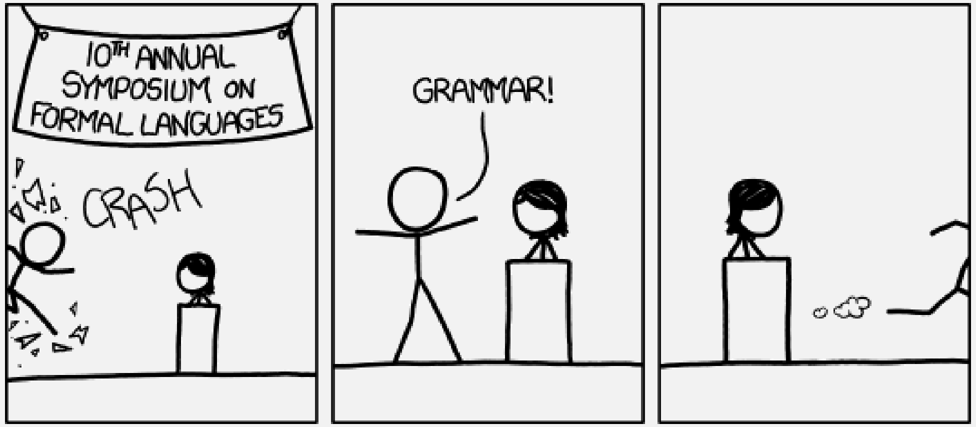
UMBCS622

Pushdown Automata (PDAs)

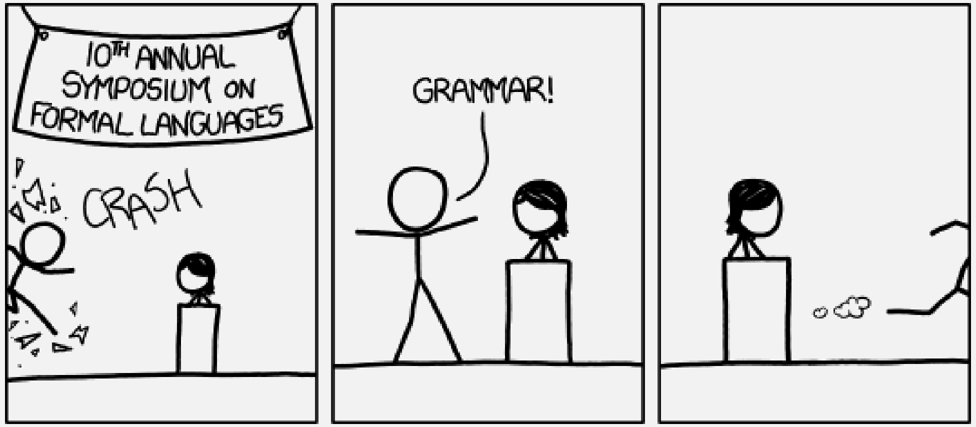
Monday, October 4, 2021



Announcements

- No class next Monday 10/11

- HW4 released
  - Due Sun 10/18 11:59pm EST
  - <u>Note</u>: this is a **2 week** assignment!



Last Time:

Regular Languages	Context-Free Languages (CFLs)
Regular Expression	Context-Free Grammar (CFG)
A Reg expr describes a Regular lang	A CFG describes a CFL

Today:

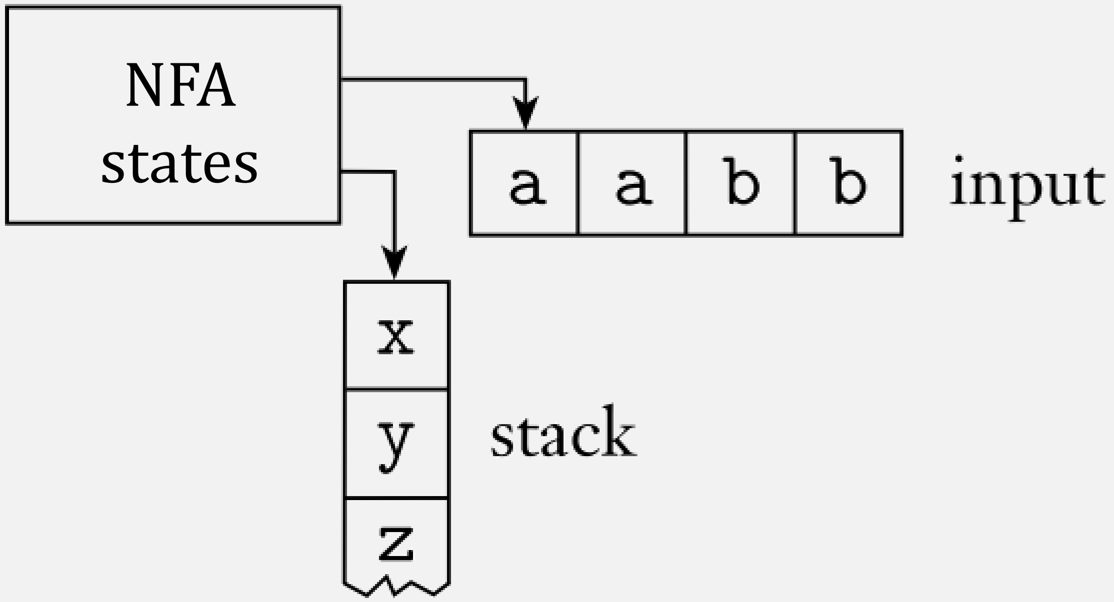
Regular Languages	Context-Free Languages (CFLs)
Regular Expression	Context-Free Grammar (CFG)
A Reg expr <u>describes</u> a Regular lang	A CFG <u>describes</u> a CFL
	TODAY:
Finite automaton (FSM)	Push-down automaton (PDA)
An FSM <u>recognizes</u> a Regular lang	A PDA <u>recognizes</u> a CFL

Today:

Regular Languages	Context-Free Languages (CFLs)
Regular Expression	Context-Free Grammar (CFG)
A Reg expr describes a Regular lang	A CFG describes a CFL
	TODAY:
Finite automaton (FSM)	Push-down automaton (PDA)
An FSM recognizes a Regular lang	A PDA recognizes a CFL
DIFFERENCE:	**DIFFERENCE:**
A Regular lang is defined with a FSM	A CFL is defined with a CFG
Must prove: Reg expr ⇔ Reg lang	*Must prove*: PDA ⇔ CFL

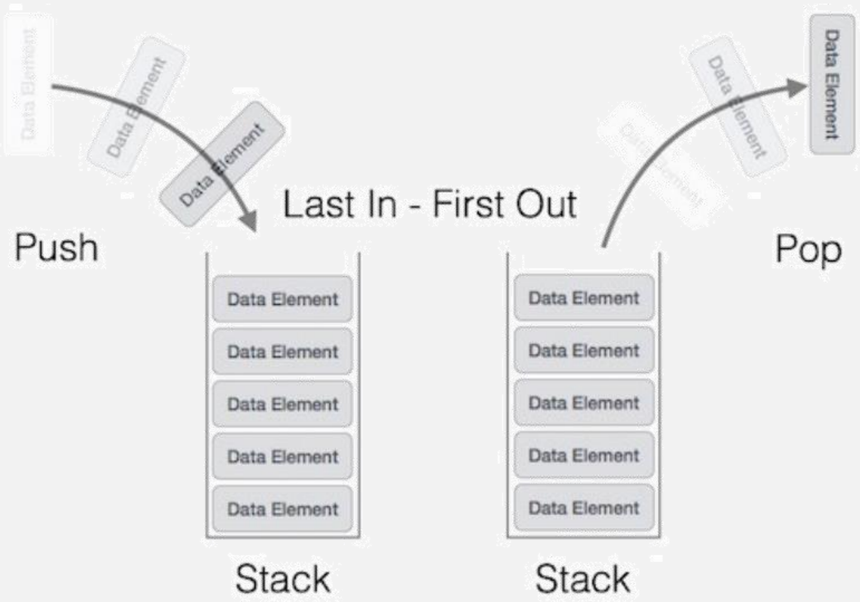
Pushdown Automata (PDA)

- PDA = NFA + a stack



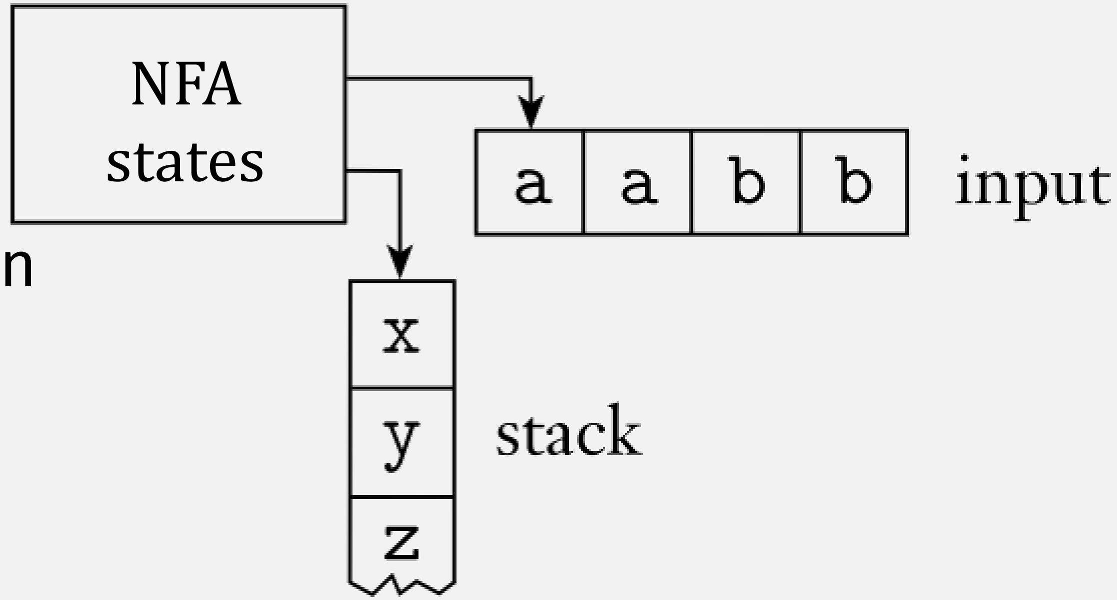
What is a Stack?

