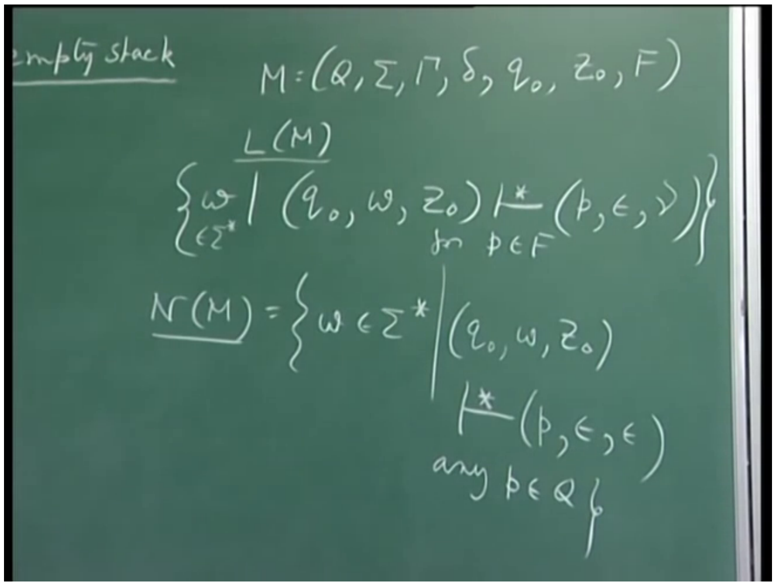
Course on Theory of Computation
By Professor Somenath Biswas
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
Lecture 33
Module 1
Equivalence of acceptance by empty stack and acceptance by final state.

Last time we had explained that we have two notions of acceptance for PDAs.

(Refer Slide Time: 0:28)



mpty stack

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$\underline{L(M)}$

$\{ \underset{\in \Sigma^*}{w} \mid (q_0, w, Z_0) \vdash^* (p, \epsilon, \gamma) \text{ for } p \in F \}$

$\underline{N(M)} = \{ w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon) \text{ any } p \in Q \}$

So suppose I have a PDA and which is as a set of states as this is sigma is the set of input symbols and gamma is the stack alphabet, delta is the transition function, q0 is the initial state and z0 is the initial stack symbol and the F is the set of final states and for this for such a machine we had explained what was the language accepted by final states, right? That means any string which this will consist of all those strings which will which can take the machine from the initial state to one of the final states, right?

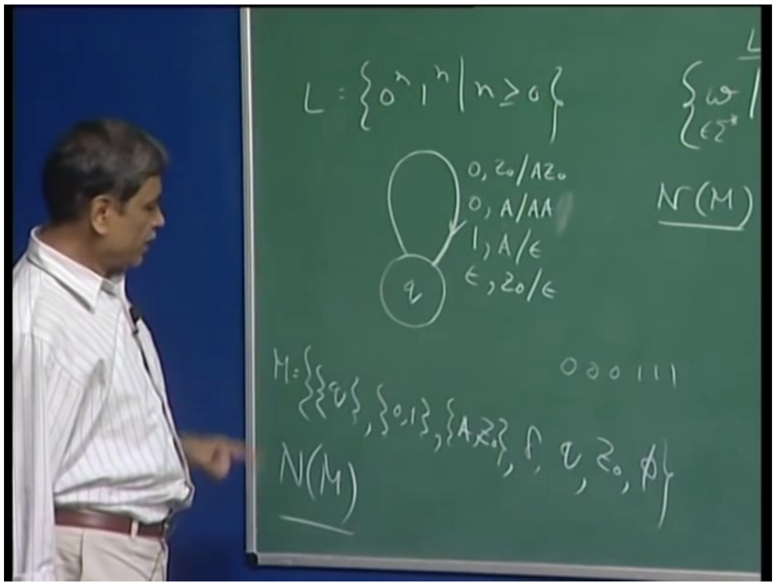
So we said ending in a, yeah right, so in one of the final states so in terms of the instantaneous description it was that q0 w is going to be in the language provided if the initial ID is always q0 that w and z0 and the this after sometime you reach a p, epsilon and we do not care whatever is the in the stack, right? So we said that language accepted by M by final state is the set of all strings w such that this condition holds. Now as oppose to that we also defined last time that there is another language associated with such a machine and we said we use the notation N(M) and this is the set of strings which are these are the language accepted by empty stack.

