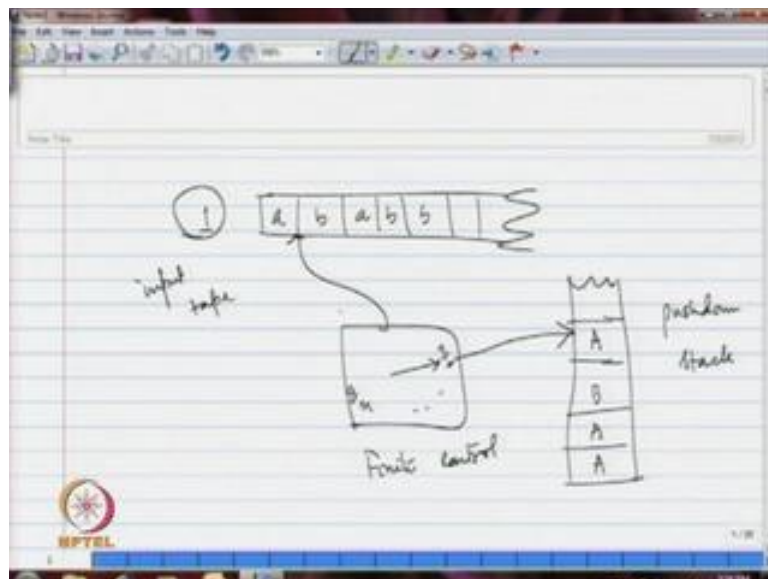


Formal Languages and Automata Theory
Prof. Dr. K.V. Krishna
Department of Mathematics
Indian Institute of Technology, Guwahati

Module - 10
Pushdown Automata
Lecture - 1
Pushdown Automata

So far whatever the concept that you have learnt related to finite automata. You understood the limitations of finite automata that they cannot accept context free languages. Because particularly we have observed that a context free language you know a power n b power n context free language that we have discussed that finite automata cannot accept that. Now, we will introduce a notion called pushdown automaton is this concept will use to accept context free languages. So, for that purpose we will introduce this.

(Refer Slide Time: 01:07)



So, in push down automaton because if you remember that we have finite control; it is a pointer pointing to states say some and reading and writing head and tape cells. So, we are giving the input on this say for example, a b a and b b etcetera for example. Now, the transition whatever the transition function defined for this finite control that takes care of this input. So, corresponding to whatever is there the next state it will the pointer will move to that and so on. So, this reading and writing that reading head will move towards

right after in each transition one by the end of the tape that mean by end of the input depending on the state; if it is final state or non-final state. We say the string is accepted that is how we have considered finite automata.

Now, with this mechanism we have limitation that we cannot accept the language of the form $a^n b^n$ or some other context free languages, which are not regular. We have observed that this kind of mechanism can accept regular languages. Further, we have talked about some finite automata two way finite automata also we have talked about. There the reading the reading head can move left and right. So, the two way finite automata when we have discussed even by giving that we are unable to you know improve the power of finite automata it access precisely regular languages only.

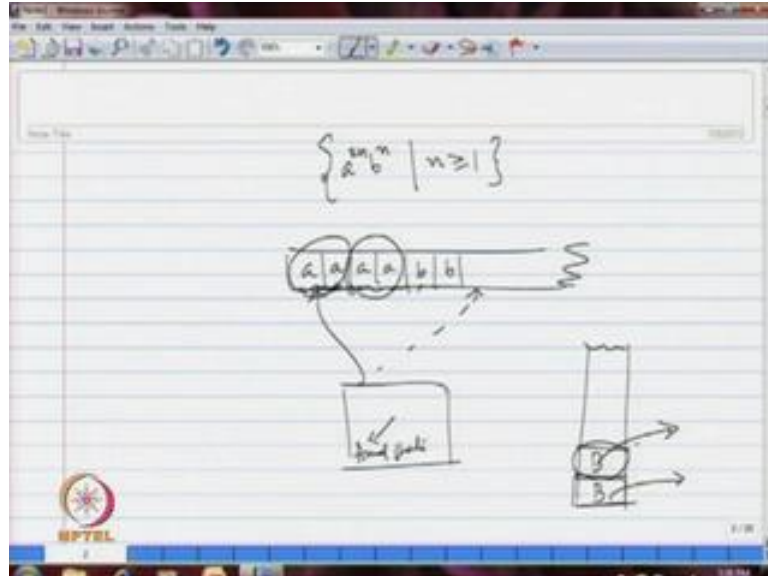
Now, what do we do? We give some more facility to this system this automaton and define pushdown automata and there we consider context free languages. So, for that purpose what we give another tape let me write like this; why do I write like this I will mention. So, divide into cells like this and here a reading and writing heads out of here we give and here the mechanism is the symbols that you write in this say naturally you may write symbols. Let me just distinguish here say capital A capital A capital B like that you keep writing in this and you will be have access to the top most symbol only.

So, you will read the top most symbol or next to that next to that and so on. So, that means, say for example, you have written this a b b a. I will be able to read a then b then a like that so; that means, this is essentially stack principle. So, this tape we will consider it as stack and the rest of the mechanism is similar. That means, you know in the read this is a reading head on the first tape; this is the input tape you may call input tape and this I call pushdown stack; this is the finite control the finite control. Now, the reading head reads one after other symbol and according to the stack and the input and what are the transition defined here the next state it goes to the next state and keep on reading the input and write something to the stack or read something to the from the stack and so on.

So, the mechanism is essentially with respect to this phenomenon and accordingly we will be defining the corresponding transition in pushdown automata. So, once again I reemphasize on this point here, in addition to the finite automaton what we have learnt here; we are giving one auxiliary memory called stack I called it a pushdown stack. So, in which you will be that we will be used sort of rough work; that means, what are the

symbols that you are reading that corresponding to that you can put something in the memory and accordingly you can read and match the things and read the input.

(Refer Slide Time: 05:41)



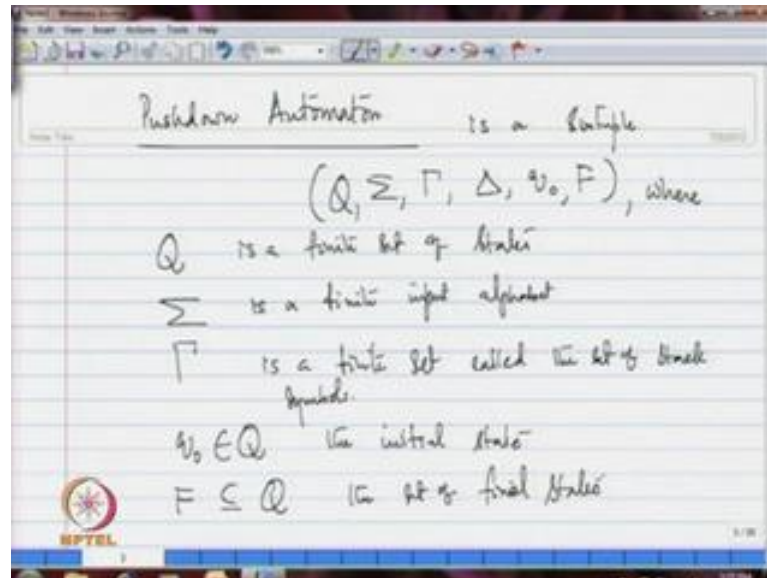
For example, if you take say a power 2 n b power n such that n greater than or equal to 1; if this this kind of language you know this is clearly context free, but not regular you can you can observe that this is not regular, in which case what is the expectation in the tape. For example, if you have say a a tape say a a a a and 2 b's; what do I do? I keep reading this symbols and corresponding to say if i read 2 2 a's then corresponding to that I may put say for example, some capital b and I keep on moving to right and next 2 a's. I will put another b now what do I do when I encounter b I remove this I pop this b from this stack.

