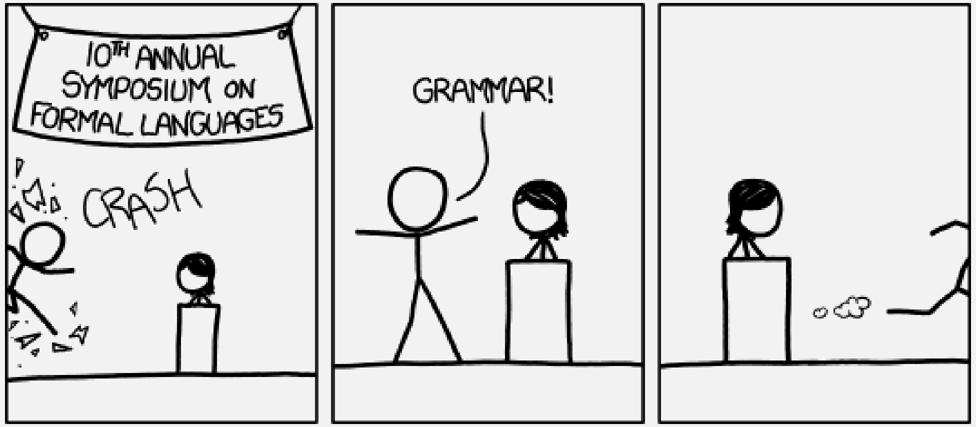
UMB CS 622

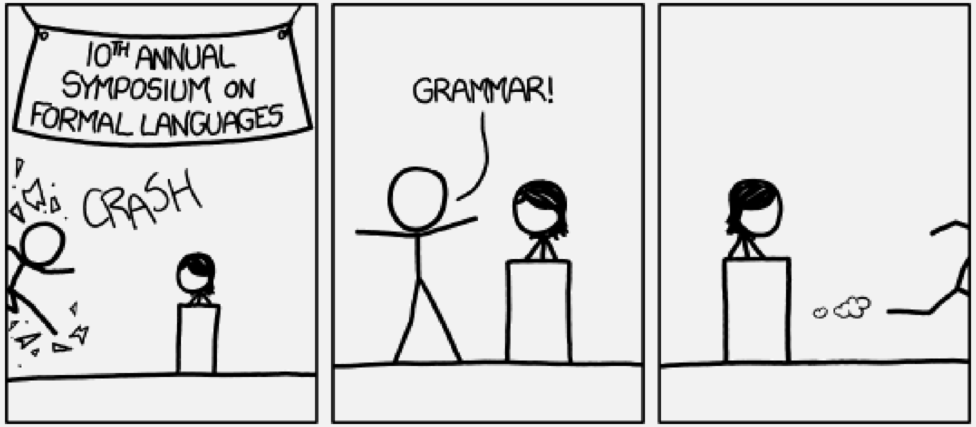
PDA Computation

Friday, March 22, 2024



Announcements

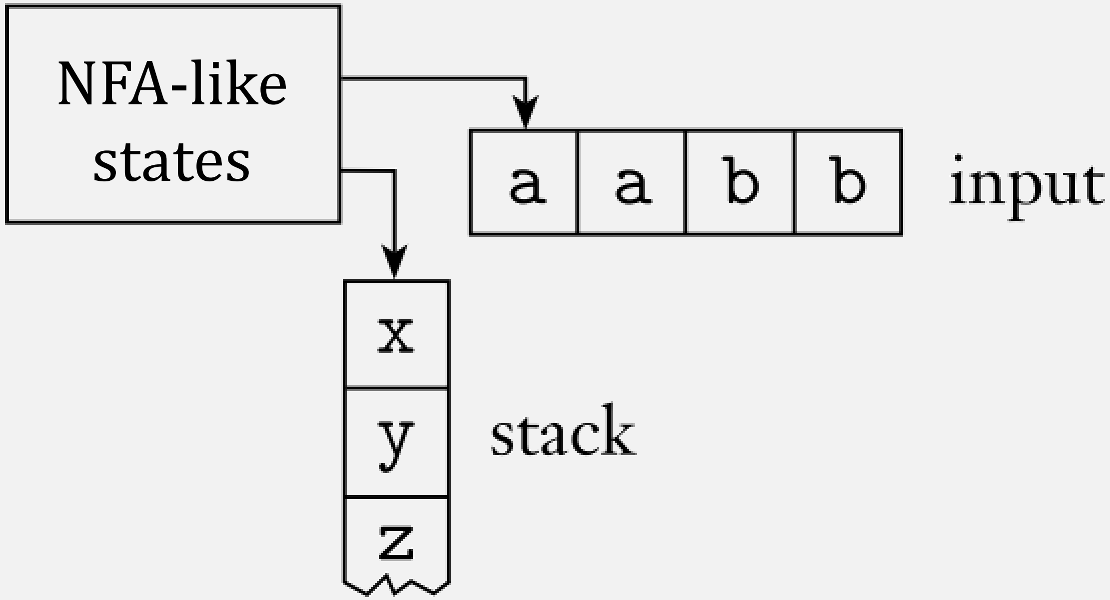
- HW 5 out
  - Due Mon 3/25 12pm noon



Last Time:

Pushdown Automata (PDA)

- **PDA** = NFA + a **stack**
  - Infinite memory
  - push/pop top location only

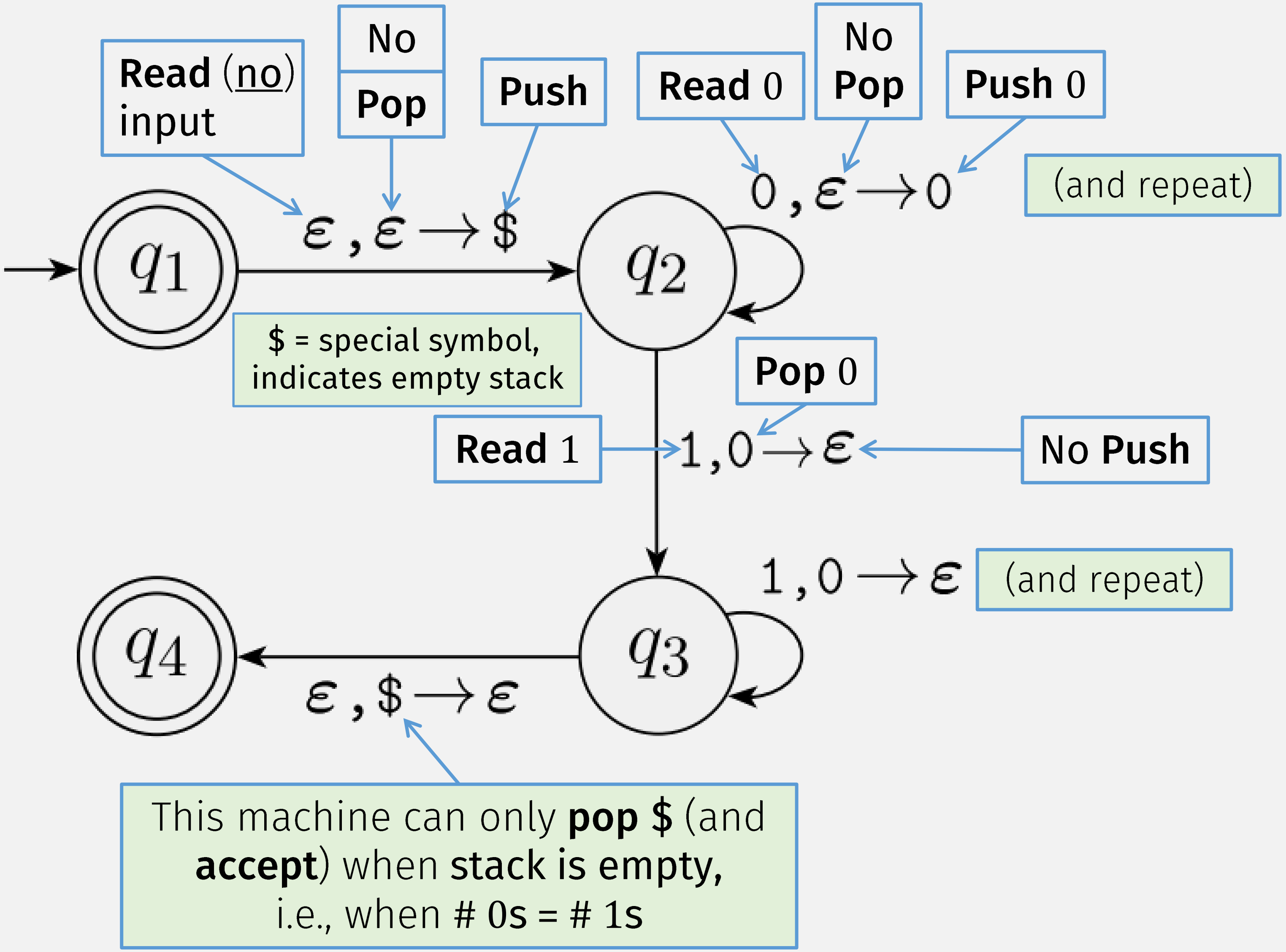


Last Time:

An Example PDA

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

A PDA transition has 3 parts:
- **Read**
- **Pop**
- **Push**



Read (no) input

No **Pop**

Push

Read 0

No **Pop**

Push 0

(and repeat)