Now let us see how we can define it is again the all strings w in sigma star such that hear also of course I should have written sigma star all strings w in sigma star such that from the initial ID which is again q0, w, z0 after sometime we reach a state p and. So this p can be any state any p in q, okay as oppose to of course in this case the p was one of the final states for some p in the final states set of final states. So contrast these two languages which are associated with a PDA M in the first case it is all those strings in sigma star which can take the machine from the initial ID to an ID where the entire input is consumed.

So what is left is epsilon and the state is one of the final states and we do not care what is the stack contents that time. As oppose to in this case this is the language accepted by empty stack here we are saying this language consist of all those strings in the alphabet sigma over the alphabet sigma such that from the initial ID with that string w in the input the machine can go to this particular instantaneous description, and what is that description? That we do not care what the this state p is but the stack should be empty as well as the input should be completely exhausted, right?

So these are the two notions of acceptance by a PDA and so you notice that the for the same PDA M actually we have two languages which are associated with M so I can talk of the language accepted by M by final state and here it is the language accepted by empty stack. Now as it happens one may think this is more natural because you know we are used to acceptance by final state as we had seen in the case of finite state machines but often it is easier to describe a language acceptance by this notion.

(Refer Slide Time: 6:03)



$L : \{0^n 1^n \mid n \geq 0\}$

$0, Z_0/AZ_0$
$0, A/AA$
$1, A/\epsilon$
$\epsilon, Z_0/\epsilon$

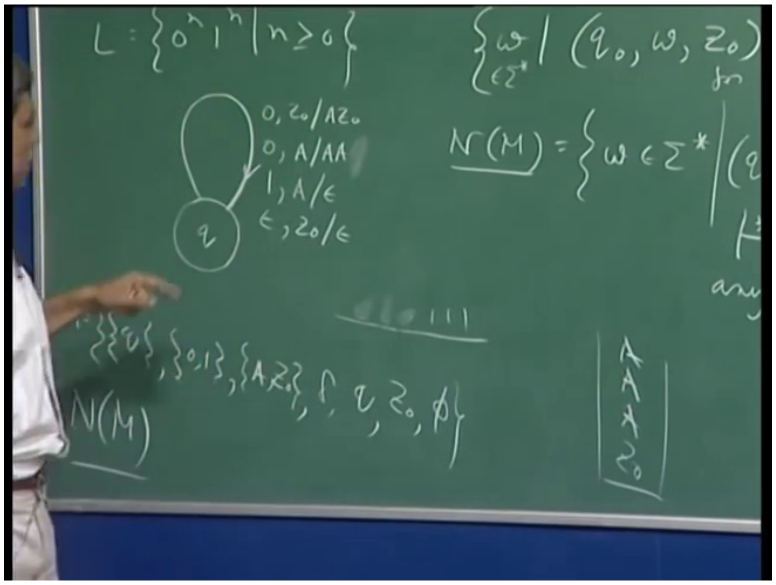
$M = \{\{q\}, \{0,1\}, \{A, Z_0\}, \delta, q, Z_0, \phi\}$

000111

$\underline{N(M)}$

$\{ w \mid$
$\in \Sigma^*$

$\underline{N(M)}$



$L : \{0^n 1^n \mid n \geq 0\}$

$\{ \underset{\in \Sigma^*}{w} \mid (q_0, w, Z_0)$
for

$0, Z_0/AZ_0$
$0, A/AA$
$1, A/\epsilon$
$\epsilon, Z_0/\epsilon$

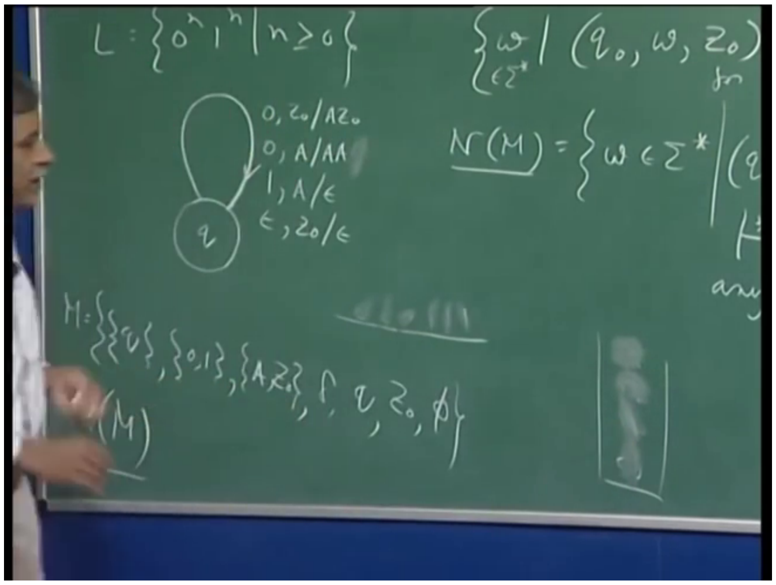
$\underline{N(M)} = \{ w \in \Sigma^* \mid (q$
$\vdash^*$
any

