

# CS 301

## Lecture 13 – Closure properties of context-free languages

Stephen Checkoway

March 5, 2018



# CFLs and PDAs

## Theorem

*Every context-free language can be recognized by some PDA.*

## Proof idea.

- 1 Use the PDA's stack to perform a left-most derivation of a word in the language
- 2 Match the PDA's input symbols against the stack, popping each one
- 3 Accept if stack is empty when there's no more input

## What we'd like to do

Consider the language  $A = \{w \mid w \in \{a, b\}^* \text{ and } w \text{ is not a palindrome}\}$

What CFG generates that language?

## What we'd like to do

Consider the language  $A = \{w \mid w \in \{a, b\}^* \text{ and } w \text{ is not a palindrome}\}$

What CFG generates that language?

$$S \rightarrow aSa \mid bSb \mid aTb \mid bTa$$

$$T \rightarrow aT \mid bT \mid \varepsilon$$

## What we'd like to do

Consider the language  $A = \{w \mid w \in \{a, b\}^* \text{ and } w \text{ is not a palindrome}\}$

What CFG generates that language?

$$S \rightarrow aSa \mid bSb \mid aTb \mid bTa$$

$$T \rightarrow aT \mid bT \mid \varepsilon$$

A left-most derivation of the string  $abaaa$  is

$$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow abaTaa \Rightarrow abaaa.$$

We want to start by pushing  $S$  on the stack, then performing the derivation step by step so that  $abaaa$  ends on the stack, and then match the input

## What we'd like to do

Consider the language  $A = \{w \mid w \in \{a, b\}^* \text{ and } w \text{ is not a palindrome}\}$

What CFG generates that language?

$$S \rightarrow aSa \mid bSb \mid aTb \mid bTa$$

$$T \rightarrow aT \mid bT \mid \varepsilon$$

A left-most derivation of the string  $abaaa$  is

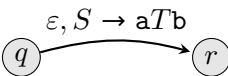
$$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow abaTaa \Rightarrow abaaa.$$

We want to start by pushing  $S$  on the stack, then performing the derivation step by step so that  $abaaa$  ends on the stack, and then match the input

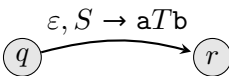
There are two complications

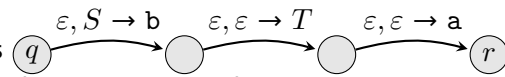
- 1 The first step in the derivation  $S \Rightarrow aSa$  requires popping one symbol and pushing three
- 2 We can only replace symbols at the top of the stack

## Pushing multiple symbols

We would like to write a transition like   
but  $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$  doesn't allow that

## Pushing multiple symbols

We would like to write a transition like   
but  $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$  doesn't allow that

Instead, use multiple transitions   
Note that the symbols are pushed on in reverse order



## We can only replace symbols at the top of the stack

Rather than first deriving the whole string on the stack and then matching the input,

- If the top of the stack is a terminal, match it to the next input symbol



- If the top of the stack is a variable, replace it with the RHS of a corresponding rule

# We can only replace symbols at the top of the stack

Rather than first deriving the whole string on the stack and then matching the input,

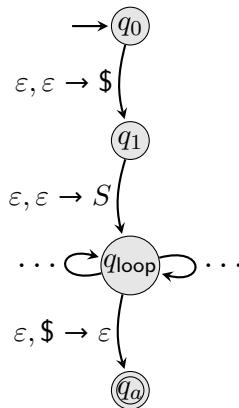
- If the top of the stack is a terminal, match it to the next input symbol



- If the top of the stack is a variable, replace it with the RHS of a corresponding rule

In fact, we only need four main states plus any additional states necessary to push multiple symbols

The  $q_{\text{loop}}$  state is where all the real work happens

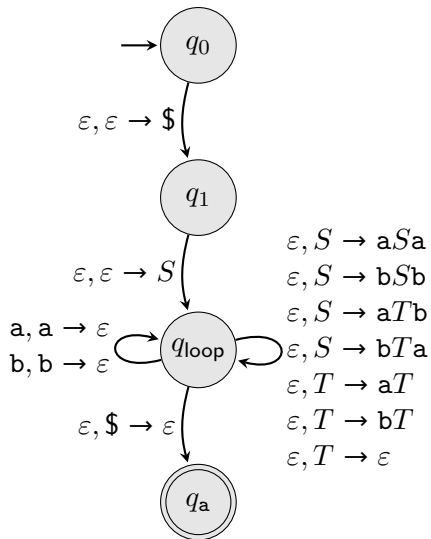


## Example

$$S \rightarrow aSa \mid bSb \mid aTb \mid bTa$$

$$T \rightarrow aT \mid bT \mid \varepsilon$$

- ① For each  $t \in \Sigma$ , add the transition  $t, t \rightarrow \varepsilon$  from  $q_{\text{loop}}$  to  $q_{\text{loop}}$
- ② For each rule  $A \rightarrow u_1 u_2 \cdots u_n$  for  $u_i \in V \cup \Sigma$ , add  $n - 1$  new states (if  $n > 1$ ) and transitions to pop  $A$  and push  $u_1, u_2, \dots, u_n$  on in reverse order



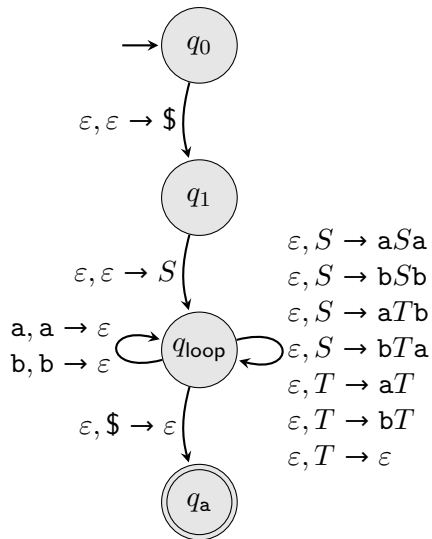
[The rules on the right need 10 extra states to make this a proper PDA]

# Running the PDA on some input

Consider running the PDA on the input abaaa. The stack is shown on the right after each step

➊ push \$;

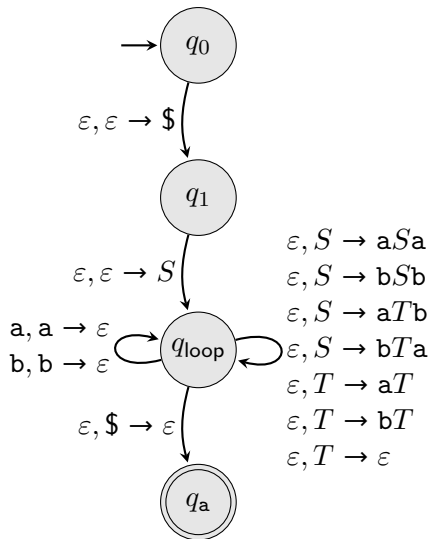
\$



## Running the PDA on some input

Consider running the PDA on the input abaaa. The stack is shown on the right after each step

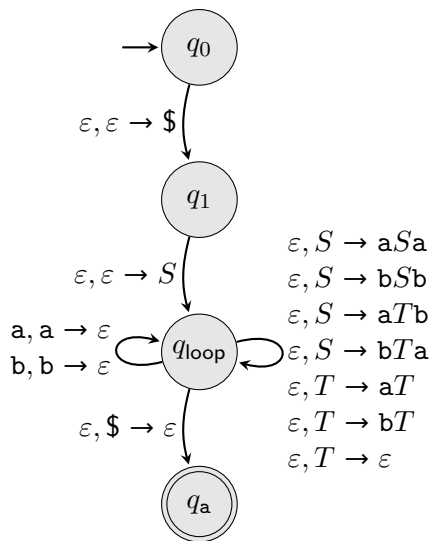
- ① push \$; \$
- ② push  $S$ ;  $SS$



# Running the PDA on some input

Consider running the PDA on the input  $abaaa$ . The stack is shown on the right after each step