$ = special symbol, indicates empty stack

Pop 0

Read 1

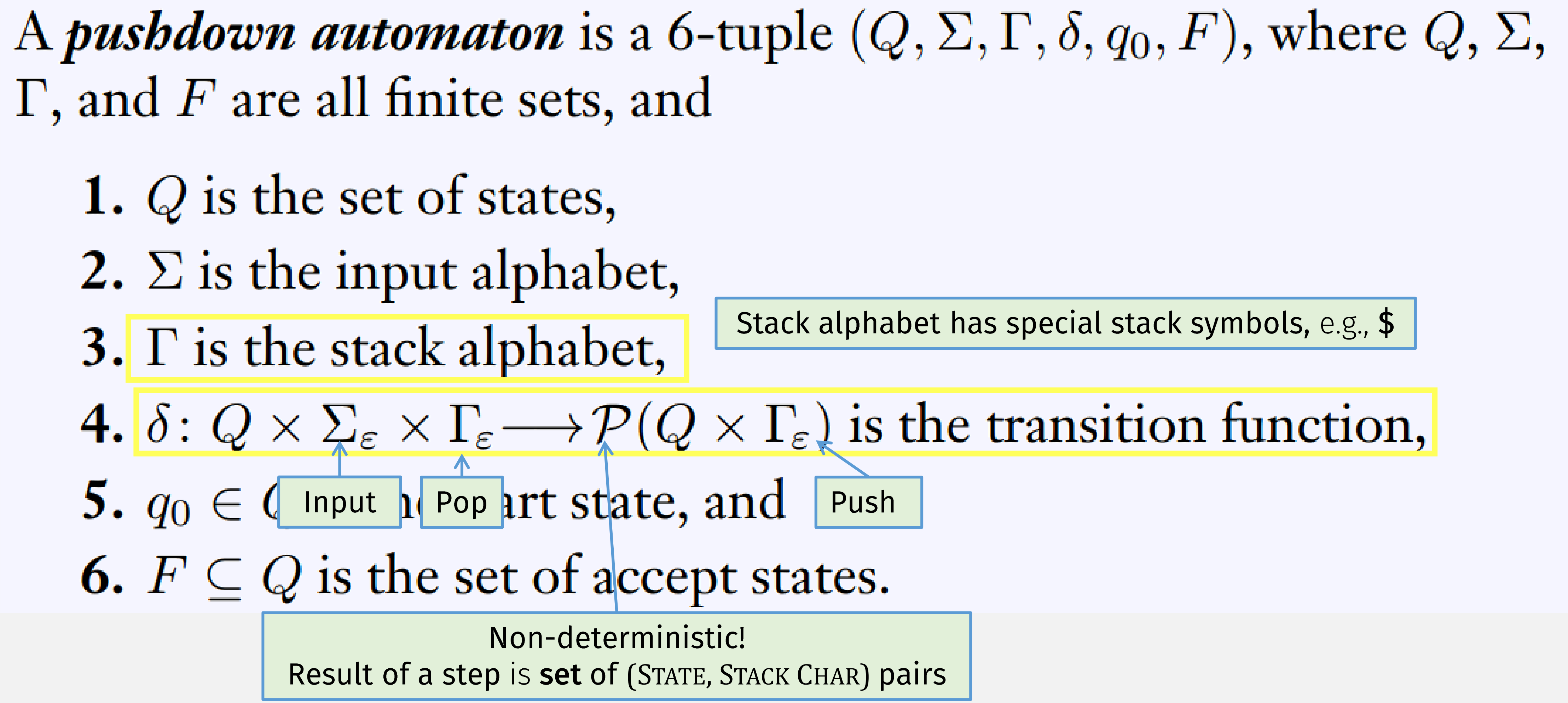
No **Push**

(and repeat)

This machine can only **pop $** (and **accept**) when **stack is empty,** i.e., when **# 0s = # 1s**

Last Time:

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. $q_0 \in$ art state, and
6. $F \subseteq Q$ is the set of accept states.

Stack alphabet has special stack symbols, e.g., $

Input

Pop

Push

Non-deterministic!
Result of a step is **set** of (STATE, STACK CHAR) pairs

PDA Formal Definition Example

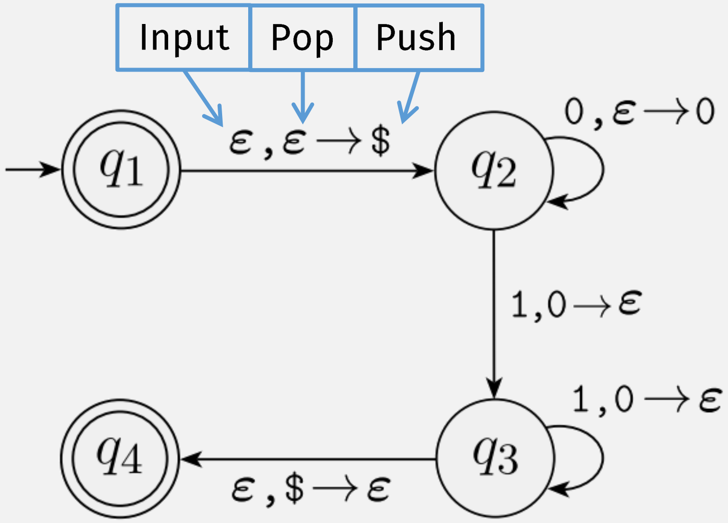
$Q = \{q_1, q_2, q_3, q_4\},$

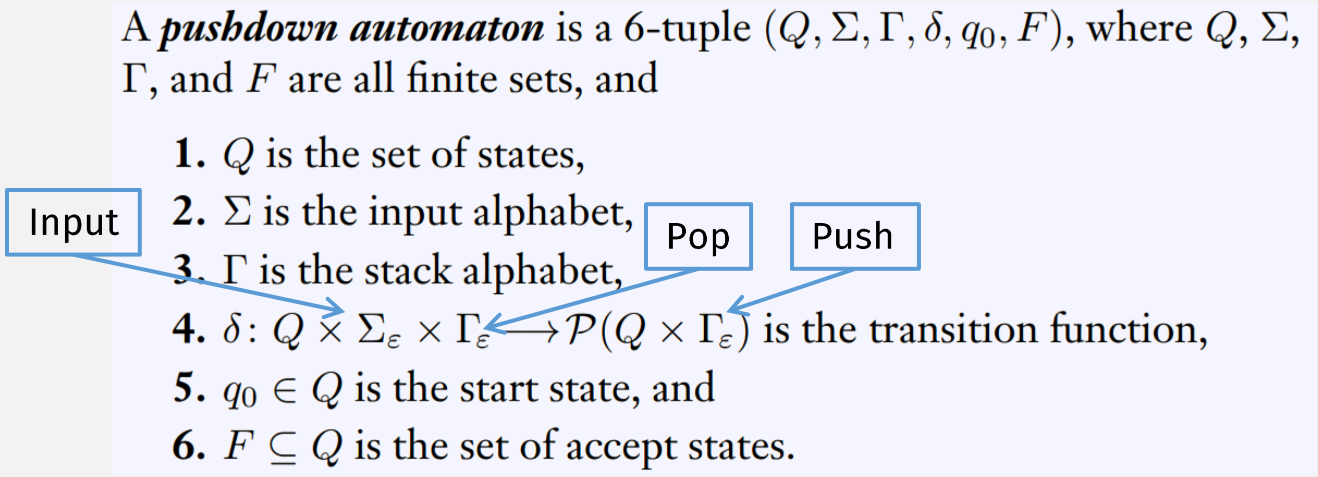
$\Sigma = \{0,1\},$

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

Stack alphabet has special stack symbol $

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




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

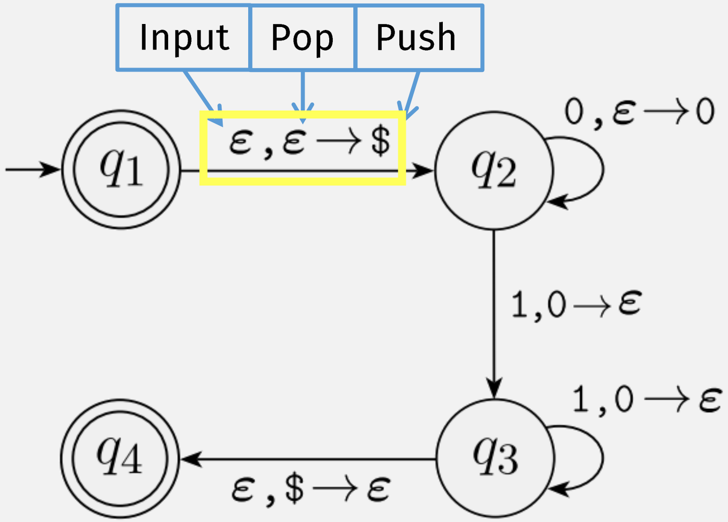
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

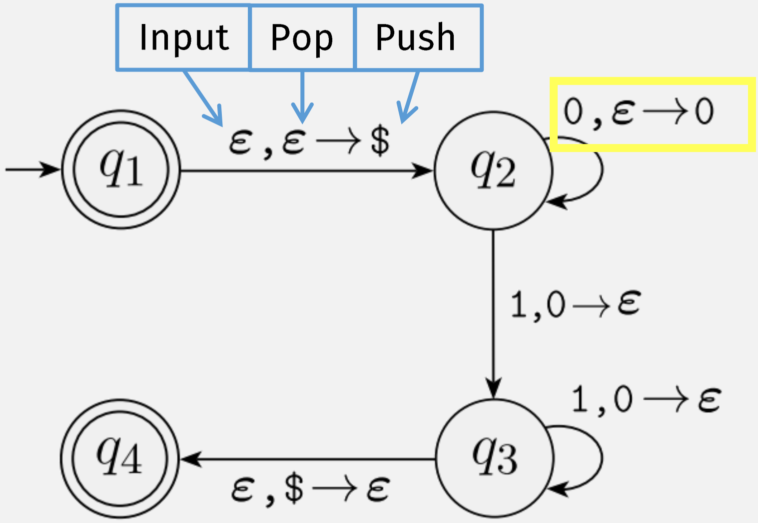
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

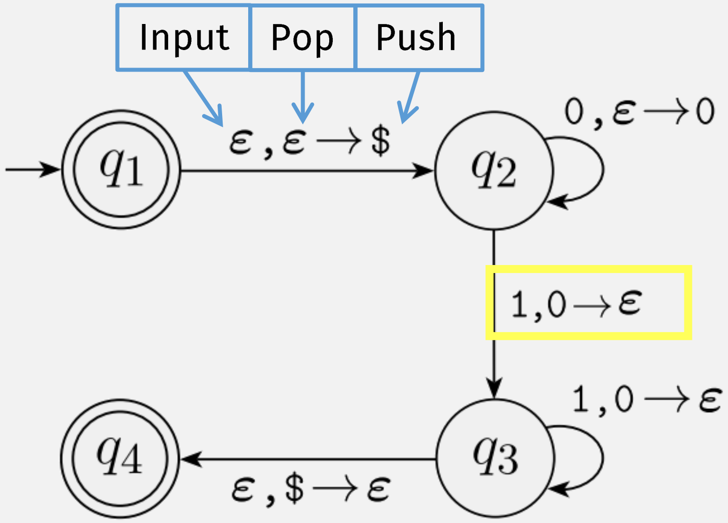
δ is given by the following table, wherein blank entries signify $\emptyset$.

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				$\{(q_4, \varepsilon)\}$	
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

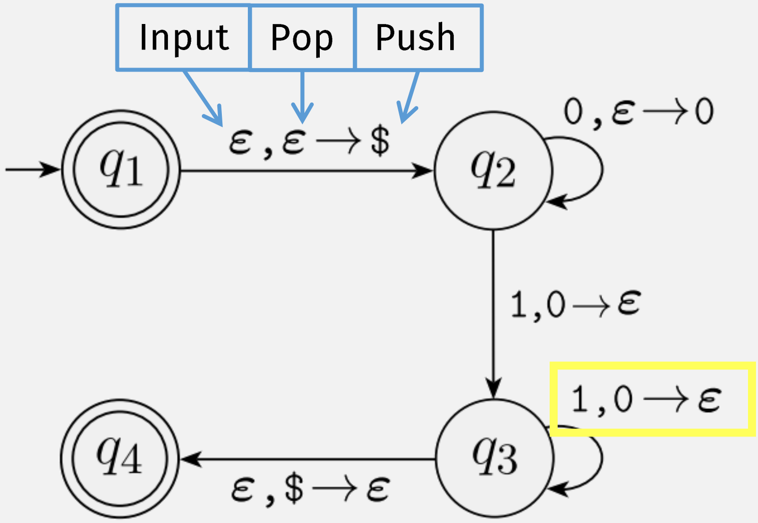
δ is given by the following table, wherein blank entries signify $\emptyset$.