$\{\{q\}, \{0,1\}, \{A, Z_0\}, \delta, q, Z_0, \phi\}$

$\underline{N(M)}$

111

A
A
A
Z_0


$L: \{0^n 1^n \mid n \geq 0\}$
0, z0/Az0
0, A/AA
1, A/ε
ε, z0/ε
q
$M = \{\{q\}, \{0,1\}, \{A, z_0\}, \delta, q, z_0, \phi\}$
$N(M) = \{\omega \in \Sigma^* \mid$
$\{\omega \in \Sigma^* \mid (q_0, \omega, z_0)$

For example consider this language you know simple language we know it is not a regular language and just consider this particular machine PDA which has exactly one state and the so there is only one arrow it has to go from the state to this and what it can do is if it sees a 0 and on the when it sees the stack symbol is z0 it will push let us say A, right? Now if it sees a 0 and an A here it pushes on the top of the stack is A so we just push as another A so instead of A you have the string AA A the top of the stack symbol A is replaced by AA and now if it is 1 and if it is A then this is this top of the stack symbol is simply popped and it can also go basically do this, right?

So it is not difficult what is happening you see that for example if you had the string 000 111 so initially I should have said the initial stack symbol is z0 so let me describe this particular machine M in this manner that it has only one state q and input alphabet is of course 0, 1 and the stack alphabet is A and z0 and this is delta and delta is what I have described here in terms of this diagram and the initial state is q because that is the only state we have and initial stack symbol is z0 and we do not care suppose I just say the we have an empty set of final states which is okay because I am going to associate a language by empty stack acceptance.

So this is the language I am interested in so far as this machine is concerned, so since the acceptance is empty stack we are not concern with any final state and so therefore I do not describe any state to be I do not mark any state to be a final state. And now on this input what will happen initially the machine has z0 and then you see the machine is going to see this symbol top of the stack symbol is z0 so it just pushes an A here and after consuming this particular symbol then it sees another 0 and now we can make use of in fact the only transition that we can make use of is this.

So this particular 0 on this 0 it pushes another A this 0 is consumed and similarly one more time we have the same situation so this that particular 0 was consumed and then A was pushed and now what you have is a string of three 1's and for every 1 you see and if the top of the stack is A that particular symbol top of the stack symbol is popped this is what it means, right? That if I have the input symbol 1 top of the stack symbol is A then that particular symbol is replaced by epsilon. So after consuming this 1 the situation will be this because this particular A will be consumed I mean popped and similarly for on this one also will be popped and this one also will be pooped.

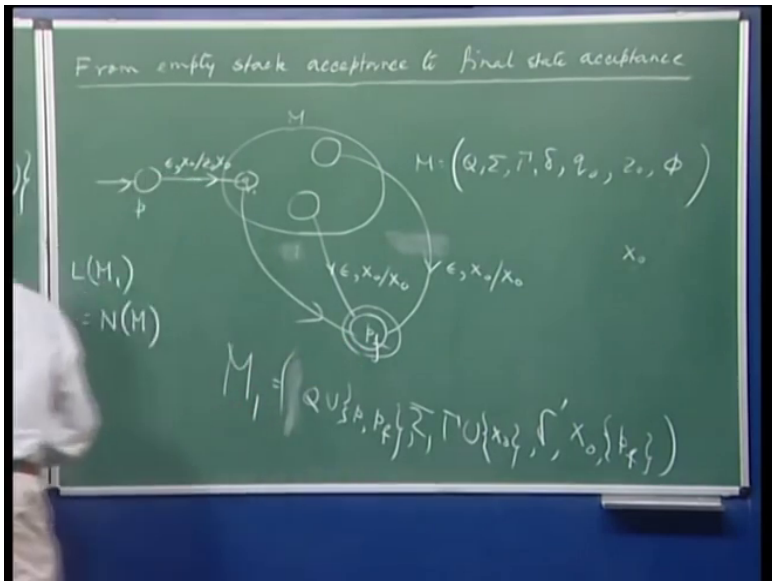
So what we have is the whatever was the input that is totally exhausted and now I can make use of this particular transition to even remove this z0. So what has happened notice this machine starting with the initial state and in this case there was only one state q on the input 000 111 with the top of the stack symbol z0 as the only contents of the stack we manage to come to the ID in which the stack is empty as well as the entire input string has been consumed. And therefore that particular string 000 111 would be in the language N(M) which is associated with this particular machine and it is not difficult to see that this particular machine will accept all strings which are in this language, right?