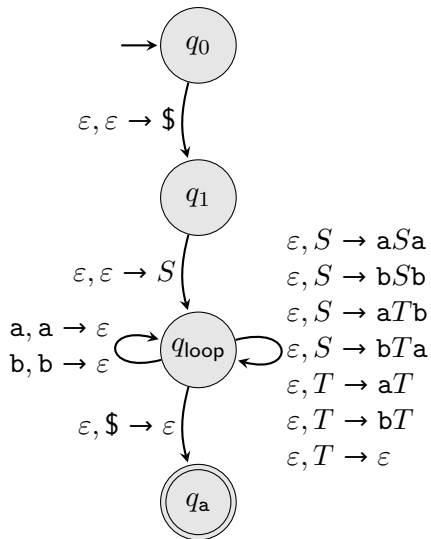
- ➊ push  $\$$ ;  $\$$
- ➋ push  $S$ ;  $S\$$
- ➌ pop  $S$ , push  $aSa$ ;  $aSa\$$



# Running the PDA on some input

Consider running the PDA on the input  $abaaa$ . The stack is shown on the right after each step

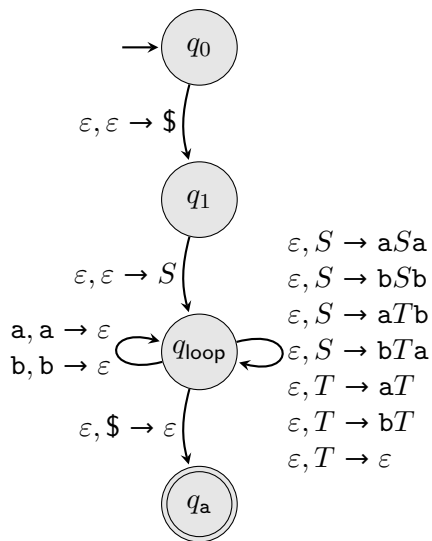
- ➊ push  $\$$ ;  $\$$
- ➋ push  $S$ ;  $S\$$
- ➌ pop  $S$ , push  $aSa$ ;  $aSa\$$
- ➍ read and pop  $a$ ;  $Sa\$$



# Running the PDA on some input

Consider running the PDA on the input  $abaaa$ . The stack is shown on the right after each step

- ➊ push  $\$$ ;  $\$$
- ➋ push  $S$ ;  $S\$$
- ➌ pop  $S$ , push  $aSa$ ;  $aSa\$$
- ➍ read and pop  $a$ ;  $Sa\$$
- ➎ pop  $S$ , push  $bTa$ ;  $bTaa\$$

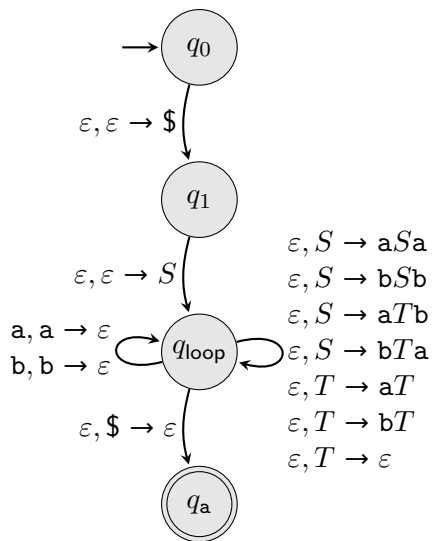




# Running the PDA on some input

Consider running the PDA on the input  $abaaa$ . The stack is shown on the right after each step

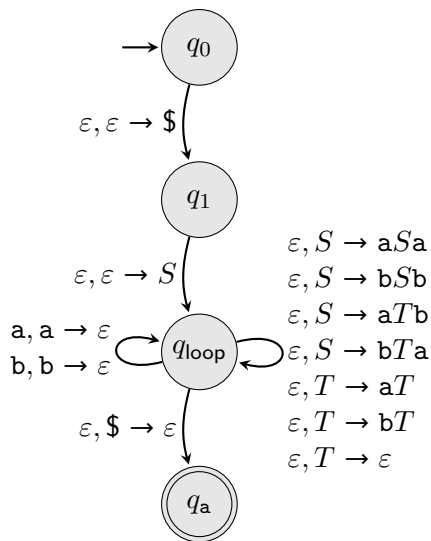
- |                          |          |
|--------------------------|----------|
| ➊ push $\$$ ;            | $\$$     |
| ➋ push $S$ ;             | $S\$$    |
| ➌ pop $S$ , push $aSa$ ; | $aSa\$$  |
| ➍ read and pop $a$ ;     | $Sa\$$   |
| ➎ pop $S$ , push $bTa$ ; | $bTaa\$$ |
| ➏ read and pop $b$ ;     | $Taa\$$  |



# Running the PDA on some input

Consider running the PDA on the input  $abaaa$ . The stack is shown on the right after each step

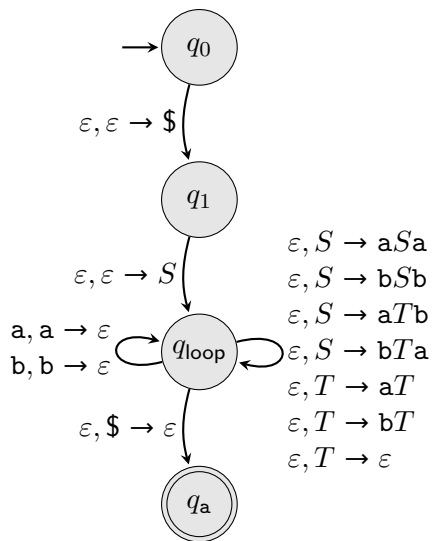
- |                          |           |
|--------------------------|-----------|
| ➊ push \$;               | \$        |
| ➋ push $S$ ;             | $S$ \$    |
| ➌ pop $S$ , push $aSa$ ; | $aSa$ \$  |
| ➍ read and pop $a$ ;     | $Sa$ \$   |
| ➎ pop $S$ , push $bTa$ ; | $bTaa$ \$ |
| ➏ read and pop $b$ ;     | $Taa$ \$  |
| ➐ pop $T$ , push $aT$ ;  | $aTaa$ \$ |



# Running the PDA on some input

Consider running the PDA on the input  $abaaa$ . The stack is shown on the right after each step

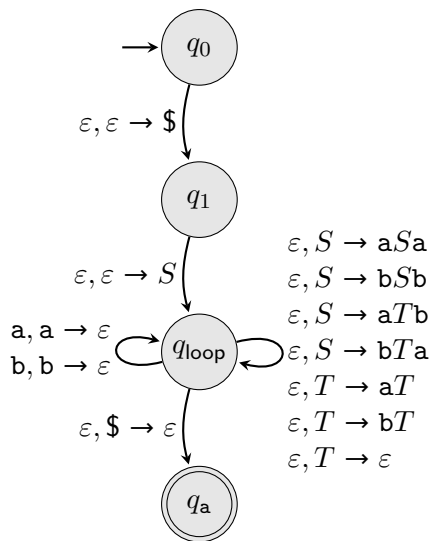
- |                          |           |
|--------------------------|-----------|
| ➊ push \$;               | \$        |
| ➋ push $S$ ;             | $S$ \$    |
| ➌ pop $S$ , push $aSa$ ; | $aSa$ \$  |
| ➍ read and pop $a$ ;     | $Sa$ \$   |
| ➎ pop $S$ , push $bTa$ ; | $bTaa$ \$ |
| ➏ read and pop $b$ ;     | $Taa$ \$  |
| ➐ pop $T$ , push $aT$ ;  | $aTaa$ \$ |
| ➑ read and pop $a$ ;     | $Taa$ \$  |