- Access to top element only
- 2 Operations: push, pop



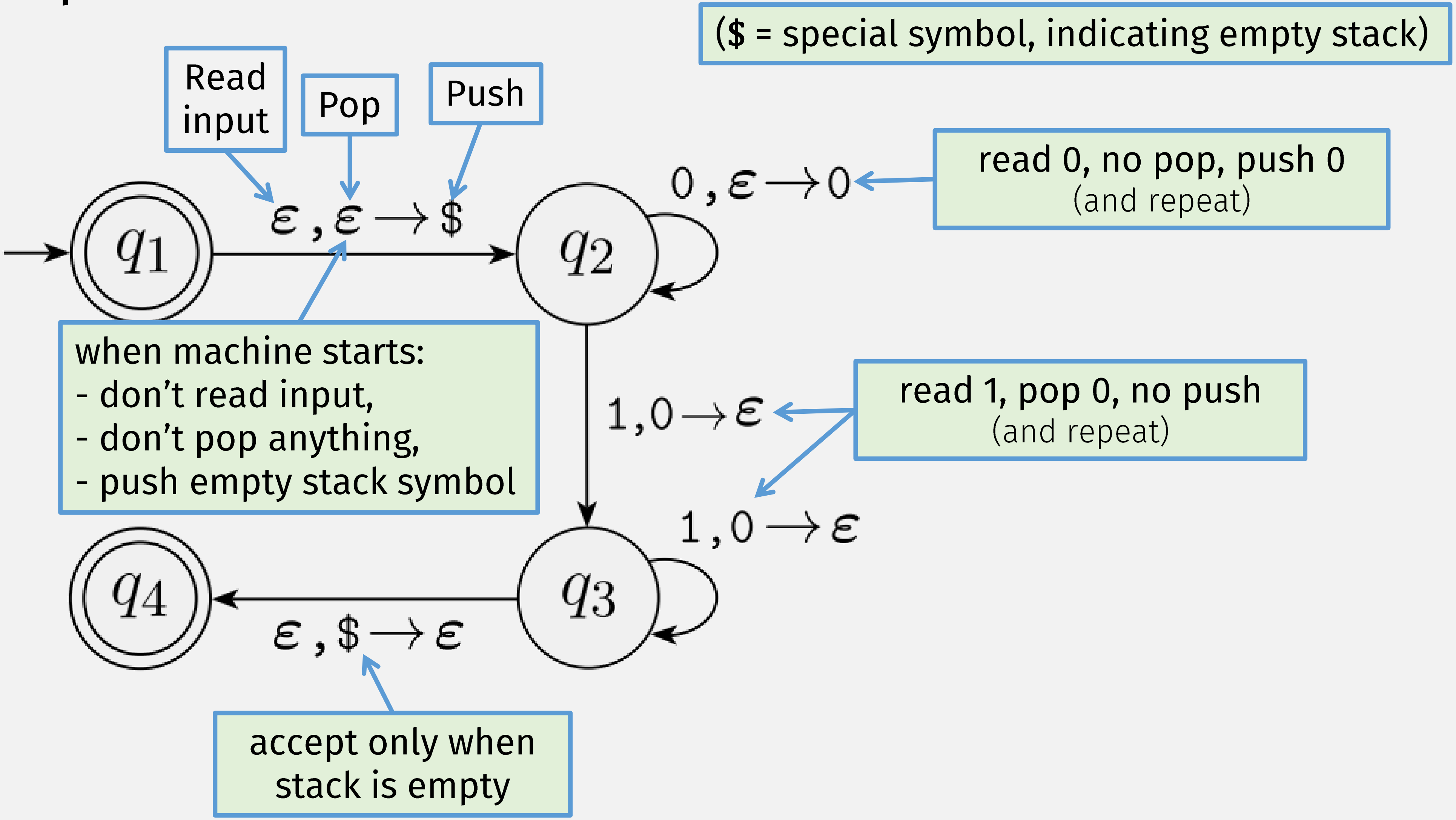
Pushdown Automata (PDA)

- PDA = NFA + a stack
  - Infinite memory
  - Can only read/write top location
    - Push/pop



An Example PDA

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



Formal Definition of PDA

A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Stack alphabet can have special stack symbols, e.g., $

Input

Pop

Push

Non-deterministic: produces a **set** of (STATE, STACK CHAR) pairs

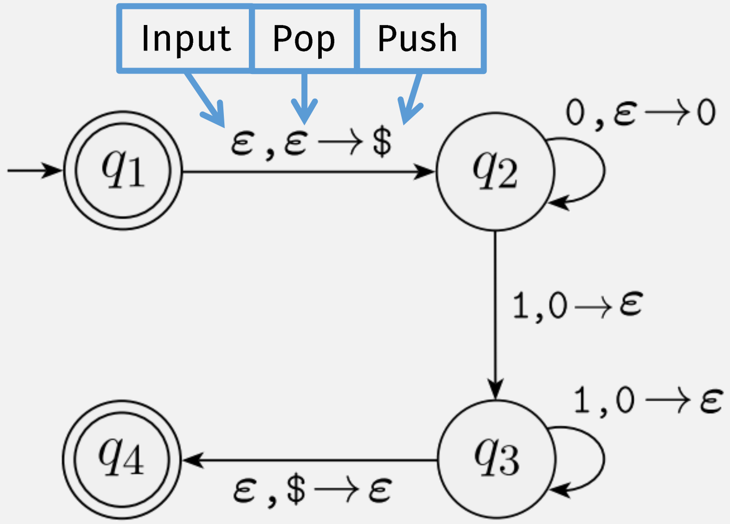
PDA Formal Definition Example

$$Q = \{q_1, q_2, q_3, q_4\},$$

$$\Sigma = \{0,1\},$$

$$\Gamma = \{0, \$\},$$

$$F = \{q_1, q_4\},$$



A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Input

Pop

Push

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0,1\}$,

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and

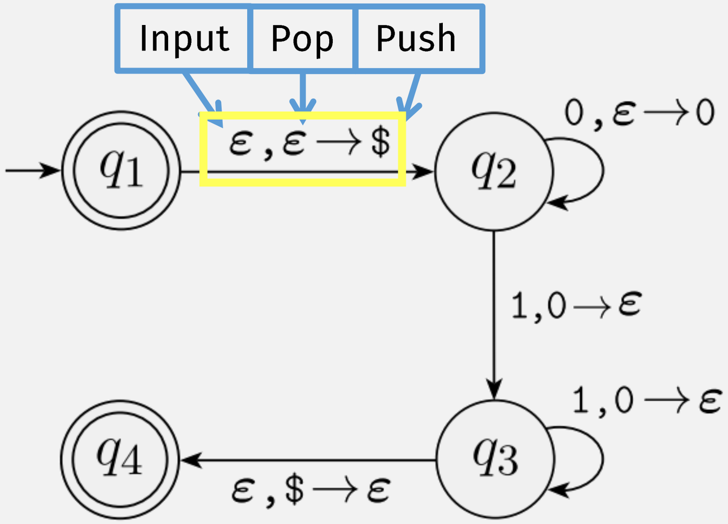
δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$ 2				4	5
q_3			1	$\{(q_3, \varepsilon)\}$ 3				$\{(q_4, \varepsilon)\}$	
q_4									

Input

Pop

Push



A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Input Pop Push

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0,1\}$,

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and

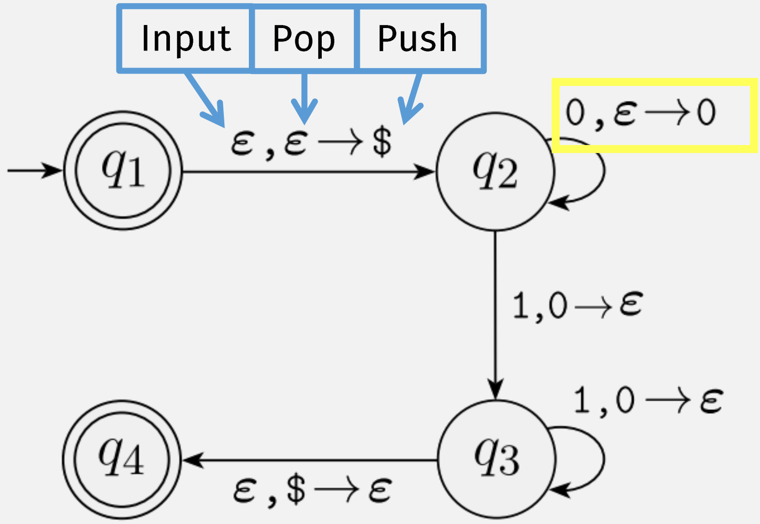
δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\boxed{\{(q_2, 0)\}}$	$\{(q_3, \varepsilon)\}$ 2				4	5
q_3			1	$\{(q_3, \varepsilon)\}$ 3				$\{(q_4, \varepsilon)\}$	
q_4									

Input

Pop

Push



A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta\colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Input

Pop

Push

$$Q = \{q_1, q_2, q_3, q_4\},$$

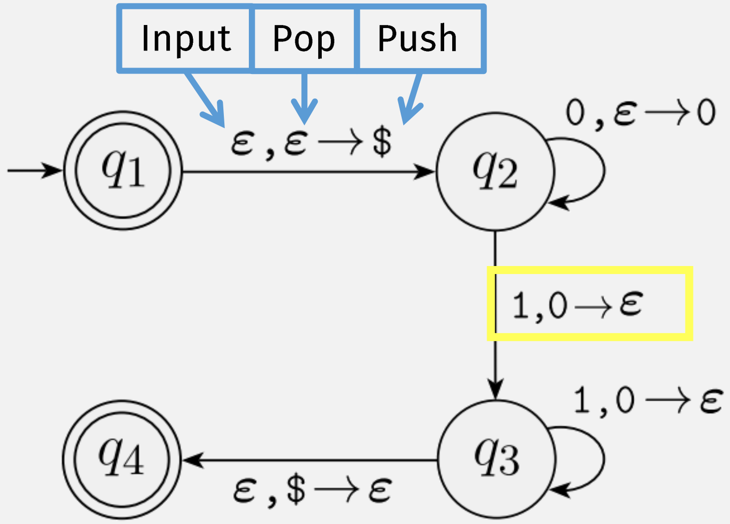
$$\Sigma = \{0,1\},$$

$$\Gamma = \{0, \$\},$$

$$F = \{q_1, q_4\}, \text{ and}$$

δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0			1			ε (Input)		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε (Pop)
q_1									$\{(q_2, \$)\}$ (Push)
q_2			$\{(q_2, 0)\}$	$\boxed{\{(q_3, \varepsilon)\}}$ 2				4	5
q_3			1	$\{(q_3, \varepsilon)\}$ 3				$\{(q_4, \varepsilon)\}$	
q_4									