In fact actually the string epsilon is also there in the language and you can see if only epsilon is there the initial ID will be p, sorry q epsilon and z0 then you can just use this particular transition and we can remove that z0 and go to a the ID so that would mean the epsilon would be accepted, right? So now I have two notions of acceptance and correspondingly I have two languages with associated possibly with the same machine M, right? So what is the by the way I mean it is a trivial kind of question what is the language accepted by M by final state that language is empty there is no string will be accepted because there is no final state p and so therefore no string in this set.

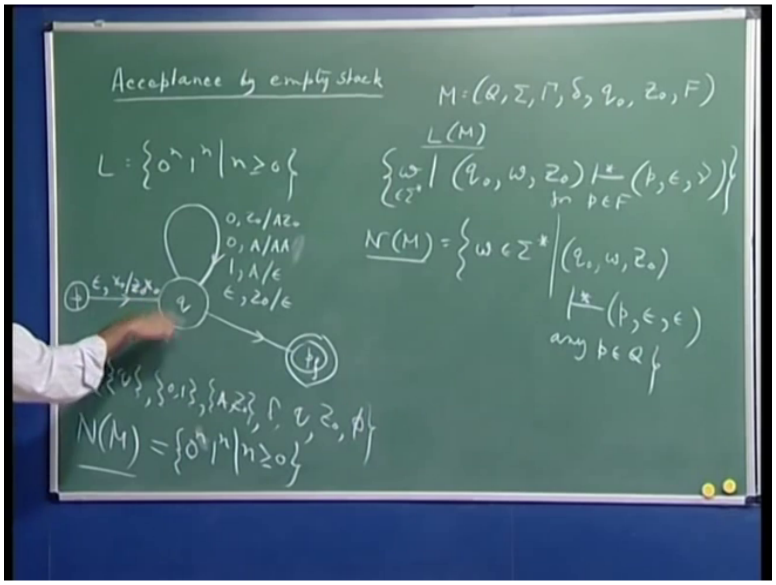
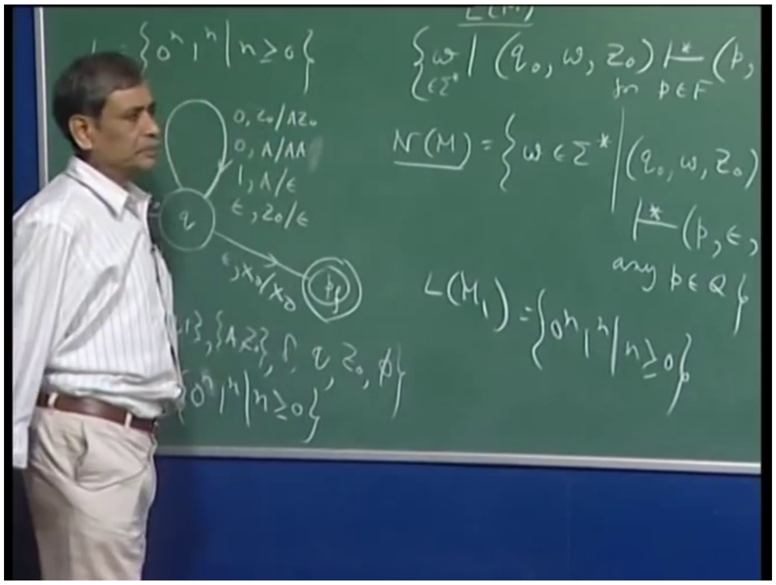
So in any case the point is that for every machine M we can talk of two languages which are associated with that particular PDA, now the question therefore is that is it possible that I have a language for which I can define a PDA which will accept that language using one of these notions and not the other, then we have a some bit of problem because then I cannot talk of both these notions of acceptance is in the same manner and the same breadth so to say as it happens that is not the case, the point is that if for a language there is a PDA which accepts that language while empty state, I am sorry empty stack then there will be another PDA which will accept the same language by final state and vice versa.