# Running the PDA on some input

Consider running the PDA on the input  $abaaa$ . The stack is shown on the right after each step

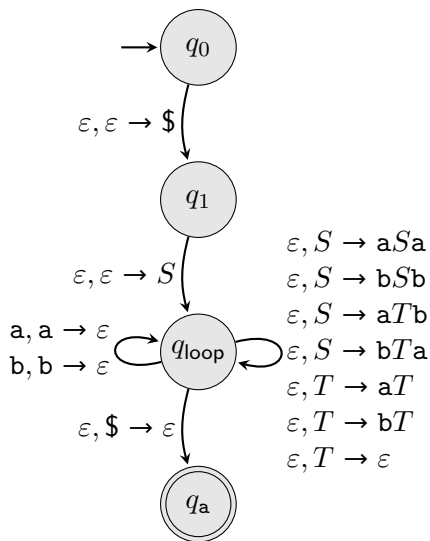
- |                               |          |
|-------------------------------|----------|
| ➊ push $\$$ ;                 | $\$$     |
| ➋ push $S$ ;                  | $S\$$    |
| ➌ pop $S$ , push $aSa$ ;      | $aSa\$$  |
| ➍ read and pop $a$ ;          | $Sa\$$   |
| ➎ pop $S$ , push $bTa$ ;      | $bTaa\$$ |
| ➏ read and pop $b$ ;          | $Taa\$$  |
| ➐ pop $T$ , push $aT$ ;       | $aTaa\$$ |
| ➑ read and pop $a$ ;          | $Taa\$$  |
| ➒ pop $T$ , push $\epsilon$ ; | $aa\$$   |



# Running the PDA on some input

Consider running the PDA on the input  $abaaa$ . The stack is shown on the right after each step

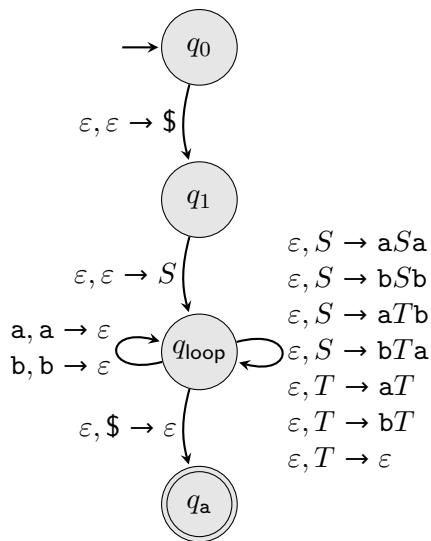
- |                               |          |
|-------------------------------|----------|
| ➊ push $\$$ ;                 | $\$$     |
| ➋ push $S$ ;                  | $S\$$    |
| ➌ pop $S$ , push $aSa$ ;      | $aSa\$$  |
| ➍ read and pop $a$ ;          | $Sa\$$   |
| ➎ pop $S$ , push $bTa$ ;      | $bTaa\$$ |
| ➏ read and pop $b$ ;          | $Taa\$$  |
| ➐ pop $T$ , push $aT$ ;       | $aTaa\$$ |
| ➑ read and pop $a$ ;          | $Taa\$$  |
| ➒ pop $T$ , push $\epsilon$ ; | $aa\$$   |
| ➓ read and pop $a$ ;          | $a\$$    |



# Running the PDA on some input

Consider running the PDA on the input  $abaaa$ . The stack is shown on the right after each step

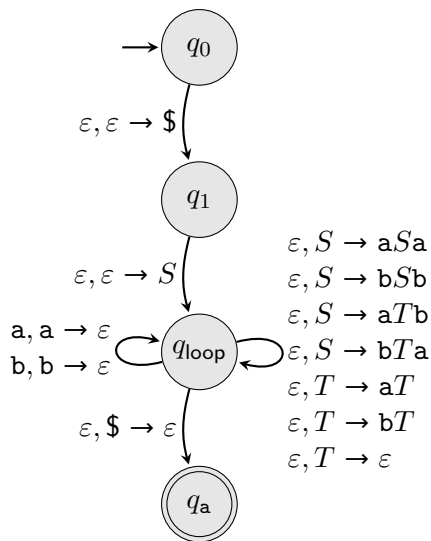
- |                               |           |
|-------------------------------|-----------|
| ➊ push \$;                    | \$        |
| ➋ push $S$ ;                  | $S$ \$    |
| ➌ pop $S$ , push $aSa$ ;      | $aSa$ \$  |
| ➍ read and pop $a$ ;          | $Sa$ \$   |
| ➎ pop $S$ , push $bTa$ ;      | $bTaa$ \$ |
| ➏ read and pop $b$ ;          | $Taa$ \$  |
| ➐ pop $T$ , push $aT$ ;       | $aTaa$ \$ |
| ➑ read and pop $a$ ;          | $Taa$ \$  |
| ➒ pop $T$ , push $\epsilon$ ; | $aa$ \$   |
| ➓ read and pop $a$ ;          | $a$ \$    |
| ➔ read and pop $a$ ;          | \$        |



# Running the PDA on some input

Consider running the PDA on the input  $abaaaa$ . The stack is shown on the right after each step

- |                               |            |
|-------------------------------|------------|
| 1 push \$;                    | \$         |
| 2 push $S$ ;                  | $S$ \$     |
| 3 pop $S$ , push $aSa$ ;      | $aSa$ \$   |
| 4 read and pop $a$ ;          | $Sa$ \$    |
| 5 pop $S$ , push $bTa$ ;      | $bTaa$ \$  |
| 6 read and pop $b$ ;          | $Taa$ \$   |
| 7 pop $T$ , push $aT$ ;       | $aTaa$ \$  |
| 8 read and pop $a$ ;          | $Taa$ \$   |
| 9 pop $T$ , push $\epsilon$ ; | $aa$ \$    |
| 10 read and pop $a$ ;         | $a$ \$     |
| 11 read and pop $a$ ;         | \$         |
| 12 pop \$ and accept;         | $\epsilon$ |



# Proving that every CFL is recognized by a PDA

Proof.

Let  $A$  be a CFL generated by a CFG  $G = (V, \Sigma, R, S)$ .



# Proving that every CFL is recognized by a PDA

Proof.

Let  $A$  be a CFL generated by a CFG  $G = (V, \Sigma, R, S)$ .

Construct the PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$  with states  $Q = \{q_0, q_1, q_{\text{loop}}, q_a\} \cup E$  where  $E$  are the extra states we need for each rule and  $\Gamma = V \cup \Sigma \cup \{\$\}$ .

# Proving that every CFL is recognized by a PDA

Proof.

Let  $A$  be a CFL generated by a CFG  $G = (V, \Sigma, R, S)$ .

Construct the PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$  with states  $Q = \{q_0, q_1, q_{\text{loop}}, q_a\} \cup E$  where  $E$  are the extra states we need for each rule and  $\Gamma = V \cup \Sigma \cup \{\$\}$ .

Start with then transitions

$\varepsilon, \varepsilon \rightarrow \$$  from  $q_0$  to  $q_1$ ,

$\varepsilon, \varepsilon \rightarrow S$  from  $q_1$  to  $q_{\text{loop}}$ , and

$\varepsilon, \$ \rightarrow \varepsilon$  from  $q_{\text{loop}}$  to  $q_a$

# Proving that every CFL is recognized by a PDA

Proof.

