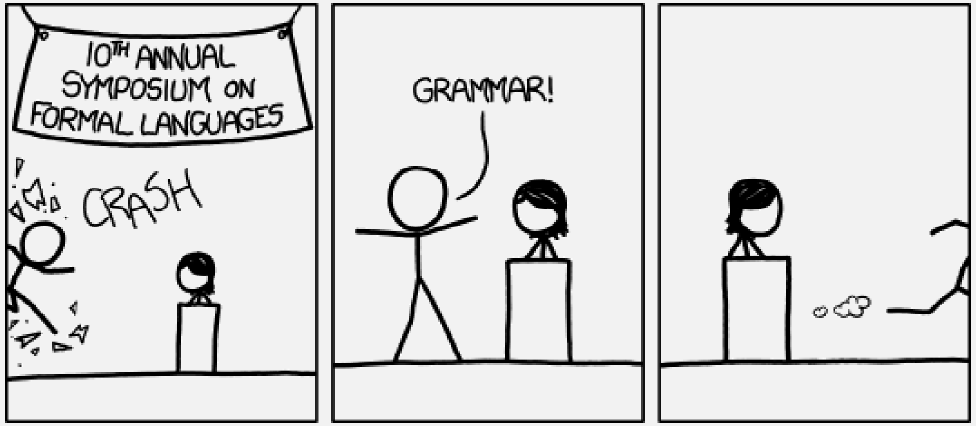
Pushdown Automata (PDAs)

Wednesday, March 3, 2021

Announcements



- HW5 deadline extended
  - Now due: Wed 3/10 11:59pm EST

- Reminder: Spring Break Mon 3/15 – Sun 3/21
  - No classes

Last Time:

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

Today

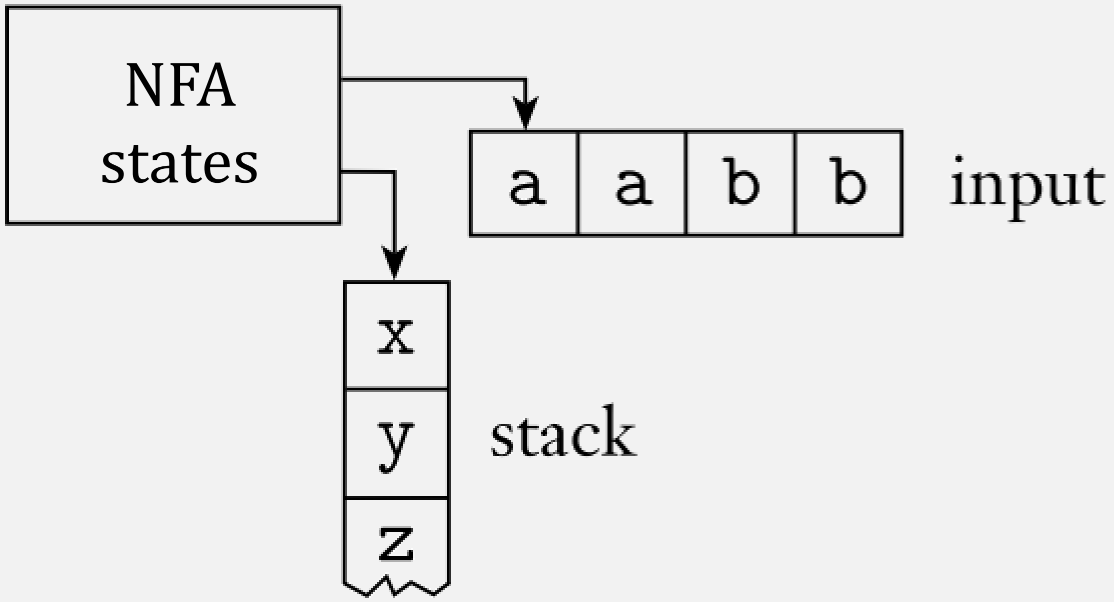
Regular Languages	Context-Free Languages (CFLs)
Regular Expression (Regexp)	Context-Free Grammar (CFG)
A Reg expr describes a Regular lang	A CFG describes a CFL
	TODAY:
Finite automaton (FSM)	Push-down automaton (PDA)
An FSM recognizes a Regular lang	A PDA recognizes a CFL

Today

Regular Languages	Context-Free Languages (CFLs)
Regular Expression (Regexp)	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

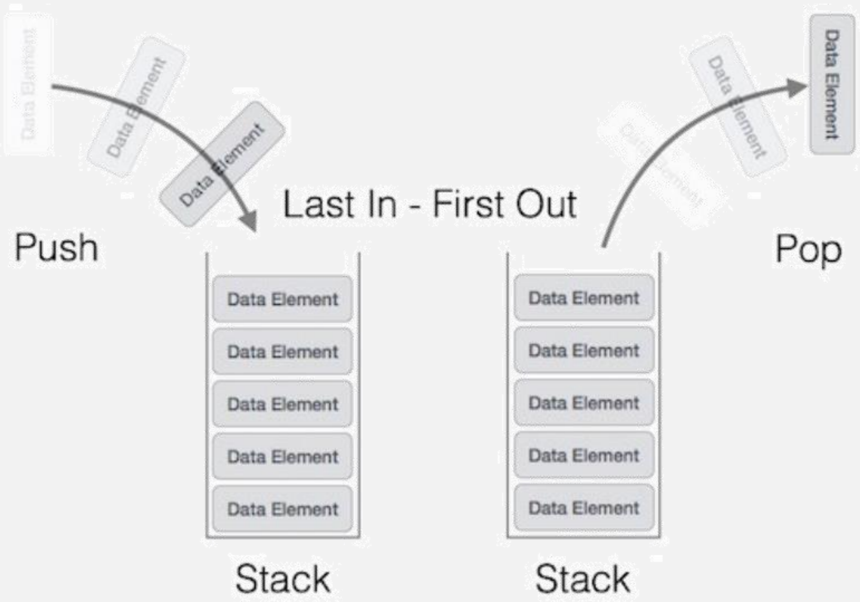
Pushdown Automata (PDA)