So the point is if a language is at all accepted by PDA using either of these two notions of acceptance then that language is also accepted by some other PDA possibly using the other notion of acceptance we will prove this now.

(Refer Slide Time: 15:12)



From empty stack acceptance to final state acceptance
M
$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \Phi)$
$\epsilon, X_0/z_0X_0$
p
q_0
$\epsilon, X_0/X_0$
$\epsilon, X_0/X_0$
X_0
$L(M_1) = N(M)$
p_f
$M_1 = (Q \cup \{p, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta', X_0, \{p_f\})$


Acceptance by empty stack
$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
$L(M)$
$L = \{0^n 1^n \mid n \geq 0\}$
$\{w \in \Sigma^* \mid (q_0, w, z_0) \vdash^* (p, \epsilon, \gamma)$ for $p \in F\}$
$0, z_0/Az_0$
$0, A/AA$
$1, A/\epsilon$
$\epsilon, z_0/\epsilon$
$N(M) = \{w \in \Sigma^* \mid (q_0, w, z_0) \vdash^* (p, \epsilon, \epsilon)$ any $p \in Q\}$
$\epsilon, x_0/z_0x_0$
$\{q\}, \{0,1\}, \{A, z_0\}, \delta, q, z_0, \phi\}$
$N(M) = \{0^n 1^n \mid n \geq 0\}$


What I would like to show you now that suppose I have a PDA which accepts some language by empty stack we would like to construct another PDA from that old PDA which will accept that same language the new PDA will accept that same language using final state acceptance.

So let us see what is the situation, suppose I have this is my PDA M and which is accepting some language with a empty stack acceptance so you know schematically it is something like this that this is the initial state of that PDA so let me say M is q, sigma, gamma, delta, q0 so this state is q0 and z0 and since it is accepting by empty stack you know we can just say that the set of final states for this machine is empty. Now what I will like to do is essentially that create a machine which when it senses that this particular machine has empty (())(16:45) stack it goes to a final state it can go to it is final state.

Now you might say that why not have the situation that from such a state it should go to a state which is final state the problem is you see you have to we must appreciate that when this machine has emptied its stack then there can be no other transition, why? Because every transition requires something in the stack some symbol some stack symbol as the top of the stack symbol so the if the stack is empty the machine cannot proceed any further because the way we have defined our PDAs you note is that for every transition we require that some symbol should be there as the top of the stack symbol.

So the point I am trying to say is that once this particular machine has emptied its stack this particular machine cannot do anything, now if are just running this machine and once that has emptied its stack then of course you it nothing can be done, right? So one possibility is how do I to figure out that situation as happened that we create another machine M1, right? Where we have a different initial stack symbol and so another state new state p and from that state p, right? On without consuming any symbol that on so in that machine the initial stack symbol is x0, so x0 is replaced not replaced by when the x0 is there we can push this z0 which is the top of the stack symbol of M, right? So is it clear what we are saying? We are saying that imagine this old machine which is M to that I add another new state and the really the role only role this state plays is to push this stack symbol start stack symbol start of z0 onto its own initial stack symbol x0, right?

And now and then it goes from this state p to the state initial state of this machine M then this machine let us say starts working and let us say at some points of point of time this machine has emptied it stack then what will happen he will this composite machine is going to find the top of the stack symbol to be x0, right? Is it? Because this machine which kind of takes over once this state is reached if it ever empties its stack that time this machine's empty stack means that the old top of the stack I mean initial stack symbol is exposed now which is x0.

So in such a situation what we want is from every state if we sense that if we find that now from this state if you find the top of the stack symbol is x0 then you may write this x0, I will do not write it here, and go to a state let me call it pf for final state, right? So similarly from here also this transition is possible on no without consuming any symbol if you can go from this state to this particular state writing as the same thing and basically the same situation.

So what we are saying is that from every state in this machine this extra transition is there to a new another new state, so you see we are adding two states one is the new initial state and a final state and now I claim that this particular machine if this which was inside this circle or this ellipse was the old machine M, let me write it and this entire machine is M1, right? So now is it clear what is M1? That M1 is essentially the this set of states union to other states that we are putting p and pf, right?