Let  $A$  be a CFL generated by a CFG  $G = (V, \Sigma, R, S)$ .

Construct the PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$  with states  $Q = \{q_0, q_1, q_{\text{loop}}, q_a\} \cup E$  where  $E$  are the extra states we need for each rule and  $\Gamma = V \cup \Sigma \cup \{\$\}$ .

Start with then transitions

$\varepsilon, \varepsilon \rightarrow \$$  from  $q_0$  to  $q_1$ ,

$\varepsilon, \varepsilon \rightarrow S$  from  $q_1$  to  $q_{\text{loop}}$ , and

$\varepsilon, \$ \rightarrow \varepsilon$  from  $q_{\text{loop}}$  to  $q_a$

For each  $t \in \Sigma$ , add the transition  $t, t \rightarrow \varepsilon$  from  $q_{\text{loop}}$  to  $q_{\text{loop}}$ .

# Proving that every CFL is recognized by a PDA

Proof.

Let  $A$  be a CFL generated by a CFG  $G = (V, \Sigma, R, S)$ .

Construct the PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$  with states  $Q = \{q_0, q_1, q_{\text{loop}}, q_a\} \cup E$  where  $E$  are the extra states we need for each rule and  $\Gamma = V \cup \Sigma \cup \{\$\}$ .

Start with then transitions

$\varepsilon, \varepsilon \rightarrow \$$  from  $q_0$  to  $q_1$ ,

$\varepsilon, \varepsilon \rightarrow S$  from  $q_1$  to  $q_{\text{loop}}$ , and

$\varepsilon, \$ \rightarrow \varepsilon$  from  $q_{\text{loop}}$  to  $q_a$

For each  $t \in \Sigma$ , add the transition  $t, t \rightarrow \varepsilon$  from  $q_{\text{loop}}$  to  $q_{\text{loop}}$ .

For each rule  $A \rightarrow u$  add the states and transitions necessary to pop  $A$  and push  $u$  in reverse order from  $q_{\text{loop}}$  to  $q_{\text{loop}}$ .

## Proof continued

Consider running  $M$  on input  $w = w_1w_2\cdots w_n$  for  $w_i \in \Sigma$ .

The first time  $M$  enters state  $q_{\text{loop}}$ , the stack is  $S\$$  and no input has been read.

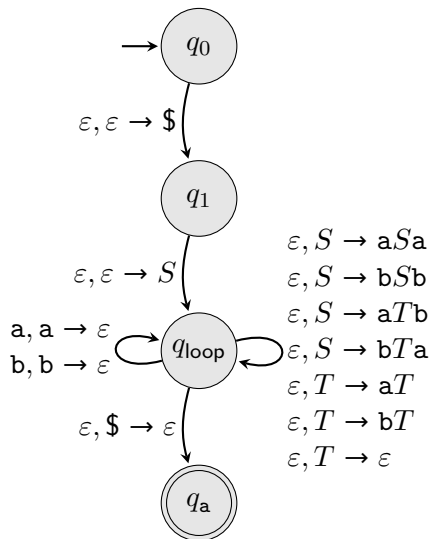
Every subsequent time it enters  $q_{\text{loop}}$ , the input read so far concatenated with the stack is a step in some left-most derivation of  $w$  (followed by a  $\$$ ).

I.e., if  $k$  symbols have been read from the input and the stack is  $s$ , then  $w_1w_2\cdots w_k s$  is a step in the derivation of  $w$

## Returning to the example

$$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow abaTaa \Rightarrow abaaa$$

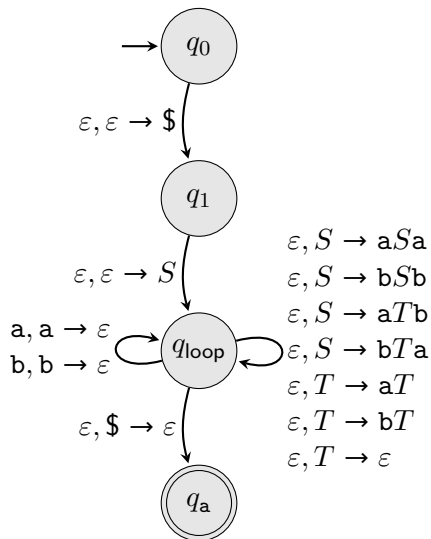
State	Action	Input read	Stack
$q_0$	push \$	$\epsilon$	\$



## Returning to the example

$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow abaTaa \Rightarrow abaaa$

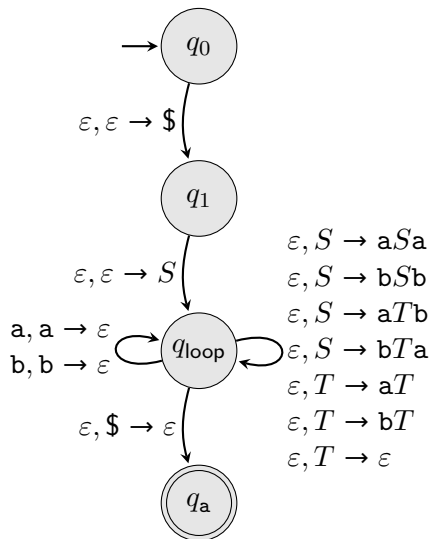
State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$



## Returning to the example

$S \Rightarrow \text{aSa} \Rightarrow \text{abTaa} \Rightarrow \text{abaTaa} \Rightarrow \text{abaaa}$

State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{\text{loop}}$	pop $S$ , push $\text{aSa}$	$\epsilon$	$\text{aSa}\$$

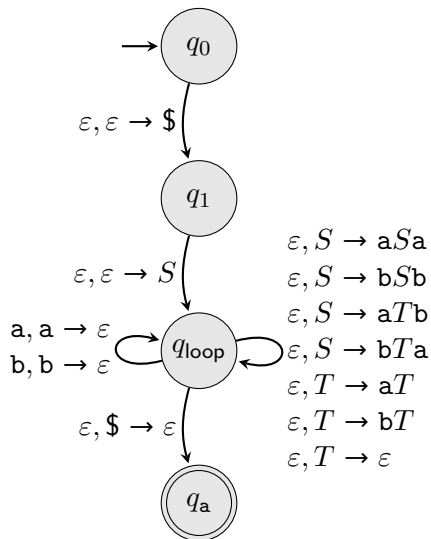




## Returning to the example

$S \Rightarrow \text{aSa} \Rightarrow \text{abTaa} \Rightarrow \text{abaTaa} \Rightarrow \text{abaaa}$

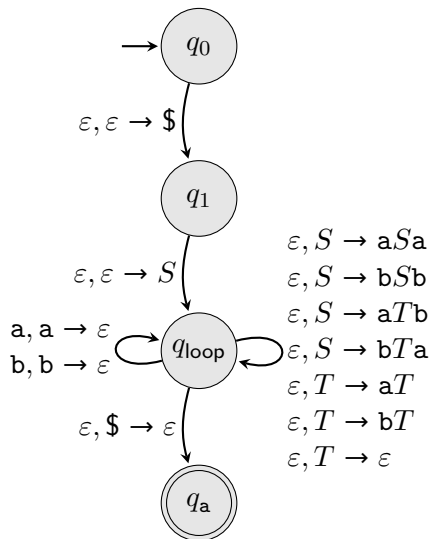
State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{\text{loop}}$	pop $S$ , push $\text{aSa}$	$\epsilon$	$\text{aSa}\$$
$q_{\text{loop}}$	read and pop $\text{a}$	$\text{a}$	$\text{Sa}\$$



## Returning to the example

$S \Rightarrow aSa \Rightarrow \textcolor{red}{abTaa} \Rightarrow abaTaa \Rightarrow abaaa$