- PDA = NFA + a stack



A (Mathematical) Stack Specification

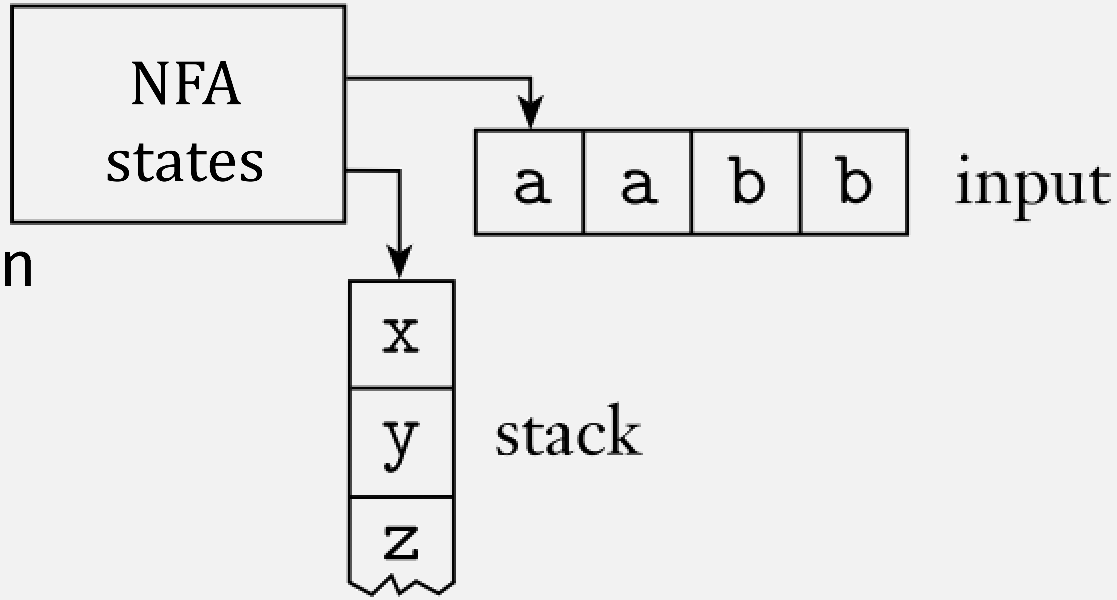
- Access to top element of stack only
- Operations: push, pop



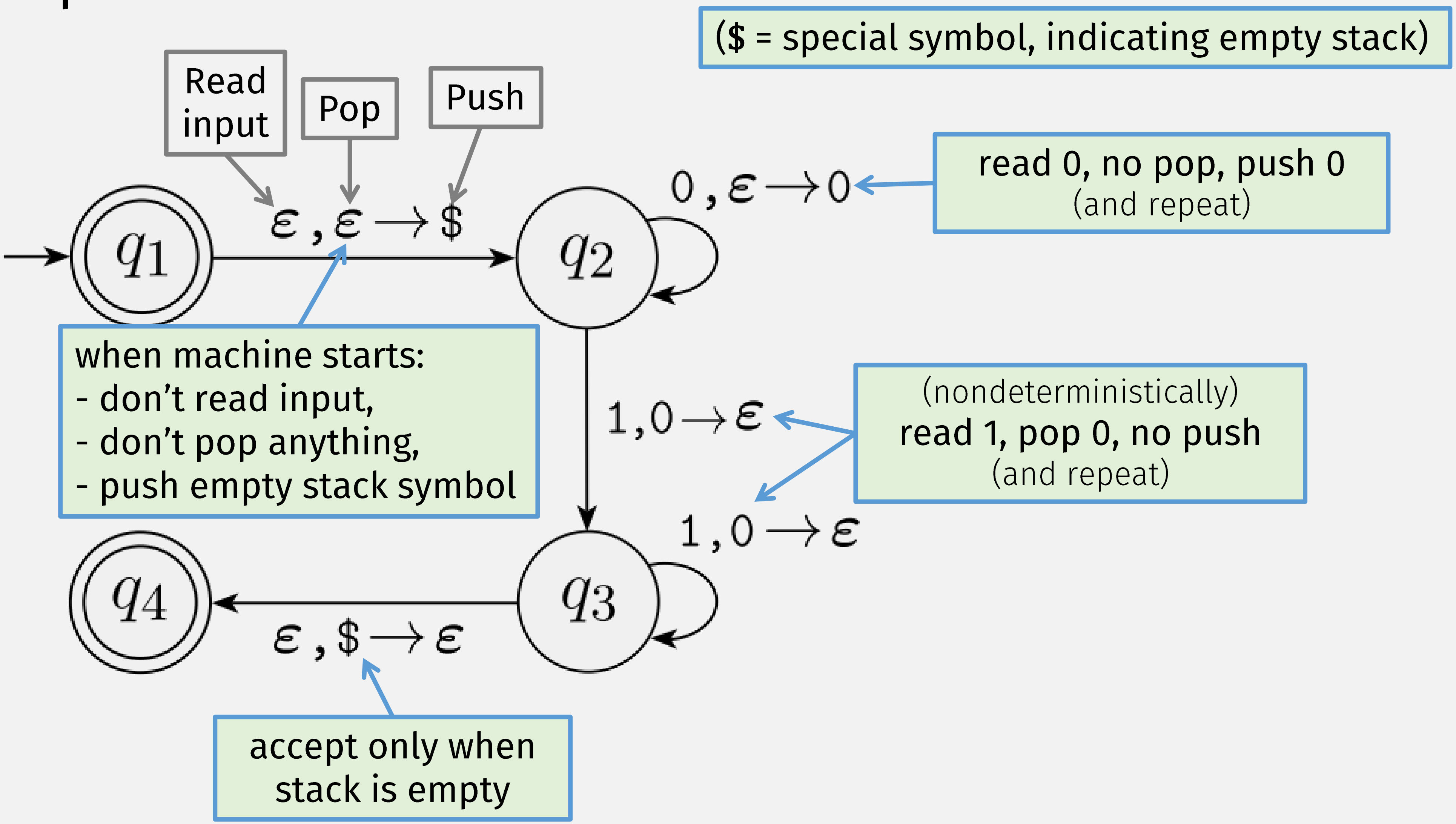
- (What could be a possible <u>data representation</u> in code?)

Pushdown Automata (PDA)

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



An Example PDA $\{0^n 1^n | n \geq 0\}$



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.

