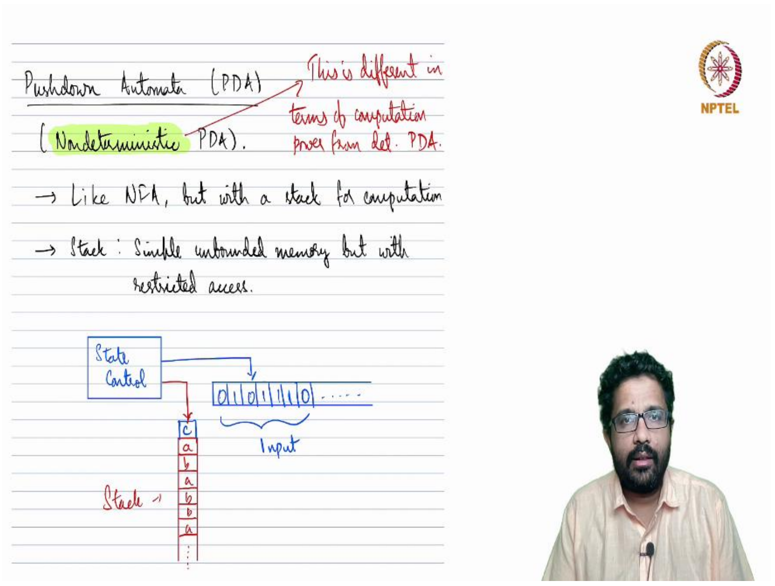
Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Normalizations in PDA and Intersection of Regular Language & CFL

(Refer Slide Time: 00:16)



Pushdown Automata (PDA)

(Nondeterministic PDA).

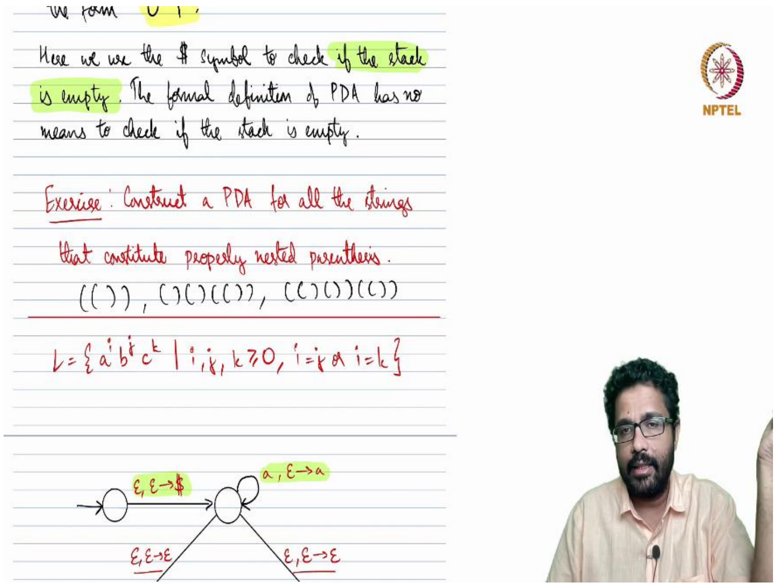
This is different in terms of computation power from det. PDA.

→ Like NFA, but with a stack for computation

→ Stack : Simple unbounded memory but with restricted access.

Hello and welcome to lecture 21 of the course Theory of Computation. In lecture 20, we saw Pushdown Automata. Pushdown Automata or PDAs were a new machine model that combine the powers of NFA and an additional stack. So, it is like NFA with an additional stack. This stack is an unbounded memory, but with restricted access. You can only access the top entry on the stack.

(Refer Slide Time: 00:48)



the form 0 1.

Here we use the $ symbol to check if the stack is empty. The formal definition of PDA has no means to check if the stack is empty.

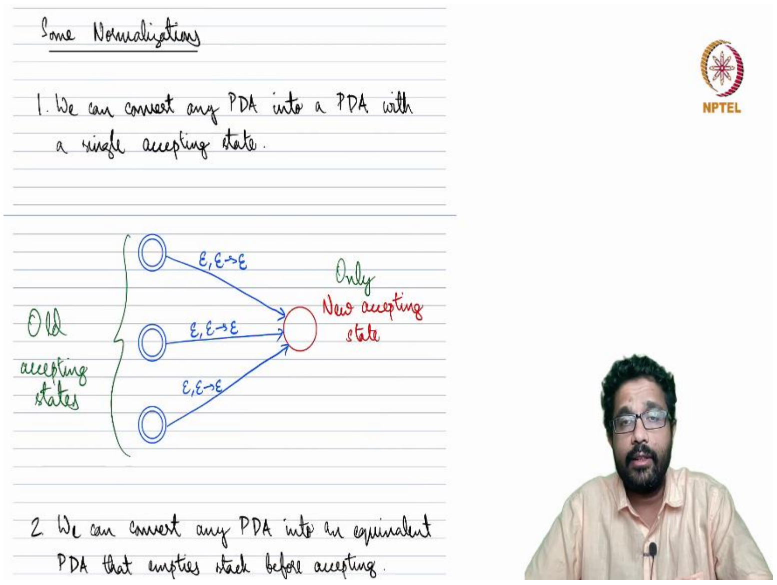
Exercise : Construct a PDA for all the strings that constitute properly nested parenthesis.

(( )) , ( )( )(( )) , (( )( ))(( ))

$L = \{a^i b^j c^k \mid i, j, k \geq 0, i = j \text{ or } i = k\}$

And we saw that it can recognise languages such as 0^n1^n, we saw examples. We saw the rules. We saw that it can recognise languages 0^n1^n also other languages which were not regular. Meaning an NFA can only recognise regular languages, but a PDA, we saw that it could do more things. We said that eventually we will show that PDAs are going to be equivalent in power to context free languages, but that is not going to be shown in this lecture. We will see it in the upcoming lectures.