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			$\{(q_4, \varepsilon)\}$	
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

Last Time:

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

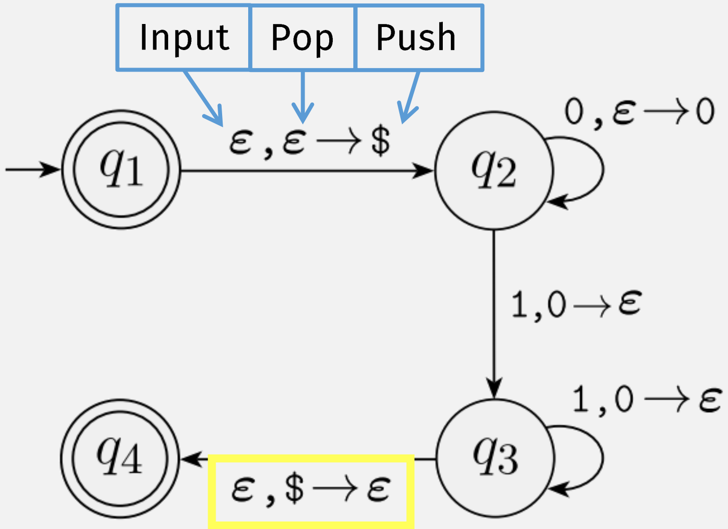
$\Sigma = \{0,1\}$,

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

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

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

Input:	0			1			ε			Input
Stack:	0	\$	ε	0	\$	ε	0	\$	ε	Pop
q_1									$\{(q_2, \$)\}$	Push
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$ 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. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ 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

In-class exercise: Fill in the blanks

$Q =$

$\Sigma =$

$\Gamma =$

$F =$

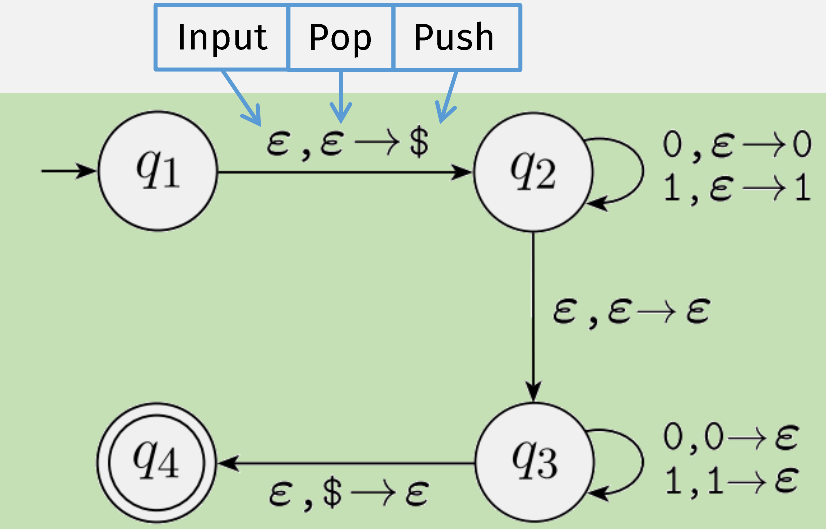
δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0	1	ε
Stack:	???	???	???
?			
?			

Input

Pop

State/ Push



PDA M_3 recognizing the language $\{ww^{\mathcal{R}} | w \in \{0,1\}^*\}$

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

In-class exercise:
Fill in the blanks

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

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

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

$F = \{q_4\}$

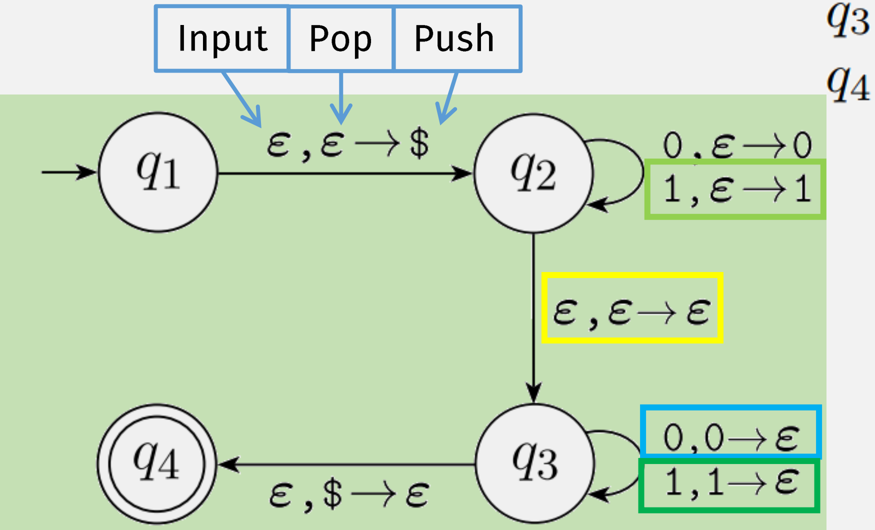
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

Input

Pop

State/ Push



PDA M_3 recognizing the language $\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$

Flashback

DFA Computation Rules

Informally

Given

- A **DFA** (~ a "Program")
- and **Input** = **string of chars**, e.g. "`1101`"

A **DFA** computation (~ "Program run"):

- Start in *start state*

- Repeat:
  - Read **1 char** from **Input**, and
  - Change state according to *transition rules*

Result of **computation**:

  - Accept if **last state** is *Accept state*
  - Reject otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

A DFA **computation** is a sequence of states:

- specified by $\hat{\delta}(q_0, w)$ where:

- *M* **accepts** *w* if $\hat{\delta}(q_0, w) \in F$
- *M* **rejects** otherwise

Flashback

$\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***

DFA Multi-step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \to Q$$

- <u>Domain</u> (inputs):
  - state $q \in Q$
  - string $w = w_1 w_2 \cdots w_n \text{ where } w_i \in \Sigma$
- <u>Range</u> (output):
  - state $q \in Q$

A DFA **computation** is a <u>sequence of states:</u>

(Defined recursively)

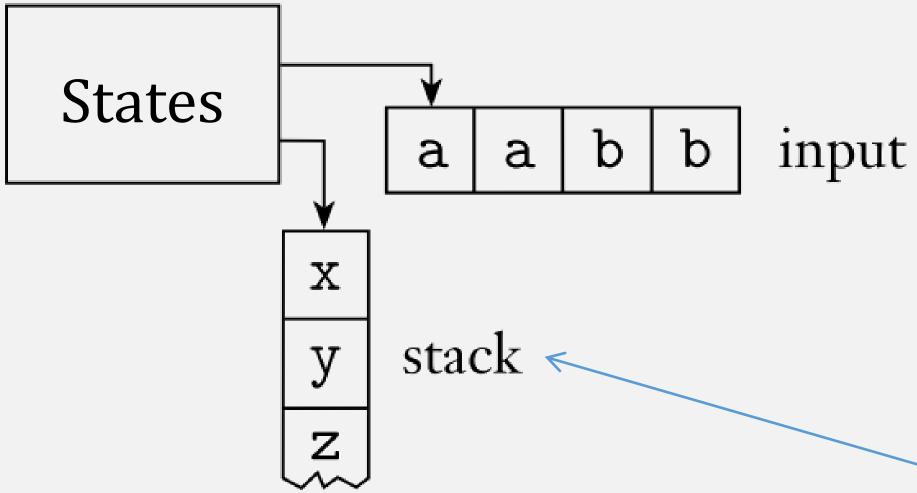
Base case $\hat{\delta}(q, \varepsilon) = q$

Recursive Case $\hat{\delta}(q, w' w_n) = \delta(\hat{\delta}(q, w'), w_n)$

$\text{where } w' = w_1 \cdots w_{n-1}$

PDA Computation?

- **PDA** = NFA + a stack
  - Infinite memory
  - Push/pop top location only



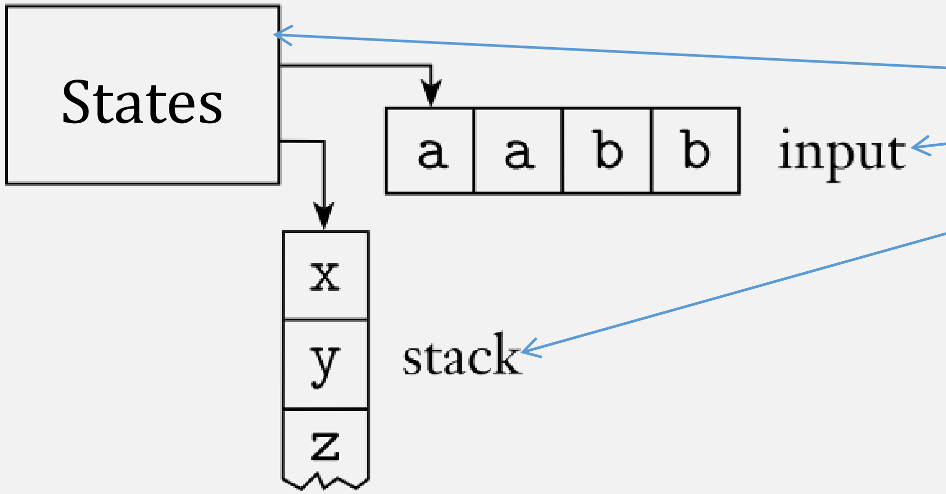
A DFA **computation** is a <u>sequence of states</u> …

A PDA **computation** is <u>not</u> just a <u>sequence of states</u> …

… because the **stack contents can change too!**

PDA Configurations (IDs)

- A **configuration** (or **ID**) is a "<u>snapshot</u>" of a **PDA's computation**



A **configuration** (q, w, γ) has 3 components:

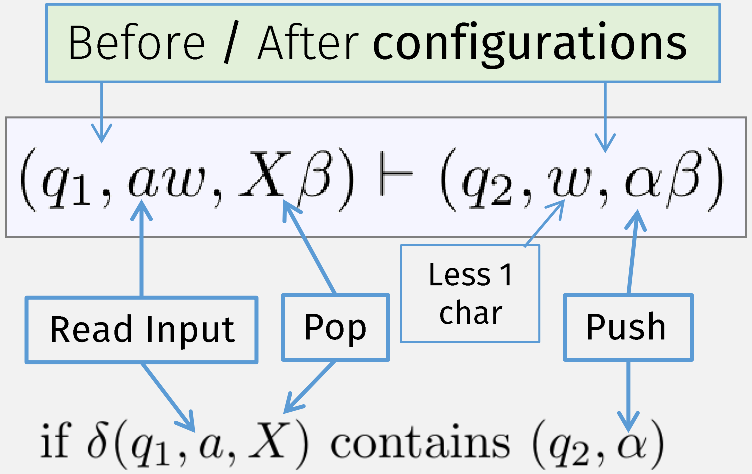
q = the current state
w = the remaining input string
γ = the stack contents

A **sequence of configurations** represents a **PDA computation**

PDA Computation, Formally

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

Single-step



Before / After **configurations**

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

Read Input

Pop

Less 1 char

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

Multi-step

- Base Case

0 steps

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

- Recursive Case

> 0 steps

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

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

Single step

Recursive “call”

This specifies the **sequence of configurations** for a **PDA** computation

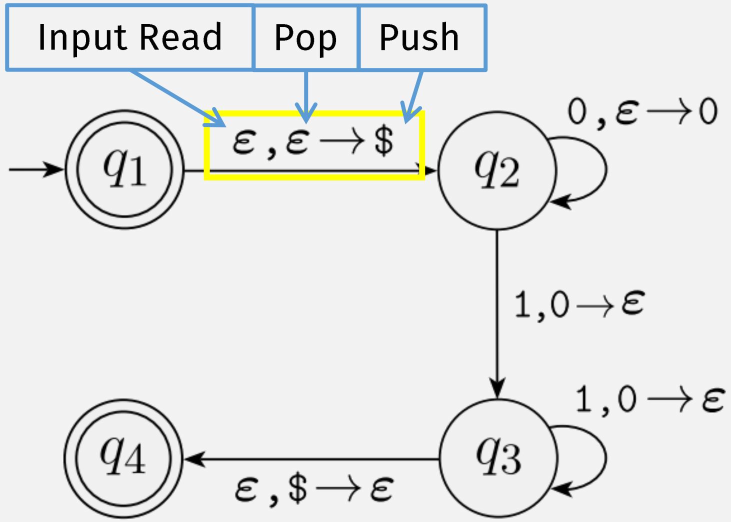
PDA Running Input String Example

State

Remaining Input

Stack

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



PDA Running Input String Example

State

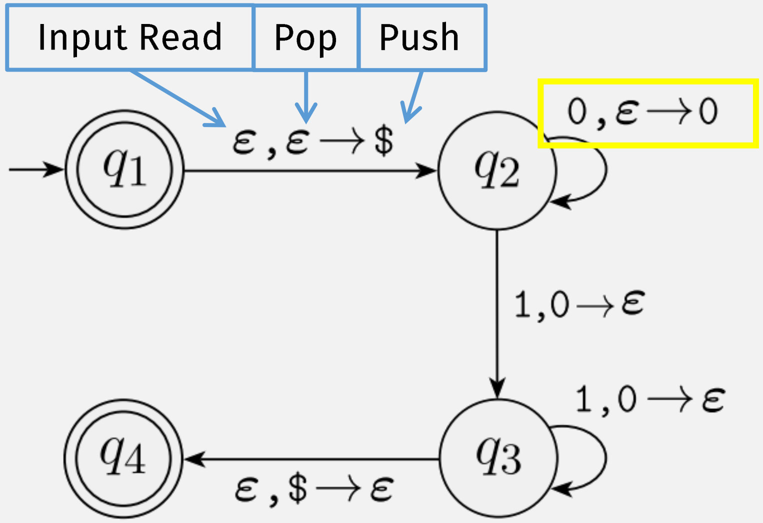
Remaining Input

Stack

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

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

Read 0, push 0



PDA Running Input String Example

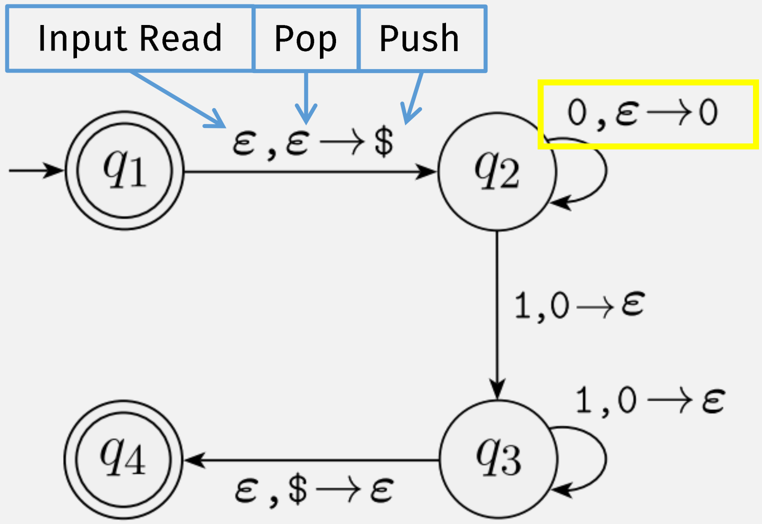
State

Remaining Input

Stack

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

Read 0, push 0



PDA Running Input String Example

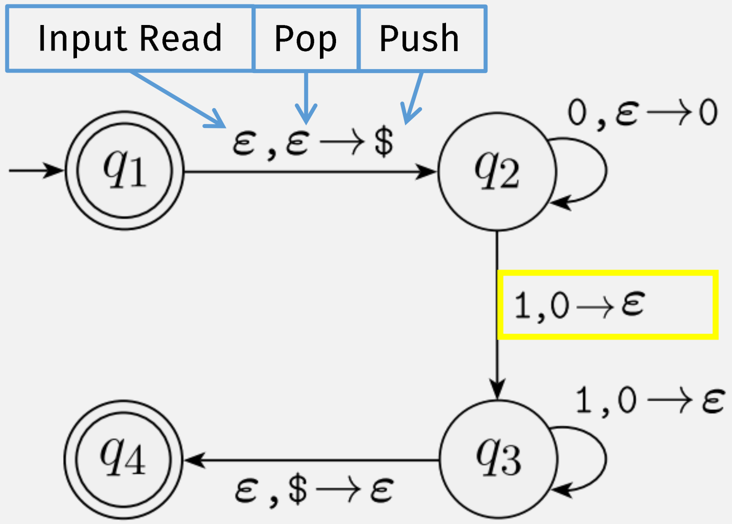
State

Remaining Input

Stack

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

Read 1, pop 0

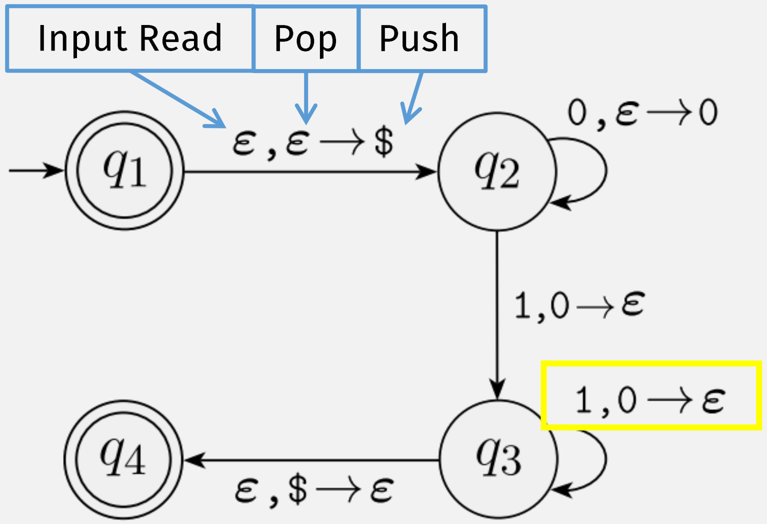


PDA Running Input String Example

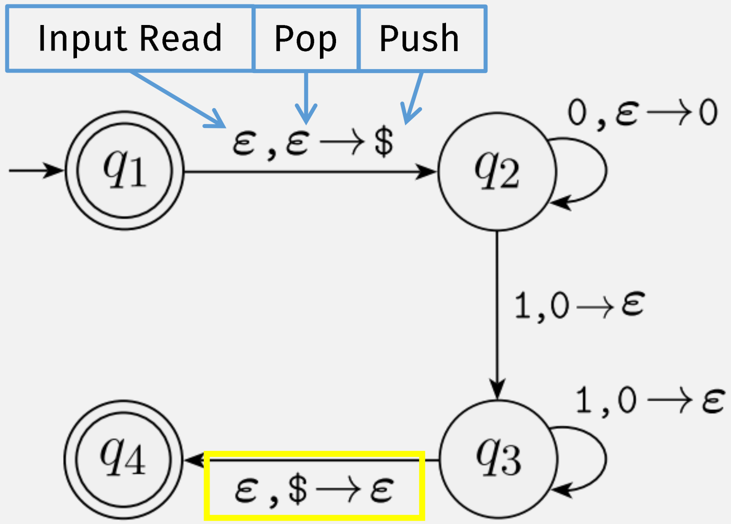
State | Remaining Input | Stack

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

Read 1, pop 0



PDA Running Input String Example



State Remaining Input Stack

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

pop empty stack symbol

Flashback: Computation and Languages

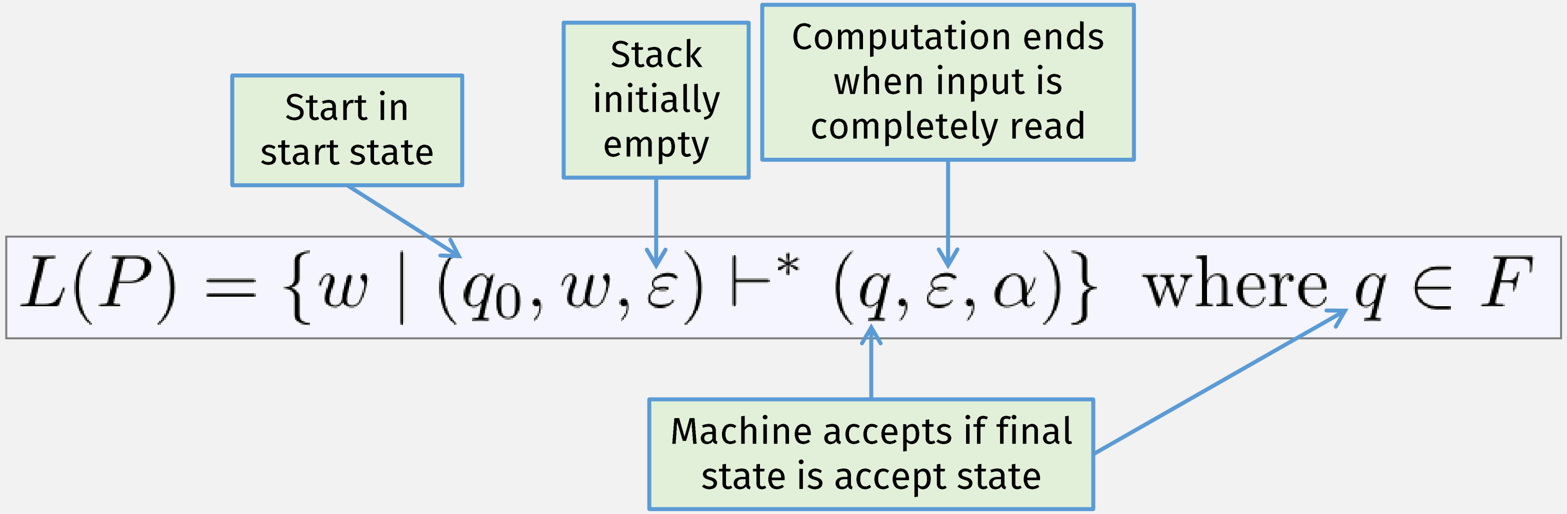
- The **language** of a **machine** is the **set of all strings that it accepts**

