

Theory of Computation
Professor. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Equivalence of Context Free Grammars and Push Down Automata - Part 03

(Refer Slide Time: 00:16)

Equivalence of CFG's and PDA's A_{pq}

The rules of A are

→ For each $p, q, r, s \in Q, t \in \Gamma, a, b \in \Sigma$,
 If $(r, t) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, t)$
 add $A_{pq} \rightarrow ahs b$

→ For each $p, q, r \in Q$
 add $A_{pq} \rightarrow A_{pr} A_{rq}$

→ For all $p \in Q$, add $A_{pp} \rightarrow \epsilon$

Claim 2.30: If $A_{p,q}$ generates x , then x can take P from p with empty stack to q with


→ For each $p, q, r \in Q$
 add $A_{pq} \rightarrow A_{pr} A_{rq}$


→ For all $p \in Q$, add $A_{pp} \rightarrow \epsilon$


Claim 2.30: If $A_{p,q}$ generates x , then x can take P from p with empty stack to q with empty stack. (or retaining the stack)


Claim 2.31: If x can bring P from p with empty stack to q with empty stack, then A_{pq} generates x .

Proof of Claim 2.30: Induction on the number of steps in the derivation of x .









Hello, and welcome to Lecture 24 of the course Theory of Computation. In Lectures 22 and 23, we have been going through the proof of the equivalence of context-free grammars and push down automata. In 22, we saw that for given any grammar, we can construct an equivalent push down automata. In 23, we started proving that given any push down automata, we can construct

a context-free grammar. So, we in fact constructed the grammar in Lecture 23. And in, and there was some part of the proof that was yet to complete, which we will complete in Lecture 24.

So, the claim was that given a push down automata we can construct a context-free grammar that is equivalent to that. So, the rules of the grammar were the following. So, given a push down automata P , the grammar that was constructed included variables like this A_{pq} where we wanted the variable A_{pq} to derive all the strings that can take the push down automata from the state p to the state q on an empty stack. So, this is what the variables were. And the rules of the grammar were the following, which we said in that previous lecture. So, if p, q, r, s are states of the PDA, and t is a symbol in the stack alphabet of the PDA and a, b are from Σ_ϵ , which means they are part of the input alphabet, they are symbols of the input alphabet or they could be empty strings as well.

So, if p, q, r, s are states, t is a symbol in the stack alphabet and a and b are symbols in the input alphabet or empty strings, then if we check if (r, t) is part of $\delta(p, a, e)$ and (q, ϵ) is part of $\delta(s, b, t)$, if these two are there then you add the rule A_{pq} gives a $A_{rs}b$. So, a is a symbol of the input alphabet, b is a symbol of the input alphabet, a and b could either or both be an empty string(ϵ) and A_{rs} is another variable.

And the second rule was that for all the states p, q, r , we add the rule A_{pq} gives A_{pr} and A_{rq} . And the final set of rules was that for all the states p , we add A_{pp} yields epsilon. So, these are the three set of rules that we are adding. And for each set, there could be multiple such rules. For instance, the last rule, for each state in the PDA, we will add one such rule. And what we wanted to show was that if A_{pq} generates x then x can take the PDA from the state p with empty stack to the state q with empty stack and the converse of that. So, these are in Claim 2.31, and 2.30 and 2.31. So, first let us try to prove the proof of Claim 2.30.

So, statement is that if A_{pq} generates x then x can take the PDA from p with empty stack to q with empty stack. So, like I said in the previous lecture, if a string can take a PDA from p to q on an empty stack, it can also take it from p to q by retaining whatever the stack contents were. So, this is just to just to recall.

(Refer Slide Time: 04:32)

number of moves $\leq n$: maximum on the number of steps in the derivation of x .

Base Case: A_{pq} generates x in one step. The only possible one step derivation is $A_{pq} \rightarrow \epsilon$. Clearly E takes the PDA P from state p to p with empty stack. Base case is true.