A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta\colon Q \times \underbrace{\Sigma_\varepsilon}_{\text{Input}} \times \underbrace{\Gamma_\varepsilon}_{\text{Pop}} \longrightarrow \mathcal{P}(Q \times \underbrace{\Gamma_\varepsilon}_{\text{Push}})$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0,1\}$,

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and

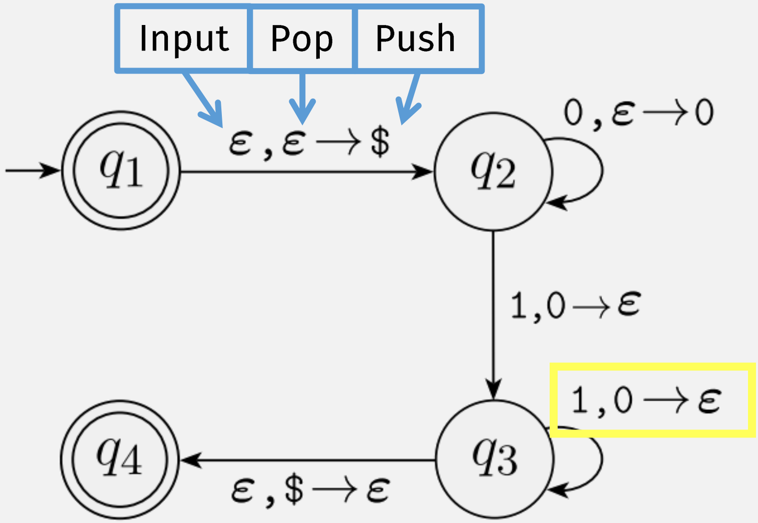
δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$ 2				4	5
q_3			1	$\boxed{\{(q_3, \varepsilon)\}}$ 3				$\{(q_4, \varepsilon)\}$	
q_4									

Input

Pop

Push



A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Input

Pop

Push

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0,1\}$,

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and

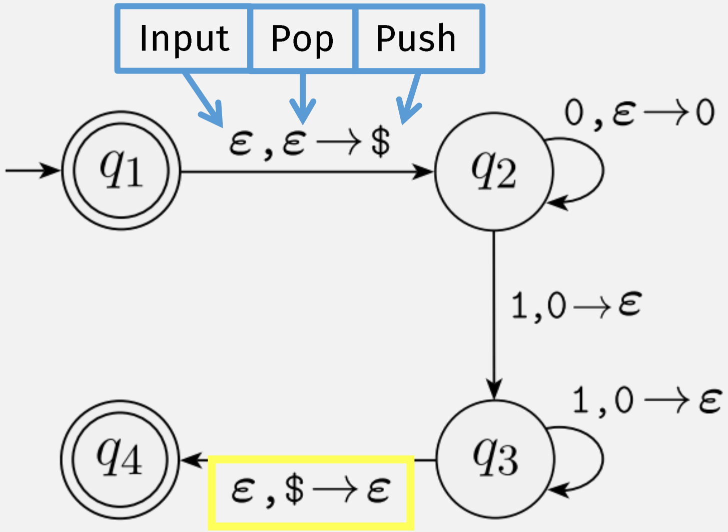
δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$ 2				4	5
q_3			1	$\{(q_3, \varepsilon)\}$ 3				$\boxed{\{(q_4, \varepsilon)\}}$	
q_4									

Input

Pop

Push



A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

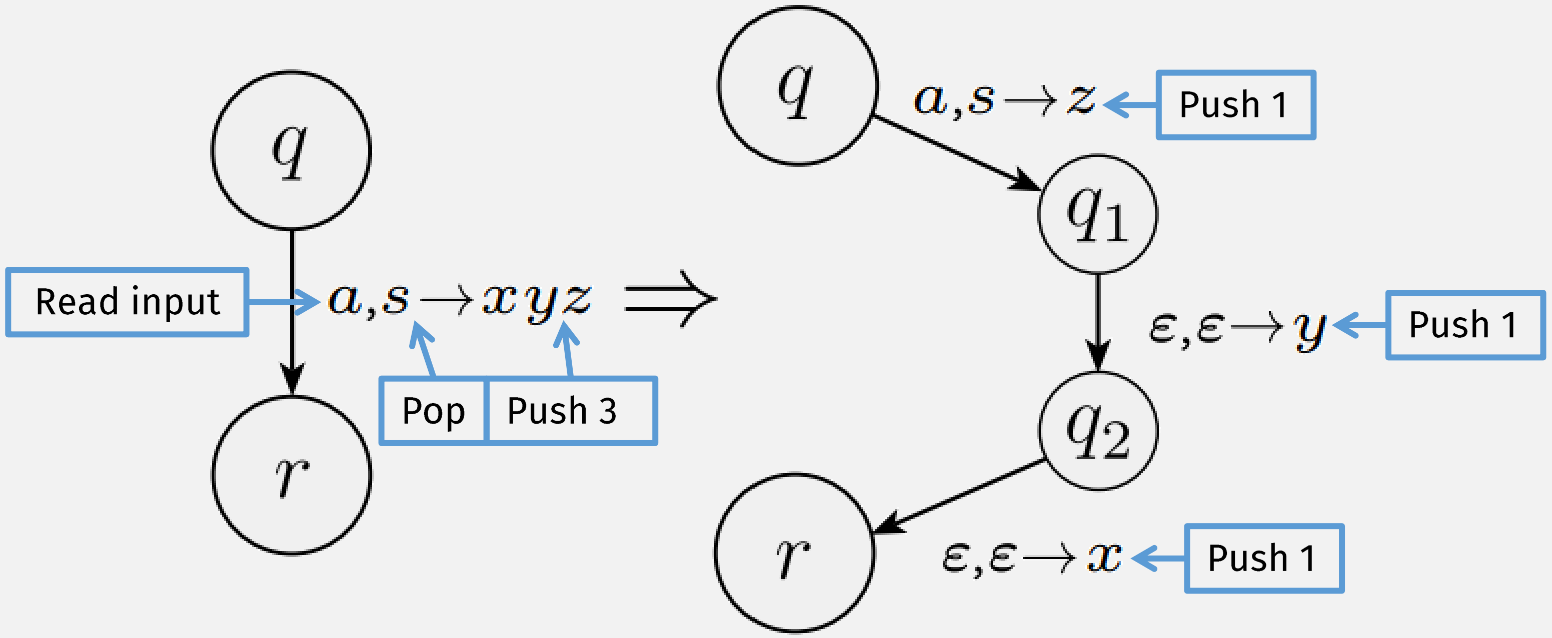
1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta\colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Input

Pop

Push

Multi-Symbol Stack Pushes



Note the reverse order of pushes

PDA Configurations (IDs)

- When "running" an input string on a PDA, a **configuration** (or **ID**) is a snapshot of some point in the computation

- A configuration (or **ID**) (q, w, γ) has three components

  q = the current state

  w = the remaining input string

  γ = the stack contents

“Running” an Input String on a PDA

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

Single-step

Before/after configurations

$$(q_1, aw, X\beta) \vdash (q_2, w, \alpha\beta)$$

Read Input

Pop

Push

$$\text{if } \delta(q_1, a, X) \text{ contains } (q_2, \alpha)$$

$$q_1, q_2 \in Q$$

$$a \in \Sigma \quad w \in \Sigma^*$$

$$X \in \Gamma \quad \beta, \alpha \in \Gamma^*$$

A configuration (q, w , γ) has three components

q = the current state

w = the remaining input string

γ = the stack contents

Extended

- Base Case

$$\boxed{I \overset{*}{\vdash} I \text{ for any ID } I}$$

- Recursive Case

$$\boxed{I \overset{*}{\vdash} J} \text{ if there exists some ID } K$$

$$\text{such that } I \vdash K \text{ and } K \overset{*}{\vdash} J$$

Language of a PDA

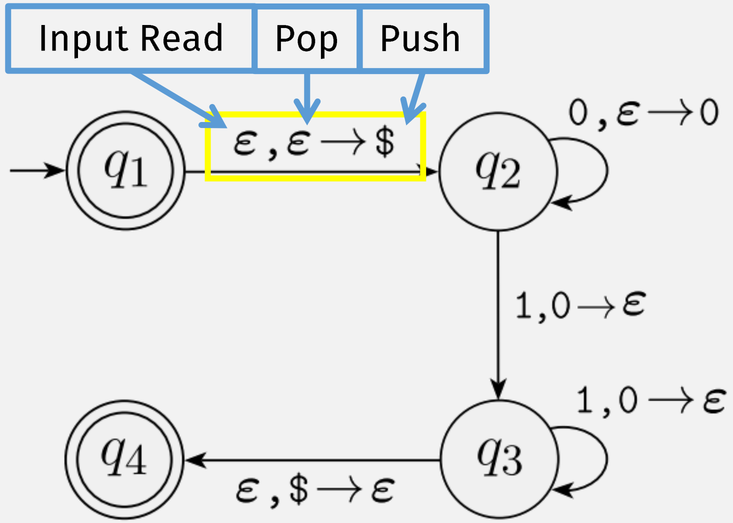
$$P = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

$$\boxed{L(P) = \{w \mid (q_0, w, \varepsilon) \vdash^* (q, \varepsilon, \alpha)\} \text{ where } q \in F}$$