(Refer Slide Time: 01:25)



Some Normalizations

1. We can convert any PDA into a PDA with a single accepting state.

2. We can convert any PDA into an equivalent PDA that empties stack before accepting.

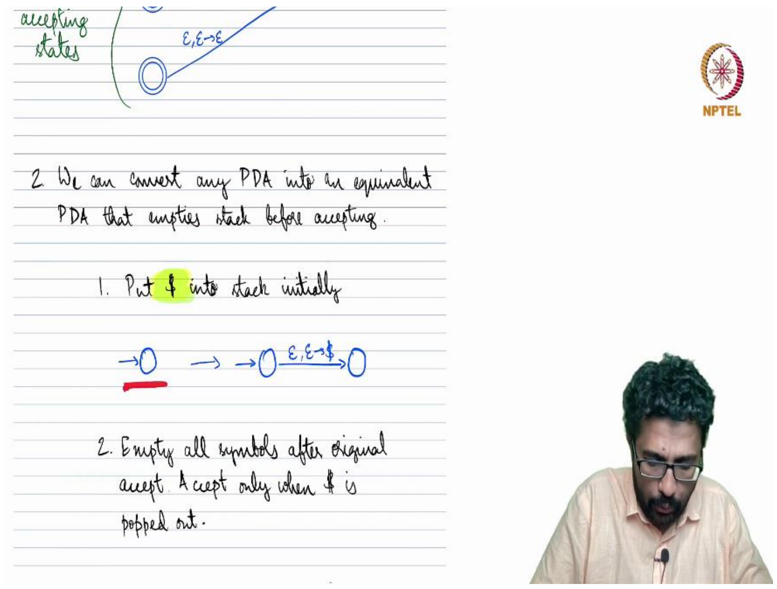
$\epsilon, \epsilon \to \epsilon$
$\epsilon, \epsilon \to \epsilon$
$\epsilon, \epsilon \to \epsilon$
Old accepting states
Only New accepting state
NPTEL

So, what I wanted to just quickly mention in this lecture, which is going to be a brief lecture, are some modifications that we could do in case of PDAs. Sat to, number one is this. If you recall we had said that if you have an NFA and it has multiple accepting states, we could convert it into an NFA that has only a single accepting state. We can pretty much do the same thing in a PDA. So, these are the old accepting states and as you can see, there are multiple accepting states.

So, the ones in blue are old accepting states. We can have ϵ transitions. We create a new state, this red one, and we make ϵ transitions from the old accepting states to the new accepting state. The old accepting states are there, but they are no longer accepting states. So, this becomes the only new accepting state. The other ones, the blue ones, are made into normal non-accepting states.

Whatever string leads to any of the blue states can take an ϵ transition and go to the new accepting state. Hence, the new PDA with this modification has only one accepting state. This is fact number one. So, given any PDA we can convert it into a PDA which has a single accepting state.

(Refer Slide Time: 03:28)



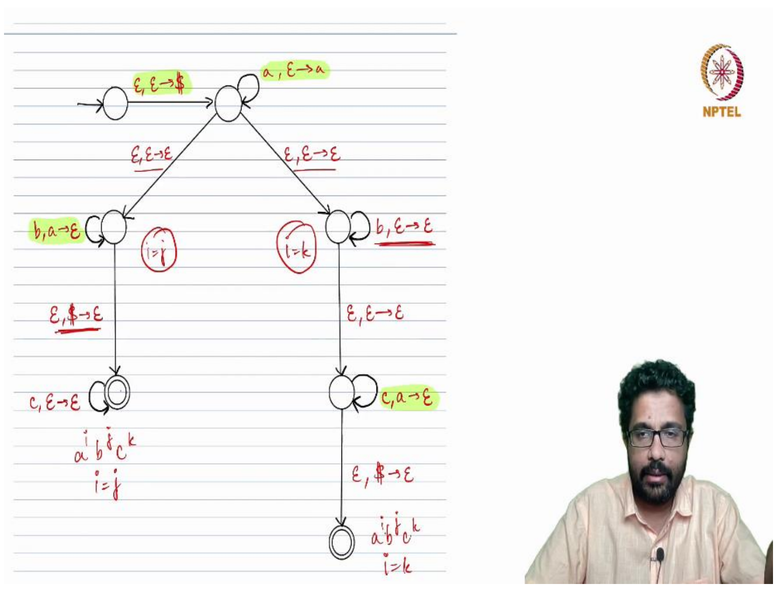
accepting states
$\varepsilon, \varepsilon \to \varepsilon$

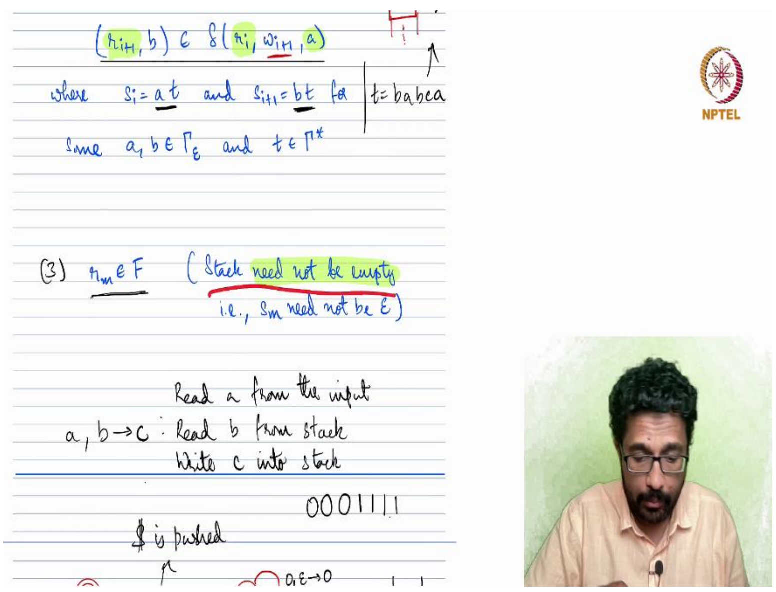
2. We can convert any PDA into an equivalent PDA that empties stack before accepting.