Suppose true for $k \geq 1$ steps.

let $A_{pq} \Rightarrow x$ with $k+1$ steps. The first step is either $A_{pq} \rightarrow a \wedge s \wedge b$ or $A_{pq} \rightarrow A_{pr} A_{rq}$

Case 1: $A_{pq} \rightarrow a \wedge s \wedge b$

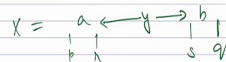
This means that $x = a y b$ and $A_{rs} \xrightarrow{k} y$ in k steps. This means y takes P from r to s on an empty stack, by induction.

\rightarrow a can take P from p to r and pushes t into the stack.

\rightarrow y takes P from r to s retaining the stack contents

\rightarrow b can take P from s to q , popping t from the stack.

So claim is true.



Case 2: $A_{pq} \rightarrow A_{pr} A_{rq}$

This means $x = y z$, where



$$\left. \begin{array}{l} A_{pn} \Rightarrow y \\ A_{nq} \Rightarrow z \end{array} \right\} \text{ both in } \leq k \text{ steps}$$

By induction, y takes P from p to n on an empty stack. And z takes P from n to q on an empty stack. So x can take P from p to q on an empty stack.

So claim is true in this case as well.

$p \longleftarrow y \rightarrow x \longleftarrow z \rightarrow q$

Claim 2.31: If x can bring P from p with empty stack to q with empty stack, then A_{pq} generates x .



\rightarrow b can take P from s to q , popping t from the stack.

So claim is true.

$x = \begin{array}{c} a \longleftarrow y \rightarrow b \\ \hline p \quad h \quad s \quad q \end{array}$

Case 2: $A_{pq} \rightarrow A_{pn} A_{nq}$
 $\Rightarrow \dots \Rightarrow yz = x$

This means $x = yz$, where

$$\left. \begin{array}{l} A_{pn} \Rightarrow y \\ A_{nq} \Rightarrow z \end{array} \right\} \text{ both in } \leq k \text{ steps}$$

By induction, y takes P from p to n on an



And the proof is by induction on the number of steps in the derivation of x . So, the claim is that if A_{pq} generates x , then it can do this. So, how many steps does the generation of x from A_{pq} take. So, this is the induction. So, before getting into the proof, I just want to say one more thing. So, the proof of Claim 2.30 and that of Claim 2.31 may seem like long proofs, but they are actually not that difficult, they are kind of very standard approaches and both of them use induction, and both of them are fairly straightforward things. So, do not be kind of scared by the proof or scared by the kind of terminology that is used in the proof. It is basic common sense that is applied to some settings. So, once you get past the notation and understand what is going on, it should not be that difficult. So, coming back to 2.30's proof, it is induction on the number of

steps in the derivation of x . So, when you say derivation, derivation from A_{pq} . So, the base case is that A_{pq} generates x in one step. So, if you look at the rules, these are the three types of rules. A_{pq} gives a A_{rs} b , A_{pq} gives $A_{pr}A_{rq}$ and A_{pp} gives epsilon. The only way that a variable generates a string comprising of entirely terminals in one step is if you take the third rule. So, the base case is that it generates string in one step. So, the only way we can get a string in one step is by taking the third rule.

So, the only possible one step derivation is the third rule, A_{pp} gives epsilon, for some state p . Which means only the string that is derived is empty string. So, the requirement is that if A_{pp} generates epsilon, we want to show that epsilon can take the PDA from the state p to itself on an empty stack. So, epsilon is empty string, an empty string kind of trivially keeps the PDA in the same state by retaining whatever the stack was. So, this is true in the base case. So, base case is true.

Now, let us move to the induction step. So, suppose it is true for some k steps. So, now we want to take the case where A_{pq} derives the string x in $k+1$ steps. Let us see what is the first step of the derivation, A_{pq} is a single variable, so the first step of the derivation is a rule of these types, A_{pq} gives a $A_{rs}b$ or A_{pq} gives $A_{pr}A_{rq}$. So, this gives us two cases, which we will see.

So, let us say the first step was this A_{pq} gives a $A_{rs}b$. So, recall that eventually, we want to get to x . So, somehow we want to get to x . Which means a $A_{rs}b$ is somehow derived into x . Which means x begins with a and ends with b . Again, recall that a and b are either symbols of the input alphabet or it could be empty strings as well. But they are not like longer strings of the input alphabet.