So, in the beginning for corresponding to these 2 a's I put one I push 1 b and another 2 a's I push another b in the stack and when I get b when I read b here I will pop this b I read another b I will pop this b and when the stack is empty by the time. You finish input by the time this reading and writing had that reading had reaches input tape here and if the pointer is pointing to some final state, some final state then I say the string is accepted.

Now, you understand this stack concept how I will be using. So, similarly say for example, a power n b power n, if you consider. So, number of a's corresponding to number of a's you put some symbols inside that and you can match with that in rest of

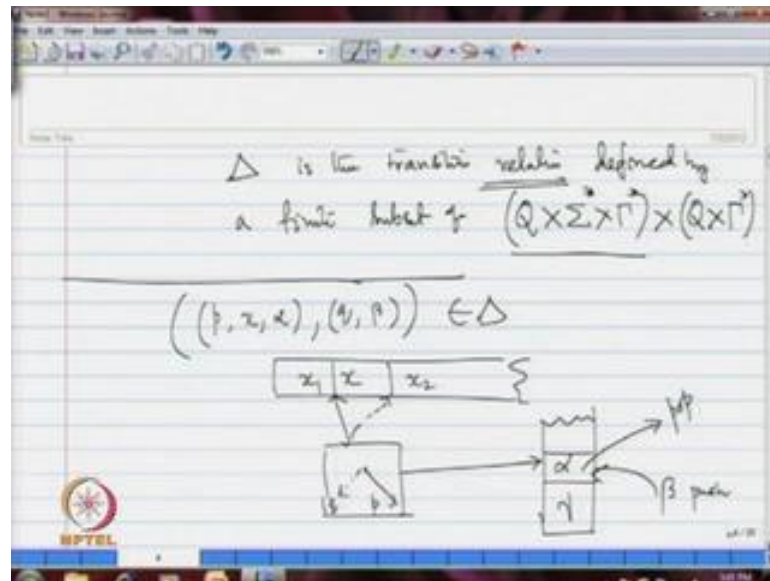
the half. Similarly, for palindromes or you know context free languages that you are familiar with for you will find this is a very convenient automaton a convenient device to understand the context free languages; I will give you better examples.

(Refer Slide Time: 07:42)



Now, this is this mechanism let me define formally the notion called push down automaton. I introduce that as a six tuple push down automaton is a six tuple I write it as $Q, \sigma, \gamma, \delta, q_0, F$, where Q as earlier finite set of states σ set of states, σ is a finite alphabet input alphabet. Now, this μ symbol γ that is a finite set again finite set called the set of stack symbols. So, whatever that I wanted to put in the stack; I will choose from this set and of course, as earlier q_0 the initial state, F is a set of final states the set of final states.

(Refer Slide Time: 09:37)



Now, what is delta? This delta I will give you little more general than whatever the example I have discussed. This is the transition relation I do not give it as a function non-determinism; I introduce here is the transition relation defined by the finite subset a finite subset of given a state and input string and a stack string, given these three what is the next state and what is that you want to put in the stack. So, that is an element of $Q \times \Sigma^* \times \Gamma^*$. We take this as a finite subset of this particular set; that means, given a state input string and a stack string we give what the corresponding next state and this I am giving you this as a relation is a transition relation.

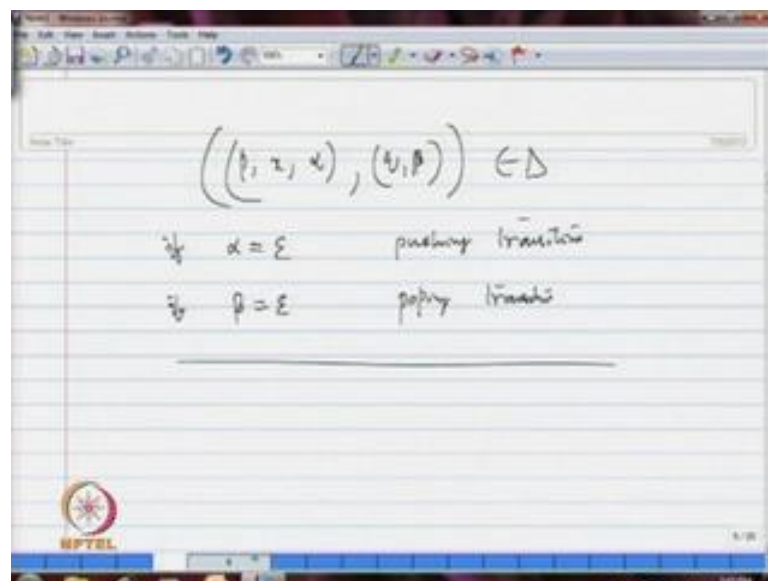
So; that means, I am now considering non-determinism so; that means, given a state and input string and a stack string corresponding to that I may assign finitely many possibilities as next state and stack strings that I would like put inside the stack. So, let me just elaborate what I mean here if say $p \times \alpha \rightarrow q \beta$ if this is an element in delta; that means, if this is a transition defined in a pushdown automaton. Then, the meaning of this is on the tape let me assume somewhere here you have x before that say some x_1 . After this string say for example, x_2 you have and in the finite control this is pointing to the state p and the reading head is above to read x ; I am just marking it like this and in this stack something is there, but I am able to access α on the top of the stack.

Something else may be there inside say for example, γ if this is the situation the top of the stack is α that may be α may be a combination symbols say string

essentially. If this is the transition corresponding to that the next the next picture is it will read x ; that means, it will move to the end of this x ; that means, you are about to read x 2 the rest of the string and your removing alpha from the stack and you are putting in this place beta in the stack; that means, in the stack now you have this gamma beta that is the stack string and in the remaining in the remaining input you have x 2.

Now, the state changes to the state q ; so this is what in the meaning once again if you state p x alpha corresponding to that given q beta in the transition. We complete x in the input from the current position and change to the state q corresponding to that you accessing top of the stack with as alpha as a string. Your remove that alpha is removed this need to be you know pop this alpha and push this beta into the stack. This is what is this is what is the meaning of that particular transition.