PDA Running Input String Example

State | Remaining Input | Stack

$$(q_1, 0011, \varepsilon)$$

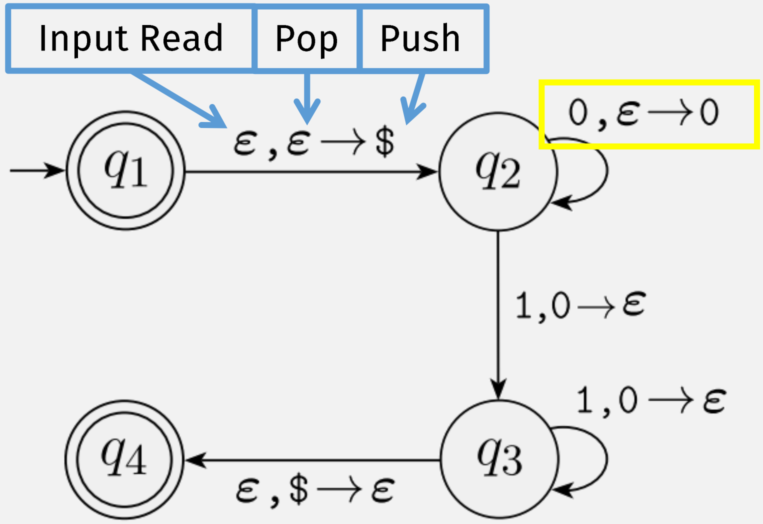


PDA Running Input String Example

State | Remaining Input | Stack

$$(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$)$$

$$\boxed{\vdash (q_2, 011, 0\$)}$$



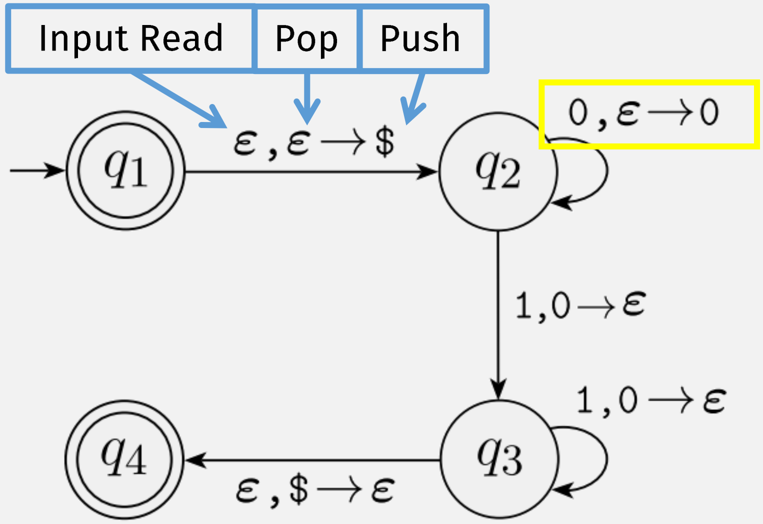
PDA Running Input String Example

State | Remaining Input | Stack

$$(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$)$$

$$\vdash (q_2, 011, 0\$)$$

$$\boxed{\vdash (q_2, 11, 00\$)}$$

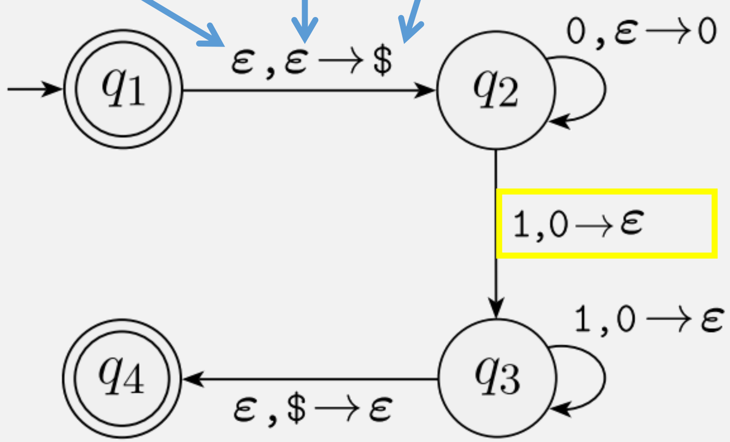


PDA Running Input String Example

State | Remaining Input | Stack

$$(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$)$$
$$\vdash (q_2, 011, 0\$)$$
$$\vdash (q_2, 11, 00\$)$$
$$\boxed{\vdash (q_3, 1, 0\$)}$$

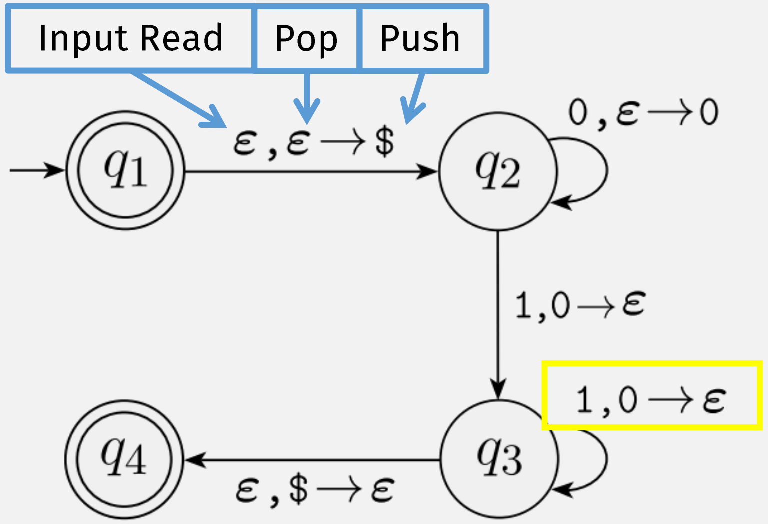
Input Read | Pop | Push



PDA Running Input String Example

State | Remaining Input | Stack

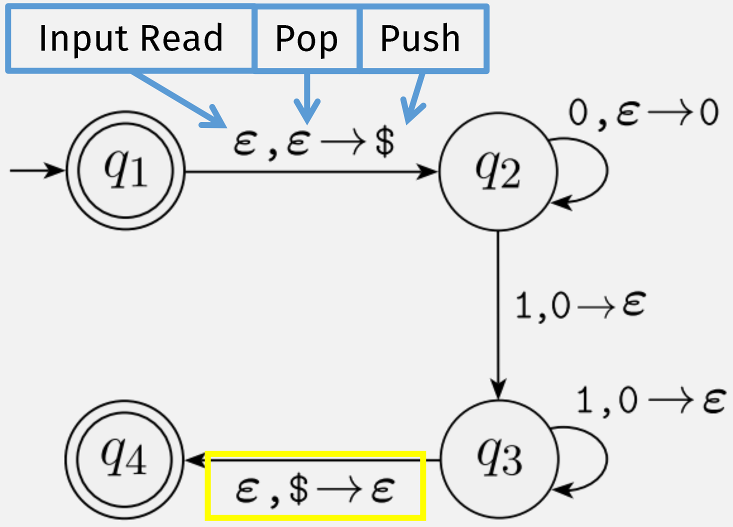
$$
\begin{aligned}
(q_1, 0011, \varepsilon) &\vdash (q_2, 0011, \$) \\
&\vdash (q_2, 011, 0\$) \\
&\vdash (q_2, 11, 00\$) \\
&\vdash (q_3, 1, 0\$) \\
&\boxed{\vdash (q_3, \varepsilon, \$)}
\end{aligned}
$$



PDA Running Input String Example

State | Remaining Input | Stack

$$
\begin{aligned}
(q_1, 0011, \varepsilon) &\vdash (q_2, 0011, \$) \\
&\vdash (q_2, 011, 0\$) \\
&\vdash (q_2, 11, 00\$) \\
&\vdash (q_3, 1, 0\$) \\
&\vdash (q_3, \varepsilon, \$) \\
&\vdash (q_4, \varepsilon, \varepsilon)
\end{aligned}
$$



A PDA Theorem

If $(q_1, x, \alpha) \vdash^* (q_n, y, \beta)$ ← Assume is true

then $(q_1, x\boldsymbol{w}, \alpha\boldsymbol{\gamma}) \vdash^* (q_n, y\boldsymbol{w}, \beta\boldsymbol{\gamma})$ ← Must prove

Adding to end of input or bottom of stack doesn't affect the computation

Proof: (by induction on the number of steps in the sequence)

- Base Case (0 steps): If $(q_1, x, \alpha) \vdash^* (q_1, x, \alpha)$ then $(q_1, x\boldsymbol{w}, \alpha\boldsymbol{\gamma}) \vdash^* (q_1, x\boldsymbol{w}, \alpha\boldsymbol{\gamma})$
  - TRUE, from definition of $\vdash^*$: $\boxed{I \vdash^* I \text{ for any ID } I}$

- Inductive Case
  - Need to prove:

IH says: if this is true … | How do we know these steps are true?

IH

If $(q_1, x, \alpha) \vdash^* (\boldsymbol{q_{n-1}}, \boldsymbol{x'}, \boldsymbol{\alpha'}) \vdash (q_n, y, \beta)$ ← From the assumption!

Then $(q_1, x\boldsymbol{w}, \alpha\boldsymbol{\gamma}) \vdash^* (\boldsymbol{q_{n-1}}, \boldsymbol{x'w}, \boldsymbol{\alpha'}\boldsymbol{\gamma}) \vdash (q_n, y\boldsymbol{w}, \beta\boldsymbol{\gamma})$ ← Left to prove

… then this is true

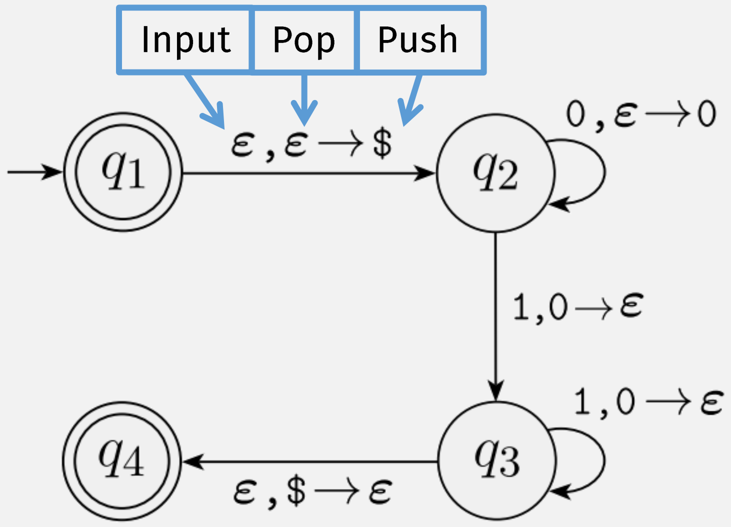
These steps must use the same δ transition, why?