So this means that x is of the form a “some string” b , where A_{rs} derives the middle string. So, this means that x is of this form, $a y b$ where a and b are either single symbols from the input alphabet or empty strings and y is some string from the input alphabet. So, where y is in Σ_* , and A_{rs} must derive y . And A_{rs} will derive y in k steps because a and b are terminals. So, the rest of the derivation is merely A_{rs} deriving y . And so since one step is already over, A_{rs} derives y in k steps.

This means that since A_{rs} derives y in k steps, and since we, by induction, we may assume that y takes the PDA from the state r to the state s on an empty stack, this is what we can get by induction. This is the statement we want to prove for $k+1$. We can assume the same for k . So, y takes the PDA from r to s on an empty stack.

So, now we know that this is a rule A_{pq} gives $aA_{rs}b$ is a rule because that is the first step. So, that must be a rule. But why is this a rule? When do we add such a rule? if these conditions are satisfied i.e the conditions being (r, t) is part of $\delta(p, a, \epsilon)$ and (q, ϵ) is part of $\delta(s, b, t)$. So, this means that the PDA can read a and go from p to the state r by pushing t into the stack.

And also, the PDA can go from state s to state q by reading b from the input and by popping t from the stack. That is what this rule states. So, a can take the PDA from State p to State r by pushing t and b can take the PDA from State s to State q by popping t . And by induction, what we have already explained, y takes the PDA from r to s by retaining the stack contents. y takes PDA from r to s on an empty stack which means y can also take the PDA from r to s by retaining the stack contents.

So, which means, this, now let us say initially, the stack was empty. By reading a , the PDA pushes t into the stack and then goes to the state r , by reading a , it goes to the state r . State p initially, and then state r . And then there is a string y . By reading y , the PDA can go to the state s on an empty stack or by retaining the stack contents. And finally, by reading the symbol b , it pops the stack content, the symbol t from the stack. And then, it takes it from s to t .

It takes the stack from s to t , which means the string $a y b$, takes the PDA from p to the state q on an empty stack. Once again, a takes this PDA from p to r by pushing the content t , y takes the PDA from r to s by retaining the stack content, and b takes the PDA from s to q by popping t . So, $a y b$ takes the PDA from p to q on an empty stack.

And we know that $a y b$ is nothing but x . So, we know that x can take the PDA from p to q on an empty stack, which is what we want to show. So, this is the Case 1 of the proof. So, the next part is that, what if the first step of derivation was this A_{pq} gives $A_{pr} A_{rs}$?

This means, like I said before, some rules are applied, and finally we should get x . But how can that happen? The only way this can happen is the variable A_{pr} generates some y and the variable

A_{rq} generates z , where x is yz . So, this means that x can be generated like this. So, the first step is already done. A_{pq} gives $A_{pr} A_{rq}$. So, the remaining number of steps is k .

So, which means A_{pr} must generate y in some number of steps, A_{rq} must generate z in some number of steps. Both the number of steps is at most k , because together, we are using only k plus 1, and 1 is already over. So, now since both of them use less than k steps, we can use induction to infer that y takes the PDA from p to r on an empty stack, and z takes the PDA from r to s on an empty stack, because we assume that for all the strings that have less than k steps of derivation, so this is what we said here, suppose it is true for k steps. And now we are trying to prove something that requires $k+1$ steps.

So, by induction y takes p from p to r on an empty stack and z takes p from r to s on an empty stack. So, x is just the concatenation of y and z , which means x takes the PDA from p to q on an empty stack. So, the claim is true in this case as well because you can think of it like this. So, y and z together form x . y takes it from p to r , and z takes it from r to q . So, the entire string yz which is the same as x , takes it from p to q on an empty stack.

So, we have shown that whatever may be the case, whether it is the first rule applied is A_{pq} gives a $A_{rs} b$ or A_{pq} gives $A_{pr} A_{rq}$, whatever be the case, we have shown that the string that is generated will take it from p to q on an empty stack. So, this completes the proof of Claim 2.30, which says that if a string is generated by A_{pq} , that string can take the PDA from p to q on an empty stack or by retaining the stack content.