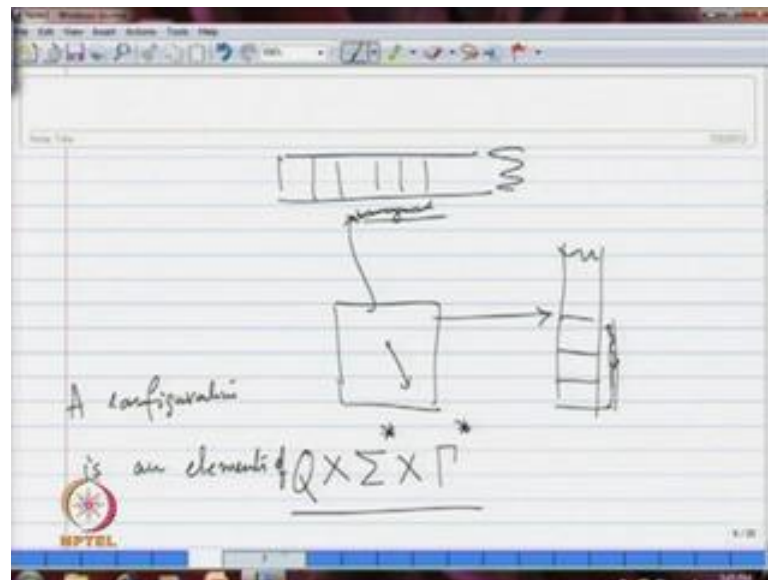
(Refer Slide Time: 13:37)



Now, if you consider a relation transition in a transition relation say for example, p x alpha is going to q beta say for example. If alpha is epsilon; that means what you are going to do when you are reading x in input in the state p without removing anything from the stack. You are pushing beta inside the stack; that means, this can be considered as this is pushing transition the pushing transition in case if beta is equal to epsilon. If beta is equal to epsilon you are not removing if alpha is epsilon; then you are pushing beta inside the stack if beta is equal to epsilon you are reading alpha from the stack and you are not pushing anything inside.

So, this is a popping transition. So, now, you notice that you notice that in a transition a transition can be a pushing transition or a transition can be a popping transition or in a transition in a stack you can push you can pop something and push something inside the stack. So, you can have this kind of mechanism.

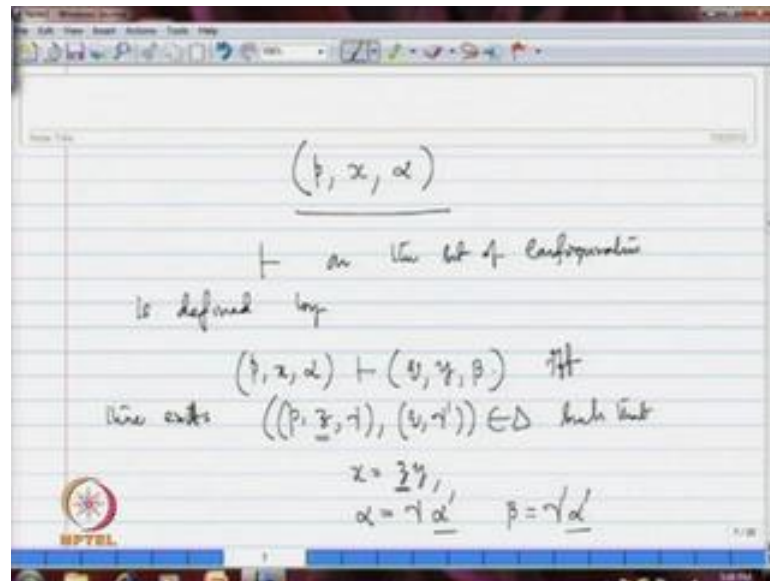
(Refer Slide Time: 15:05)



Now, to talk about computation in a pushdown automaton; suppose pushdown automaton it is given like this. A snapshot or a configuration as earlier we have we need to define, so what essentially the information that you require on the tape. What is the information in a pushdown automaton? Whatever the string that you are going to read in the tape that you require and what is there in the stack that is what you would require. So, this information you would require and what you are going to read that is required and the current state.

So, if you know these three things depending on the transition that you are applying you may. Here, in the tape this reading head is not going back and forth is not going left and right only it goes to right; if you read something after that you know what ever the remaining information it is going to read; the stack information you would require and the current state. That means, merely you would require this information and therefore, this kind of picture can be seen as an element of $Q \times \Sigma^* \times \Gamma^*$. So, that means a configuration is an element of a configuration is an element of this set.

(Refer Slide Time: 16:38)



That means, you have to give me the current state and what is the string that I am going that it is left over on the input tape and what is the string inside the stack; if you know this is a configuration corresponding to given snapshot or configuration we call. Now, for computation we would require to the computation relation that we define as earlier in case of finite automaton also. So, the binary relation on the set of configurations so; that means, we are defining on $Q \times \Sigma^* \times \Gamma^*$ is defined by this is a binary relation; if you consider the transitions transition between two configurations.

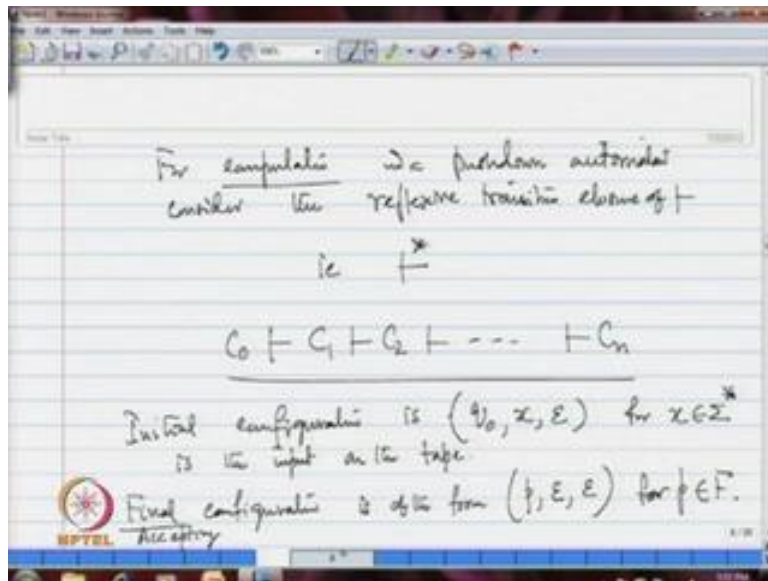
Say for example, if say $p x \alpha$ goes to say $q y \beta$; these are two configurations if and only if there exists there exists a transition. That is in the state p for example, some $z \gamma q \gamma \delta$ such that this x is equal $z y$ and what happens this the in the stack you when you have α ; you are making that β by removing γ from the stack; that means, this is of the form say $\gamma \alpha \delta$ and then β will be this $\gamma \delta \alpha \delta$. So; that means, what you are doing here this configuration $p x \alpha$ gives the configuration $q y \beta$;

That means, you should have a transition related transition in the transition relation; say $p z \gamma$ goes to $q \gamma \delta$ such that the x ; you are finishing the input z because z is shown here. In the stack you have α of the form say $\gamma \alpha \delta$. So, that you remove γ from the stack and you push $\gamma \delta$ inside the stack. That

means, the resultant string beta should be this gamma dash alpha dash because alpha dash is already in the stack.