- E.g., A **DFA** M **accepts** w **if** $\hat{\delta}(q_0, w) \in F$

- **Language** of M = $L(M)$ = { w | M **accepts** w }

Language of a PDA

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



Start in start state

Stack initially empty

Computation ends when input is completely read

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

Machine accepts if final state is accept state

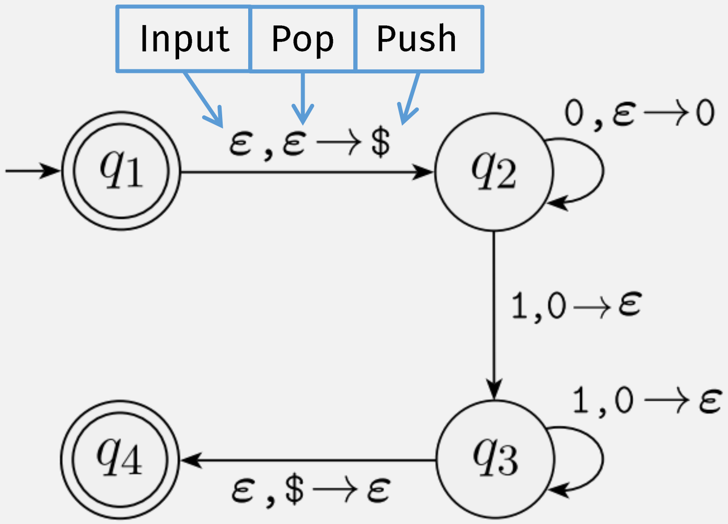
A **configuration** (q, w, γ) has three components

q = the current state

w = the remaining input string

γ = the stack contents

PDAs and CFLs?



- **PDA** = NFA + a stack
  - Infinite memory
  - Push/pop top location only
- Want to prove: PDAs represent CFLs!
- We know: a CFL, by definition, is a language that is generated by a CFG
- Need to show: PDA ⇔ CFG
- Then, to prove that a language is a CFL, we can either:
  - Create a CFG, or
  - Create a PDA

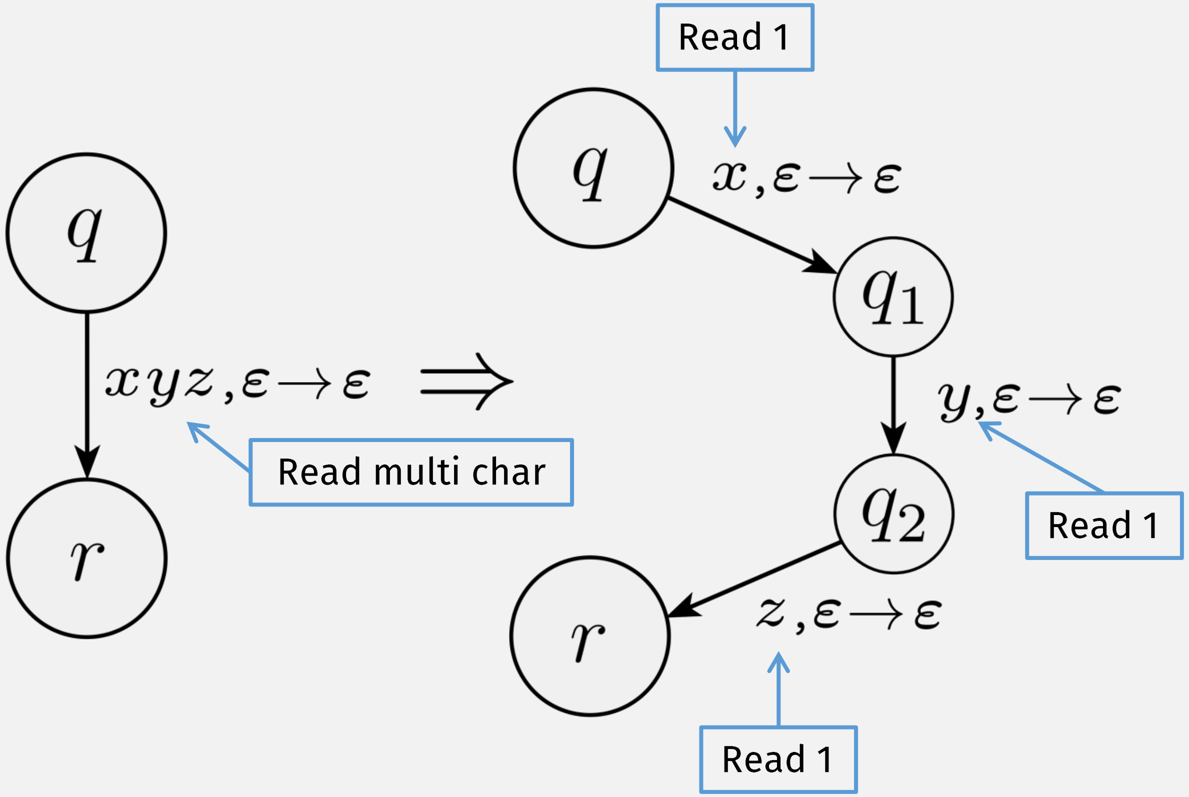
A lang is a CFL **iff** some PDA recognizes it

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

- We know: A CFL **has a CFG describing it** (definition of CFL)
- To prove this part: **show the** CFG **has an equivalent PDA**