Same state, input char, and stack top!

CFL $\Leftrightarrow$ PDA

Pushdown Automata (PDA)

- PDA = NFA + a stack
    - Infinite memory
    - Can only read/write top location
        - Push/pop



- <u>Want to prove</u>: PDA ⇔ CFG

- Then, to prove that a language is context-free, we can either:
    - Create a CFG, or
    - Create a PDA

A lang is a CFL iff some PDA recognizes it

⇒ If a language is a CFL, then a PDA recognizes it

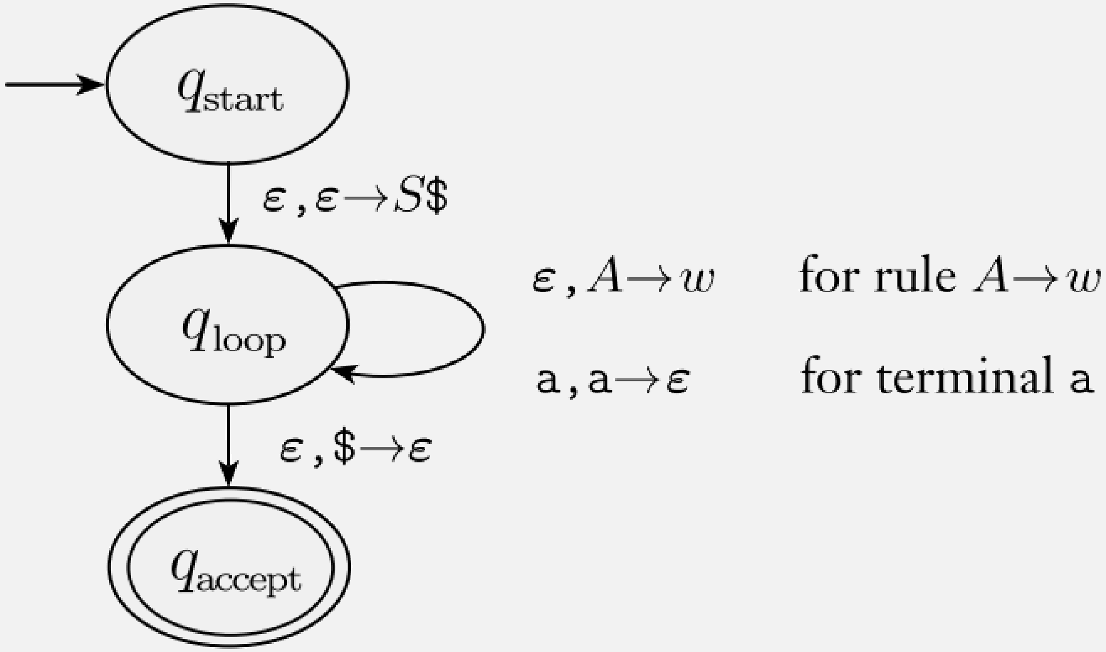
- (Easier)
- We know: A CFL has a CFG describing it (definition of CFL)
- To prove forward dir: Convert **CFG→PDA**

⇐ If a PDA recognizes a language, then it's a CFL

CFG→PDA

- Construct a PDA from CFG such that:
  - PDA accepts input string only if the CFG can generate that string

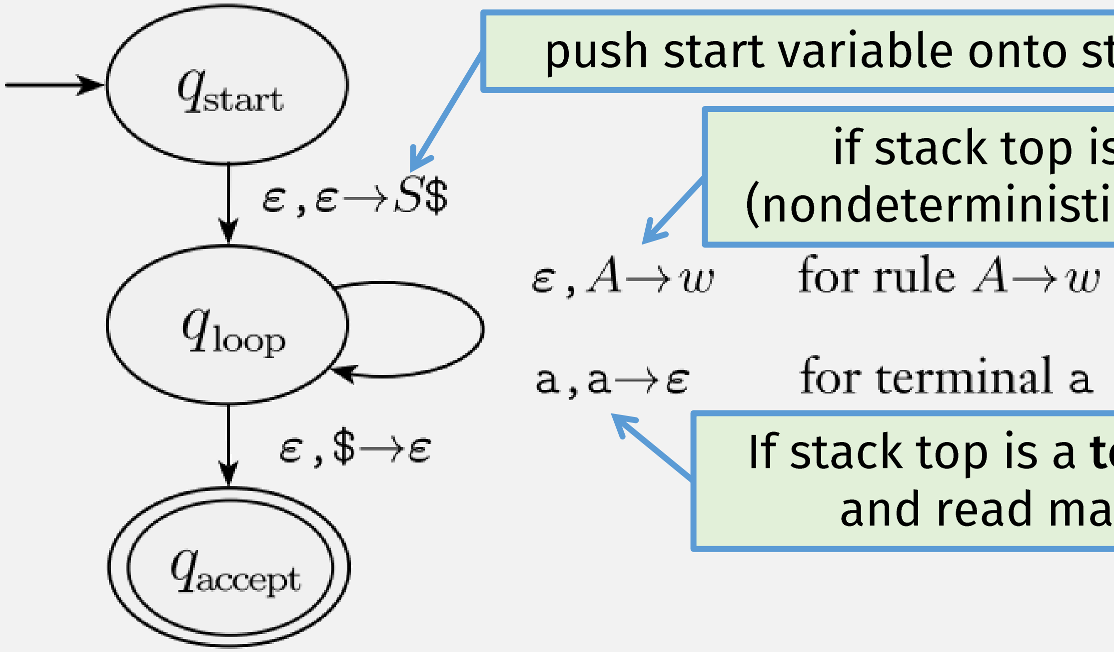
- Intuitively, PDA will <u>nondeterministically</u> try all rules



CFG→PDA

- Construct a PDA from CFG such that:
  - PDA accepts input string only if the CFG can generate that string

- Intuitively, PDA will <u>nondeterministically</u> try all rules



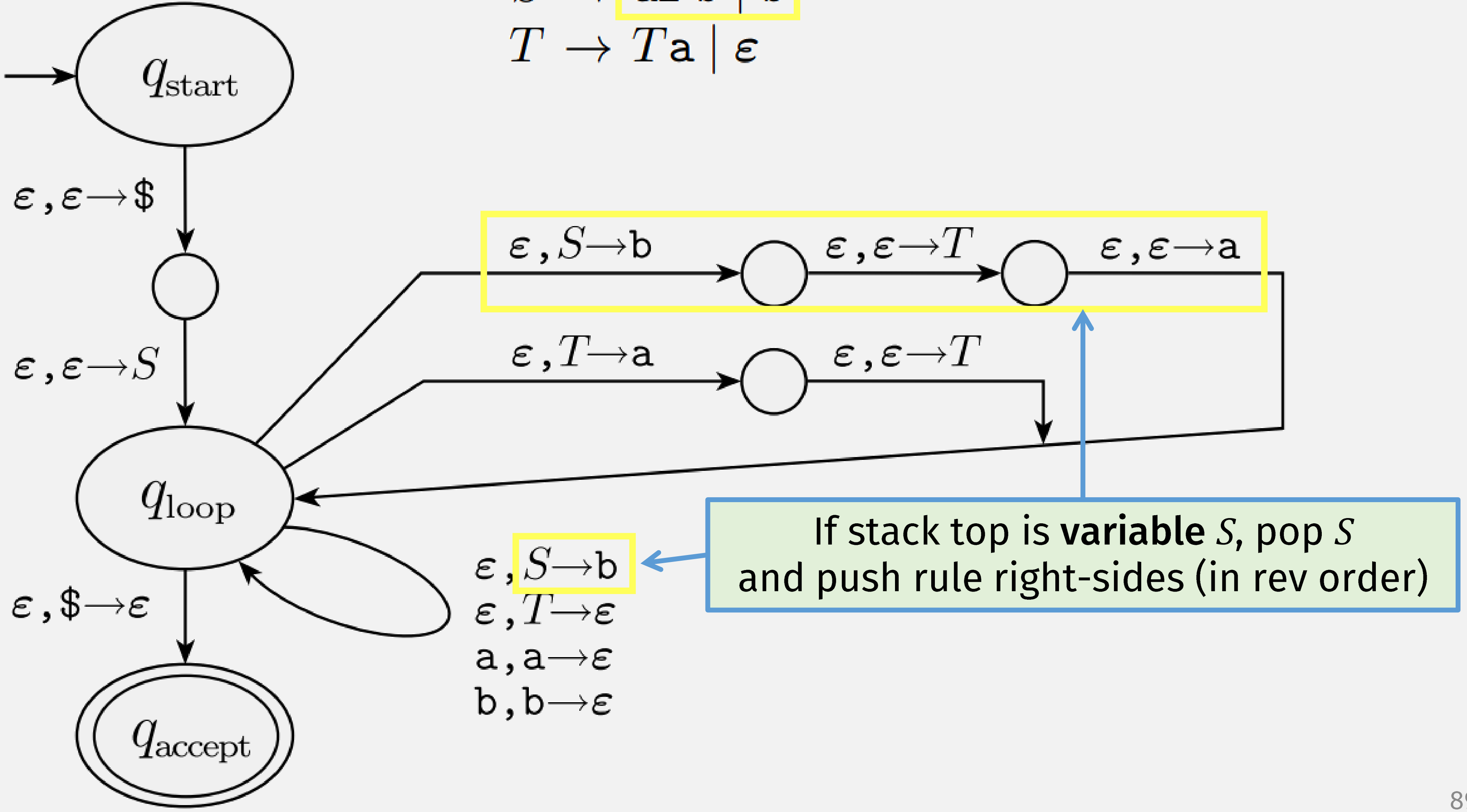
push start variable onto stack

if stack top is a **variable** A, pop it and (nondeterministically) push rule's right-sides