1. Put $ into stack initially

$$\begin{tikzcd} \arrow[r] & \bigcirc & \arrow[r] & {} & \arrow[r] & \bigcirc \arrow[r, "{\varepsilon, \varepsilon \to \$}"] & \bigcirc \end{tikzcd}$$

2. Empty all symbols after original accept. Accept only when $ is popped out.





$(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$

where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$

$t = babca$

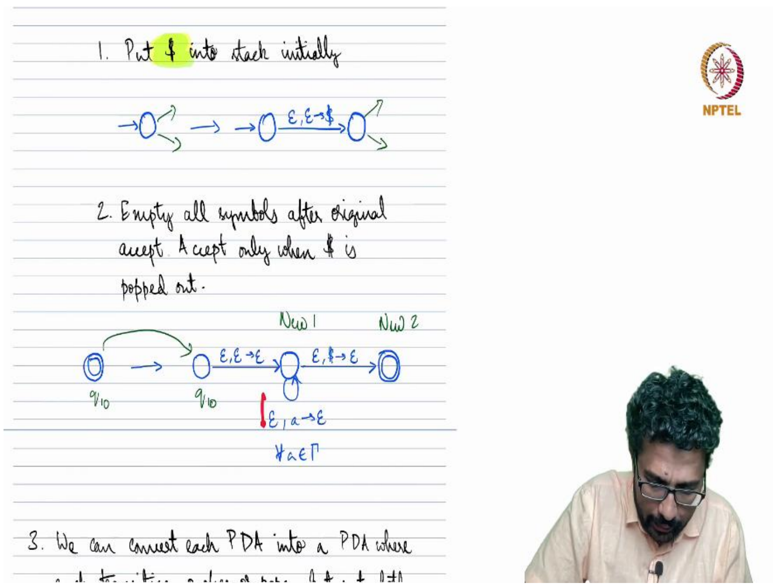
(3) $r_m \in F$ (Stack need not be empty i.e., s_m need not be ε)

$a, b \to c$: Read a from the input
Read b from stack
Write c into stack

000111

\$ is pushed

$a, \varepsilon \to 0$



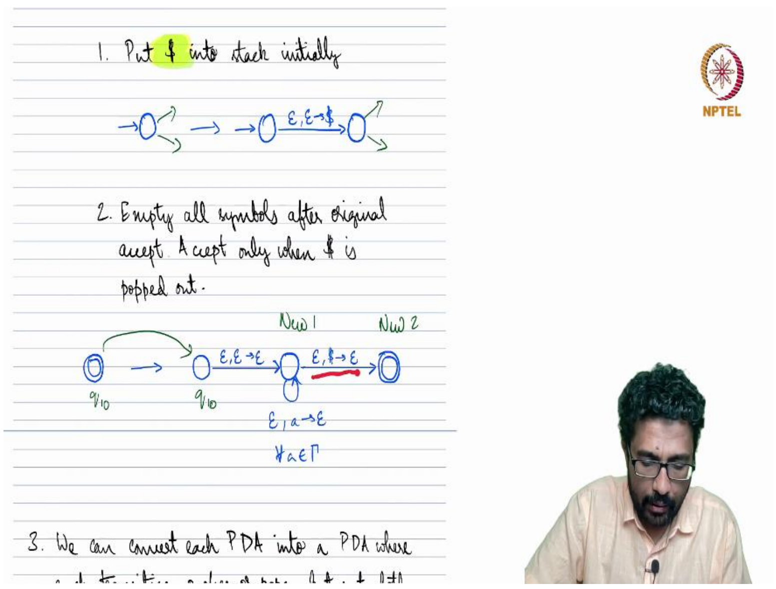
1. Put \$ into stack initially

$$\begin{tikzcd} \arrow[r] & \bigcirc \arrow[r, phantom, "\longrightarrow"] & {} \arrow[r] & \bigcirc \arrow[r, "{\varepsilon, \varepsilon \to \$}"] & \bigcirc \end{tikzcd}$$

2. Empty all symbols after original accept. Accept only when \$ is popped out.

$$\begin{tikzcd} \underset{q_{10}}{\circledcirc} \arrow[r, phantom, "\longrightarrow"] & \underset{q_{10}}{\bigcirc} \arrow[r, "{\varepsilon, \varepsilon \to \varepsilon}"] & \overset{\text{New 1}}{\bigcirc} \arrow[loop below, "{\varepsilon, a \to \varepsilon \ \forall a \in \Gamma}"] \arrow[r, "{\varepsilon, \$ \to \varepsilon}"] & \overset{\text{New 2}}{\circledcirc} \end{tikzcd}$$