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

Shorthand: Multi-Symbol Read Transition



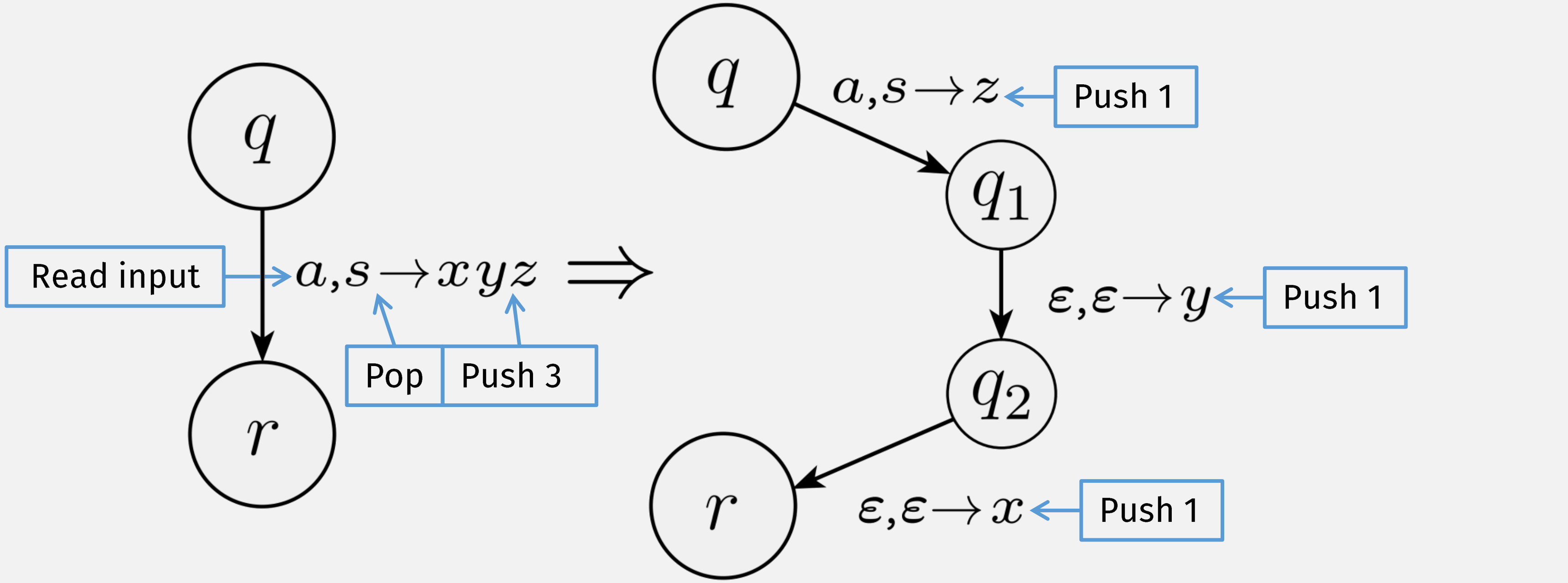
Read multi char

Read 1

Read 1

Read 1

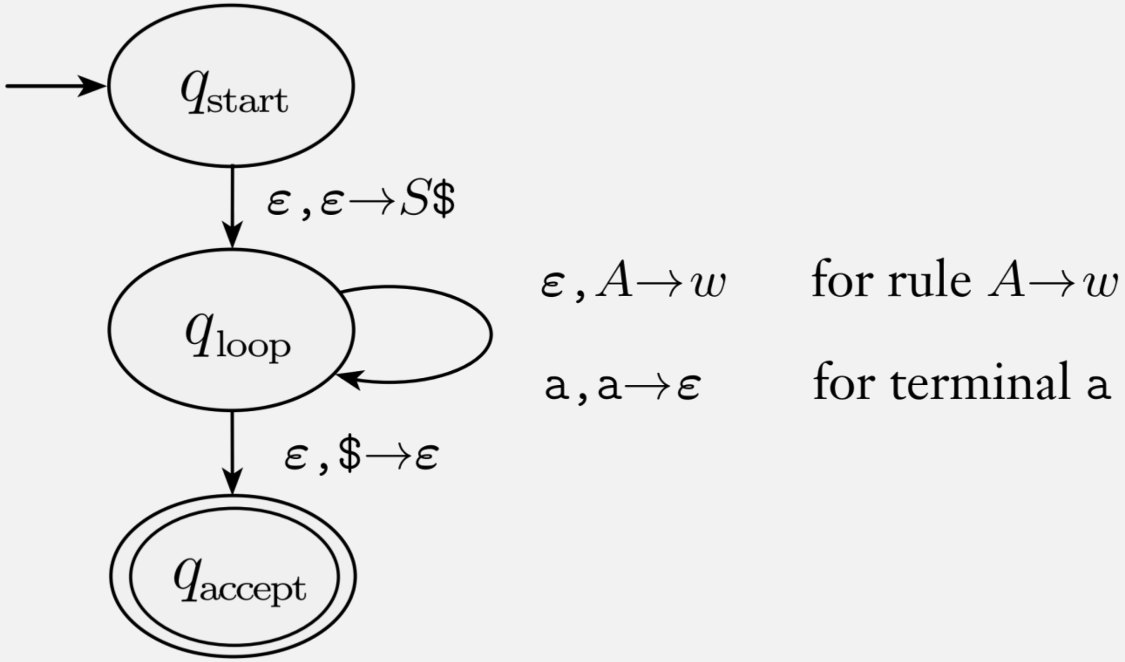
Shorthand: Multi-Stack Push Transition



Note the reverse order of pushes

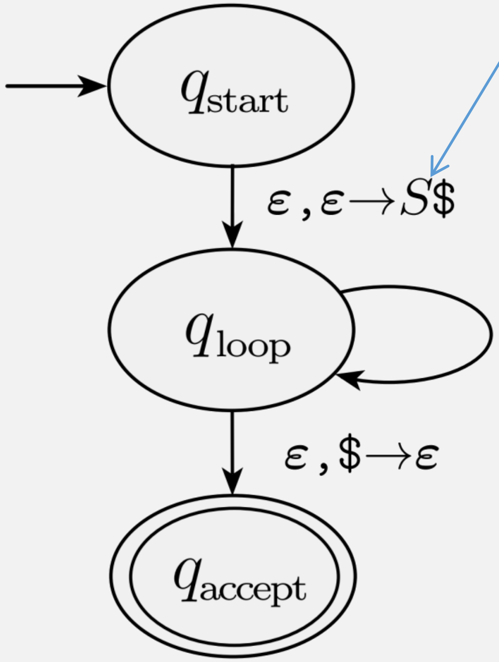
CFG→PDA (sketch)

- Construct PDA from CFG such that:
  - PDA accepts input only if CFG generates it
- PDA:
  - simulates generating a string with CFG rules
  - by (nondeterministically) trying all rules to find the right ones



CFG→PDA (sketch)

- **Construct PDA from CFG** such that:
  - PDA accepts input only if CFG generates it
- PDA:
  - simulates generating a string with CFG rules
  - by (nondeterministically) trying all rules to find the right ones



push **start variable** onto stack

If: stack top **is variable** A, pop and …

… push **rule's right-sides** (nondeterministically)

$\varepsilon, A \to w$ for rule $A \to w$

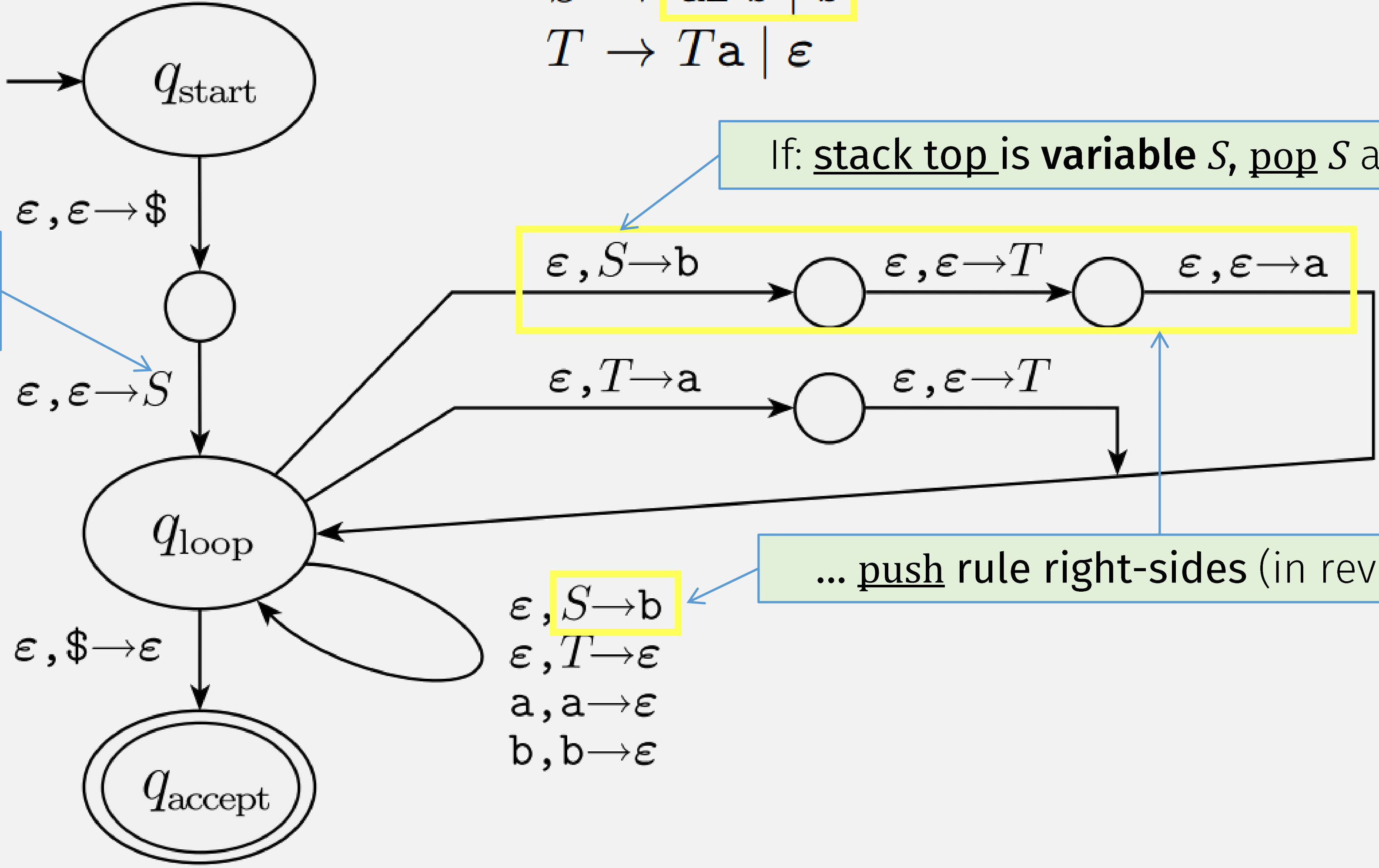
$\mathtt{a}, \mathtt{a} \to \varepsilon$ for terminal $\mathtt{a}$

If: stack top **is terminal** a, pop and …

… read matching input

Example **CFG→PDA**

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



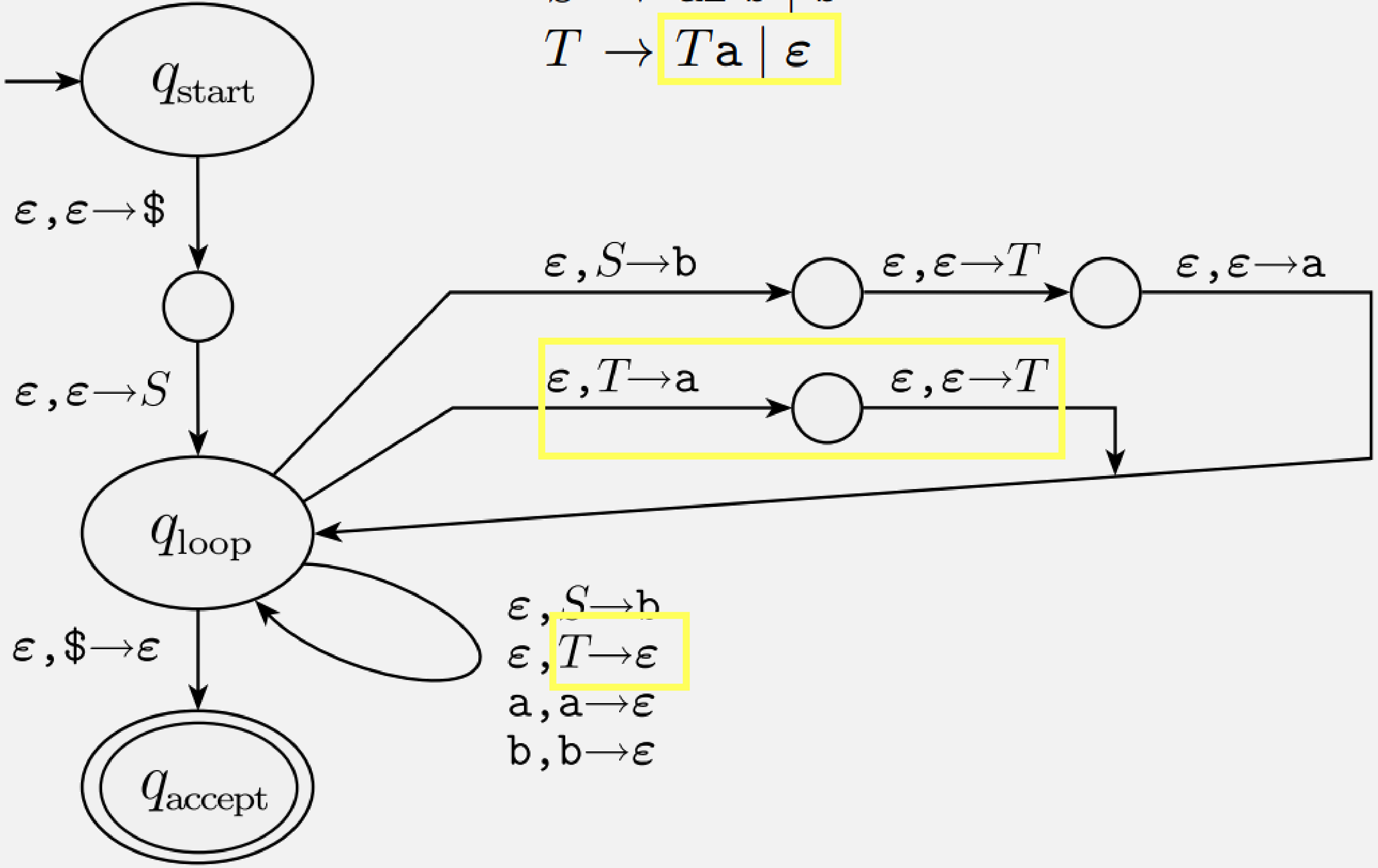
push **start variable** onto stack

If: stack top **is variable** *S*, pop *S* and …

… push **rule right-sides** (in rev order)