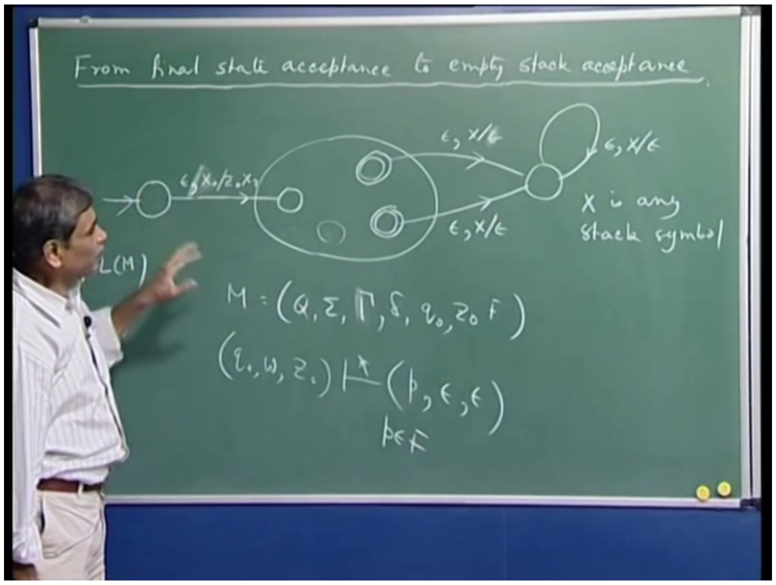
Of course it will have the same set of input alphabet same input alphabet and what is its gamma it is the old gamma union we have a new bottom of the a new start symbol the symbol stack symbol with which you start which is the start stack symbol x0 so which we this is the new stack alphabet so basically new stack alphabet is obtained by just adding a adding this x0 to the old set of stack symbols delta dash delta dash is a new transition function which retains all the old transition and adds a few more so one is to go from this state p to the old initial state of machine M or rather this machine's initial state and the other set of transitions will can will be for taking the machine this composite machine from any state to this final state on top of the stack symbol if found to be x0, right?

That means the this old machine M would have emptied its stack and so therefore we go to a final state the only final state that we have so this is delta dash and now for this composite machine the initial stack symbol is x0 and it has a final state which is this state, I am sorry I am writing this as these are tuples so these are not set, right? So we should have written like this, okay so it is not too difficult to prove that the language accepted by M1 on final state is the language accepted by M on empty stack so that is why it is possible from any PDA given any PDA which accepts a language by empty stack then using this construction to obtain a new PDA M1 which will accept the same language by final state. And as an example we could have done this construction for this particular PDA which was accepting this language 0 n 1 n by empty stack.

So it should have meant since there is only one state so this is the old initial state there is only one state here and this is our new p, right? And from here we can go to a pf which is a final state so let me write like this then here also I should have put a circle to say that this is the final state and here what should we what is the transition without consuming any symbol on its initial stack symbol which is now x0 we push z0, right? So now we start the this machine and time the stack is empty that would mean that we are going to see x0 and we come here.

So you see that from the old machine just by adding these transitions we get a machine new machine M1 and it is not difficult to see that L(M1) is also 0 n and n greater than 0.

(Refer Slide Time: 27:39)


From final state acceptance to empty stack acceptance
ε, X0/Z0X0
ε, X/ε
ε, X/ε
ε, X/ε
X is any stack symbol
L(M)
M = (Q, Σ, Γ, δ, q0, Z0, F)
(q0, w, Z0) ⊢* (p, ε, ε)
p∈F

Let us now prove the converse of what we proved just now, so what we want to show that suppose I have a PDA, okay this is accepting language by final state, right? By reaching a final state and from this PDA what I would like to do is to construct another PDA which will accept the same language by empty stack.

So it is kind of tempting to say the look why not have transitions from every final state so in this case you know I have marked two of these states as final to a particular state, right? Where the job of this particular state once you go there once the machine chooses to go here the this in this state what we are going to do this machine is going to do is just empty the stack, so how will you do that? Basically that here is on epsilon, right? On any x to epsilon.

So what it means is that whatever x is any stack symbol, right? And when we go here when we have reached some final state and the machine has the capability of making this transition and for making this transition of course it does not have to does not need to or does not wish to consume any input symbol presumably by then if it has accepted the string it has reached this final state there is no input symbol to be consumed, right? And so whatever be the stack symbol here writes the same stack symbol or it can even just remove that pop that stack symbol and similarly from here also it can do the same thing.