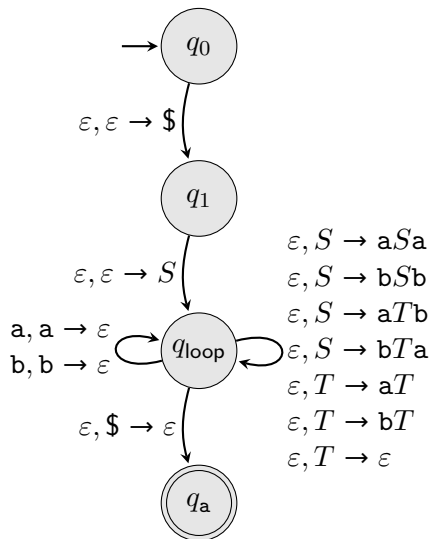
State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{\text{loop}}$	pop $S$ , push $aSa$	$\epsilon$	$aSa\$$
$q_{\text{loop}}$	read and pop $a$	$a$	$Sa\$$
$q_{\text{loop}}$	pop $S$ , push $bTa$	$\textcolor{red}{a}$	$\textcolor{red}{bTaa}\$$



## Returning to the example

$S \Rightarrow aSa \Rightarrow \textcolor{red}{abTaa} \Rightarrow abaTaa \Rightarrow abaaa$

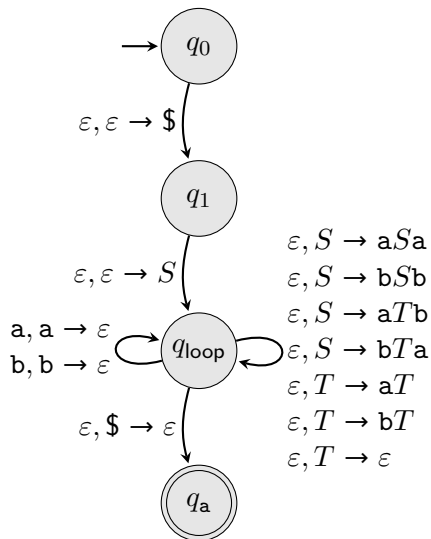
State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{\text{loop}}$	pop $S$ , push $aSa$	$\epsilon$	$aSa\$$
$q_{\text{loop}}$	read and pop $a$	$a$	$Sa\$$
$q_{\text{loop}}$	pop $S$ , push $bTa$	$a$	$bTaa\$$
$q_{\text{loop}}$	read and pop $b$	$\textcolor{red}{ab}$	$\textcolor{red}{Taa}\$$



## Returning to the example

$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow \textcolor{red}{abaTaa} \Rightarrow abaaa$

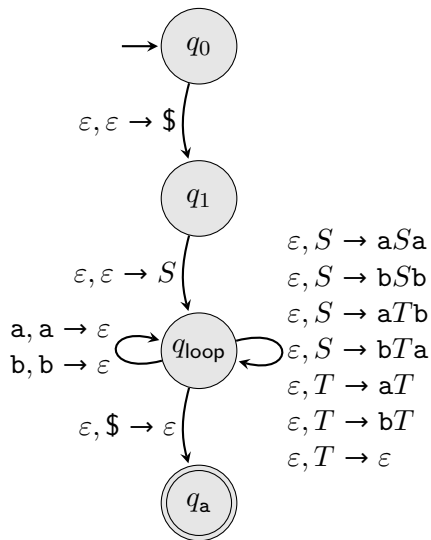
State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{\text{loop}}$	pop $S$ , push $aSa$	$\epsilon$	$aSa\$$
$q_{\text{loop}}$	read and pop $a$	$a$	$Sa\$$
$q_{\text{loop}}$	pop $S$ , push $bTa$	$a$	$bTaa\$$
$q_{\text{loop}}$	read and pop $b$	$ab$	$Taa\$$
$q_{\text{loop}}$	pop $T$ , push $aT$	$\textcolor{red}{ab}$	$\textcolor{red}{aTaa\$}$



## Returning to the example

$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow \textcolor{red}{abaTaa} \Rightarrow abaaa$

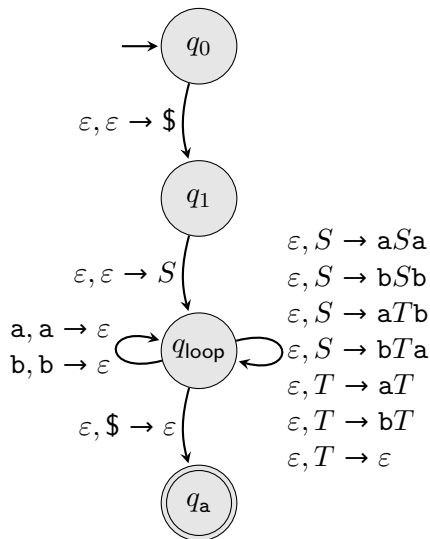
State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{loop}$	pop $S$ , push $aSa$	$\epsilon$	$aSa\$$
$q_{loop}$	read and pop $a$	$a$	$Sa\$$
$q_{loop}$	pop $S$ , push $bTa$	$a$	$bTaa\$$
$q_{loop}$	read and pop $b$	$ab$	$Taa\$$
$q_{loop}$	pop $T$ , push $aT$	$ab$	$aTaa\$$
$q_{loop}$	read and pop $a$	$\textcolor{red}{aba}$	$\textcolor{red}{Taa\$}$



## Returning to the example

$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow abaTaa \Rightarrow \text{abaaa}$

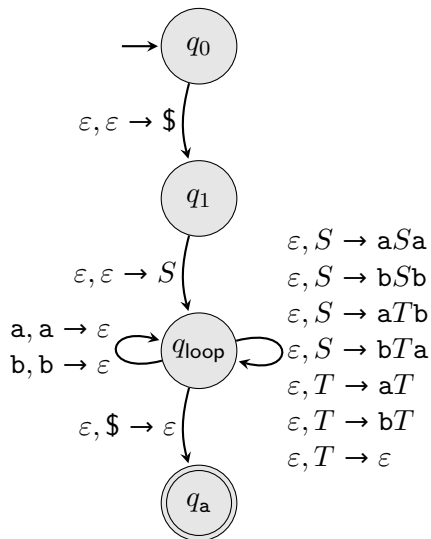
State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{\text{loop}}$	pop $S$ , push $aSa$	$\epsilon$	$aSa\$$
$q_{\text{loop}}$	read and pop $a$	$a$	$Sa\$$
$q_{\text{loop}}$	pop $S$ , push $bTa$	$a$	$bTaa\$$
$q_{\text{loop}}$	read and pop $b$	$ab$	$Taa\$$
$q_{\text{loop}}$	pop $T$ , push $aT$	$ab$	$aTaa\$$
$q_{\text{loop}}$	read and pop $a$	$aba$	$Taa\$$
$q_{\text{loop}}$	pop $T$ , push $\epsilon$	$\text{aba}$	$\text{aa}\$$



## Returning to the example

$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow abaTaa \Rightarrow \text{abaaa}$