If stack top is a **terminal a**, pop it and read matching input

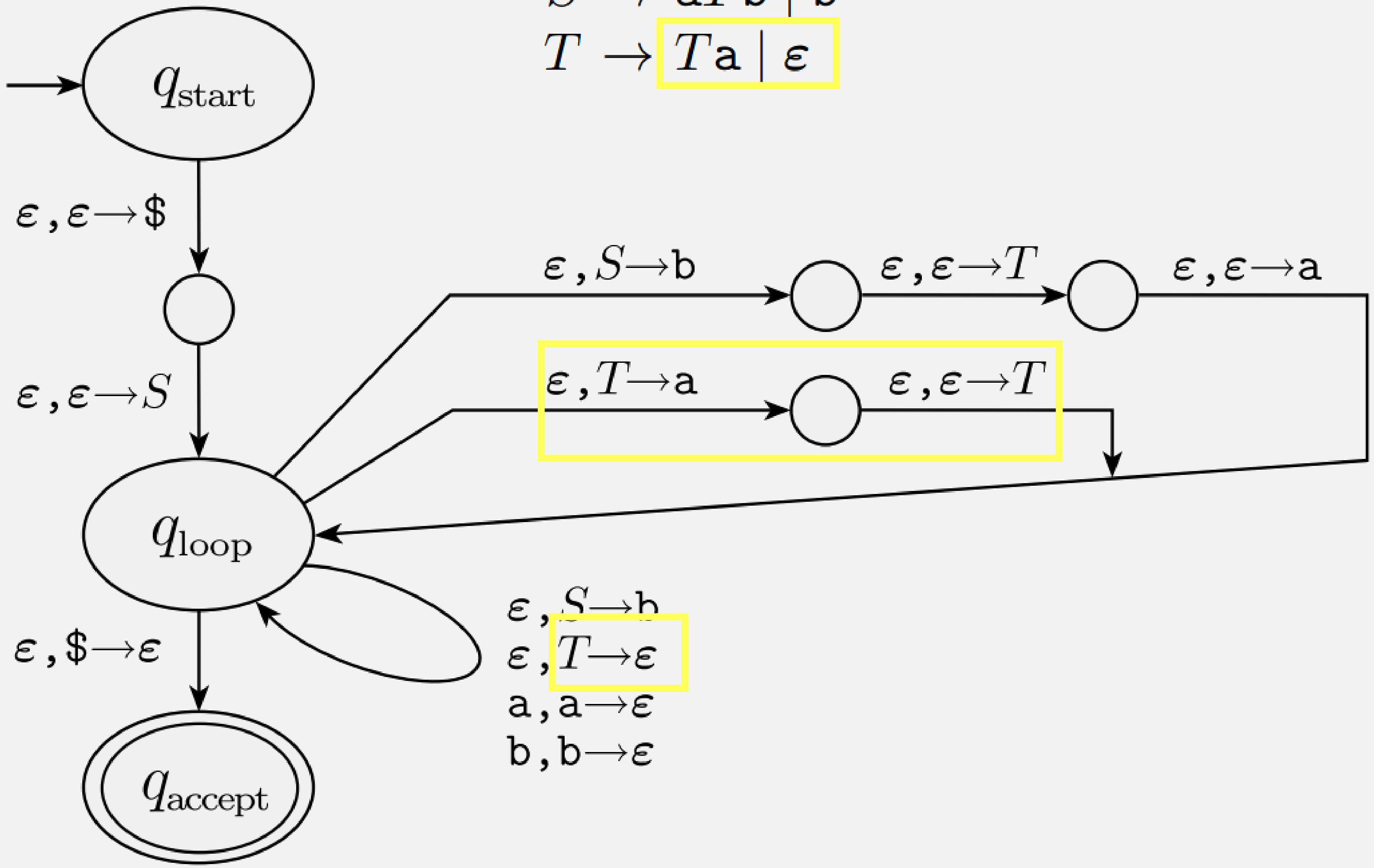
Example **CFG→PDA**

$$S \to \boxed{\mathtt{a}T\mathtt{b} \mid \mathtt{b}}$$
$$T \to T\mathtt{a} \mid \varepsilon$$



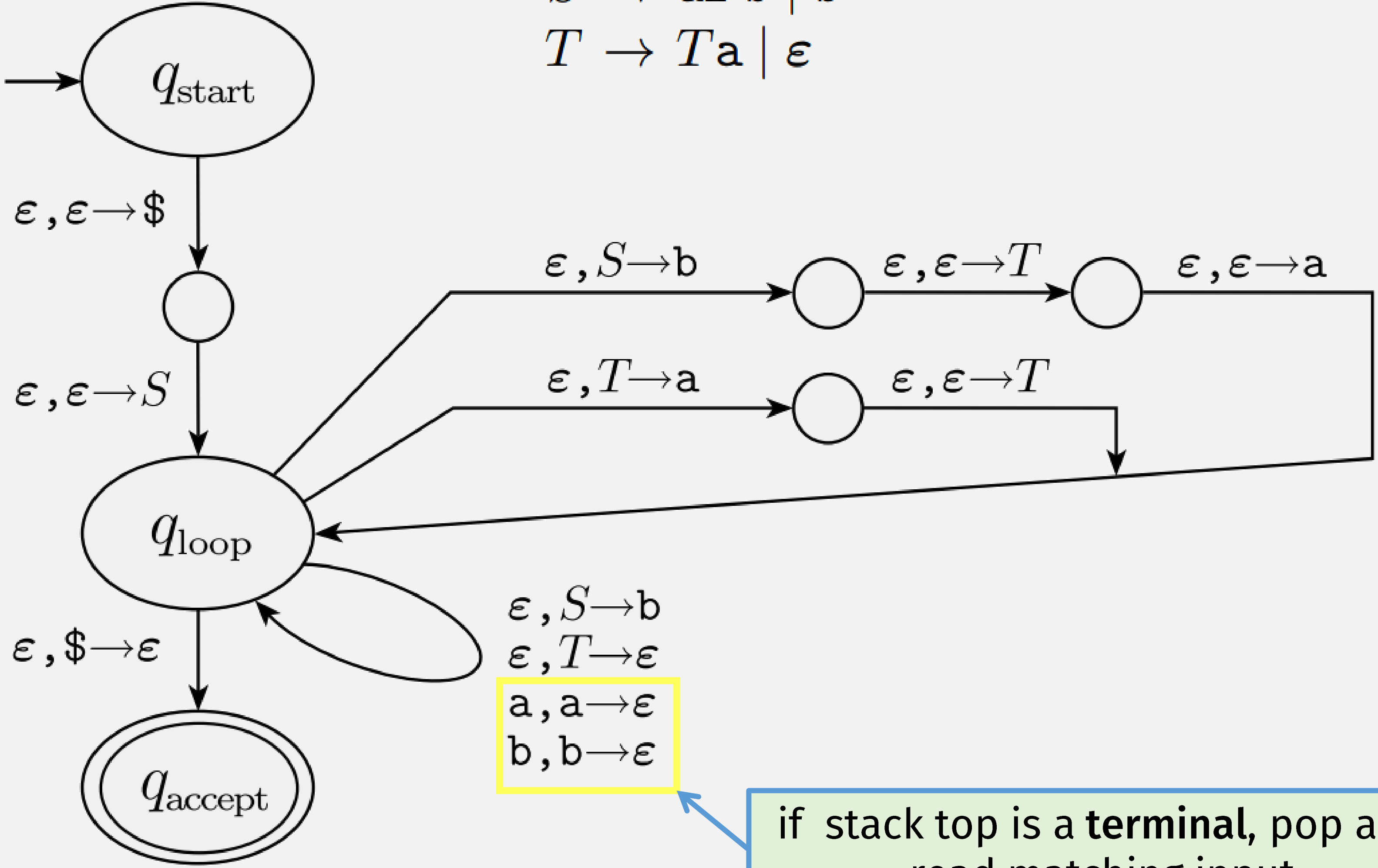
Example CFG→PDA

$$S \to \mathtt{a}T\mathtt{b} \mid \mathtt{b}$$
$$T \to \boxed{T\mathtt{a} \mid \varepsilon}$$