3. We can convert each PDA into a PDA where

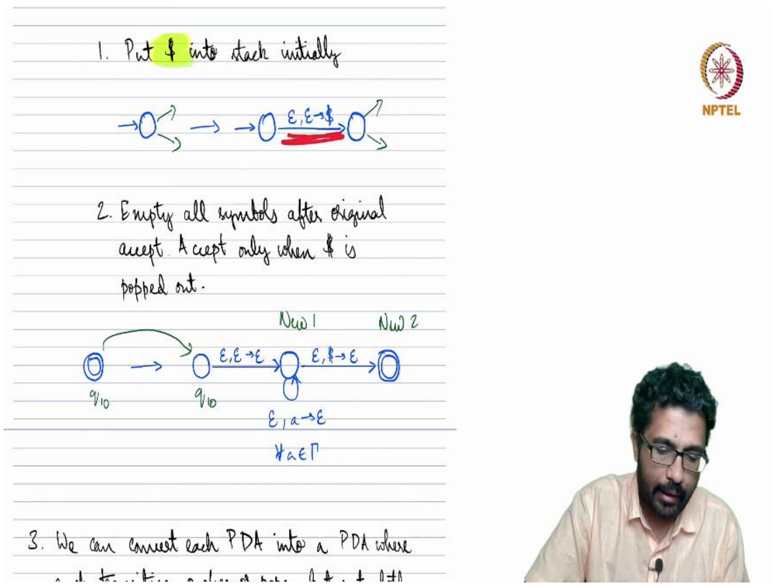


1. Put \$ into stack initially

2. Empty all symbols after original accept. Accept only when \$ is popped out.

New 1 New 2 q_{10} q_{10} $\varepsilon, \varepsilon \to \varepsilon$ $\varepsilon, \$ \to \varepsilon$ $\varepsilon, a \to \varepsilon$ $\forall a \in \Gamma$

3. We can convert each PDA into a PDA where



1. Put $ into stack initially

2. Empty all symbols after original accept. Accept only when $ is popped out.

3. We can convert each PDA into a PDA where

$\varepsilon, \varepsilon \to \$$
New 1
New 2
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
q_{10}
q_{10}
$\varepsilon, a \to \varepsilon$
$\forall a \in \Gamma$
NPTEL

Number two is this. We saw two PDAs in the previous lecture. In both cases, the stack became empty for the PDA upon acceptance. So, in both cases we push the dollar symbol and then at the end we removed the dollar symbol. But at the beginning we said that, when we define the PDA, we said that the stack need not be empty.

What I am saying is that given a PDA, suppose it accepts. So, it need not check the status of the stack while accepting. The claim is that if there is a PDA which does not ensure that the stack gets emptied while accepting because that is not something that needs to be ensured, we can modify that PDA into a PDA which always empties the stack before accepting.

So, suppose there is a PDA P that need not necessarily empty the stack, we can transform it into a PDA P' which always empties the stack. So, how do we do that? Basically, we artificially ensure that the stack gets emptied. So, we need to do two things. We first insert a special symbol into the stack first. So, let us say the dollar. So, this special symbol is not something that is used by the PDA already. If the PDA already uses the dollar, you find another symbol which is not used by the PDA. Basically, this has to be unique.

Before the PDA starts operating, we push the dollar into the stack. That is what I show here. Suppose this was the old start state. Now we modify it by pushing the dollar symbol and adding an extra state. So whatever transitions the old one had, let us say there are two transitions, those two transitions will happen from the new state here. So, it is exactly the old PDA but we are pushing a new symbol into the stack at first.

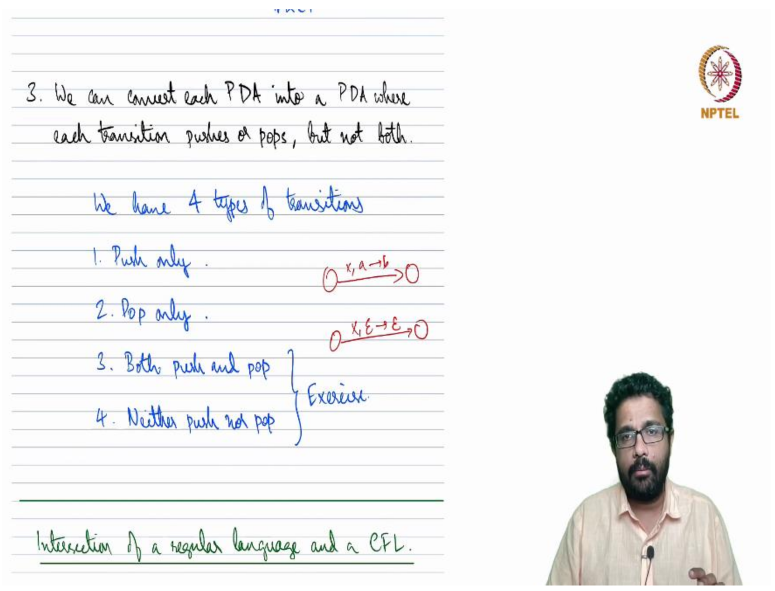
Now we have this dollar at the bottom of the stack, which is a unique symbol which is not used by the PDA. The thing that we need to address is, suppose the PDA accepted without the stack being empty. Now, we will force it to empty the stack. So, how will we do that? So, suppose this was the old accepting state. Now for this accepting state, we modify it like this. This state becomes the first state out of the three that I have drawn here. We replace it by a series of three states.

So, first let us say this is state q_{10}. Then this is q_{10} and these are some two new states. This is new, and this is also new. What we do is, from the q_{10} we make an ϵ transition to a new state. So that there is nothing read from the input and no movement to the stack. And from the new state, we are able to move to another new state. So maybe I will call it New1 and New2. From New1, we are able to move to New2 and New2 is the accepting state.

And what we do before going to New2 is that we ensure that the stack gets emptied. So, if there is a loop on New1, where we are able to empty the entire contents of the stack, whatever was the original stack alphabet. The only way to move to New2 is by popping out the dollar that we had originally inserted at the beginning.