So, that will be there in place of gamma you are pushing gamma dash. So, beta should be equal to gamma dash alpha dash if this happens we say this configuration is related to here. So, given two configurations there should exist a transition satisfying this condition; then we say these two configurations are related as earlier.

(Refer Slide Time: 19:59)



Now, for computation consider for computation in the pushdown automaton for computation in a push down automaton. Consider the reflexive transitive closure of this relation one step relation that is through this relation; the reflexive transitive closure of the one step relation will talk about the computation. So, computation in a computation in a pushdown automaton is of the form a finite sequence of configurations related by one step relations $c_0 c_1 c_2$ and so on; c_n for n greater than or equals to 0. This can be considered as a computation in a pushdown automaton. So, with respect to this binary relation what we have defined we talk about this computation.

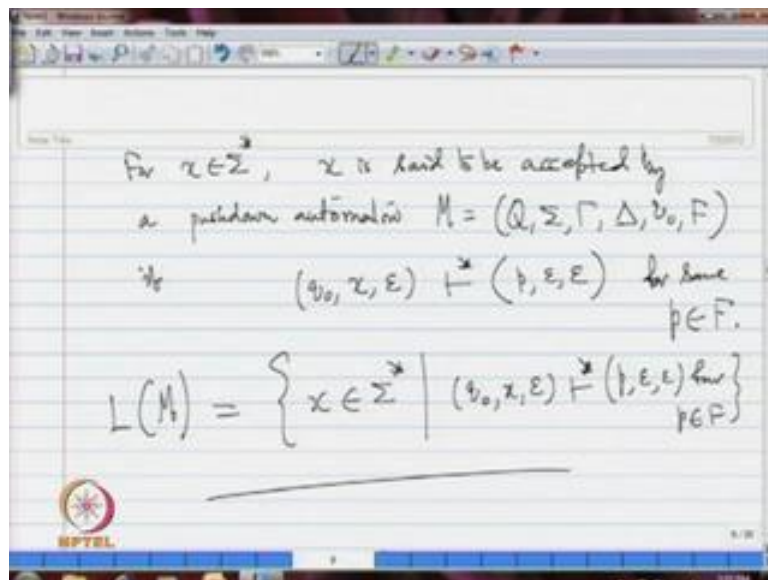
Now, what is initial configuration and when do you say a string is accepted by a pushdown automaton that motion I am going to define now. Because you have observed the mechanism a very first place when I have explained a power n a power $2n$ b power n this particular example; when I have talked about this particular example you realize that by the time you finish the input; you know stack we are making it empty and we are

expecting final state as to be there so. That means, you should not have anything to read on the input tape and you are starting with the empty stack and we are making the stack also empty.

When you are starting from initial state with this input by the time you finish the input stack. We are expecting it to be empty expecting it to be empty by and you should reach to final state. So, this is the expectation that I have made here in this example. So, let me consider initial configuration is of the form is the initial state q_0 and whatever is the input that we are taking x epsilon for x belongs to Σ^* this is given as this is entire string on the input tape for x belongs to Σ^* is the input on the tape and final configuration is of the form.

I mean, when you are accepting some final state you would require say for example, p and you should finish the tape that should not be anything and the stack should also reach to empty for some state p in F . So, this is an accepting configuration. In fact, accepting configuration is of this form.

(Refer Slide Time: 23:39)



Now, you can define for x belongs to Σ^* given input string epsilon x is said to be accepted by a pushdown automaton. A pushdown automaton as was writing earlier $M = (Q, \Sigma, \Gamma, \Delta, q_0, F)$; if the following happens; that means, in the initial state you give this as its input after finitely many steps; you should reach to a final state by

completing this input and stack should main I mean should come back to empty stack for some p in F.

Now, as usual as earlier the language accepted by pushdown automaton M is set of all x in sigma star x is accepted by n; that means, you should have this relation after finitely many steps you will get this for p in F for some p in F. So, the language accepted by a pushdown automaton as earlier the set of all those strings accepted by a pushdown automaton.

(Refer Slide Time: 25:38)

$$L_1 = \{a^n b^n \mid n \geq 1\}$$

$$\Delta = \left\{ \begin{aligned} &((q_0, a, \epsilon), (q_0, A)), \\ &((q_0, b, A), (q_1, \epsilon)), \\ &((q_1, b, A), (q_1, \epsilon)) \end{aligned} \right\}$$

$$\text{Set } M = (Q, \Sigma, \Gamma, \Delta, q_0, \{q_1\})$$

Note that $L(M) = L_1$

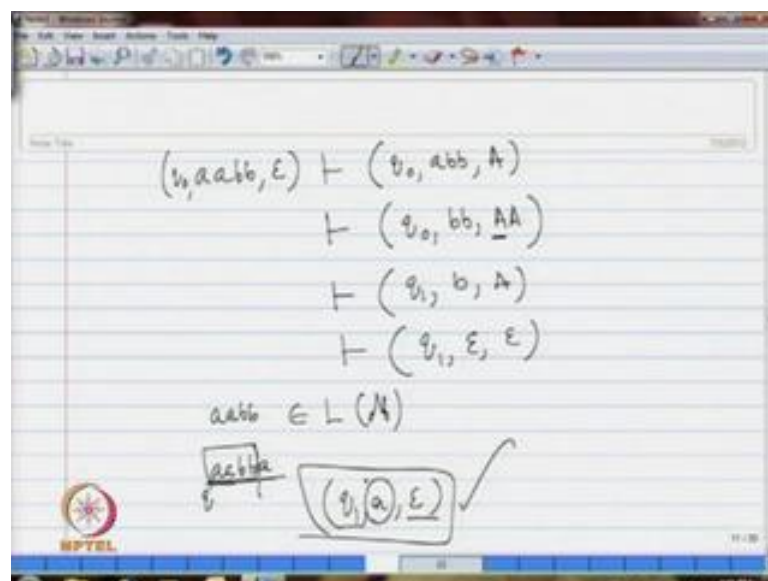
Now, let me consider some examples a natural candidate that as we have discussed if you consider L 1 equal to a power n b power n such that n greater than or equal to 1. As I gave the hint earlier what do you do you push a's inside corresponding to that symbols. You put you remember those number of a's through certain symbols and then and you are getting b's you match with them and accept the string. So, let me write the transitions instead of giving that six triple; first I will just give you the transitions. Let me say q naught is in initial state any way n greater than or equal to 1 I had put. So, at least one a has to come.