So you might say that this should work, right? Why? Because let us let this within this ellipse whatever be the you know states and transition which where there that shows the old machine and suppose now the old machine has gone to a final state so that means it has reached to one of these final states and now it should indicate that string to be accepted by empty stack. So all we are doing is that we are going to a state and emptying the stack, right?

This would work almost, why? Why is this is not the correct conversion for a machine which accepts a language by final state to empty stack because seems this is alright, reason it is not correct is that the old machine you know this situation is possible that you see that this machine suppose the old machine was M which again we will write Q, sigma, gamma, delta, q0, z0, F. Now it is possible that this machine empties its stack consumes the entire input, okay so suppose in this case what I am trying to say that suppose from the initial ID q0, w, z0 it goes to some state p epsilon and p is not a final state, okay.

So such a w will not be accepted by the old machine such a w is not so I can say that w is not in language L(M). Now you look at this composite machine which you are trying to what we said our first attempt to convert M to a machine accepting the same language on empty stack. Do you see in this machine what is going to happen? That w has taken the machine from here to some state p which is not a final state but the stack is empty the input is empty.

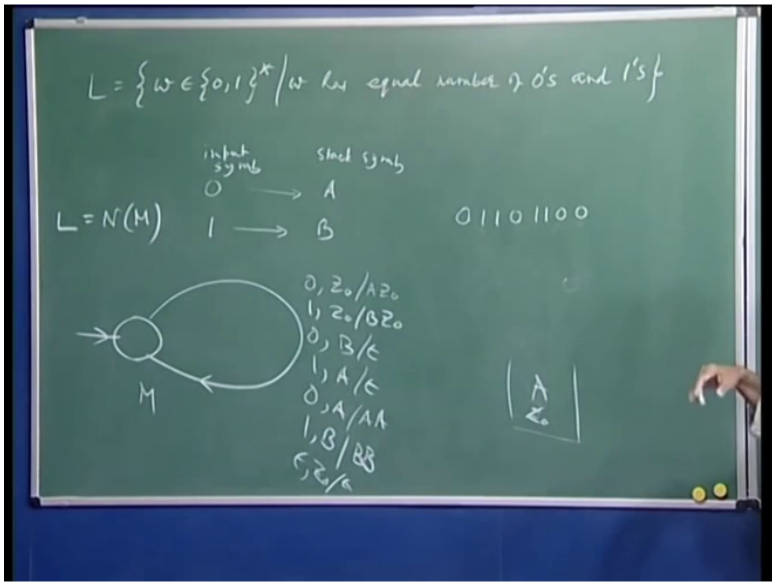
So then that means the machine has reached this state reached this ID and so if we are having the notion of acceptance by empty stack then this w would be accepted, this situation we want to avoid, right? It should not happen this particular string w should not be accepted then how would you take care of such situations? The only way I mean one way of doing I would be that you add another initial state to this and again the role of this initial state is to put a new see this new machine will use x0 as its initial stack symbol and from x0 it just puts z0 on top of the stack x0.

So now we will see what is going to happen, so what is going to happen is if this situation happened in this machine of course this machine will also reach this state p the input will be exhausted but the stack would not be empty because all the stack will now show this old top of the stack symbol x0 here, is it? So is it clear what we are saying that this particular machine had z0 as its initial stack symbol we are creating a new machine where we will use a new symbol stack symbol x0 as the you know initial stack symbol and the only thing that you do really is to write in the beginning on top of this x0 you put the initial stack symbol of this machine and then let the old machine run, so basically you are going to the initial state of the old machine which accepted the language by empty stack.

And now if the old machine, sorry the old machine was accepting the language by final state and now if the old machine had emptied the stack exhausted all the input and in a is in a state which is not a final state. So what is going to happen? Right now the this particular machine the composite machine is going to see x0 on the top of the stack, right? So the stack is not empty and therefore there is no way this machine this composite machine is going to remove that x0, so therefore that w is not going to be accepted.

So this is the correct conversion from a machine which is just this which was accepting some language by final state to a machine which accepts the same language by empty stack, what did we do? We added an extra state here as the initial state as well as we added another new state whose job was to basically you know flush the stack and that state you would reach you can reach only from one of the final states and the accidental emptying of stack is not going to now matter because we have this x0 in the stack, so this is the conversion. So we added two states added a new stack symbol and we can now you now it is easy to get the definition of the empty stack acceptance machine from the old machine.