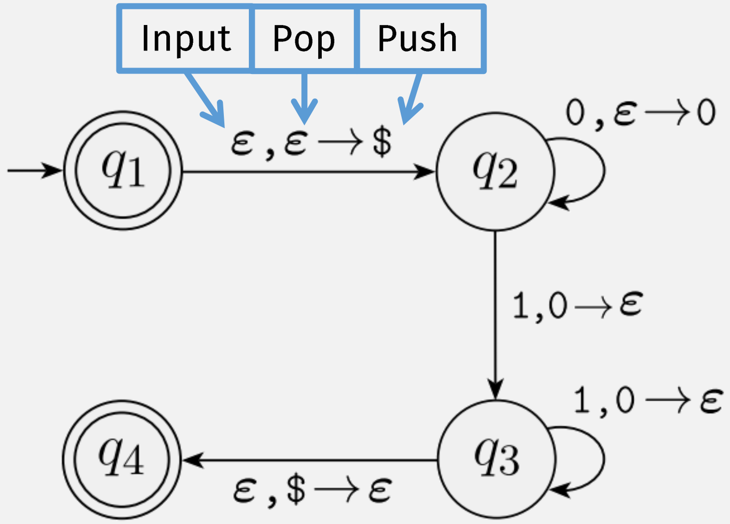
Stack alphabet can have special stack symbols, e.g., $

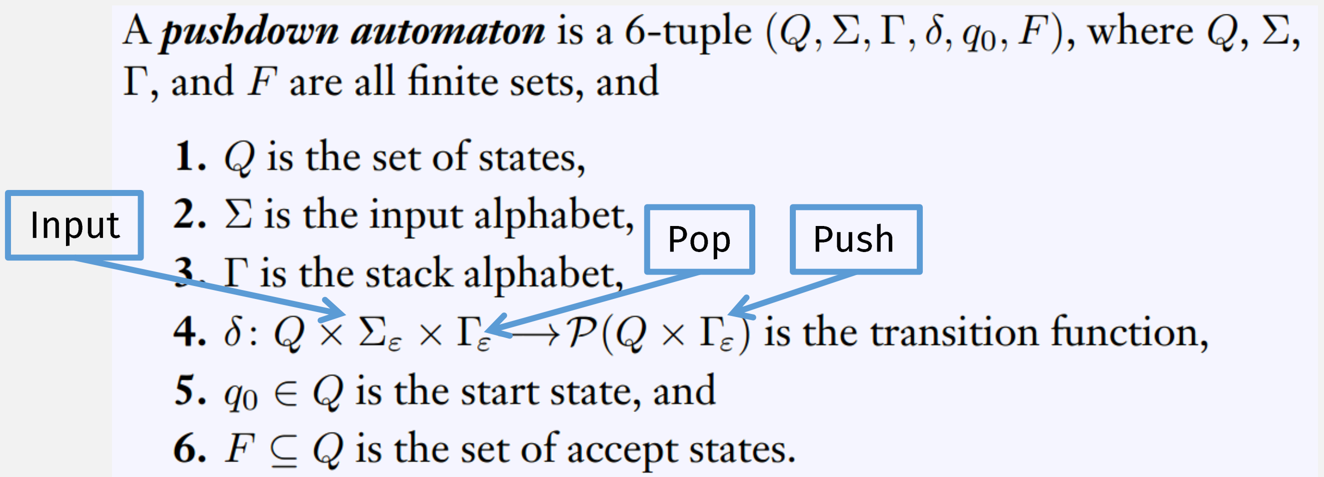
Input

Pop

Push

In-class example





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.

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

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

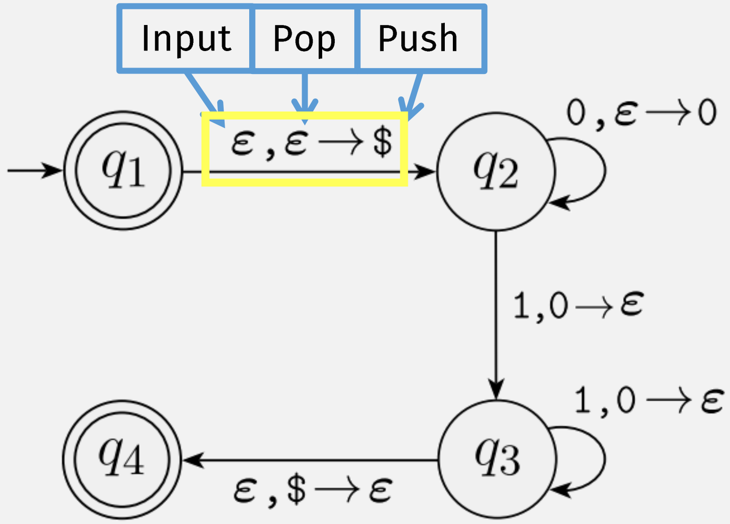
$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, 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			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Input
Pop
Push
1 2 3 4 5



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

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

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

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

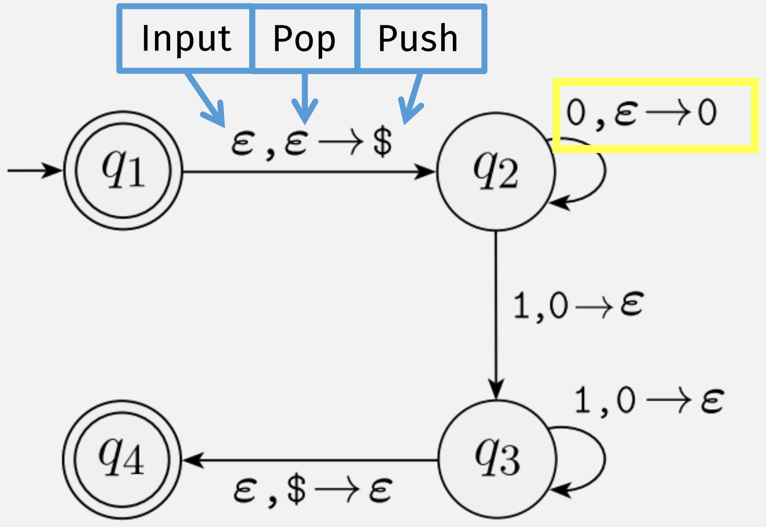
$F = \{q_1, q_4\}$, and

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			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Input Pop Push

1 2 3 4 5



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

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

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

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

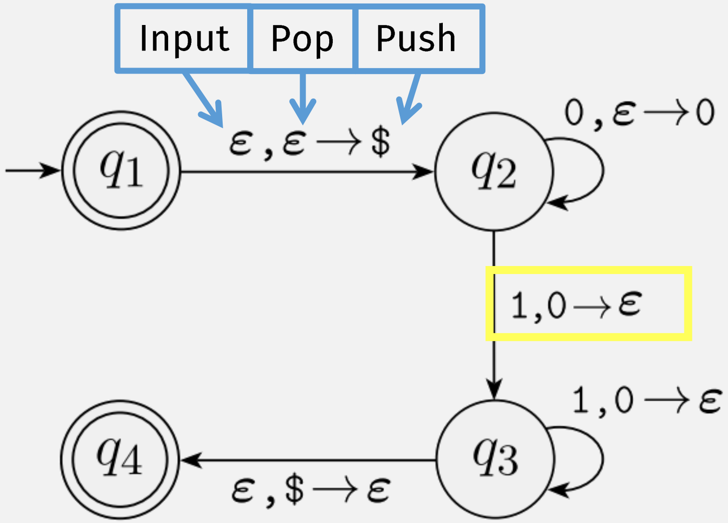
$F = \{q_1, q_4\}$, and

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			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Input Pop Push