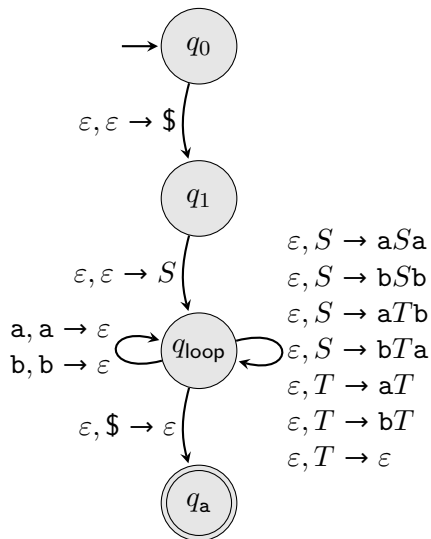
State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{\text{loop}}$	pop $S$ , push $aSa$	$\epsilon$	$aSa\$$
$q_{\text{loop}}$	read and pop $a$	$a$	$Sa\$$
$q_{\text{loop}}$	pop $S$ , push $bTa$	$a$	$bTaa\$$
$q_{\text{loop}}$	read and pop $b$	$ab$	$Taa\$$
$q_{\text{loop}}$	pop $T$ , push $aT$	$ab$	$aTaa\$$
$q_{\text{loop}}$	read and pop $a$	$aba$	$Taa\$$
$q_{\text{loop}}$	pop $T$ , push $\epsilon$	$aba$	$aa\$$
$q_{\text{loop}}$	read and pop $a$	$\text{abaa}$	$a\$$



## Returning to the example

$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow abaTaa \Rightarrow \text{abaaa}$

State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{\text{loop}}$	pop $S$ , push $aSa$	$\epsilon$	$aSa\$$
$q_{\text{loop}}$	read and pop $a$	$a$	$Sa\$$
$q_{\text{loop}}$	pop $S$ , push $bTa$	$a$	$bTaa\$$
$q_{\text{loop}}$	read and pop $b$	$ab$	$Taa\$$
$q_{\text{loop}}$	pop $T$ , push $aT$	$ab$	$aTaa\$$
$q_{\text{loop}}$	read and pop $a$	$aba$	$Taa\$$
$q_{\text{loop}}$	pop $T$ , push $\epsilon$	$aba$	$aa\$$
$q_{\text{loop}}$	read and pop $a$	$abaa$	$a\$$
$q_{\text{loop}}$	read and pop $a$	$\text{abaaa}$	$\$$

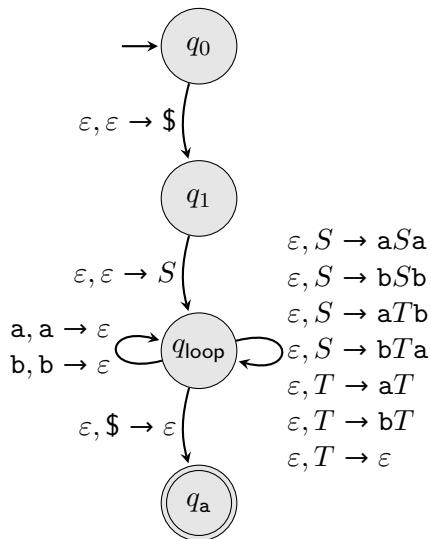




## Returning to the example

$$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow abaTaa \Rightarrow abaaa$$

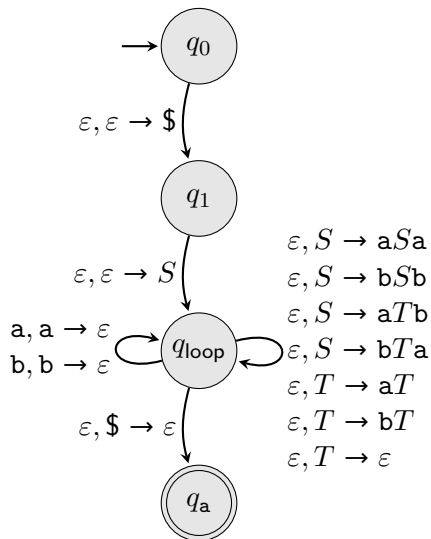
State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{loop}$	pop $S$ , push $aSa$	$\epsilon$	$aSa\$$
$q_{loop}$	read and pop $a$	$a$	$Sa\$$
$q_{loop}$	pop $S$ , push $bTa$	$a$	$bTaa\$$
$q_{loop}$	read and pop $b$	$ab$	$Taa\$$
$q_{loop}$	pop $T$ , push $aT$	$ab$	$aTaa\$$
$q_{loop}$	read and pop $a$	$aba$	$Taa\$$
$q_{loop}$	pop $T$ , push $\epsilon$	$aba$	$aa\$$
$q_{loop}$	read and pop $a$	$abaa$	$a\$$
$q_{loop}$	read and pop $a$	$abaaa$	$\$$
$q_{loop}$	pop $\$$	$abaaa$	$\epsilon$



## Returning to the example

$$S \Rightarrow aSa \Rightarrow abTaa \Rightarrow abaTaa \Rightarrow abaaa$$

State	Action	Input read	Stack
$q_0$	push $\$$	$\epsilon$	$\$$
$q_1$	push $S$	$\epsilon$	$S\$$
$q_{loop}$	pop $S$ , push $aSa$	$\epsilon$	$aSa\$$
$q_{loop}$	read and pop $a$	$a$	$Sa\$$
$q_{loop}$	pop $S$ , push $bTa$	$a$	$bTaa\$$
$q_{loop}$	read and pop $b$	$ab$	$Taa\$$
$q_{loop}$	pop $T$ , push $aT$	$ab$	$aTaa\$$
$q_{loop}$	read and pop $a$	$aba$	$Taa\$$
$q_{loop}$	pop $T$ , push $\epsilon$	$aba$	$aa\$$
$q_{loop}$	read and pop $a$	$abaa$	$a\$$
$q_{loop}$	read and pop $a$	$abaaa$	$\$$
$q_{loop}$	pop $\$$	$abaaa$	$\epsilon$
$q_a$	accept	$abaaa$	$\epsilon$



## Back from example

Consider running  $M$  on input  $w = w_1w_2\cdots w_n$  for  $w_i \in \Sigma$ .

The first time  $M$  enters state  $q_{\text{loop}}$ , the stack is  $S\$$  and no input has been read.

Every subsequent time it enters  $q_{\text{loop}}$ , the input read so far concatenated with the stack is a step in some left-most derivation of  $w$  (followed by a  $\$$ ).

I.e., if  $k$  symbols have been read from the input and the stack is  $s$ , then  $w_1w_2\cdots w_k s$  is a step in the derivation of  $w$

## Back from example

Consider running  $M$  on input  $w = w_1w_2\cdots w_n$  for  $w_i \in \Sigma$ .

The first time  $M$  enters state  $q_{\text{loop}}$ , the stack is  $S\$$  and no input has been read.

Every subsequent time it enters  $q_{\text{loop}}$ , the input read so far concatenated with the stack is a step in some left-most derivation of  $w$  (followed by a  $\$$ ).

I.e., if  $k$  symbols have been read from the input and the stack is  $s$ , then  $w_1w_2\cdots w_k s$  is a step in the derivation of  $w$

$M$  accepts  $w$  once the derivation is complete and all terminals have been matched.  
Therefore, each string accepted by  $M$  is in  $A$ .

## Back from example

Consider running  $M$  on input  $w = w_1w_2\cdots w_n$  for  $w_i \in \Sigma$ .

The first time  $M$  enters state  $q_{\text{loop}}$ , the stack is  $S\$$  and no input has been read.

Every subsequent time it enters  $q_{\text{loop}}$ , the input read so far concatenated with the stack is a step in some left-most derivation of  $w$  (followed by a  $\$$ ).

I.e., if  $k$  symbols have been read from the input and the stack is  $s$ , then  $w_1w_2\cdots w_k s$  is a step in the derivation of  $w$

$M$  accepts  $w$  once the derivation is complete and all terminals have been matched. Therefore, each string accepted by  $M$  is in  $A$ .

For each  $w \in A$ , there is some left-most derivation of  $w$  by  $G$ . By construction,  $M$  performs the derivation on the stack while matching leading terminals.

Thus  $L(M) = A$ .



## Going the other direction

### Theorem

*If a language is recognized by a PDA, then it is context-free.*

Proof idea.