So, when a is coming I do not do anything with the stack and in the state say some capital A remembering A I will put inside the stack. So, this is one transition I define and this continues because I am in the same state whenever I get a without taking anything out of this stack; I simply put a inside the stack and when I get first b in the state q

naught I expect A in the stack in which case, I will match with them and change the state. Because at least one a and one b has to be there a a b has to be at least in in the input. So, then I will change the state say for example, q 1 and pop that a. This is one transition I defined in another transition what do I do; I continue this in q 1 when I get a and one more when I get b if I have capital A in the stack; then I simply remove that also I keep doing this.

So, I consider delta to be in the set of these three transitions and now I set M to be I have two states here q naught q 1 sigma is a b and I have considered gamma I have only one symbol used capital a 2 sigma gamma delta has defined above and q naught the initial state and the singleton q 1. In the final state suppose if I set M to be this and note that observe that the language accepted by M is L 1. Let me just do some computation in this PDA in this pushdown automaton.

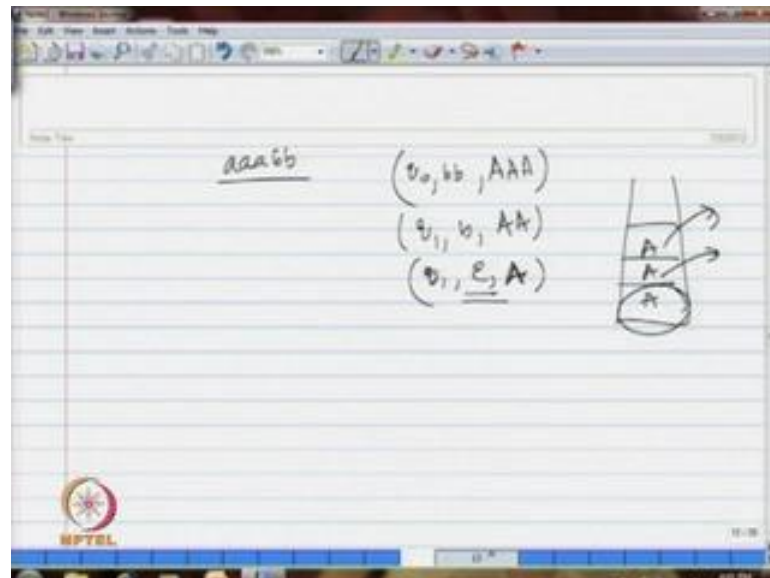
(Refer Slide Time: 28:29)



So, this pushdown automaton for example, if you give a b b in the initial state q naught this stack empty. As we have defined in q naught if you read a without touching anything in the stack; I am simply push putting a inside the stack so; that means, what happens here q naught a b b capital A and in the next step I continue the same. So, I have b b A A I have put the second A here and then when I have b in the input I expect a on the top of the stack in the state q naught to change to the state q 1.

So, I have the remaining here this. Look here if I have b in the input and a on the top of the stack I remove that capital A from the stack and read b to change to the state q 1. So, second transition I am using here. So, by this you get and in q 1 I continue with this. So, q 1 I remove A from the stack. So, clearly as we have mentioned q 1 is a final state and thus this string a a b b is in language accepted by M.

(Refer Slide Time: 30:05)



Now, for example, if I consider a a a b b suppose if I consider you understand first I will put three a's inside the stack by the time I get this. So, this kind of configuration you get after three steps. Now, in q naught if I read b I can have this situation this will be in another step and after that you can of course, pop this also you will get epsilon you have epsilon in the input and you have this much left over. So, q 1 epsilon, so you do not have anything to read here and we have something left over in the stack. So, the stack you have put three a's in the stack and you have read you have removed corresponding to those two b's this and something is left over in the stack.

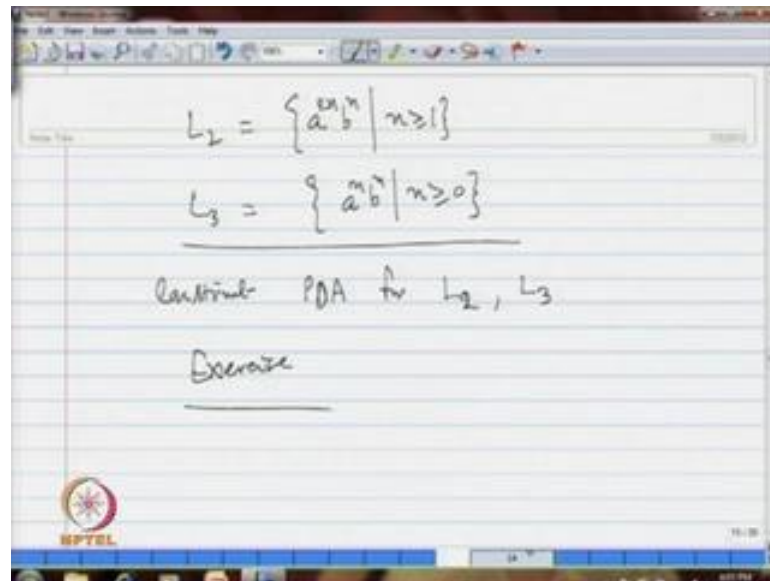
For example, if you have less number of a's and more number of b's the input will be left over and when the input is left over I do not have any transition defined corresponding to b and if you have any mixture of a's and b's now what will happen. If you have any mixer of a's and b's corresponding to a you have removed b's and for example if you get a then what will happen let us see that part. See for example, now if you have a a b b after that say for example, a in q naught I can put these two a's in the stack. So, it will be

a a b b corresponding to this and when a is left over I am in the state q_1 and the stack empty.

As we have observed here a a b b; now if I give a a b b after that this or may be after one b if I give a in this state q_1 in the state q_1 if I have a then what is happening after getting as long as we are getting is we are putting them inside. Once you get b you are popping them and if there is a mismatch suppose assume you have match for a a b b and after that say for example, if you have a then what will happen q_1 a epsilon. Now, you see in q_1 we have not defined. So, the delta as a flexibility we have fixed with the pushdown automaton; I abbreviated as PDA pushdown automaton if you know some symbol is here for which I have not defined I have this flexibility because we have considered transition relation this is non-determinism that we introduced.

So, through this non-determinism you know I have the flexibility for not defining for certain symbols in the particular state. So, I have not defined. So, you will reach to this configuration and the computation you know hands over there and you can see that this final configuration this is the final configuration corresponding to the initial configuration with a with input a a b b a. So, with this it will stop you are in the final stage, but some input is left over all other stack is empty the some input is left over. So, this is not an accepting configuration. So, if you get an accepting configuration then only we say the string accepted. So, thus with this simple explanation about the computation in this particular PDA I can say that the language accepted by this particular push down automaton is $a^n b^n$.