1 2 3 4 5



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

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

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

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

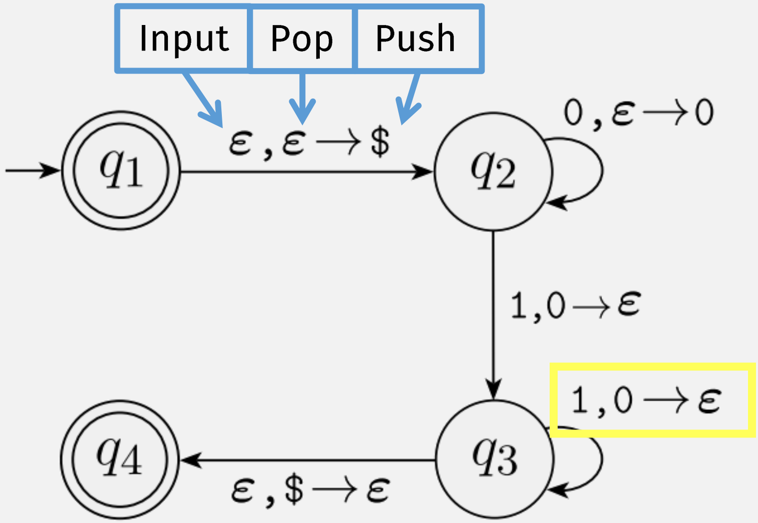
$F = \{q_1, q_4\}$, and

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			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Input — Pop — Push

1 2 3 4 5



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

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

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

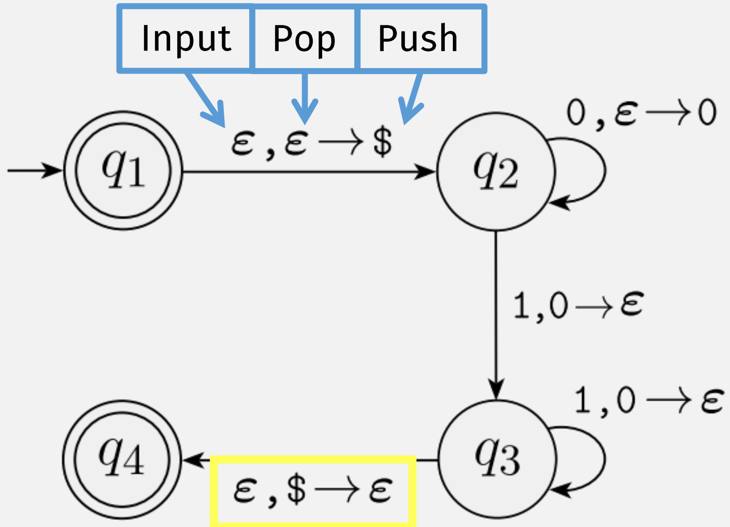
$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, 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									$\{(q_2, \$)\}$ 5
q_2			$\{(q_2, 0)\}$ 1	$\{(q_3, \varepsilon)\}$ 2					
q_3				$\{(q_3, \varepsilon)\}$ 3				4 $\{(q_4, \varepsilon)\}$	
q_4									

Input
Pop
Push



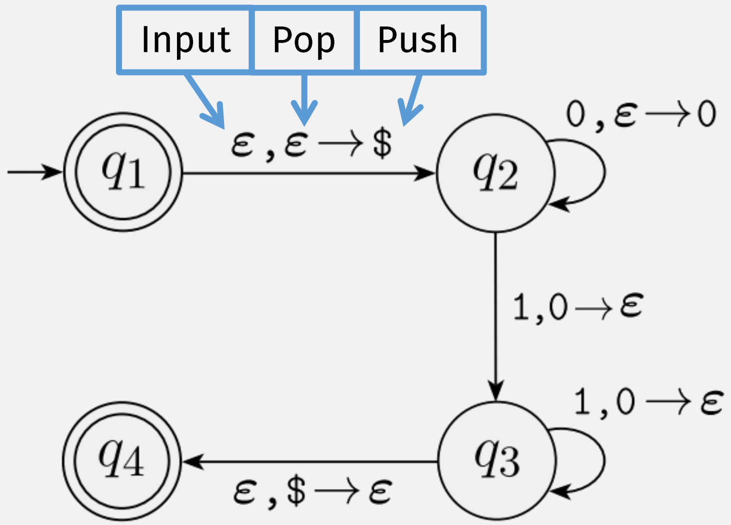
A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

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

Pushdown Automata (PDA)