(Refer Slide Time: 17:43)



Claim 2.31: If x can bring P from p with empty stack to q with empty stack, then A_{pq} generates x .

Proof: Induction on the number of steps in the computation of P .

Base case: The computation has zero steps.

This means P has to start and end at the same state, say p . In zero steps, P can only process the empty string ϵ , so $x = \epsilon$. We know that $A_{pp} \rightarrow \epsilon$ is a rule for all states p .

Base case: The computation has zero steps.

This means P has to start and end at the same state, say p . In zero steps, P can only process the empty string ϵ , so $x = \epsilon$. We know that $A_{pp} \rightarrow \epsilon$ is a rule for all states p .

Induction: Assume claim is true when P needs $\leq k$ steps. Let P have a computation where x brings P from p to q with empty stack using $k+1$ steps. We need to show that A_{pq} generates x . There are two cases.

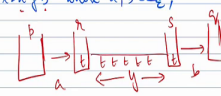
Case 1: The stack never becomes empty. This means that the symbol pushed in step 1 is

$1 \ 2 \ 3 \ \dots \ k \ \dots \ 1 \ 2 \ 3 \ \dots \ k$



Popped in step $k+1$. say the symbol is $t \in \Sigma$
 let the first symbol read from the input be
 a and the last symbol read is $b \in \Sigma$
 $\epsilon \in \Sigma$

This means that $x = ayb$ where $a, b \in \Sigma$,
 and $y \in \Sigma^*$



$\rightarrow a$ takes P from p to r by pushing t
 $\rightarrow b$ takes P from s to q by popping t .

This implies that y takes P from r to s
 retaining the stack in $k-1$ steps.

By induction, we have $A_{rs} \Rightarrow y$

By assumption, $(r, t) \in \delta(p, a, \epsilon)$
 and $(q, \epsilon) \in \delta(s, b, t)$

$\rightarrow b$ takes r from s to q by popping t .

This implies that y takes P from r to s
 retaining the stack in $k-1$ steps.

By induction, we have $A_{rs} \Rightarrow y$

By assumption, $(r, t) \in \delta(p, a, \epsilon)$
 and $(q, \epsilon) \in \delta(s, b, t)$

These two mean that $A_{pq} \rightarrow a A_{rs} b$
 is a rule. \Downarrow^*
 $a y b$

Since $x = ayb$, this means $A_{pq} \Rightarrow x$.

Case 2: The stack empties in the middle. let
 the stack become empty when P is at the state r .



So, next thing to be proved is Claim 2.31 which is the other direction. Suppose there is a string x that can take the PDA from p to q on an empty stack. Then that string is generated by A_{pq} . Like I said before, this is also an induction which is straightforward. So, here the assumption is that the string takes the PDA from p to q on an empty stack. So, how many steps of the PDA, how many step, transitions of the PDA is required to generate that string. So, the induction is on that number of steps.

So, induction on the number of steps required by the PDA in the PDA computation. So, the simplest case is when the computation has zero steps, the PDA does not do anything. So, when, in zero steps the PDA is not able to read anything, which means the string corresponding to that

is the only one string that can be read without any number of steps, which is the empty string. So, the empty string keeps the PDA at the same state or can keep the PDA in the same state because that is the only thing that it can do because we assume that the computation has zero steps.

So, in zero steps, the PDA can only process the empty string. So, the empty string is empty. And let us say it was in state, some state small p , and we know that, we know that A_{pp} gives empty string is a rule. So, this empty string is generated by the grammar. So if the computation of the PDA has zero steps, the only string possible is empty string, and that is generated by the grammar. So, now, we have to consider the computations of a bigger number.

So, suppose the claim is true whenever the PDA requires some k steps for some k . So, now we want to show it for $k+1$ steps. So, now we have a string x which takes the PDA from p to q on an empty stack using $k+1$ steps. And then we have to show that A_{pq} generates x . So, like in the other claim, we have two cases. So, assumption is that for all the strings that take PDA from one state to another using k steps or less, the grammar, the corresponding variable in the grammar generates this.