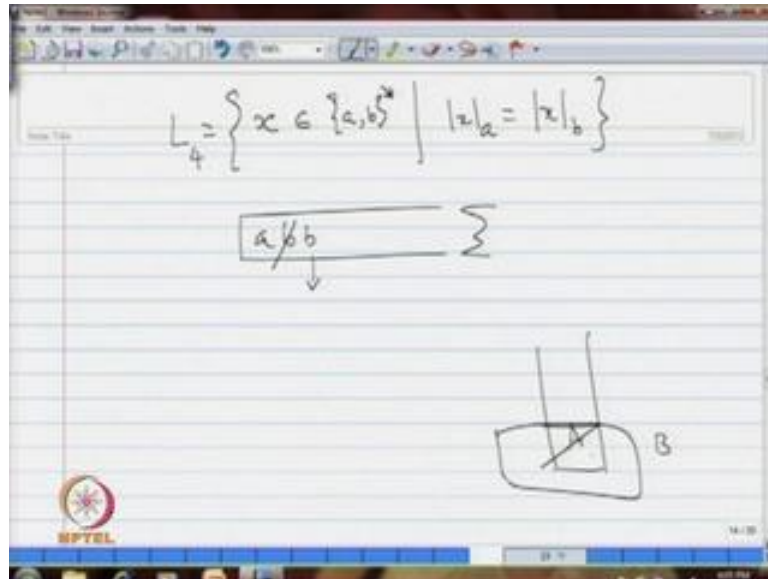
(Refer Slide Time: 33:42)



Now, you can try other what I have mentioned earlier say for example, a power 2 n b power n such that n greater than or equal to 1. So, you can try other what I have mentioned earlier say for example, and let me give this a power n b power n such that n greater than or equal to 0. Why I am putting 0 here I have working with greater than or equal to 1. So, I know that there has to be at least one b and accordingly I have defined the transition here.

So, q 1 b a I have defined this now you have to carefully think that since it is zero epsilon can be accepted and now when you have to change the state that you have to be careful with this. So, construct pushdown automata for L 2 and L 3 pushdown automata for this take this is an exercise.

(Refer Slide Time: 34:53)



I will discuss one more example for instance if you consider set of all x in a^*b^* such that let me consider this. Say, L_4 number of a 's in x is same as number of b 's in x . You know this is a context free language you would have construct a context free grammar for this. So, now you first think about the logic when you are constructing a pushdown automaton. So, what is the pattern that you are expecting and how you what are the things that you have to remember you have to memories or you have to put it in the memory. So, that first you decide then based on that you can construct the respective you know you can define the transitions in the respective PDA.

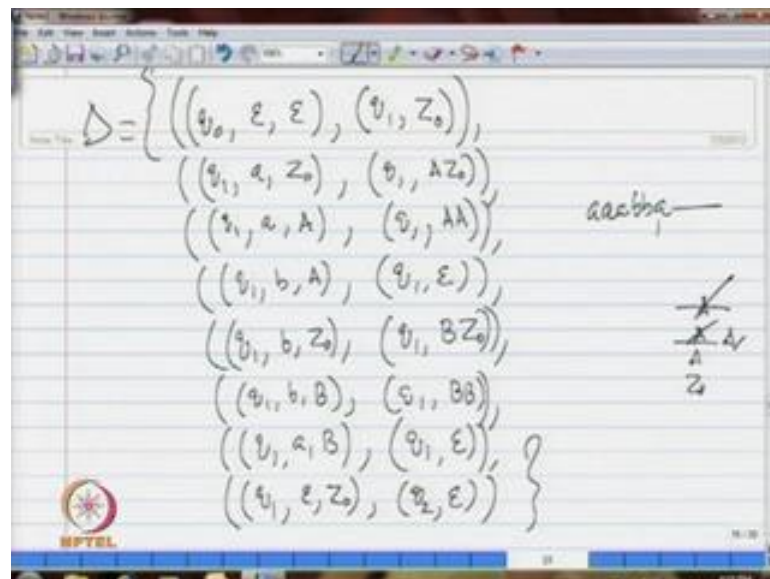
So, one logic can be now when I get a I put say for example, a in the stack and I get now b I will match with this when I get b one more b ; I will put b inside this. Now, the question is now the question is how I know that you know here things are things get over. For example if this is a situation a^nb^n suppose this is situation corresponding to a I had put inside a and I have match when I have got b I have removed this. When I get b the first place I know I am putting an inside that when I get b know how do I know that tank became an empty. Now, I have to put say for example, some b inside this.

So, that is that will be a question say for example, if you are getting any pattern $a^n b^n$ sort of pattern that earlier version will work. Now, it can be a mixer you can have first b if the first symbol as for has the first symbol concerned may be from the initial state; we can go ahead with that and you know you can put that particular symbol,

but in between when certain some after some match when the stack becomes empty. How do I know that a new symbol a another type of symbol has come and I have to put inside the stack.

So, for that purpose you can now introduce you know some special symbol in the beginning itself, we put inside the stack and then whenever you are touching that symbol you know that stack become the empty; that means, the symbol corresponding to that you have already a match now another symbol has come; so now corresponding to that you want to put inside that this.

(Refer Slide Time: 37:24)



So, that is how you can define push down automaton is for this particular purpose. So, let me use the technique and say in the beginning without reading any input and of course, stack is any way empty in the beginning I simply put some special symbol. Let me call it as say Z naught pushes inside the stack so that means, in the beginning in the stack I put this. Now, in q 1 if I get a and top of the stack is Z naught then what do I do z naught I do not remove, but corresponding to that I put A inside and Z naught is that remains there. So, this is the transition that I defined now if it is Z naught is a top of the stack.

Now, I have already one a if I get one more a; that means, this is the situation I have then this a I do not remove. So, in same q 1 I will continue and I put one more A corresponding to this A. I hope you have got the point. So, in the beginning we have put some special symbol and if I get first A's and the top of the stack is naught, then I push

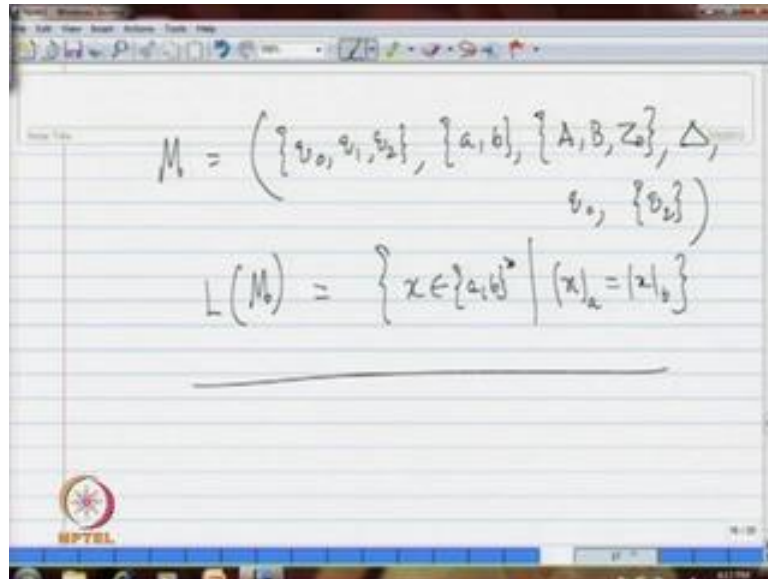
A's one after another and if I am getting. So, many A's and I will keep pushing those s inside that. Now, in q 1 corresponding to this a if I get b know and if I access A on the top of the stack what I do I will pop that. So, and this is matching configuration corresponding to b this is a matching transition.