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



- Want to prove: PDA ⇔ CFG

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

CFL $\Leftrightarrow$ PDA

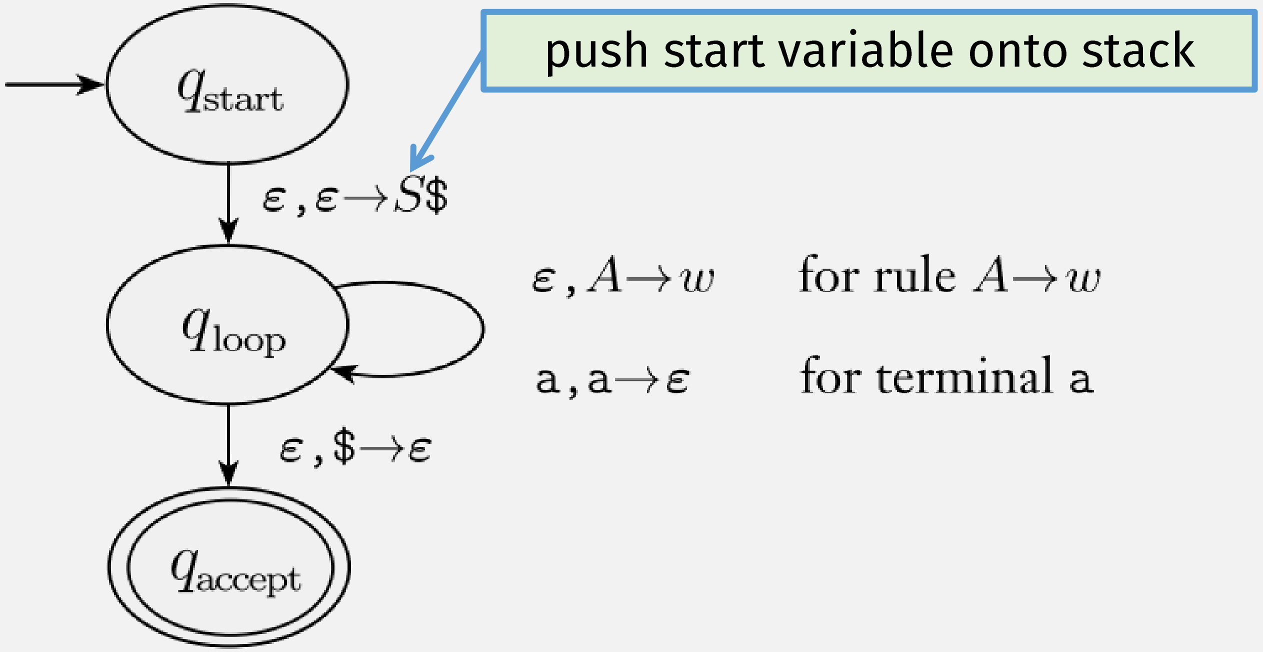
A lang is a CFL **iff** some PDA recognizes it