# Going the other direction

## Theorem

*If a language is recognized by a PDA, then it is context-free.*

Proof idea.

- 1 First, convert the PDA to one that
  - has a single accepting state  $q_a$ ;
  - empties its stack before accepting; and
  - either pushes a symbol or pops a symbol, but not both, on each transition

# Going the other direction

## Theorem

*If a language is recognized by a PDA, then it is context-free.*

Proof idea.

- ① First, convert the PDA to one that
  - has a single accepting state  $q_a$ ;
  - empties its stack before accepting; and
  - either pushes a symbol or pops a symbol, but not both, on each transition
- ② Next, construct a CFG that
  - has variables that are pairs of states  $\langle q, r \rangle$  from the PDA;
  - has start variable  $\langle q_0, q_a \rangle$ ;
  - has rules  $\langle q, q \rangle \rightarrow \varepsilon$  for each  $q \in Q$ ;
  - has rules  $\langle p, r \rangle \rightarrow \langle p, q \rangle \langle q, r \rangle$  for each  $p, q, r \in Q$ ; and
  - has rules  $\langle p, q \rangle \rightarrow a \langle r, s \rangle b$  for  $p, q, r, s \in Q$  and  $a, b \in \Sigma_\varepsilon$  if  $(r, u) \in \delta(p, a, \varepsilon)$  and  $(q, \varepsilon) \in \delta(s, b, u)$



# Going the other direction

## Theorem

*If a language is recognized by a PDA, then it is context-free.*

Proof idea.

- ① First, convert the PDA to one that
  - has a single accepting state  $q_a$ ;
  - empties its stack before accepting; and
  - either pushes a symbol or pops a symbol, but not both, on each transition
- ② Next, construct a CFG that
  - has variables that are pairs of states  $\langle q, r \rangle$  from the PDA;
  - has start variable  $\langle q_0, q_a \rangle$ ;
  - has rules  $\langle q, q \rangle \rightarrow \varepsilon$  for each  $q \in Q$ ;
  - has rules  $\langle p, r \rangle \rightarrow \langle p, q \rangle \langle q, r \rangle$  for each  $p, q, r \in Q$ ; and
  - has rules  $\langle p, q \rangle \rightarrow a \langle r, s \rangle b$  for  $p, q, r, s \in Q$  and  $a, b \in \Sigma_\varepsilon$  if  $(r, u) \in \delta(p, a, \varepsilon)$  and  $(q, \varepsilon) \in \delta(s, b, u)$
- ③ Prove (by induction) that each variable  $\langle q, r \rangle$  has the property  $\langle q, r \rangle \xRightarrow{*} x \in \Sigma^*$  iff starting  $M$  in state  $q$  with an empty stack and running on input  $x$  causes  $M$  to move to state  $r$  and end with an empty stack

# Going the other direction

## Theorem

*If a language is recognized by a PDA, then it is context-free.*

Proof idea.

- ① First, convert the PDA to one that
  - has a single accepting state  $q_a$ ;
  - empties its stack before accepting; and
  - either pushes a symbol or pops a symbol, but not both, on each transition
- ② Next, construct a CFG that
  - has variables that are pairs of states  $\langle q, r \rangle$  from the PDA;
  - has start variable  $\langle q_0, q_a \rangle$ ;
  - has rules  $\langle q, q \rangle \rightarrow \varepsilon$  for each  $q \in Q$ ;
  - has rules  $\langle p, r \rangle \rightarrow \langle p, q \rangle \langle q, r \rangle$  for each  $p, q, r \in Q$ ; and
  - has rules  $\langle p, q \rangle \rightarrow a \langle r, s \rangle b$  for  $p, q, r, s \in Q$  and  $a, b \in \Sigma_\varepsilon$  if  $(r, u) \in \delta(p, a, \varepsilon)$  and  $(q, \varepsilon) \in \delta(s, b, u)$
- ③ Prove (by induction) that each variable  $\langle q, r \rangle$  has the property  $\langle q, r \rangle \xRightarrow{*} x \in \Sigma^*$  iff starting  $M$  in state  $q$  with an empty stack and running on input  $x$  causes  $M$  to move to state  $r$  and end with an empty stack
- ④ Conclude that  $\langle q_0, q_a \rangle \xRightarrow{*} w$  iff  $w \in L(M)$

# Closure properties of CFLs

The class of context-free languages is closed under

- Union
- Concatenation
- Kleene star
- PREFIX
- SUFFIX
- Reversal
- Intersection with a regular language
- Quotient by a string
- Quotient by a regular language

We proved closure under union, concatenation, Kleene star, and PREFIX previously

# Reversal

## Theorem

*Context-free languages are closed under reversal.*

**Proof.** Let  $B$  be a context-free language generated by a CFG  $G = (V, \Sigma, R, S)$ .

Construct CFG  $G' = (V, \Sigma, R', S)$  where

$$R' = \{A \rightarrow u^{\mathcal{R}} \mid A \rightarrow u \text{ is a rule in } R\}.$$

# Reversal

## Theorem

*Context-free languages are closed under reversal.*

**Proof.** Let  $B$  be a context-free language generated by a CFG  $G = (V, \Sigma, R, S)$ .

Construct CFG  $G' = (V, \Sigma, R', S)$  where

$$R' = \{A \rightarrow u^{\mathcal{R}} \mid A \rightarrow u \text{ is a rule in } R\}.$$

To prove that  $L(G') = B^{\mathcal{R}}$ , we want to show that for each variable  $A \in V$  and  $u \in (V \cup \Sigma)^*$ ,  $A \xRightarrow{*}_G u$  in  $n$  steps iff  $A \xRightarrow{*}_{G'} u^{\mathcal{R}}$  in  $n$  steps.

Let's write  $\xRightarrow{k}$  to mean  $\xRightarrow{*}$  in exactly  $k$  steps.

## Proof continued

Base case  $n = 0$ . If  $A \xRightarrow{0}_G u$ , then  $u = u^{\mathcal{R}} = A$  so  $A \xRightarrow{0}_{G'} u^{\mathcal{R}}$ , and vice versa.

## Proof continued

Base case  $n = 0$ . If  $A \xRightarrow{0}_G u$ , then  $u = u^{\mathcal{R}} = A$  so  $A \xRightarrow{0}_{G'} u^{\mathcal{R}}$ , and vice versa.

Inductive step. Assume that for all  $n > 0$ ,  $A \in V$ , and  $u \in (V \cup \Sigma)^*$ ,  $A \xRightarrow{n-1}_G u$  iff  $A \xRightarrow{n-1}_{G'} u^{\mathcal{R}}$ .

## Proof continued

Base case  $n = 0$ . If  $A \xRightarrow{0}_G u$ , then  $u = u^{\mathcal{R}} = A$  so  $A \xRightarrow{0}_{G'} u^{\mathcal{R}}$ , and vice versa.

Inductive step. Assume that for all  $n > 0$ ,  $A \in V$ , and  $u \in (V \cup \Sigma)^*$ ,  $A \xRightarrow{n-1}_G u$  iff  $A \xRightarrow{n-1}_{G'} u^{\mathcal{R}}$ .

If  $A \xRightarrow{n}_G u$ , then there is some  $C \in V$  and  $x, y, z \in (V \cup \Sigma)^*$  such that  $u = xyz$ ,  $A \xRightarrow{n-1}_G xCz$ , and  $C \Rightarrow_G y$ .



## Proof continued

Base case  $n = 0$ . If  $A \xRightarrow{0}_G u$ , then  $u = u^{\mathcal{R}} = A$  so  $A \xRightarrow{0}_{G'} u^{\mathcal{R}}$ , and vice versa.