So, for every accepting state like q_{10}, we add these two new states. So, this could be two new states added to every accepting state. So, this ensures that whenever it accepts the stack is empty. So, even though it is not a requirement that the stack is emptied while accepting, given a PDA we can convert into a PDA which always empties the stack.

(Refer Slide Time: 08:28)

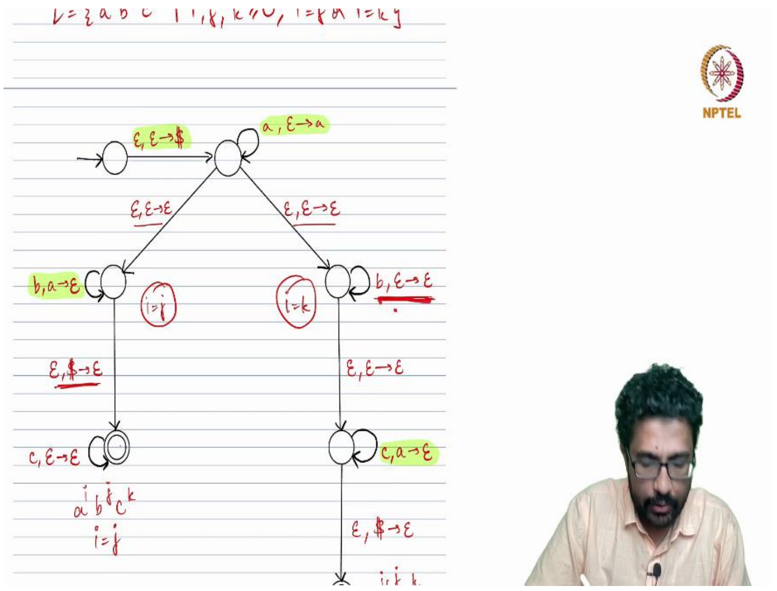

3. We can convert each PDA into a PDA where each transition pushes or pops, but not both.

We have 4 types of transitions

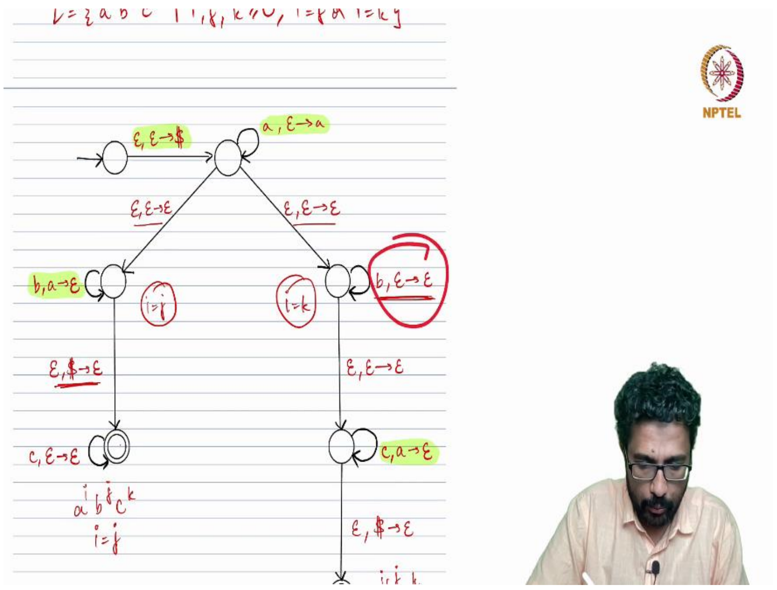
1. Push only. $\bigcirc \xrightarrow{x,\, a \to b} \bigcirc$
2. Pop only. $\bigcirc \xrightarrow{x,\, \varepsilon \to \varepsilon} \bigcirc$
3. Both push and pop
4. Neither push nor pop

} Exercise.

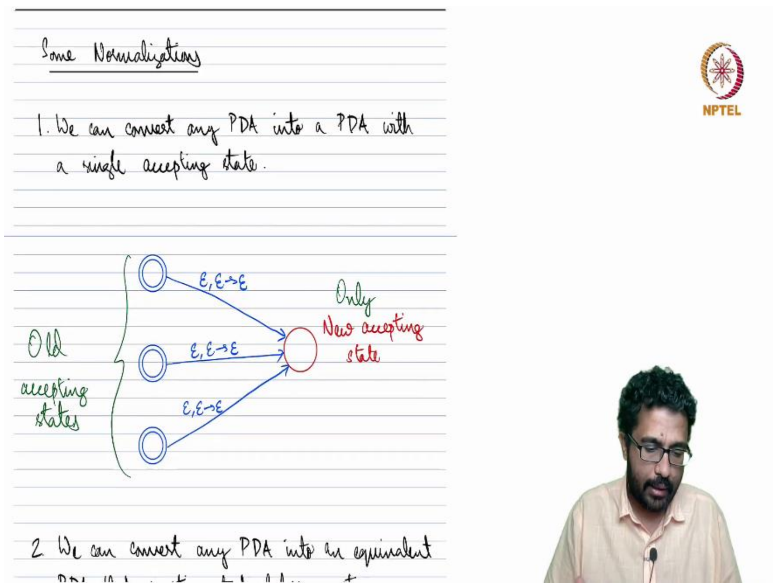
Intersection of a regular language and a CFL.