So, I keep now the something will match as many b's as long as you are getting b's it will match. Now, in q 1 for example, if you have got Z naught; that means, A's have got exhausted. So, now, you put b inside that means without removing Z naught. So, this similar to what we have done earlier for a's. So, if I get some more b's; that means this is a situation you keep putting b's inside that. So, this you consider and now corresponding to b's if you get a's in the input then you remove so match. Now, you can understand if you have say for example, three a's you have put inside the stack if you get three b's that you are matching.

Now, corresponding to if you have lesser number of b's and after that suppose if you get some more a's you are in the same state q 1. So, say for example, if you have say three a's two b's then a. Then, what will happen this three a's are put in the form a a a of course, Z naught in the bottom and now when I get b I am removing this another b I am removing this. Now, I am accessing on the top a in the top of the stack. So, when I get a now I get I will put. So, these two a's are removed; now, I put corresponding to this a another a in the stack after that whatever that you are getting accordingly that. You will be managing now if there is a match; that means, number of a's is equal to number of b's throughout whatever the order that we put and match them.

Finally, by the time you complete the input what is the expectation. So, input is empty and if there is a proper match; then on the top of the stack you would have only the special symbol Z naught whatever that we had we had put in the beginning. If you access that in q 1 you simply change the state and remove it;, so because non-determinism is helpful in this regard. So, I will change to the state q 2 without reading anything in the input; if I access Z naught I simply remove it. In q 1 I will change to state to q 2 and simply remove that Z naught.

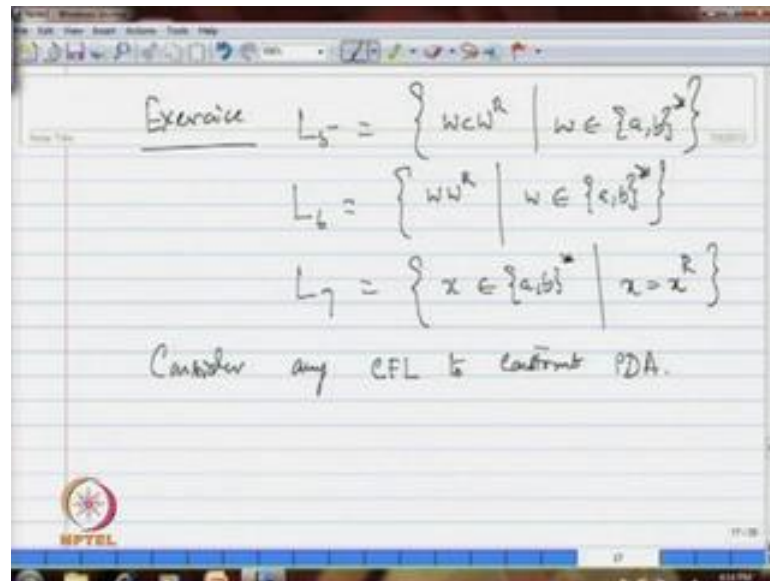
(Refer Slide Time: 41:53)


$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{A, B, Z_0\}, \Delta, q_0, \{q_2\})$$
$$L(M) = \{x \in \{a, b\}^* \mid |x|_a = |x|_b\}$$

So, this transition I put in this now with this transition if you set machine M. So, how many states I have three q_0, q_1, q_2 and what is Σ here I have a's and b's only. So, little a little b and in case of Γ I have capital A capital B and Z naught this three are their Δ has defined and q_0 is the initial state and the final state is singleton q_2 . So, if you consider this next step all; you can observe that as per the logic we have discussed you can observe that the language accessible by this machine is the required one; that means, number of a's is equal to x in a b star such that number of a's is equal to number of b's.

Now, you see the pushdown automaton through this logic you can understand systematically; how the pattern of the symbols are coming and accordingly how we are actually you know matching them for the same language. I am sure that you would have felt little bit difficult when you are constructing context free grammar, but you know depending on the pattern that you are expecting in the input and appropriate string that you have to store if you can think of then you can easily construct a pushdown automaton. In that regard non-determinism is especially useful and you will be I am sure that you will be able to construct pushdown automata for context free languages that you all learnt.

(Refer Slide Time: 43:46)



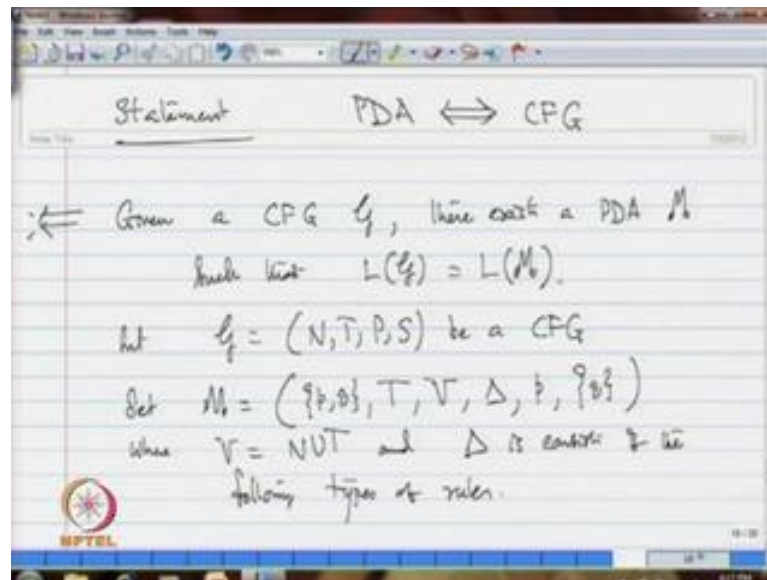
Let me give some more examples that you can try as a exercises I am sure that you have you would have constructed context free grammar for all these things, but let me just give you. So, let me consider this as L 5 here in this lecture. So, $w c w^R$ such that this w I am comparing myself to element alphabet; nevertheless you can construct for arbitrary alphabets. Now, in this case we can quickly see that what are the symbols that we are getting you put inside the stack when you are getting c because that is not in a 's and b 's and you get this symbol c ; then you know there you have to change the state and match with the reversal its reversal essentially.

Now, if i ask you $w w^R$ now you do not have any check here when you have to change for matching so; that means, here again non-determinism is useful. So, think carefully and construct for this and in general because this is even palindromes if consider if I ask you set of all x in $a b^*$ such that x is equal to x^R ; that means, all palindromes power the alphabet $a b$. Try for this languages and not only this any context free language that you have considered you have constructed context free grammar you try constructing PDA. So, consider any CFL that you would have constructed CFG for these languages and construct PDA push down automata.