Inductive step. Assume that for all  $n > 0$ ,  $A \in V$ , and  $u \in (V \cup \Sigma)^*$ ,  $A \xRightarrow{n-1}_G u$  iff  $A \xRightarrow{n-1}_{G'} u^{\mathcal{R}}$ .

If  $A \xRightarrow{n}_G u$ , then there is some  $C \in V$  and  $x, y, z \in (V \cup \Sigma)^*$  such that  $u = xyz$ ,  $A \xRightarrow{n-1}_G xCz$ , and  $C \Rightarrow_G y$ .

By the inductive hypothesis  $A \xRightarrow{n-1}_{G'} z^{\mathcal{R}} C x^{\mathcal{R}}$  and by construction  $C \Rightarrow_{G'} y^{\mathcal{R}}$ . Thus  $A \xRightarrow{n}_{G'} z^{\mathcal{R}} y^{\mathcal{R}} x^{\mathcal{R}} = (xyz)^{\mathcal{R}} = u^{\mathcal{R}}$ . Swapping  $G$  and  $G'$  shows the converse.

## Proof continued

Base case  $n = 0$ . If  $A \xRightarrow{0}_G u$ , then  $u = u^{\mathcal{R}} = A$  so  $A \xRightarrow{0}_{G'} u^{\mathcal{R}}$ , and vice versa.

Inductive step. Assume that for all  $n > 0$ ,  $A \in V$ , and  $u \in (V \cup \Sigma)^*$ ,  $A \xRightarrow{n-1}_G u$  iff  $A \xRightarrow{n-1}_{G'} u^{\mathcal{R}}$ .

If  $A \xRightarrow{n}_G u$ , then there is some  $C \in V$  and  $x, y, z \in (V \cup \Sigma)^*$  such that  $u = xyz$ ,  $A \xRightarrow{n-1}_G xCz$ , and  $C \Rightarrow_G y$ .

By the inductive hypothesis  $A \xRightarrow{n-1}_{G'} z^{\mathcal{R}} C x^{\mathcal{R}}$  and by construction  $C \Rightarrow_{G'} y^{\mathcal{R}}$ . Thus  $A \xRightarrow{n}_{G'} z^{\mathcal{R}} y^{\mathcal{R}} x^{\mathcal{R}} = (xyz)^{\mathcal{R}} = u^{\mathcal{R}}$ . Swapping  $G$  and  $G'$  shows the converse.

Thus,  $A \xRightarrow{n}_G u$  iff  $A \xRightarrow{n}_{G'} u^{\mathcal{R}}$ .

## Proof continued

Base case  $n = 0$ . If  $A \xRightarrow{0}_G u$ , then  $u = u^{\mathcal{R}} = A$  so  $A \xRightarrow{0}_{G'} u^{\mathcal{R}}$ , and vice versa.

Inductive step. Assume that for all  $n > 0$ ,  $A \in V$ , and  $u \in (V \cup \Sigma)^*$ ,  $A \xRightarrow{n-1}_G u$  iff  $A \xRightarrow{n-1}_{G'} u^{\mathcal{R}}$ .

If  $A \xRightarrow{n}_G u$ , then there is some  $C \in V$  and  $x, y, z \in (V \cup \Sigma)^*$  such that  $u = xyz$ ,  $A \xRightarrow{n-1}_G xCz$ , and  $C \Rightarrow_G y$ .

By the inductive hypothesis  $A \xRightarrow{n-1}_{G'} z^{\mathcal{R}} C x^{\mathcal{R}}$  and by construction  $C \Rightarrow_{G'} y^{\mathcal{R}}$ . Thus  $A \xRightarrow{n}_{G'} z^{\mathcal{R}} y^{\mathcal{R}} x^{\mathcal{R}} = (xyz)^{\mathcal{R}} = u^{\mathcal{R}}$ . Swapping  $G$  and  $G'$  shows the converse.

Thus,  $A \xRightarrow{n}_G u$  iff  $A \xRightarrow{n}_{G'} u^{\mathcal{R}}$ .

Therefore, for  $w \in B$ ,  $S \xRightarrow{*}_G w$  iff  $S \xRightarrow{*}_{G'} w^{\mathcal{R}}$  so  $L(G') = B^{\mathcal{R}}$ .



# Suffix

## Theorem

*Context free languages are closed under SUFFIX.*

## Proof.

Since  $\text{SUFFIX}(A) = \text{PREFIX}(A^{\mathcal{R}})^{\mathcal{R}}$  and CFLs are closed under reversal and PREFIX, CFLs are closed under SUFFIX. □

# Intersection of a CFL and a regular language

## Theorem

*The intersection of a CFL and a regular language is context-free.*

## Proof.

Let  $A$  be a CFL recognized by the PDA  $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_1, F_1)$  and  $B$  be a regular language recognized by the NFA  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ .

# Intersection of a CFL and a regular language

## Theorem

*The intersection of a CFL and a regular language is context-free.*

## Proof.

Let  $A$  be a CFL recognized by the PDA  $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_1, F_1)$  and  $B$  be a regular language recognized by the NFA  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ .

Construct the PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  where

$$Q = Q_1 \times Q_2$$

$$q_0 = (q_1, q_2)$$

$$F = F_1 \times F_2$$

$$\delta((q, r), a, b) = \{((s, t), c) \mid (s, c) \in \delta_1(q, a, b) \text{ and } t \in \delta_2(r, a)\} \quad \text{for } a \in \Sigma_\epsilon, b, c \in \Gamma_\epsilon$$

# Intersection of a CFL and a regular language

## Theorem

*The intersection of a CFL and a regular language is context-free.*

## Proof.

Let  $A$  be a CFL recognized by the PDA  $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_1, F_1)$  and  $B$  be a regular language recognized by the NFA  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ .

Construct the PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  where

$$Q = Q_1 \times Q_2$$

$$q_0 = (q_1, q_2)$$

$$F = F_1 \times F_2$$

$$\delta((q, r), a, b) = \{((s, t), c) \mid (s, c) \in \delta_1(q, a, b) \text{ and } t \in \delta_2(r, a)\} \quad \text{for } a \in \Sigma_\epsilon, b, c \in \Gamma_\epsilon$$

As  $M$  runs on input  $w$ , its stack and the first element of its state change according to  $\delta_1$  whereas the second element of its state changes according to  $\delta_2$ .

$M$  accepts  $w$  iff  $M_1$  accepts  $w$  and  $M_2$  accepts  $w$ . Therefore,  $L(M) = A \cap B$ . □

# What about intersection with another CFL?

Are context-free languages closed under intersection?



# What about intersection with another CFL?

Are context-free languages closed under intersection?

Consider  $\Sigma = \{a, b, c\}$  and

$$A = \{a^m b^m c^n \mid m, n \geq 0\}$$

$$B = \{a^m b^n c^n \mid m, n \geq 0\}$$

Both  $B$  and  $C$  are context-free. Is

$$A \cap B = \{a^n b^n c^n \mid n \geq 0\}?$$

How can we keep track of how many as and bs we've seen to ensure we get the same number of cs using a PDA?

How about trying to generate such strings with a CFG?

# What about intersection with another CFL?

Are context-free languages closed under intersection?

Consider  $\Sigma = \{a, b, c\}$  and

$$A = \{a^m b^m c^n \mid m, n \geq 0\}$$

$$B = \{a^m b^n c^n \mid m, n \geq 0\}$$

Both  $B$  and  $C$  are context-free. Is

$$A \cap B = \{a^n b^n c^n \mid n \geq 0\}?$$

How can we keep track of how many as and bs we've seen to ensure we get the same number of cs using a PDA?

How about trying to generate such strings with a CFG?

Next time, we'll see that  $B \cap C$  is *not* context-free!