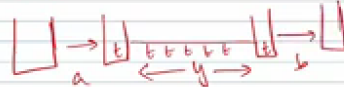
Now, we are considering such a string for which the PDA requires $k+1$ steps. So, there are two cases which we had kind of referred to in the previous lecture. First case is the stack never becomes empty. So, the PDA starts, it just takes from empty stack to empty stack. So, we know that the first step involves a push, and the last step, again we ended in empty stack, which means it must involve a pop. So, if the stack never becomes empty, the symbol that was pushed in the first step must be popped only in the last step or by $k+1$ step. So, assume that the symbol is t . So, t is something in the tape alphabet. Let the first symbol that is read from the input be a i.e when pushing that symbol t , the PDA must be reading something from the input tape, let that symbol be a and in the $k+1$ step, the symbol be b .

So, notice that, recall that a is part of Σ_ϵ and b is also part of Σ_ϵ . So, a and b could also be empty string. This means that the first symbol x is a , if a is not empty, and the last symbol of x is b . So, we can write x as $y b$ for some string y .



popped in step $k+1$. say the symbol is $t \in \Sigma$
 let the first symbol read from the input be
 a and the last symbol read is $b \in \Sigma$
 $\in \Sigma$

This means that $x = ayb$ where $a, b \in \Sigma$,
 and $y \in \Sigma^*$



→ a takes P from p to r by pushing t

→ b takes P from s to q by popping t .

This implies that y takes P from r to s
 retaining the stack in $k-1$ steps.

By induction, we have $A_{rs} \Rightarrow y$

By assumption, $(r, t) \in \delta(p, a, \epsilon)$
 and $(q, \epsilon) \in \delta(s, b, t)$



(Refer Slide Time: 23:34)

So, we know that ' a ' takes the PDA from p to r by pushing t and b takes a PDA from s to q by popping t . This also means that y takes the PDA from r to s . So, initially it was empty and upon reading the symbol a , you push a ' t '. And then you read ' y ' which eventually gives us the same situation, the same ' t ' is still retained. It never becomes empty. So, always, this ' t ' is there, for all the intermediate steps. And finally, b takes out this ' t '. So, this means y could also take the PDA from r to s . So, the state here is small $p r s$ and q . So, this means that y could also take the PDA from r to s on an empty stack because it retains the stack means it can also take it from r , an empty stack, to s , an empty stack.

So, by induction we assume y takes it in $k-1$ steps because one step, the beginning step is by reading a , and the last step is by reading b . So, y takes it from r to s by retaining the stack in $k-1$ steps. By induction assumption, so the assumption was that any string that takes k steps or less is generated by the corresponding A_{pq} . So, this means that A_{rs} generates y . And since, by already we observed that a takes p from p to r by pushing t , b takes p from popping t from s to q .

This means that because we have these two conditions satisfied, this means that A_{pq} gives $A_{rs} b$ is a rule. So, we know that A_{pq} gives a $A_{rs} b$, and we have to show that A_{pq} gives x . We

already have seen that A_{rs} gives y . Since, we already have seen that A_{rs} gives y , which means a y b is generated, but what is a y b ? We already assumed that x was a y b . So, this means A_{pq} derives x , which is what we wanted to show.

So, if the stack never became empty which meant that this symbol pushed in the Step 1 is the same as the symbol popped in this last step, now that allows us to infer that a pushed something and b popped, there are rules where you push something initially and the same thing is popped at the end, which which gave us these two conditions. And by induction we could infer that A_{rs} derives y , and combining all this, we get that A_{pq} derives x . The next case, so this is, recall, this was Case 1, the stack never becomes empty.

(Refer Slide Time: 26:59)

Case 2: The stack empties in the middle. Let the stack become empty when P is at the state r .

Let y be the substring that brings P from p to r in $l \leq k$ steps with empty stack.

Let z be the substring that brings P from r to s in $k+1-l \leq k$ steps, again with empty stack.