Now, here I give you a mechanism particularly to obtain the equivalence between this pushdown automata and context free grammar suppose if I obtain the equivalence; that means, you give me a context free grammar. I construct a push down automaton

accepting the same language that is generated by given context free grammar and vice versa. If I can do that you can understand you can understand the equivalence between push down automata and context free grammar; once again here I emphasize that the pushdown automaton that I have introduced the version it is a non-deterministic version.

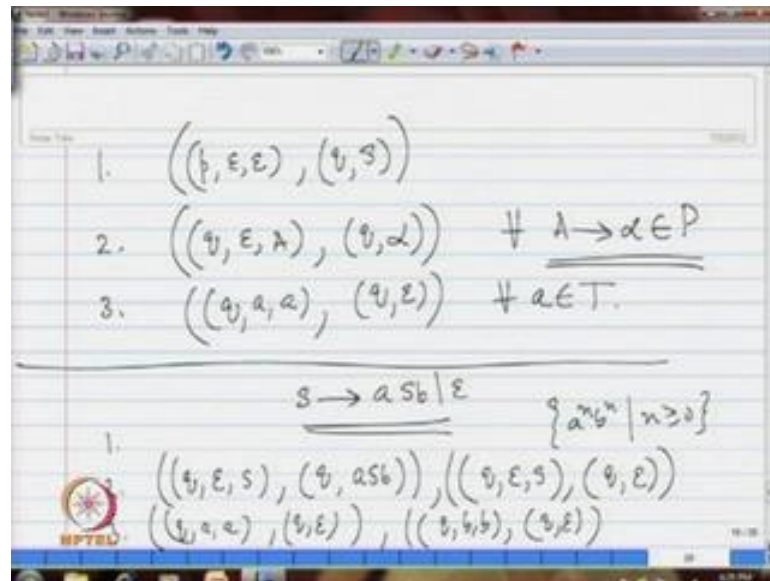
(Refer Slide Time: 46:27)



So, the statement here is the pushdown automata are equivalent to context free grammars. What do I do? Since you know context free grammars are here given a context free grammar context free grammar g given a context free grammar there exist a PDA. Let me write say M such that the language generated by g is equal to the language accepted by this M . So, this is one part this is particularly this direction similarly you give me a PDA; I have to construct a context free grammar and I have to construct a context free grammar equivalent to that. So, in this lecture I simply construct I consider this direction; that means, you give me a CFG I construct a PDA and I compute this lecture in the next lecture, I consider the converse part.

So, let it is very simple that is why I can explain in this lecture. Let $g N T P S$ is a CFG. Now, set m to be only two states I consider see how simple is that sigma here of course, the input the terminal set their and now gamma I consider all the symbols that usually we write V gamma and I define delta and p is the initial state. Singleton q I take them as finally, state where V is all the symbols of all the symbols of that grammar N union T and delta is defined by delta consist of the following types of rules.

(Refer Slide Time: 49:14)



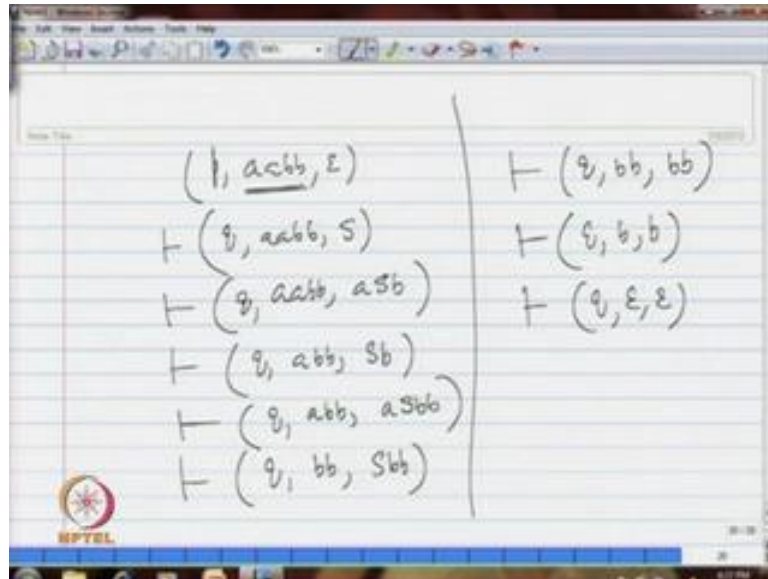
What are they? First I do this in p without reading anything I push I change the state to q and push the start symbol in to this stack. So, this is one type without reading anything in the input I do this simply I push s in the stack. Now, what do I do? In the state q if I have a non-terminal symbol on the top of the stack. So, this I push the respective. So, for all A goes to α in P . I hope you understand this transition; you take any production rule in the grammar A goes to α form. If you are having A on the top of the stack without reading anything in the input; you simply remove A and put the corresponding right hand side α ; what are the rule that you put inside the stack.

So, corresponding to each production rule you introduce one set and then what do I do for all terminal symbols; when I am getting on the top of the stack terminal symbol I will define this you match them and you remove them for all a the terminal symbols. Suppose, if you introduce these three type of rules then what are the pushdown automaton you construct; you can understand that the language accepted in push down automaton is same as language generated by the given grammar. I just consider small example a very simple example that if you consider the rules s goes to say $a s b$ epsilon you know the language generated by this grammar that is $a^n b^n$ such that n greater than or equal to zero.

You know this is the language generated by these two rules. So, what we have asked p epsilon epsilon q s this rule is anyway their 1 is there. In place of two you have two rules

that is $q \epsilon s$ is $q a s b$, this is you have to put inside this is 1 transition and $q \epsilon s$ this is non-determinism you can see that $q \epsilon s$ corresponding to the rule s goes to ϵ . This is the transition that I have to put inside that. Corresponding to three you have $q a a$ is $q \epsilon s$ this is one transition and another transition is $q b b$ is $q \epsilon s$ to remove that.

(Refer Slide Time: 52:12)



Now, you see if you are having a a b b in the input in the state p of course, this stack what do I do this $p a a b b$; we do not we simply put change to the state q simply put the start symbol s in that. Now, the way that you generate this particular string I generate in the stack and then match. So, I have the production rule $a a b b$. So, what do I do a s b this is the production rule that I use. So, you put corresponding to the transition two type two; then you match this $q a b b s b$. Now, you are accessing s you have to match another a . So, generate particular string $q a b b$ again here you insert a s b in the stack; s is removed and a s b is inserted.

So, this is the situation. Now, in q now you match with this. So, $b b s b b$ now we use this transition $q \epsilon s$ is $q \epsilon s$ the second corresponding to second production rule. So, s now will be popped. So, what do you get this $q b b$; now $b b$ now you match them q is now match this also $q \epsilon s$ ϵ . So, whatever the string that you have in the input if it can be generated by the grammar; then you generate them in the stack and match them.

Of course, here I require a special type of derivation that I am observing of course, this is particular example you do not realize; we will discuss this and see how this construction will work to show that corresponding to a given context free grammar; there exist this type of push down automaton. So, that means, this indeed this construction indeed works that is what we have to observe that we will discuss in the next lecture.