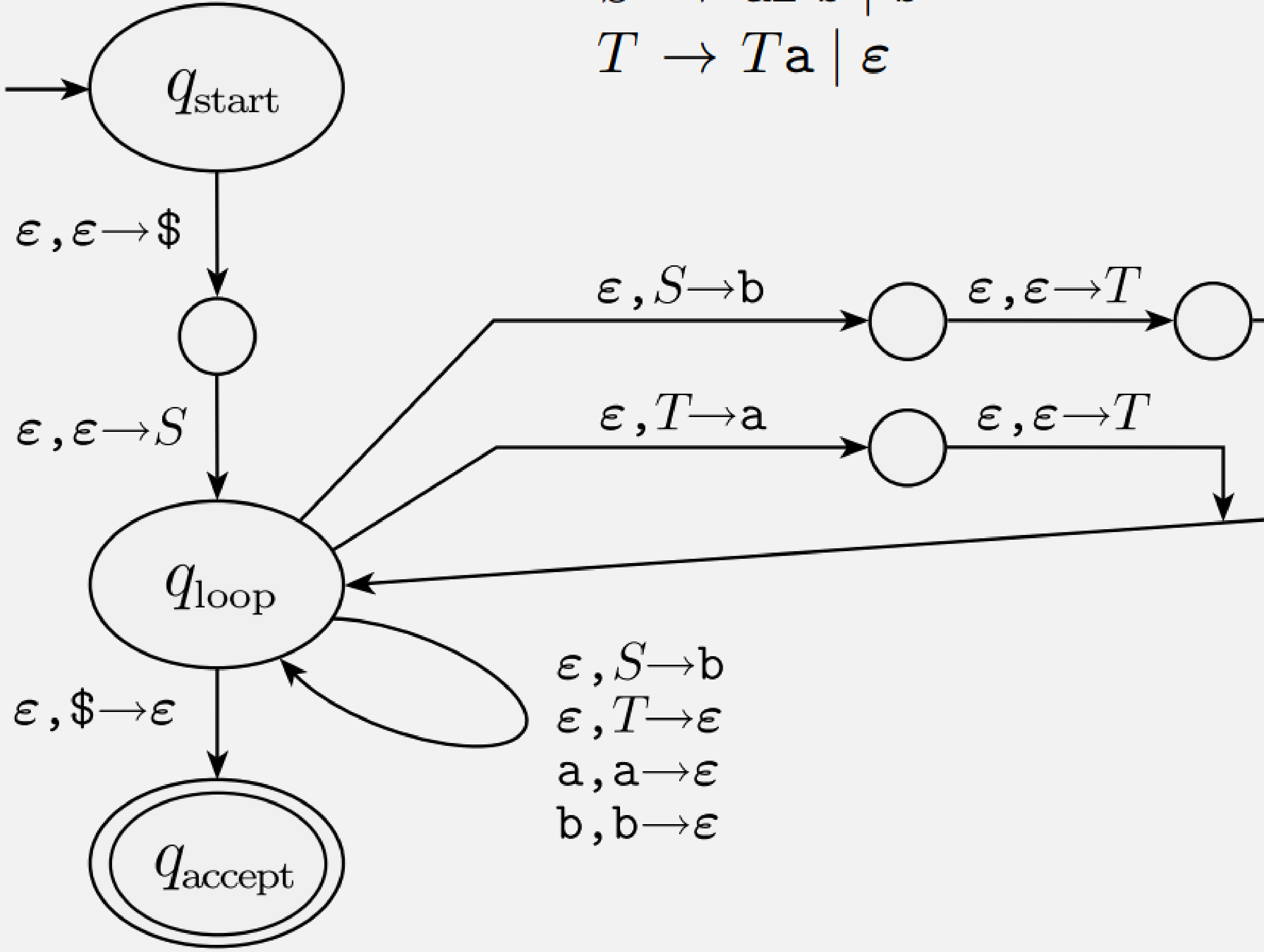
Example CFG→PDA

$$S \to \mathtt{a}T\mathtt{b} \mid \mathtt{b}$$
$$T \to T\mathtt{a} \mid \varepsilon$$



if stack top is a **terminal**, pop and read matching input

Example **CFG→PDA**



<u>Example Derivation using CFG:</u>

$S \Rightarrow \mathtt{a}T\mathtt{b}$ (using rule $S \to \mathtt{a}T\mathtt{b}$)

$\Rightarrow \mathtt{a}T\mathtt{ab}$ (using rule $T \to T\mathtt{a}$)

$\Rightarrow \mathtt{aab}$ (using rule $T \to \varepsilon$)

<u>PDA Example</u>

State	Input	Stack	Equiv Rule
q_{start}	aab		
q_{loop}	aab	$S\$$	
q_{loop}	aab	$\mathtt{a}T\mathtt{b}\$$	$S \to \mathtt{a}T\mathtt{b}$
q_{loop}	ab	$T\mathtt{b}\$$	
q_{loop}	ab	$T\mathtt{ab}\$$	$T \to T\mathtt{a}$
q_{loop}	ab	ab$	$T \to \varepsilon$
q_{loop}	b	b$	
q_{loop}		$	
q_{accept}			

Example **CFG→PDA**

Example Derivation using CFG:

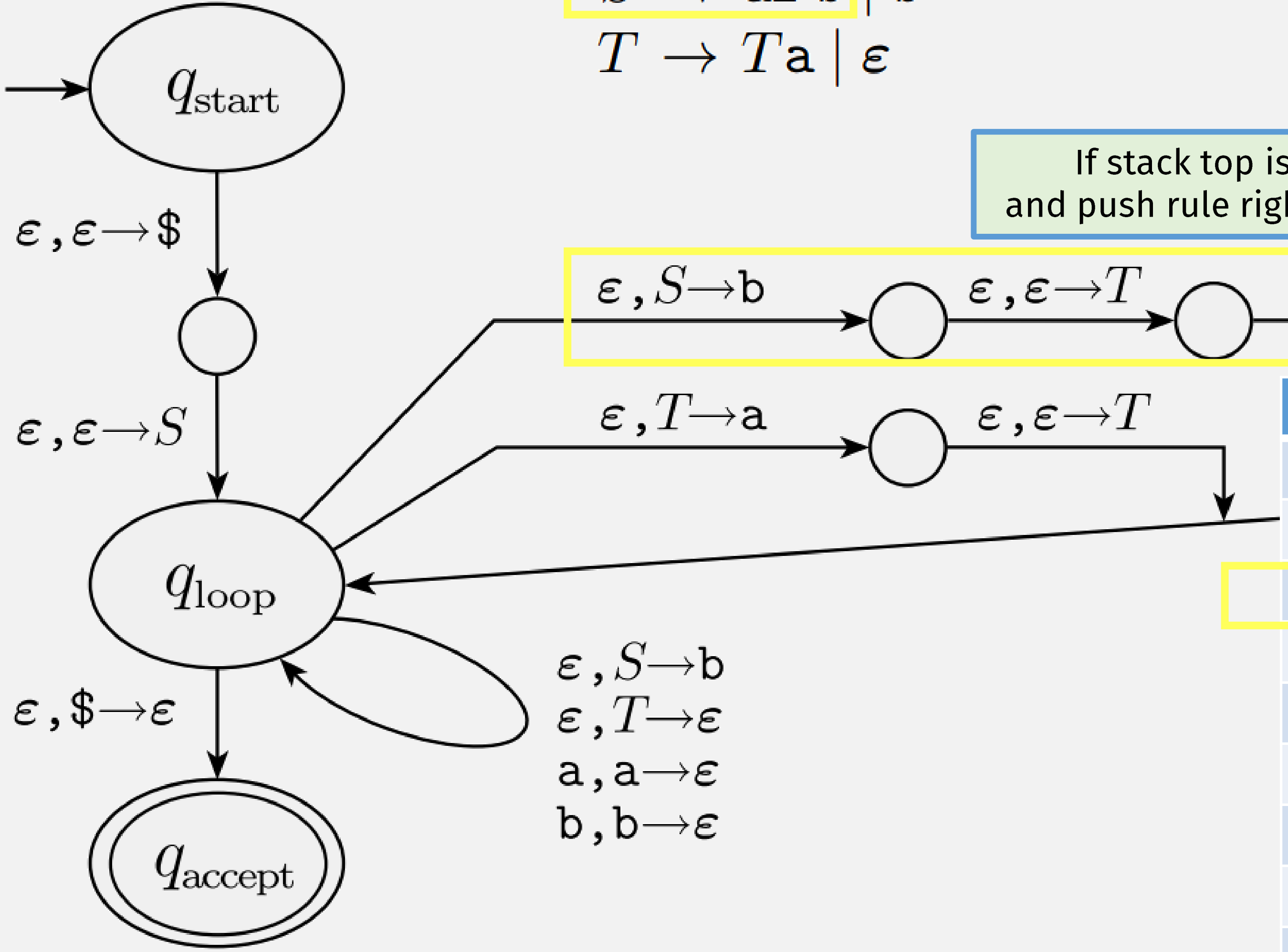
$S \Rightarrow \mathtt{a}T\mathtt{b}$ (using rule $S \rightarrow \mathtt{a}T\mathtt{b}$)

$\Rightarrow \mathtt{a}T\mathtt{ab}$ (using rule $T \rightarrow T\mathtt{a}$)

$\Rightarrow \mathtt{aab}$ (using rule $T \rightarrow \varepsilon$)

$$S \rightarrow \mathtt{a}T\mathtt{b} \mid \mathtt{b}$$
$$T \rightarrow T\mathtt{a} \mid \varepsilon$$

If stack top is **variable** S, pop S
and push rule right-sides (in rev order)



PDA Example

State	Input	Stack	Equiv Rule
q_{start}	**aab**		
q_{loop}	**aab**	$S\$$	
q_{loop}	**aab**	$\mathtt{a}T\mathtt{b}\$$	$S \rightarrow \mathtt{a}T\mathtt{b}$
q_{loop}	**ab**	$T\mathtt{b}\$$	
q_{loop}	**ab**	$T\mathtt{ab}\$$	$T \rightarrow T\mathtt{a}$
q_{loop}	**ab**	$\mathtt{ab}\$$	$T \rightarrow \varepsilon$
q_{loop}	**b**	$\mathtt{b}\$$	
q_{loop}		$\$$	
q_{accept}			

Example **CFG→PDA**

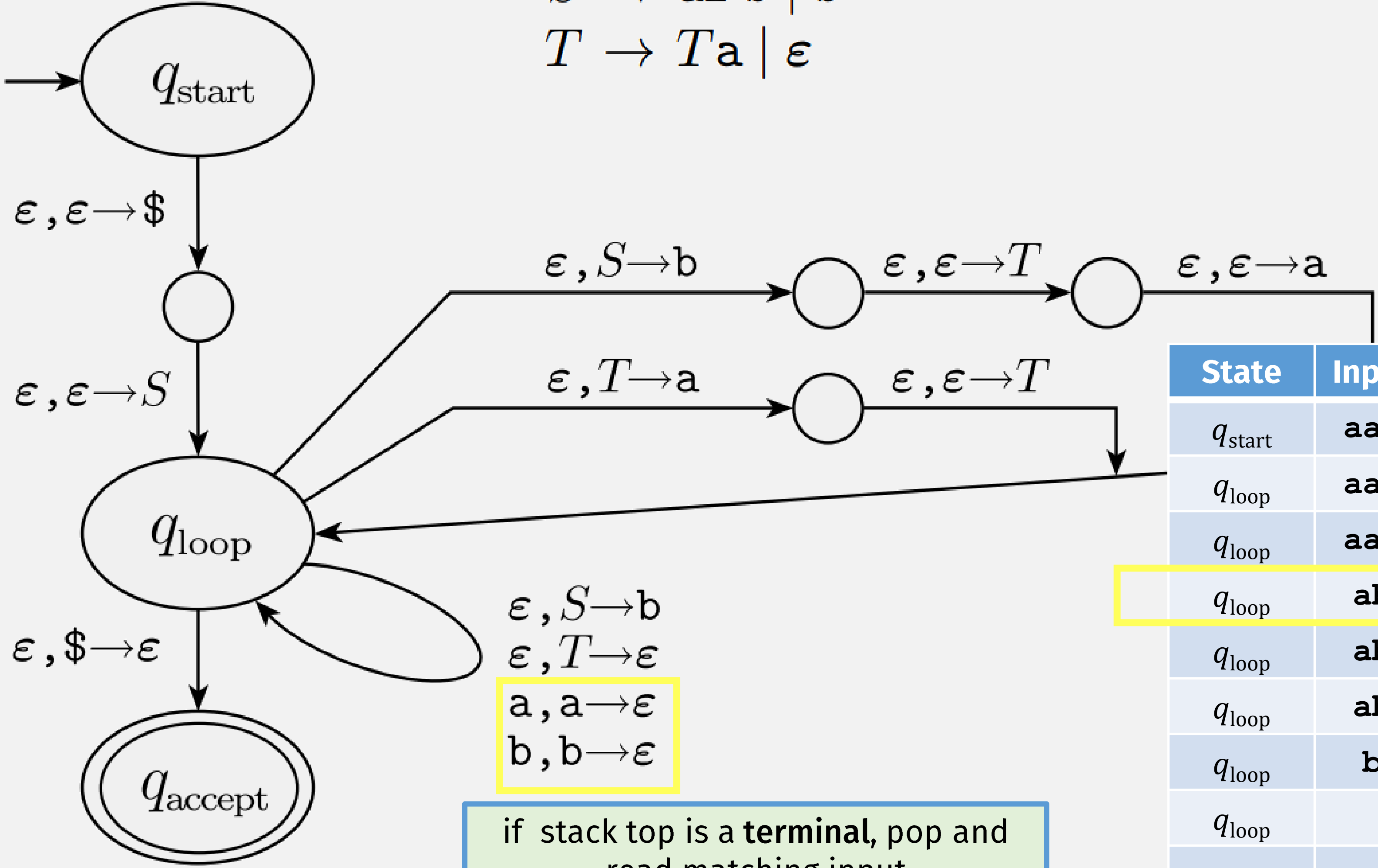
Example Derivation using CFG:

$S \Rightarrow \mathtt{a}T\mathtt{b}$ (using rule $S \to \mathtt{a}T\mathtt{b}$)

$\Rightarrow \boxed{\mathtt{a}}T\mathtt{ab}$ (using rule $T \to T\mathtt{a}$)

$\Rightarrow \mathtt{aab}$ (using rule $T \to \varepsilon$)

$$S \to \mathtt{a}T\mathtt{b} \mid \mathtt{b}$$
$$T \to T\mathtt{a} \mid \varepsilon$$



if stack top is a **terminal**, pop and read matching input

PDA Example

State	Input	Stack	Equiv Rule
q_{start}	aab		
q_{loop}	aab	$S\$$	
q_{loop}	aab	$\mathtt{a}T\mathtt{b}\$$	$S \to \mathtt{a}T\mathtt{b}$
q_{loop}	ab	$T\mathtt{b}\$$	
q_{loop}	ab	$T\mathtt{ab}\$$	$T \to T\mathtt{a}$
q_{loop}	ab	ab$	$T \to \varepsilon$
q_{loop}	b	b$	
q_{loop}		$	
q_{accept}			

Example **CFG→PDA**

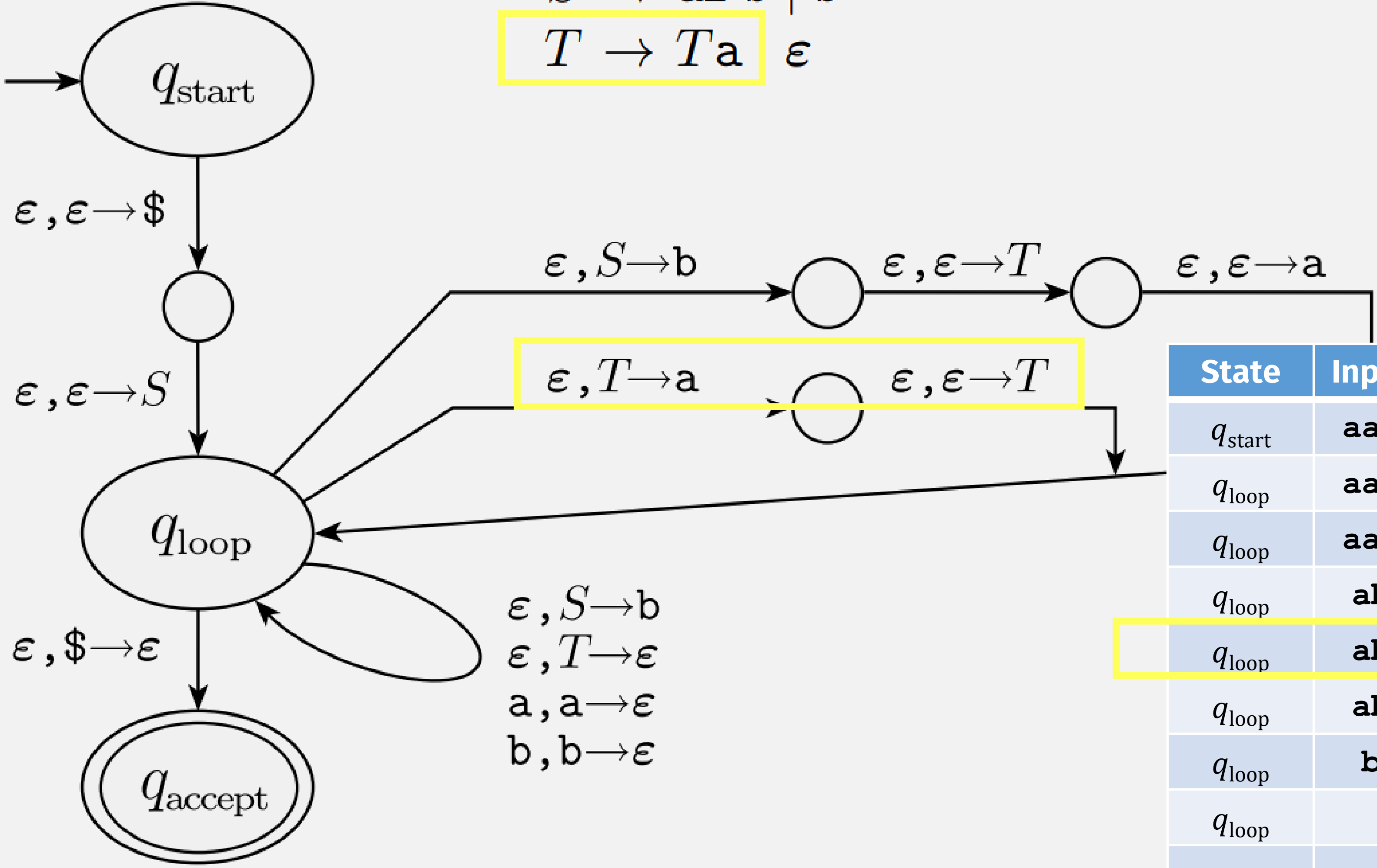
<u>Example Derivation using CFG:</u>

$S \Rightarrow \mathtt{a}T\mathtt{b}$ (using rule $S \to \mathtt{a}T\mathtt{b}$)

$\Rightarrow \mathtt{a}T\mathtt{ab}$ (using rule $T \to T\mathtt{a}$)

$\Rightarrow \mathtt{aab}$ (using rule $T \to \varepsilon$)

$$S \to \mathtt{a}T\mathtt{b} \mid \mathtt{b}$$
$$T \to T\mathtt{a} \mid \varepsilon$$



<u>PDA Example</u>

State	Input	Stack	Equiv Rule
q_{start}	$\mathtt{aab}$		
q_{loop}	$\mathtt{aab}$	$S\$$	
q_{loop}	$\mathtt{aab}$	$\mathtt{a}T\mathtt{b}\$$	$S \to \mathtt{a}T\mathtt{b}$
q_{loop}	$\mathtt{ab}$	$T\mathtt{b}\$$	
q_{loop}	$\mathtt{ab}$	$T\mathtt{ab}\$$	$T \to T\mathtt{a}$
q_{loop}	$\mathtt{ab}$	$\mathtt{ab}\$$	$T \to \varepsilon$
q_{loop}	$\mathtt{b}$	$\mathtt{b}\$$	
q_{loop}		$\$$	
q_{accept}			

A lang is a CFL **iff** some PDA recognizes it

☑ ⇒ If a language is a CFL, then a PDA recognizes it
- Convert CFG→PDA

⇐ If a PDA recognizes a language, then it's a CFL
- (Harder)
- Need to: Convert PDA→CFG

PDA→CFG: Prelims

Before converting PDA to CFG, modify it so :

1. It has a single accept state, q_{accept}.
2. It empties its stack before accepting.
3. Each transition either pushes a symbol onto the stack (a *push* move) or pops one off the stack (a *pop* move), but it does not do both at the same time.

<u>Important</u>:
This doesn't change the language recognized by the PDA
(confirm this to yourselves)

PDA P -> CFG G: Variables

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$ variables of G are $\{A_{pq} \mid p, q \in Q\}$

- Want: if P goes from state p to q reading input x, then some A_{pq} generates x

- So: For every pair of states p, q in P, add variable A_{pq} to G

- Then: connect the variables together by,
  - Add rules: $A_{pq} \rightarrow A_{pr}A_{rq}$, for each state r
  - These rules allow grammar to simulate every possible transition
  - (We haven't added input read/generated terminals yet)

- To add terminals: pair up stack pushes and pops (essence of a CFL)

PDA P -> CFG G: Generating Strings

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$

variables of G are $\{A_{pq} \mid p, q \in Q\}$

- <u>The key</u>: pair up stack pushes and pops (essence of a CFL)

if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),

put the rule $A_{pq} \to aA_{rs}b$ in G

PDA P -> CFG G: Generating Strings

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$

variables of G are $\{A_{pq} \mid p, q \in Q\}$

- <u>The key</u>: **pair up stack pushes and pops** (essence of a CFL)

if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),

put the rule $A_{pq} \rightarrow aA_{rs}b$ in G

PDA P -> CFG G: Generating Strings

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$ variables of G are $\{A_{pq} \mid p, q \in Q\}$

- <u>The key</u>: **pair up stack pushes and pops** (essence of a CFL)

if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε),

put the rule $A_{pq} \to aA_{rs}b$ in G

A language is a CFL ⇔ A PDA recognizes it

☑ ⇒ If a language is a CFL, then a PDA recognizes it

- Convert **CFG→PDA**

☑ ⇐ If a PDA recognizes a language, then it's a CFL

- Convert **PDA→CFG**

$\blacksquare$

Check-in Quiz 10/4

On Gradescope