- => 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

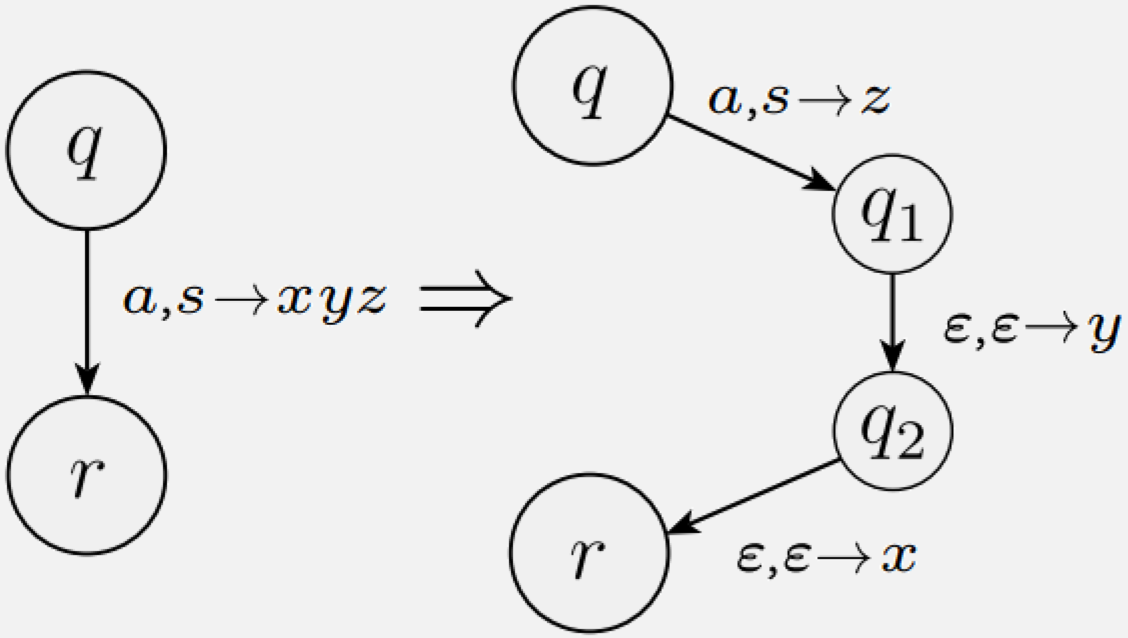
CFG -> PDA

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

- Intuitively, PDA will nondeterministically try all rules



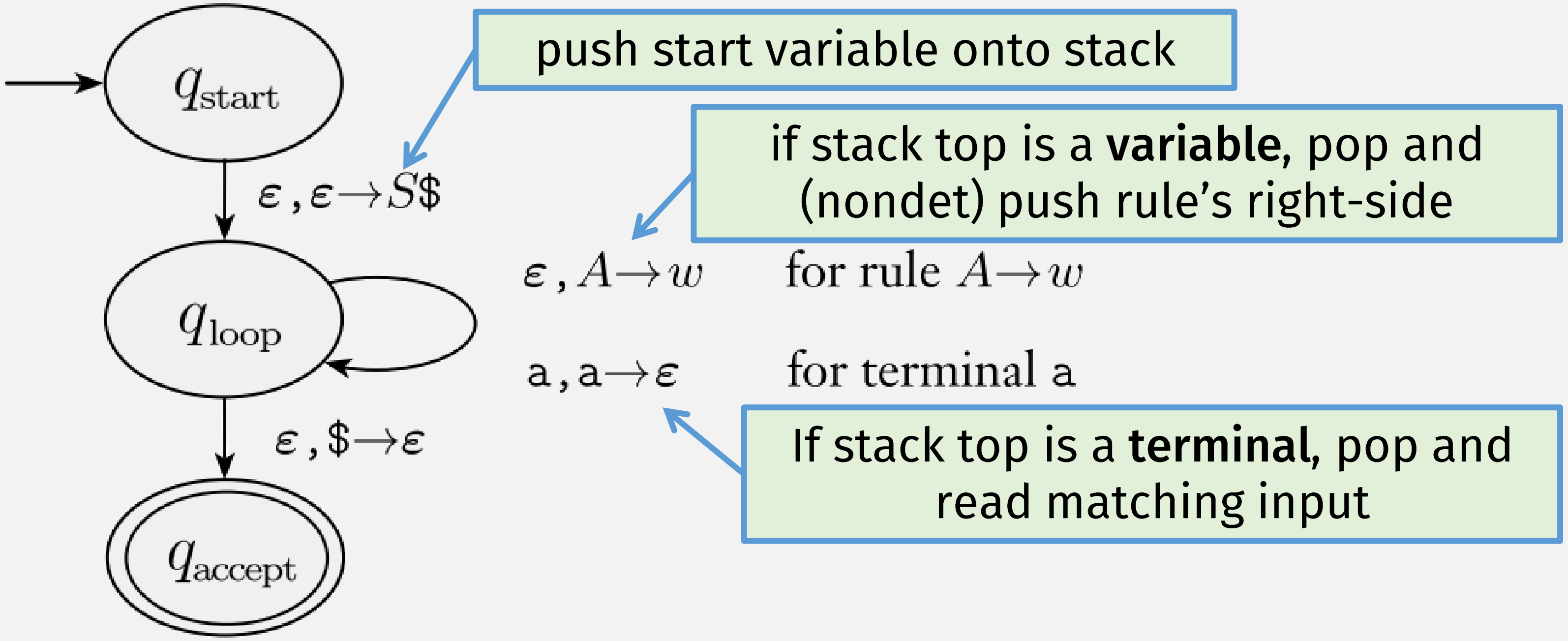
Transition with multiple stack pushes



CFG -> PDA

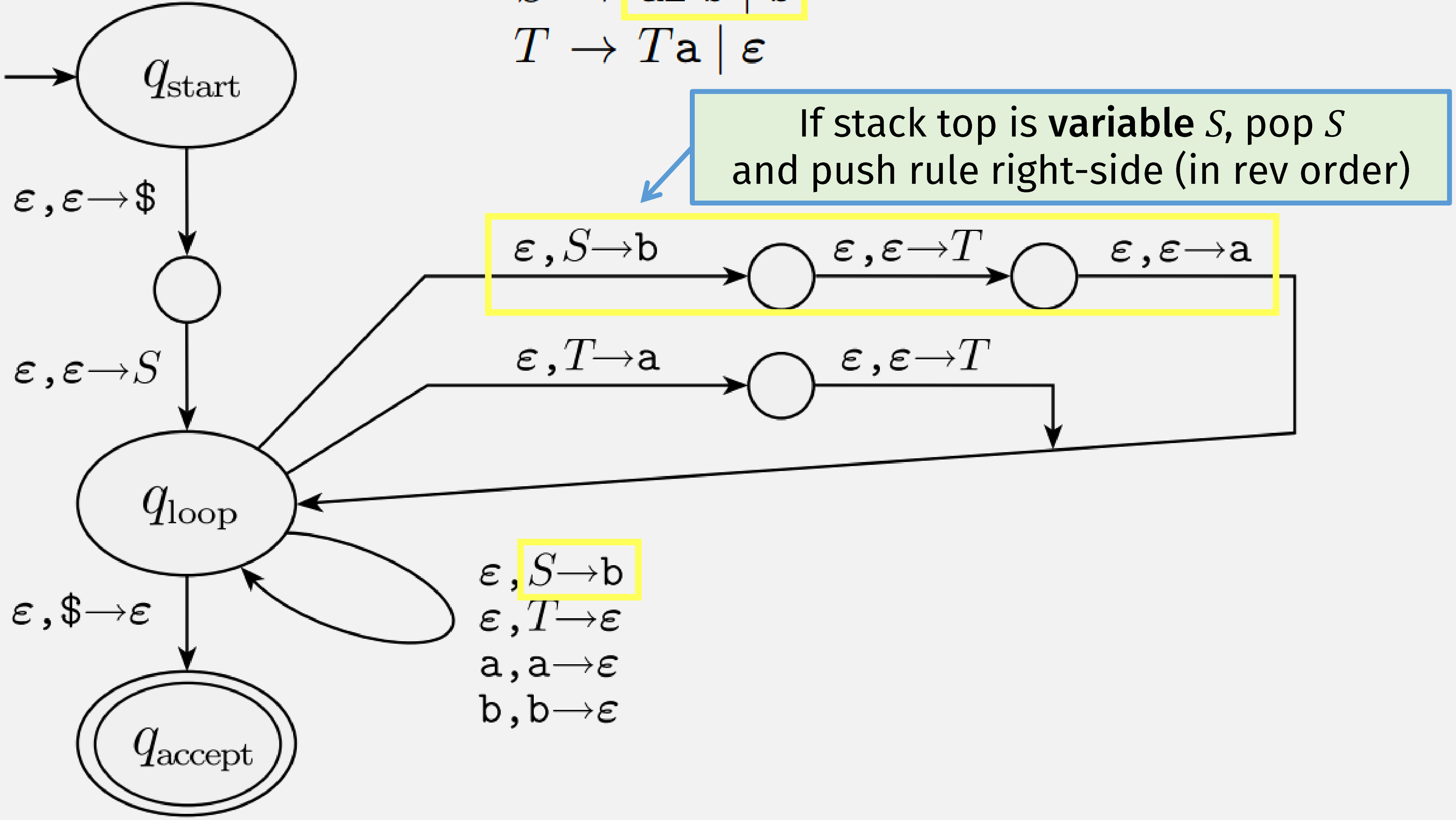
- Construct PDA from CFG such that:
  - PDA accepts input string only if the CFG can generate that string