$L = \{a^i b^j c^k \mid i, j, k \geq 0,\ i = j \text{ or } i = k\}$

$\varepsilon, \varepsilon \to \$$
$a, \varepsilon \to a$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$b, a \to \varepsilon$
$i=j$
$i=k$
$b, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$c, \varepsilon \to \varepsilon$
$c, a \to \varepsilon$
$a^i b^j c^k$
$i=j$
$\varepsilon, \$ \to \varepsilon$
NPTEL


$L = \{a^i b^j c^k \mid i, j, k \geq 0,\ i = j \text{ or } i = k\}$

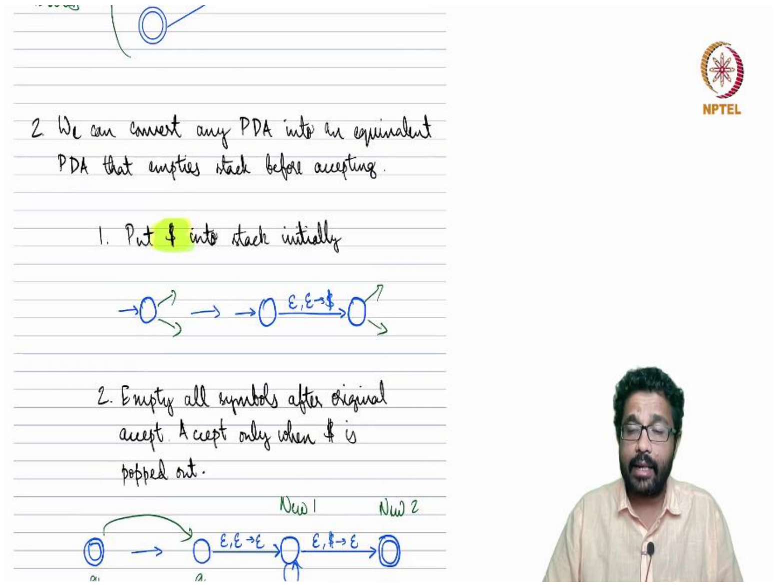
$\varepsilon, \varepsilon \to \$$
$a, \varepsilon \to a$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$b, a \to \varepsilon$
$i=j$
$i=k$
$b, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$c, \varepsilon \to \varepsilon$
$c, a \to \varepsilon$
$a^i b^j c^k$
$i=j$
$\varepsilon, \$ \to \varepsilon$
NPTEL



Some Normalizations

1. We can convert any PDA into a PDA with a single accepting state.

2. We can convert any PDA into an equivalent



2. We can convert any PDA into an equivalent PDA that empties stack before accepting.

1. Put $ into stack initially

2. Empty all symbols after original accept. Accept only when $ is popped out.

Finally, I have one more modification or one more thing that I wanted to say. Suppose there is a PDA. So, if you look here, there is a push but there is no pop. In this transition there is no push, no pop, here also there is no push no pop. I am talking about this one. And here there is a pop but there is no push. So, in any transaction it is not required that there is a push or a pop. We could have both of them happening or only one of them happening or neither of them happening.

So, the third point is that as per the rules of a PDA we are allowed to have four types of transitions. One where there is only push, one there is only pop, one there are both and one there is neither. So, the claim is that we can always convert the PDA into an equivalent PDA. Where every transition has one push or one pop but not both. Meaning every transition will be of the types 1 or 2. So if it is type 1, we can retain the transition. If it is type 2 also, we can retain the transition. If it is type 3, the transition has both push and pop. So, what I mean is if it has both push and pop. Something like $x,\ a\ \rightarrow b$.

Now, what we can do is we can split this into two steps by introducing a new state in between. First popping a and then pushing b. So, when there is both push and pop, we can introduce a new state and ensure that the push and pop are separated. When there is neither push nor pop, so, when there is a transition that does neither, it may just read the input or not even that, then we artificially push something and pop the same thing.

So, when there is neither push nor pop, so something like this $x, \epsilon \rightarrow \epsilon$. Then again, we introduce a new state in between, and we push some dummy symbol into the stack, and then we pop it. So, in the two transitions that replace this one transition, there is one push and one pop. So, it is kind of a workaround, but as a result, we can have a PDA where every transition includes a push or a pop, but not both.

So, these are the three equivalent forms, meaning given a PDA that does not satisfy this we can convert into this without any loss of generality. So, if you have a PDA that does not have a single accepting state, we can convert it into one that has a single accepting state, without changing the language.

If there is a PDA where the stack is not empty, we can convert it into a PDA where the stack is empty, again without changing the language. And if there is a PDA which sometimes does both push and pop in a transition or does neither of them in a transition, we can convert it into a PDA where every move has a push into the stack or pop out of the stack, but not both. So, these are the three points that I wanted to mention first.

(Refer Slide Time: 12:10)

4. Neither push nor pop }

Intersection of a regular language and a CFL.

We saw that CFL's are not closed under intersection.

Theorem: If A is a regular language and B is a CFL, then $A \cap B$ is a CFL as well.

Proof Sketch: We can construct a PDA for $A \cap B$. We have a DFA M such that $L(M) = A$.

We have PDA P such that $L(P) = B$.

Proof Sketch: We can construct a PDA for $A \cap B$. We have a DFA M such that $L(M) = A$.

We have PDA P such that $L(P) = B$.

We use the Cartesian product idea (originally used to show that regular languages are closed under union).

let $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and

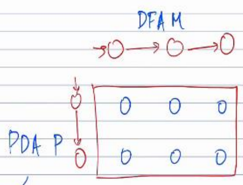
$P = (Q_2, \Sigma, \Gamma, \delta_2, q_{2}, F_2)$.