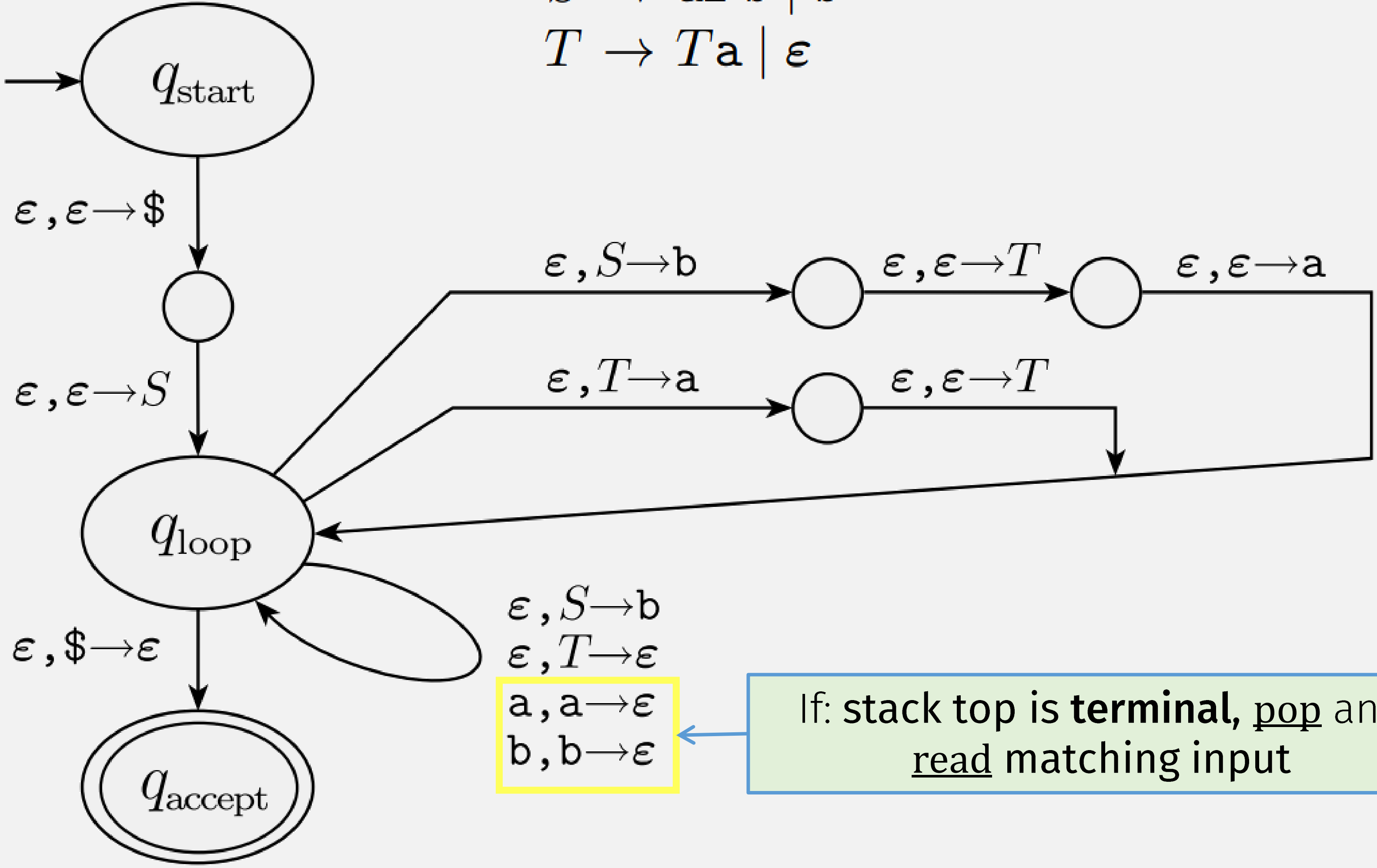
Example **CFG→PDA**

$$
\begin{aligned}
S &\to \mathtt{a}T\mathtt{b} \mid \mathtt{b} \\
T &\to T\mathtt{a} \mid \varepsilon
\end{aligned}
$$



Example **CFG→PDA**

$$\begin{aligned} S &\to \mathtt{a}T\mathtt{b} \mid \mathtt{b} \\ T &\to T\mathtt{a} \mid \varepsilon \end{aligned}$$



If: stack top is **terminal**, <u>pop</u> and <u>read</u> matching input

Example **CFG→PDA**

Example Derivation using CFG:

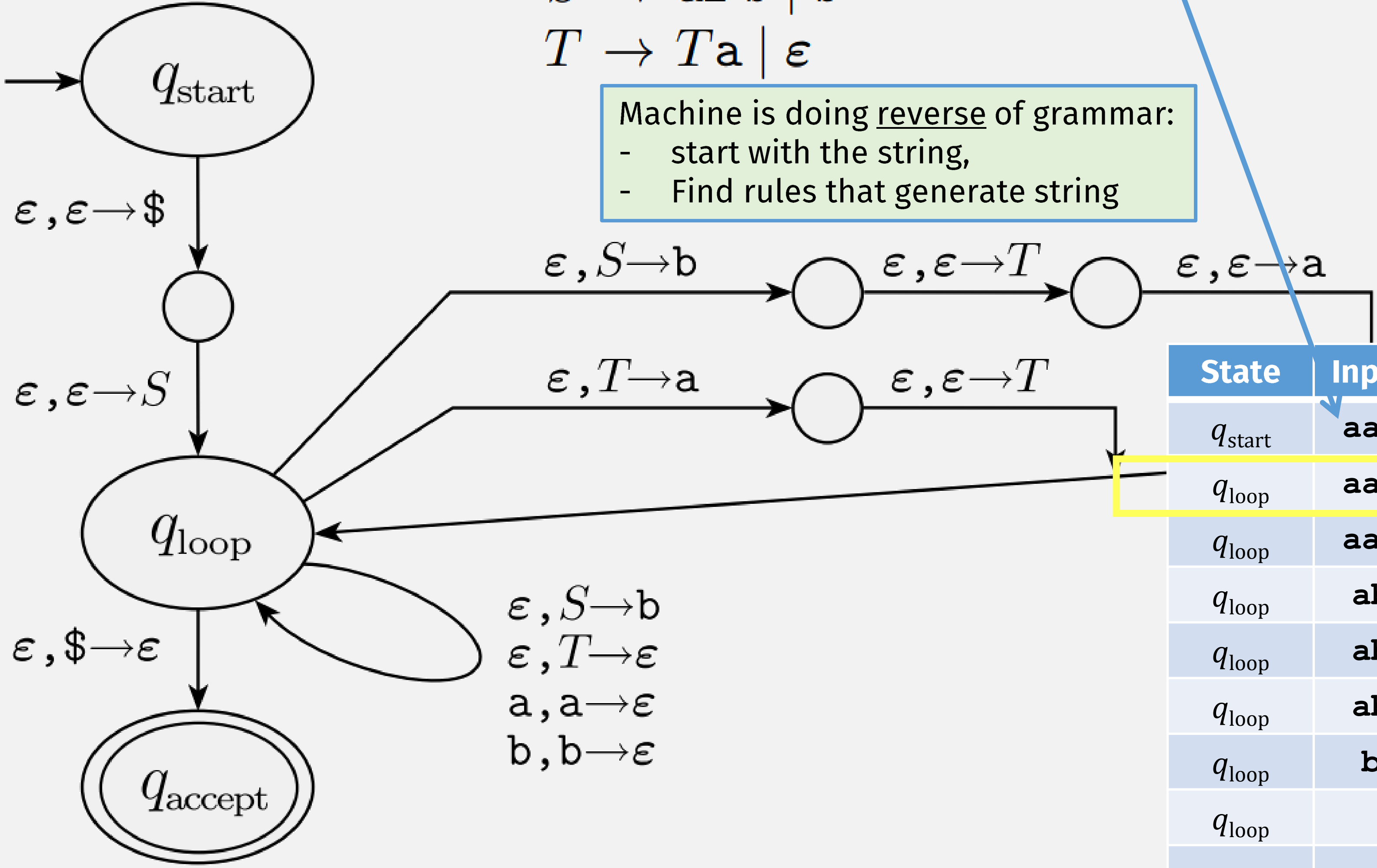
$S \Rightarrow \mathtt{a}T\mathtt{b}$ (using rule $S \to \mathtt{a}T\mathtt{b}$)

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

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

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

Machine is doing reverse of grammar:
- start with the string,
- Find rules that generate string



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**	$\mathtt{ab}\$$	$T \to \varepsilon$
q_{loop}	**b**	$\mathtt{b}\$$	
q_{loop}		$\$$	
q_{accept}			

Example **CFG→PDA**

Example Derivation using CFG:

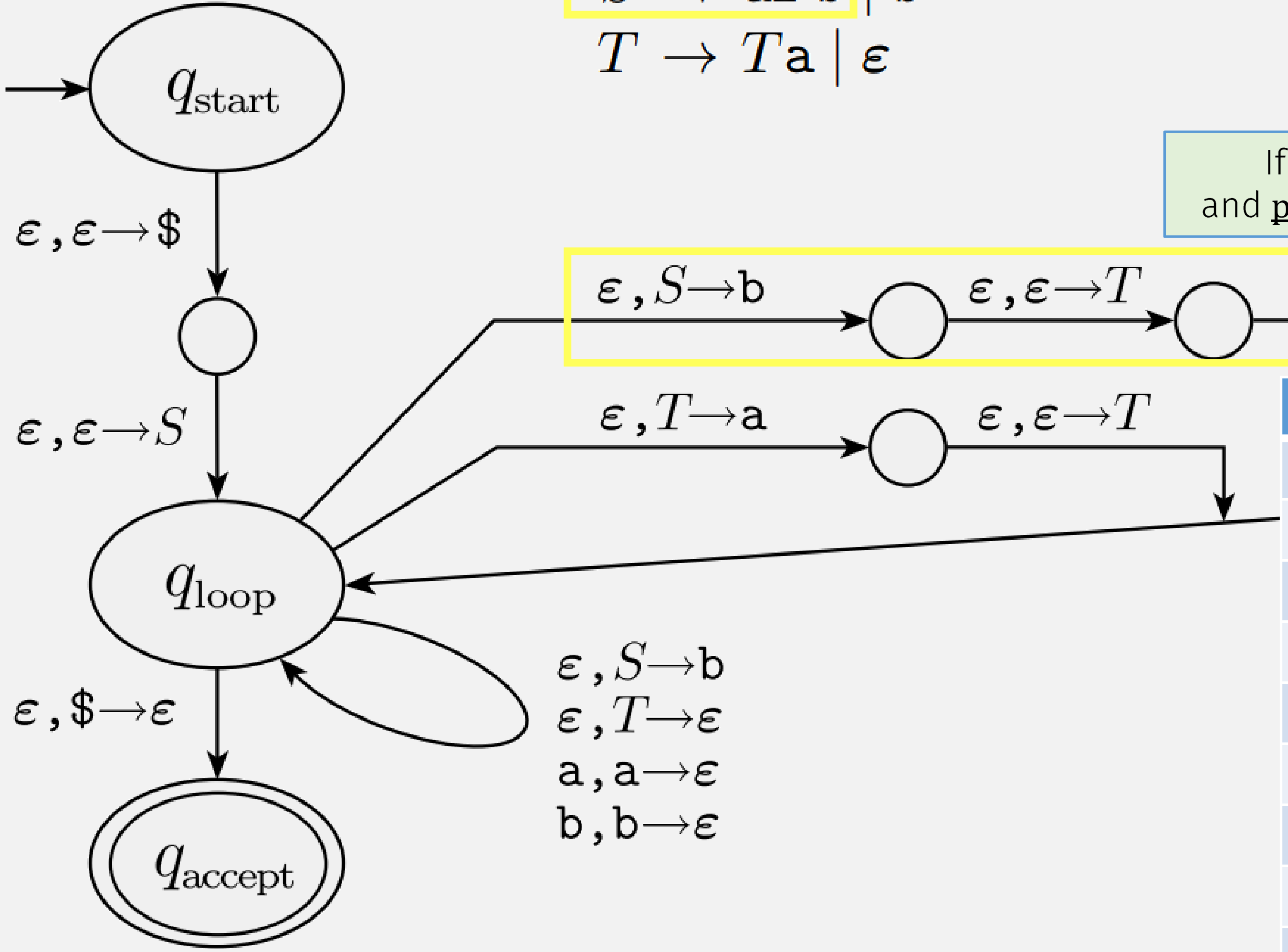
$S \Rightarrow \mathbf{a}T\mathbf{b}$ (using rule $S \rightarrow \mathbf{a}T\mathbf{b}$)