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



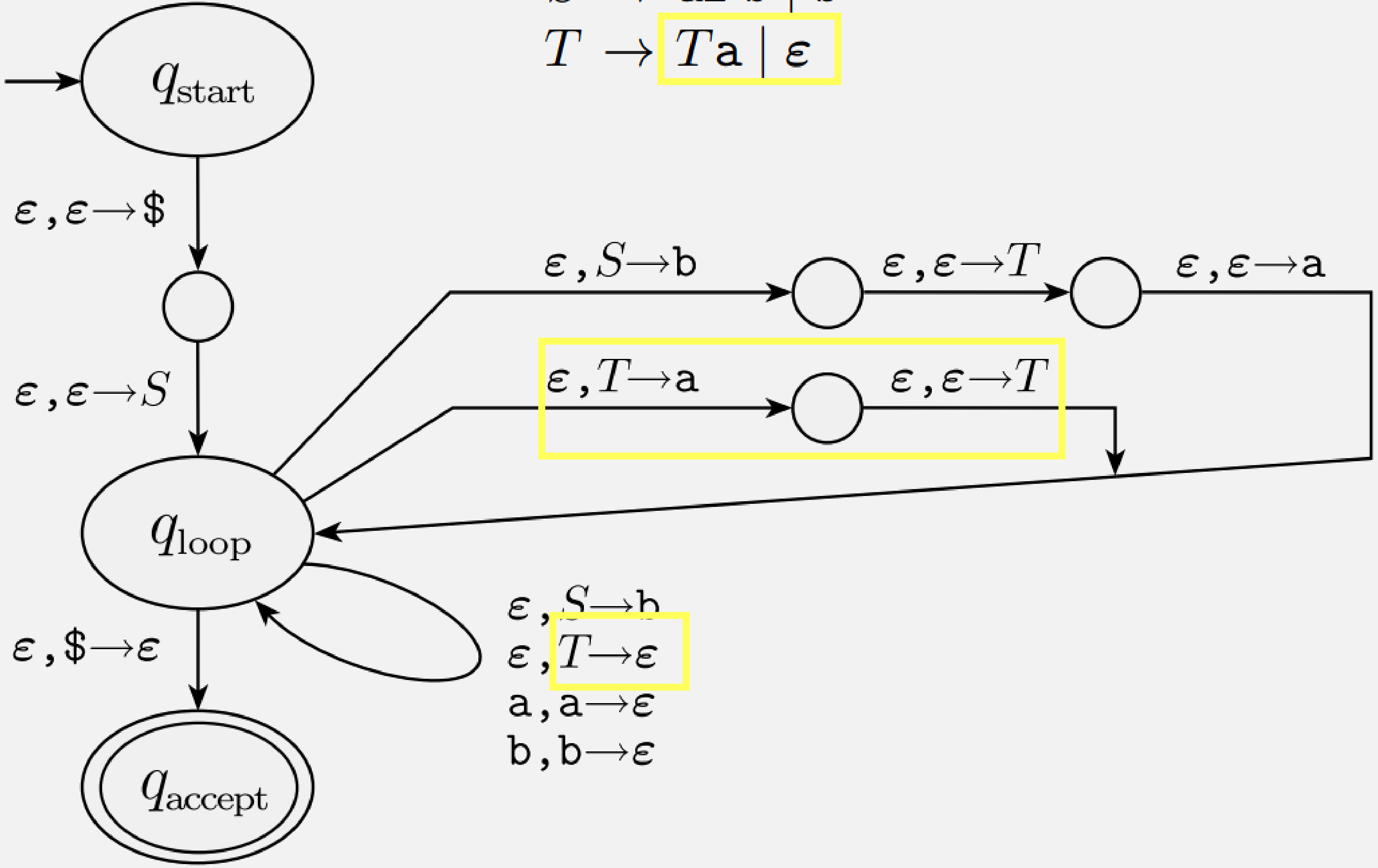
Example CFG -> PDA

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



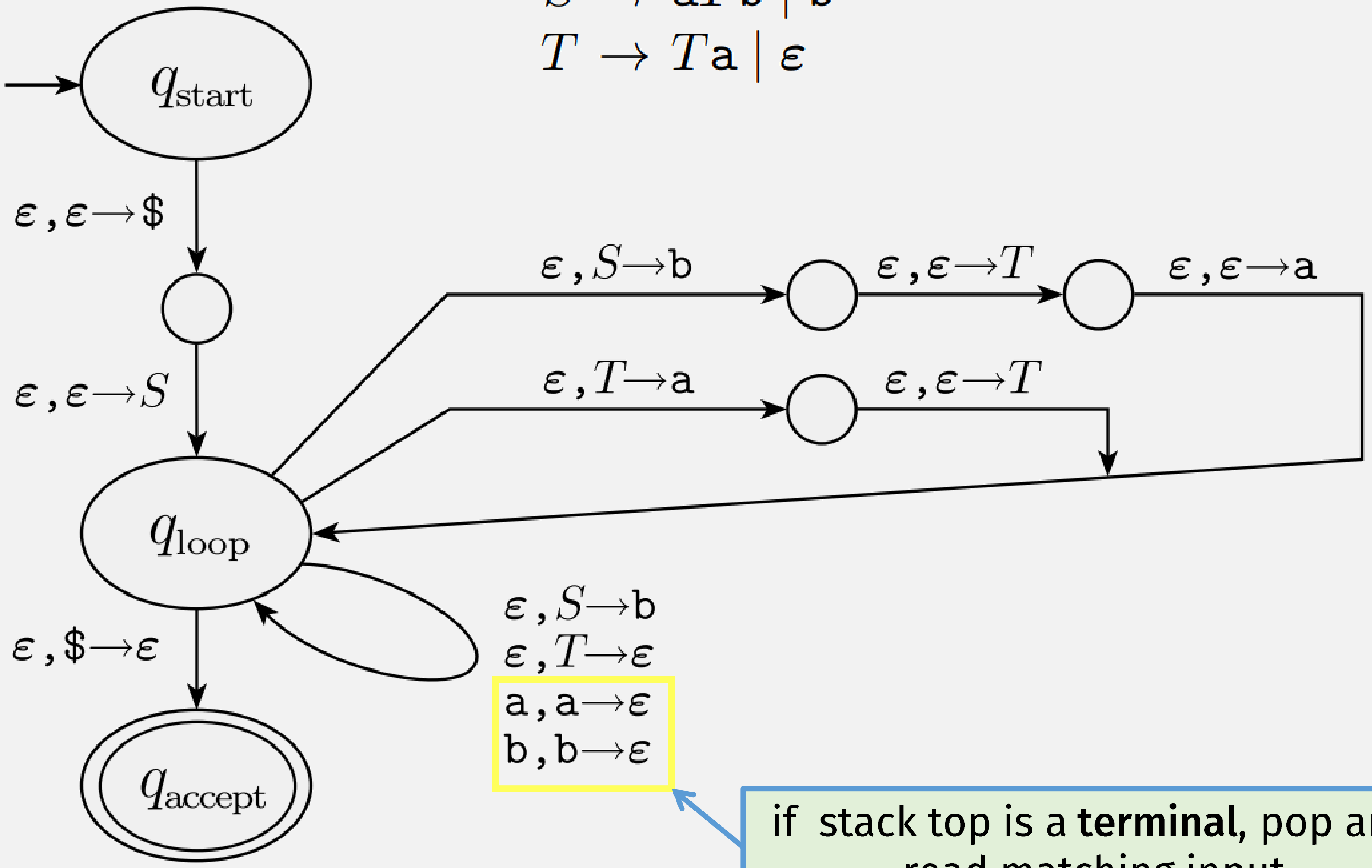
Example CFG -> PDA

$$S \rightarrow aTb \mid b$$
$$T \rightarrow Ta \mid \varepsilon$$



Example CFG -> PDA

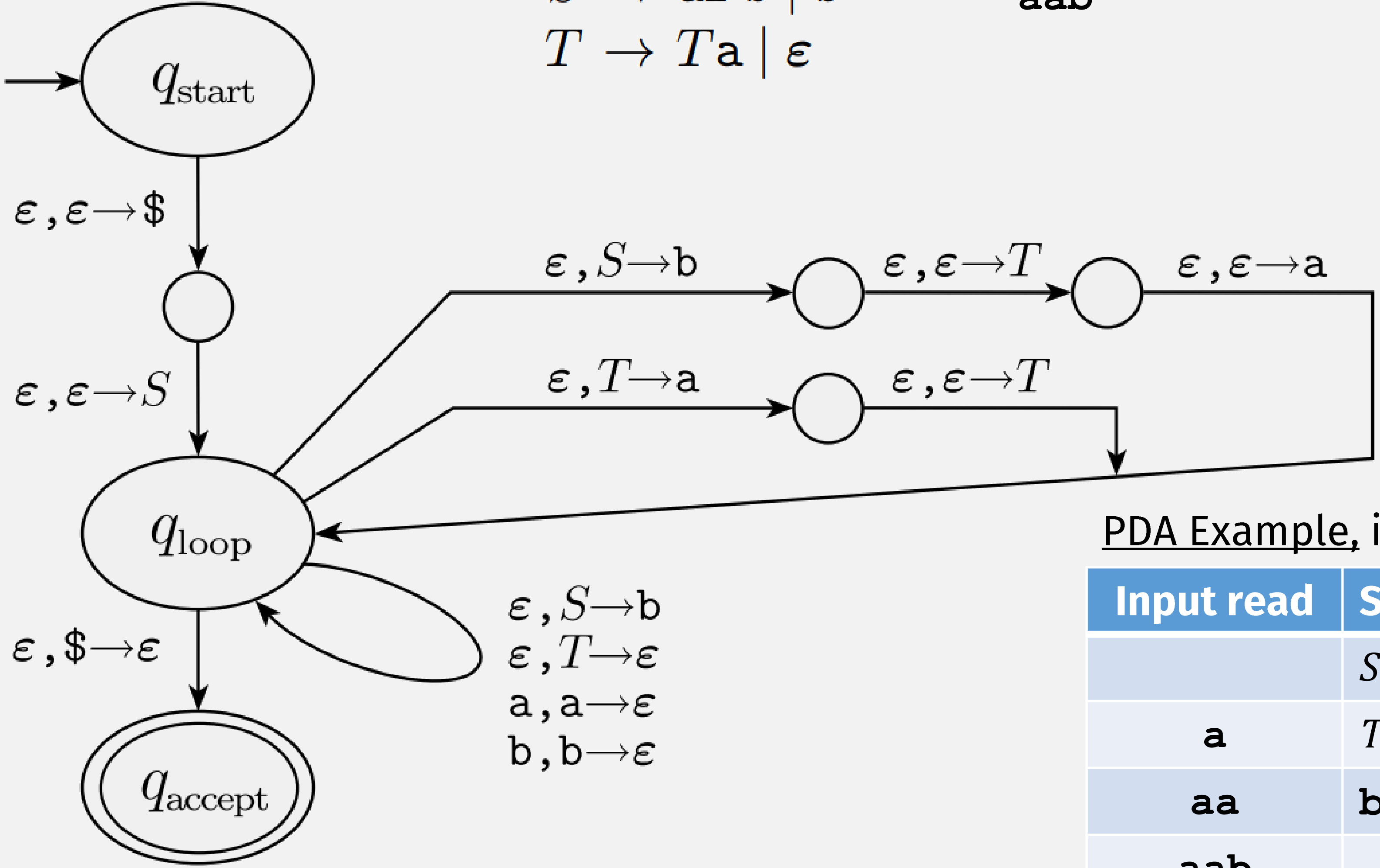
$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$



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

Example CFG -> PDA

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



Example Derivation using CFG:

S ->
a*T***b** ->
a*T***ab** ->
aab

PDA Example, input **aab**

Input read	Stack
	S -> **a***T***b** ->
a	*T***b** -> *T***ab** -> **ab** ->
aa	**b** ->
aab	

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)
  - Need to: Convert CFG -> PDA **(DONE!)**
- <= 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.

Important:
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 $\{A_{pq} | p, q \in Q\}$

- 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 $A_{pq} \to aA_{rs}b$ in G

PDA P -> CFG G: Generating Strings

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

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

- The key: **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

A language is a CFL ⇔ A PDA recognizes it

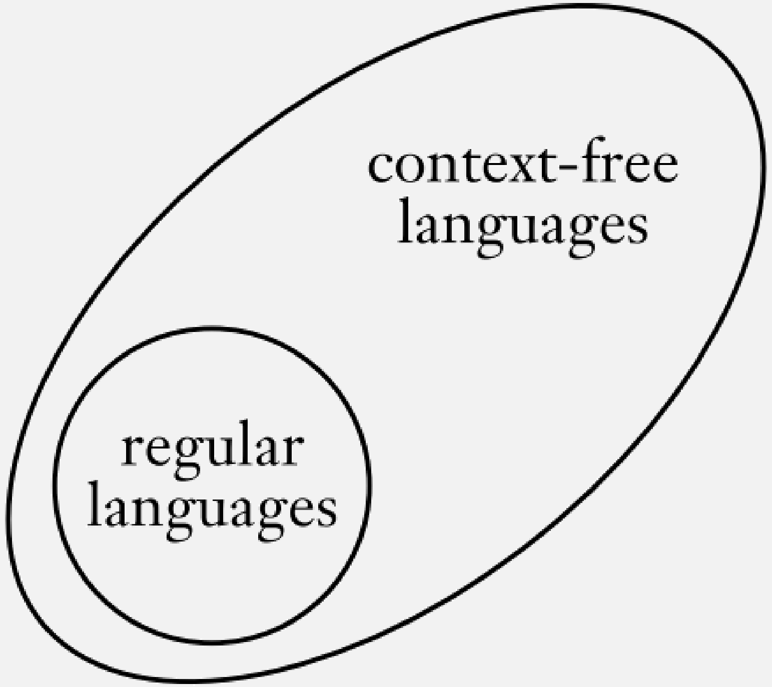
- => If a language is a CFL, then a PDA recognizes it
  - We know: A CFL has a CFG describing it (definition of CFL)
  - Need to: Convert CFG -> PDA (**DONE!**)

- <= If a PDA recognizes a language, then it’s a CFL
  - Need to: Convert PDA -> CFG (**DONE!**)

∎

Regular languages are CFLs: 3 Proofs

- NFA -> PDA (with no stack moves) -> CFG
  - Just now
- DFA -> CFG
  - Textbook page 107
- Regular expression -> CFG
  - HW5



Check-in Quiz 3/3

On Gradescope