We will construct PDA P':

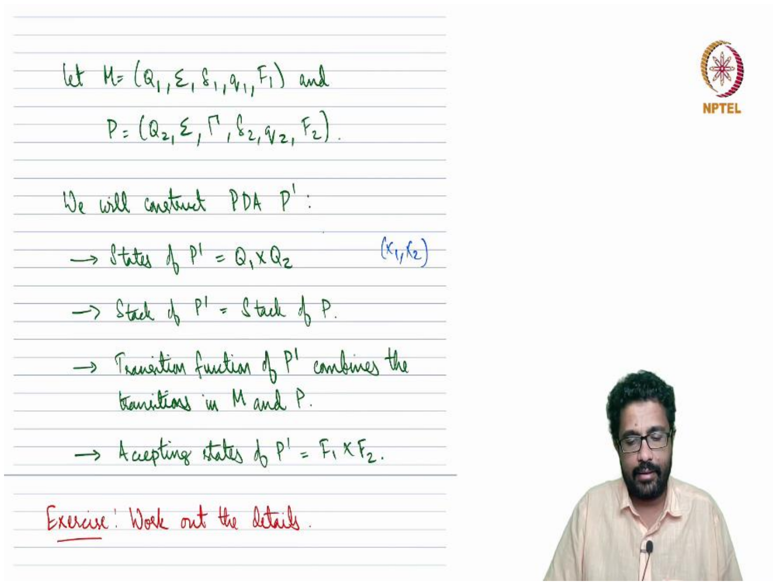
→ Transition function of P' combines the transitions in M and P.

→ Accepting states of $P' = F_1 \times F_2$.

Exercise: Work out the details.



This resulting automaton will be a PDA.



Let $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and

$P = (Q_2, \Sigma, \Gamma, \delta_2, q_2, F_2)$.

We will construct PDA P':

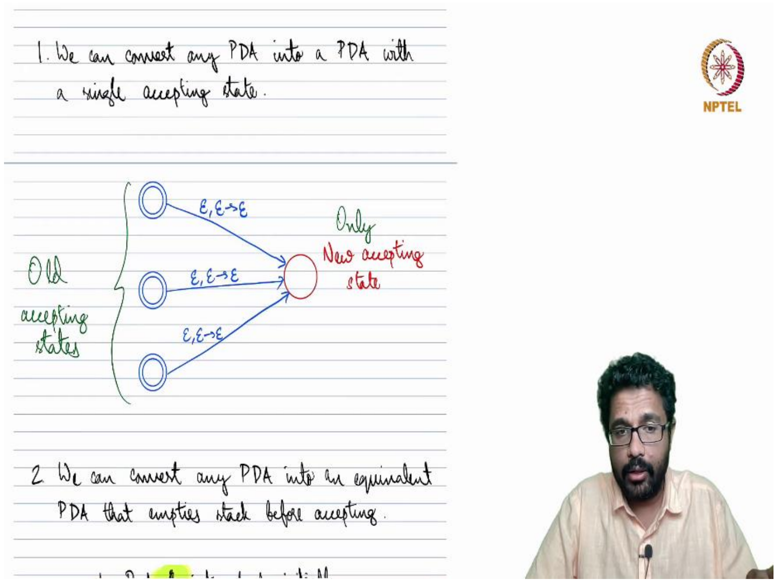
$\rightarrow$ States of $P' = Q_1 \times Q_2$ (x_1, x_2)

$\rightarrow$ Stack of P' = Stack of P.

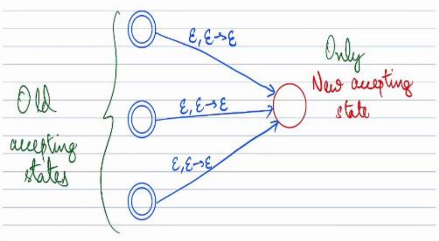
$\rightarrow$ Transition function of P' combines the transitions in M and P.

$\rightarrow$ Accepting states of $P' = F_1 \times F_2$.

Exercise: Work out the details.

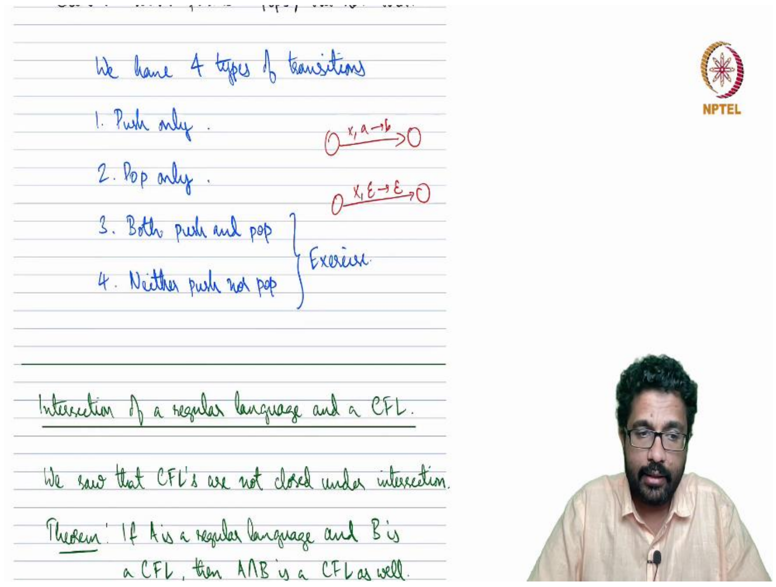


1. We can convert any PDA into a PDA with a single accepting state.



$\varepsilon, \varepsilon \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
Old accepting states
Only New accepting state

2. We can convert any PDA into an equivalent PDA that empties stack before accepting.



We have 4 types of transitions

1. Push only. $\bigcirc \xrightarrow{x,\, a \rightarrow b} \bigcirc$
2. Pop only. $\bigcirc \xrightarrow{x,\, \varepsilon \rightarrow \varepsilon} \bigcirc$
3. Both push and pop
4. Neither push nor pop

$\left.\right\}$ Exercise.