(Refer Slide Time: 37:27)



We had encountered this language before we gave this as an example of a context free language which is not a regular language.

And let us design PDA to accept this language this strategy is something very simple so let us say for 0 there is a corresponding stack symbol so this is the input symbol and the corresponding stack symbol is A and for 1 the corresponding stack symbol is B and think of some examples string in the language let us say 01101100 or something like this, so this is four 0's and four 1's which is therefore a string which is in the language what you are going to do is something like this that here is my stack there is some initial stack symbol z0 when I see a 0 I will push an A and now I see a 1 so you see what I can do is therefore that this particular 0 is can be checked off with the one that I have seen.

So on one and if the top of the stack is A, then I am going to consume that 1 and you know we are going to pop the A. So basically that means I have what I has a symbol 0 here I have seen a 1 so these two have been paired off and now the rest of the string should have the same property that it should have equal number of 0's and 1's. And now again I see a 1 so corresponding to 1 this 1 will push a B, right?

And again I see a 0 now 0 and top of the stack is B so I can consume this 0 and corresponding this to this 0 was the old 1 for which I had a B and therefore I pop it and now you see I have 1 here so I will put a B another 1 here so I will put another B, is it clear why we are doing it because you now I as yet I have no evidence that there is a 0 too so far so long I have you know come to this point I have no evidence that these two 1's can be paired off with 0's and so these two B's are recording the fact that I have seen two 1's which have not been paired off with 0's and now I see a 0 this 0 for this 0 this B will be removed this 0 new or last 0 this B will be removed and then we can remove this z0 and the stack will get empty.

So what I have described is that I have one just one state, right? And if because I have only one state that is the initial state as well as all the transitions are going back from that state to itself and let us now write down the whatever I described. So if you see a 0 and that means so far the stack is empty so what you are going to do you are just write the corresponding stack symbol and push it there, right? So A z0 similarly 1, z0 will be B z0 now if you see 0 and B what should you do you should consume this 0 and pop off this B, so epsilon 1 A epsilon and what about 0 and A, it would mean that we should put push another A and 1 and B I should push another B and finally that I should have the freedom to remove this z0.

So epsilon z0 epsilon, right? So what has happened is that I have a PDA call it M and I clam that this PDA accepts this language L by empty stack, okay. So it is not difficult to say that any string which has equal number of 0's and 1's will be accepted by empty stack by this machine, correct? And on the other hand suppose I had string like this 10010 so what is going to happen that initially of course you had z0 and when this 1 came you pushed a B, right? Then this 0 came you know you popped this B and checked off this you know consumed this 0 this 0 came you put A this 1 came so now you know 1, A.

So this A will be popped off and now a 0 is there so corresponding to that is A is there you see and now the input is exhausted and we cannot you know empty this stack because on the top I have A and such an A can be removed you see only with a 1 and that 1 is missing. So you can see that any string which is not in the language whatever I wrote there that string will not be accepted. So it is not difficult to prove that the this machine will accept this language which has equal number of 0' and 1's.

The interesting point about this particular machine is that it is accepting a kind of non-trivial language using just one state, right? And of course it is doing so this acceptance is by empty stack. So two points I want to make here that firstly you see that it is kind of easy to define

PDA's to design PDA's using the notion of acceptance by empty stack to accept some given language we will believe this as an when we do more and more problems when we will not in these lectures many more but if you try more and more problem you will find this kind of acceptance is fairly easy to use.

And the other interesting this is this language seem kind of complex, right? Because it is not a regular language and yet we could accept that language by empty stack using only state, this is not accidental what we can show in fact we are going to prove that any context free language can be accepted by a PDA with just one state using empty stack, but we will prove that in the next lecture and in the next lecture we will prove the converse of this not so much one state business the converse would be the given any PDA I can get a context free grammar such that the two devices that is the language accepted generated by the grammar will be the language accepted by the PDA that we are going to design.

So in other words the two results that we are going to show that give a PDA we can find a context free grammar to generate the same language as accepted by that PDA and given a context free grammar we can find the PDA which will accept that language which is generated by the context free grammar.

So that will show that PDA's capture exactly the set of languages generated by context free so therefore PDA's are just another way of capturing the class of context free languages this is something what we are going to do in the next class.