We have $x = yz$, where by induction

$$A_{pr} \Rightarrow y \text{ and } A_{rs} \Rightarrow z$$

$$A_{pq} \rightarrow A_{pr} A_{rs} \Rightarrow yz = x$$

Thus $A_{pq} \Rightarrow x$.



let y be the substring that brings P from p to r in $l \leq k$ steps with empty stack.

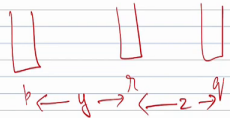
let z be the substring that brings P from r to s in $k+1-l \leq k$ steps, again with empty stack.

We have $x = yz$, where by induction

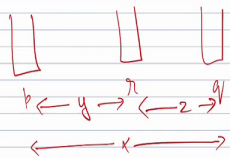
$$A_{pr} \Rightarrow y \text{ and } A_{rq} \Rightarrow z$$

$$A_{pq} \rightarrow A_{pr} A_{rq} \Rightarrow yz = x.$$

Thus $A_{pq} \Rightarrow x$.



Thus $A_{pq} \Rightarrow x$.



Theorem: A language is Context-Free if and only if it is recognized by a PDA.



The next case was when the stack becomes empty somewhere in the middle. So, which means, initially you were at State p , then finally, you are at state q and let us say it became empty in state r , somewhere in the middle. So, suppose the entire string is x . Suppose the string that took you from p to r was y , and the string that took you from r to this q was z , which means x is $y z$.

Now to process x , it requires $k+1$ steps, that is by assumption. Now, we know that y takes the PDA from p to r in less than or equal to k steps, and z takes the PDA from r to q in less than or equal to k steps because both of these movements have take at least one step. Hence, so when the stack empties by the middle, y is the substring that takes the PDA from p to r , in some l number

of steps, where l is at most k , and z takes the PDA from r to s in some $k+1-l$ steps, again which is less than or equal to k .

And we assume that x is yz . By induction, since y took the PDA from p to r in less than or equal to k steps on an empty stack, it follows that A_{pr} derives y . That is the induction assumption. Whenever there is some string that takes it from state p to state r in k steps or less, A_{pr} derives that string. So, since y takes it from p to r in k steps or less, A_{pr} derives y . And since z takes it from r to q in less than or equal to k steps, r q derives z .

We know that A_{pq} derives $A_{pr} A_{rq}$ is a rule, for all triplets p q and r such a rule is there. And we know that A_{pr} derives y and A_{rq} derives z . So, A_{pr} or A_{rq} derive y and z respectively, and that y and z together form x . So, this gives us that A_{pq} derives x . So, even in this case when the stack empties in the middle, we are able to see that any string that takes the PDA from p to q is derived by the variable A_{pq} . And that completes the proof of claim 2.31.

So, just to summarize, 2.30 said that if A_{pq} generates a string, then that string can take the PDA from p to q on an empty stack, or by retaining the stack. 2.31 said the converse. If a string x can take the PDA from p to q on an empty stack, then that string is generated by the variable A_{pq} . So, together, we get that A_{pq} generates exactly those set of variables that take this PDA from p to q on an empty stack.

And that was a missing piece for us to show that the grammar that we generated, that we constructed here is equivalent to the, to the PDA that we began with. So, this completes the proof of the equivalence of the grammar constructed with the PDA. And together with that we are also completing the proof of the fact that any PDA, or the fact that a language is context-free if and only if it is recognized by a PDA.

Which means both of them generate or both of them recognize exactly the same class of languages, which is context-free languages. So A language is context-free if and only if it is recognized by a PDA. So, the class of language recognized by PDAs are exactly the same as the class of context-free languages.

So, this completes the proof. Even though, as I said, even though this proof, 2.30 and 2.31 may have seemed long, they are not really that complex. We are just using the basic induction and we are trying to understand what is happening in the derivation of a string from the grammar and the processing of a string by the PDA. And we see that there is a direct equivalence.

And this completes the equivalence of CFGs and PDAs. In the next lecture, we will see ways to show that languages are not context-free. So, just like we had pumping lemma for regular languages, we will see pumping lemma for context-free languages in the next lecture. So, I think we will stop Lecture 24 for now, and see you in the next lecture, lecture number 25.