Intersection of a regular language and a CFL.

We saw that CFL's are not closed under intersection.

Theorem: If A is a regular language and B is a CFL, then $A \cap B$ is a CFL as well.

The next thing I want to mention is that we saw that context free languages are not closed under intersection; we saw an example. Now, however, we know that regular languages are closed under intersection. So naturally, one question is what if you take a regular language and a context free language, let us say A is a regular language and B is a context free language, and consider the intersection of A and B. So, the claim is that $A \cap B$ is going to be a context free language.

So, we know regular intersection regular is regular context free intersection context free need not be context free. We know examples where it is not context free. But here I am saying that regular intersection context free is necessarily context free. We will not do the full proof. Rather we will just give a sketch of the proof because we have seen the ideas of this proof earlier. So, suppose A is a regular language and B is a context free language.

So, which means there is a DFA M that recognises a language A and there is a PDA P that recognises the language B. If you recall, while we showed that regular languages are closed under union, we built a certain DFA. This was our first proof before we discovered NFAs where if you had two DFAs, we built a DFA where the states of the new DFA were Cartesian products or Cartesian pairs of the original DFA. So, this was DFA1 and this was DFA2.

Basically, the movement in DFA1 is mimicked by the horizontal movement and the movement in DFA2 is mimicked by the vertical movement. And this is combined in the DFA whose states are the Cartesian product of the original DFAs'. This is what we did to show that regular languages are closed under union. This was the first proof before we discussed NFAs. So, the same idea works here as well. We have a DFA M and a PDA P.

So, we make a Cartesian product of these, So, this is DFA M and this is, let us say PDA P, and the states are Cartesian pairs. So, the movement on the DFA is mimicked by the horizontal movement. And the movement on the PDA is mimicked by the vertical movement. So, if it is both moving horizontally one step and vertically one step, you do this, you move diagonally. In this figure, just as an example, I have just drawn 2 and 3 states. But it could be like the DFA has 10 states, PDA 20 states. It will be a 10×20 grid. But in this case, there are a couple of things that are different.

The PDA has a stack, while DFA does not have a stack. So, I can maintain a stack. So this resulting structure or automaton, which is a combination of DFA and PDA will be a PDA, because PDA P has a stack and now that stack can be retained. M does not have a stack, so the stack operations can be done as per PDA P. Just the state operations we follow the Cartesian thing. So, just going through the details, the DFA M has states Q_1 and transition δ_1, start state q_1 and accepting states F_1. PDA P has states Q_2 and accepting states F_2.

$$M = (Q_1, \Sigma, \delta_1, q_1, F_1)$$

$$P = (Q_2, \Sigma, \Gamma, \delta_2, q_2, F_2)$$

And in the new PDA that is constructed, the states will be cross product $Q_1 \times Q_2$ and the stack of the new PDA will faithfully reflect the stack of P. That is possible because we know M does not have a stack. And for the transition functions, we use the same idea as before. It combines the movements of the transitions in M and P. And the accepting states will be like in the case of union of languages. We said the resulting states let us say (x_1, x_2) should be such that either x_1 is an accepting state of the first machine or x_2 is an accepting state of the second machine. Or, the key thing is that we had an or operation.

However, here we are talking about intersection. So, we want both M and the PDA to accept. Both x_1 should be an accepting state of the DFA M and x_2 should be an accepting state of the PDA P. In other words, this means that the set of accepting states of the new PDA should be just basically $F_1 \times F_2$. Basically, it is an ordered pair, where the first element comes from the accepting state of M and the second element comes from an accepting state of P. And if you recall, we even mentioned this briefly while discussing the proof of 'regular languages are closed under union'. We said that by changing the accept states into $F_1 \times F_2$ we get the closure under intersection.

So, this is the same thing. The accepting states are $F_1 \times F_2$. States are $Q_1 \times Q_2$ and the stack, because only one of them has a stack, that faithfully reproduces the stack contents. That is pretty much the idea, but otherwise, the details of the proof are pretty much the same. So, my suggestion is to work out the details of this proof just to give yourself an idea of how this proof goes about. And that is pretty much what I wanted to say in this lecture. So, we saw these three modifications of the PDA.

We can do these modifications, but still, it will be equivalent to the original PDA. We can modify it. We can convert it into a PDA with a single accepting state. We can convert it into a PDA where it empties the stack upon accepting. We can convert the PDA where every transition includes a push or a pop, but not both. And then we said that if A is regular and B is context free, then $A \cap B$ is necessarily context free and we gave the sketch of the proof and I suggest you to work out the details of the proof and that completes lecture 21. It also completes the lecture of week 4.

In this week, we first saw the Chomsky Normal Form, which is a specialised form for representing context free grammars. Then we saw a CYK algorithm, which is an efficient algorithm to check whether a given string is generated from a given grammar. You can apply this algorithm as long as the given grammar is in Chomsky Normal Form. We then saw closure properties of regular context free languages, including this one, which is also kind of a closure property. Intersection of regular and context free is context free.

And then we saw PDAs. We saw definitions and we saw some examples. And that completes week 4. In week 5 we will continue discussion on context free languages. So, in fact, we will see that the class of languages recognised by PDAs are also the same class as context free languages. So, there is an equivalence here much like regular languages are those that are recognised by DFAs as well as regular expressions. So, here also context free languages are those that are recognised by context free grammars as well as PDAs.

So, there is an equivalence. One is a language grammar based model and one is a machine based model. And we will see more, like why they are equivalent and why PDAs are equivalent to context free grammars. And we will see more details about context free languages in the upcoming week. So, see you next week, and thank you.