when you see a C you go to state q 2. And in state q 2 you start removing the symbols from the stack in the proper order, if the string is different from the reversal of the initial portion the machine will halt, is that clear? So, q 1 0 G that is q 1 B G, q 1 1 G is q 1 G G you are adding a green plate, when you see a 1, q 1 C G will be you will go to q 2. But you do not do anything with the stack that is why G remains as it is.

This is with q 1 with q 2 you have very few mappings, if you are in q 2 and see a 0 and a B combination. If you have a 0 and a B combination remove the blue plate, if you should not have a 1 and a B combination in that case, you are having a wrong thing and so you will halt. So, this is q 2 removing the plate means q 2 epsilon, you are removing the plate.

Similarly delta of q 2 1 G is you are removing the green plate. So, that is q 2 epsilon you must have a 1 and a G combination, if you get a 0 and a G combination, it is not correct machine will halt. Then the last is when you have finished with everything reading the whole input, what happens the red plate alone remains on the stack you have to remove the red plate.

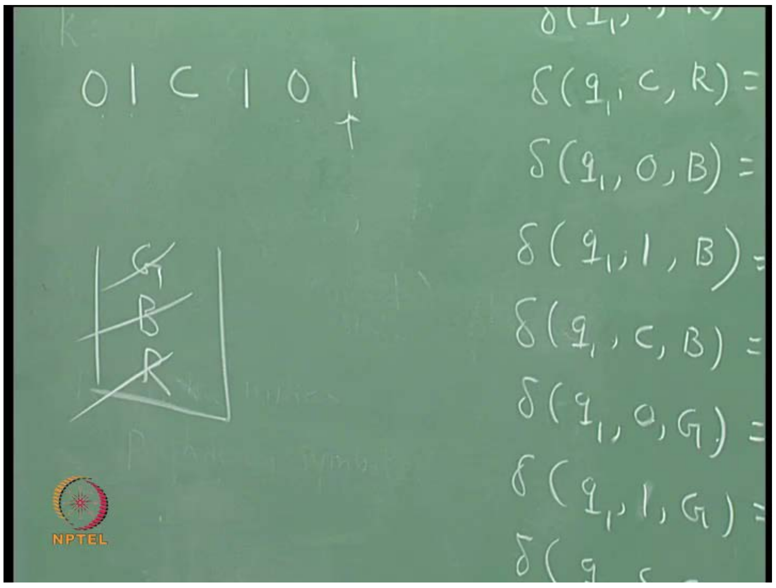
So, that move is specified by delta of q 2 epsilon R is q 2 epsilon. (No Audio Time: 41:28 to 41:34) So, let me see we have seen that, we had written the mappings how a string can be accepted, we have seen an example.

(Refer Slide Time: 41:52)



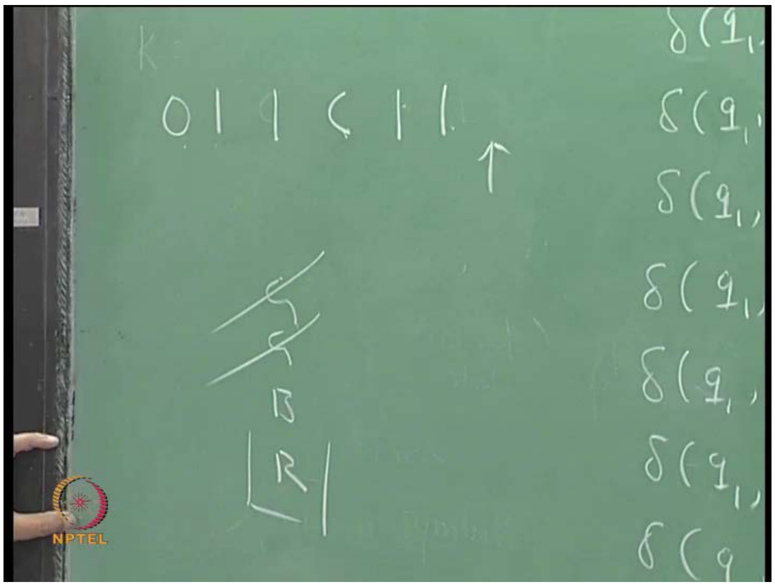
Suppose, I have 0 1 C 0 1 this is not the reversal of this. How will it reject? First you have a red plate when you see a 0 you add a blue plate, when you see a 1 you have a green plate, when you see a c from q 1 to q 2 you will go. But then you are having a 0 and a G combination which you should not get a machine will halt, it will not read the input further that is.

(Refer Slide Time: 42:25)



Suppose, I have something like this 1 0 1 the second portion is more. in that case what will happen, you have a red plate when you see a 0 add a blue plate, 1 green plate. When you see a c, change the state to q 2. Then when you see a 1 remove this, when you see a 0 remove this you are here, in q 2 you are having a red and a 1 combination. When you have q 2 and you are having red without looking for the input you will remove the red plate. So, you will remove this, then there is no way to read this one, you cannot read this one at all, the whole input cannot be read. The condition is the whole of the input has to be read.

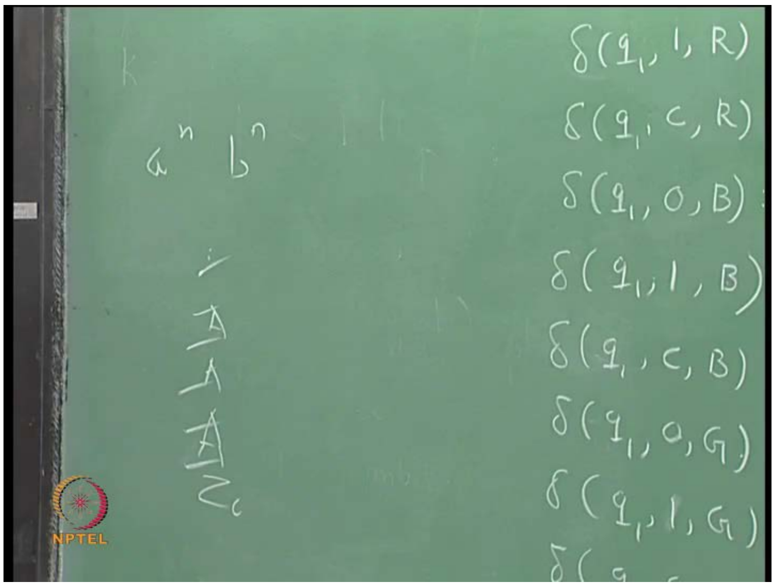
(Refer Slide Time: 43:23)



Now suppose, I have something like this, this is portion of the reversal is there but it is not complete length is shorter, second portion is shorter. What will happen here, you have a red plate, add a blue plate, add a green plate, add a green plate. When you see a c go to q 2, when you see a one remove this, when you see a one remove this, you are at the end of the input. But you are not able to empty the stack, stack still contains some, if you have a red you can remove but you cannot remove blue like that.

So, at the end you have read the whole input but you are not able to empty the stack. So, in this case the string will not be accepted. Similarly, you can construct pushdown automata, as a exercise I have to advise you to take some problems very simple one's like a power n, b power n.

(Refer Slide Time: 44:38)



What is the idea? How will you get a power n b power n accepted? Keep on adding some of the basics Z naught is there, keep on adding A A A. And then keep on removing them then you read the b, you will get the a power n b. That is the idea, basic idea is like that. Then a string having equal number of a(s) and b(s) how will you accept? See the context free language which had strings over a and b, which had add equal number of a(s) and equal number of b(s). How will you accept it? Try to write down the mappings and see. So, we shall continue in the next class.