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

$\Rightarrow \mathbf{aab}$ (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**	$\mathbf{a}T\mathbf{b}\$$	$S \rightarrow \mathbf{a}T\mathbf{b}$
q_{loop}	**ab**	$T\mathbf{b}\$$	
q_{loop}	**ab**	$T\mathbf{ab}\$$	$T \rightarrow T\mathbf{a}$
q_{loop}	**ab**	$\mathbf{ab}\$$	$T \rightarrow \varepsilon$
q_{loop}	**b**	$\mathbf{b}\$$	
q_{loop}		$\$$	
q_{accept}			

Example **CFG→PDA**

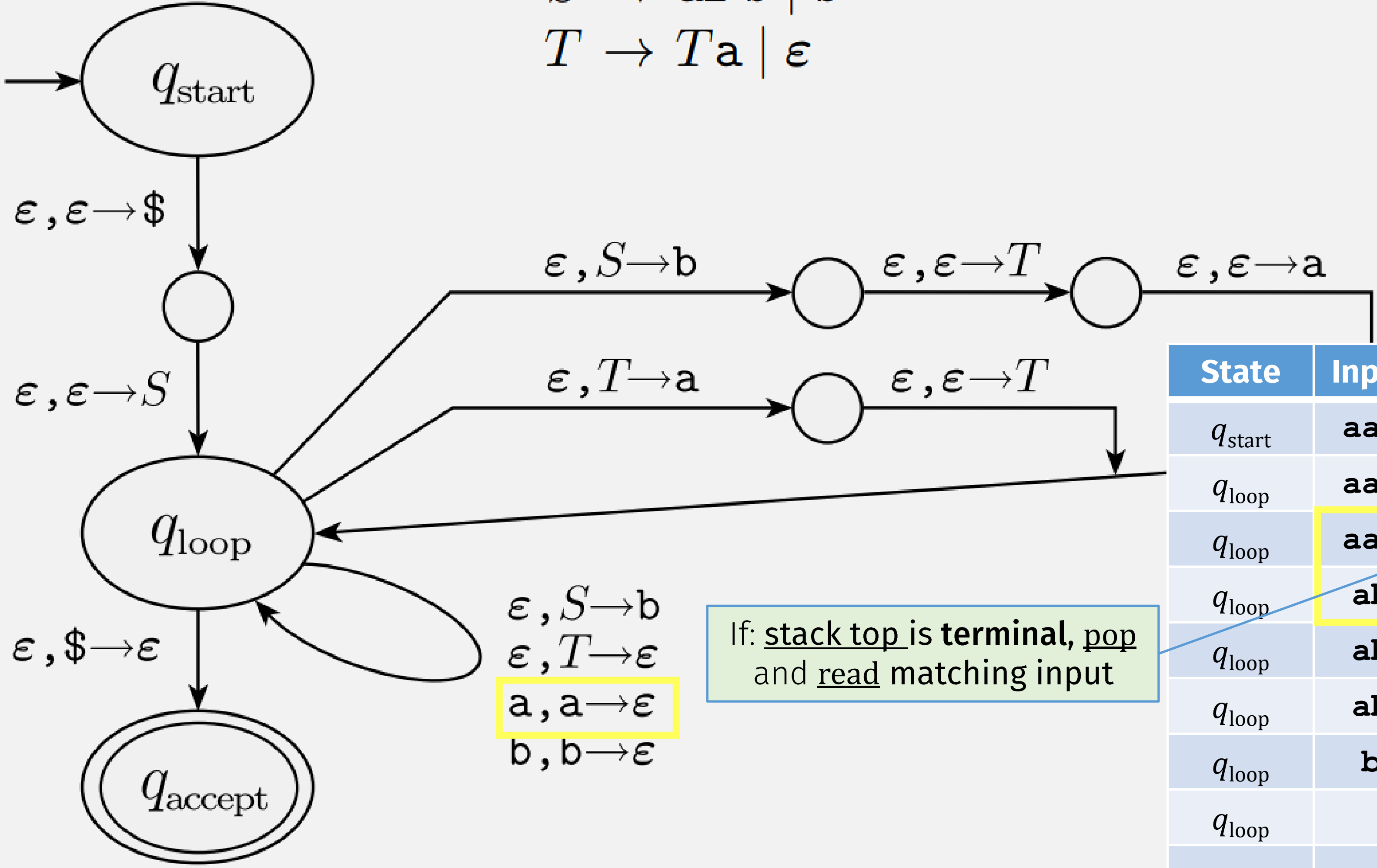
Example Derivation using CFG:

$S \Rightarrow \mathtt{a}T\mathtt{b}$ (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 **terminal**, pop and read matching input

PDA Example

State	Input	Stack	Equiv Rule
q_{start}	aab		
q_{loop}	aab	$S\$$	
q_{loop}	aab	$\mathtt{a}T\mathtt{b}\$$	$S \rightarrow \mathtt{a}T\mathtt{b}$
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:

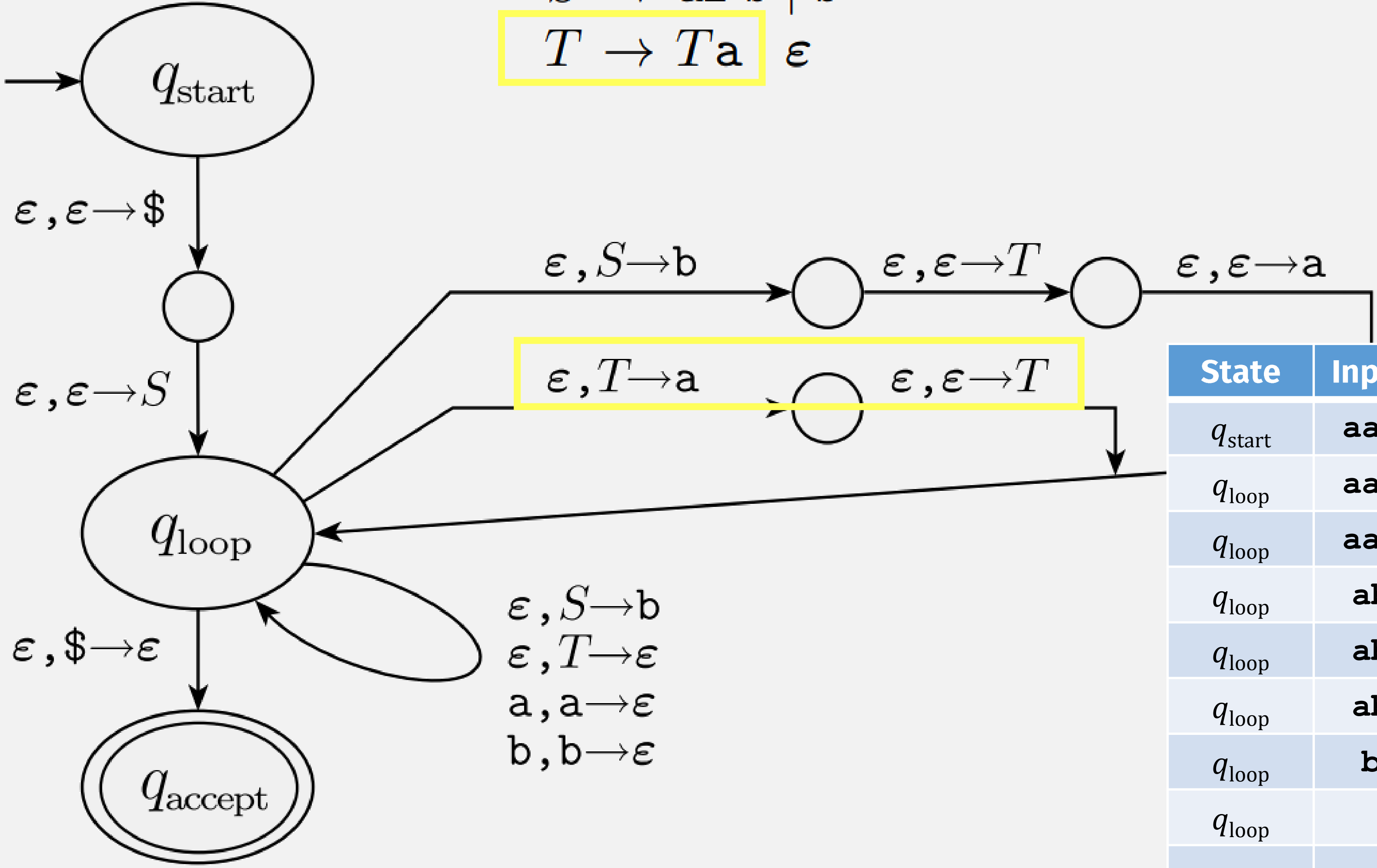
$S \Rightarrow \mathtt{a}T\mathtt{b}$ (using rule $S \to \mathtt{a}T\mathtt{b}$)

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

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

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

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



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	$\mathtt{ab}\$$	$T \to \varepsilon$
q_{loop}	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**

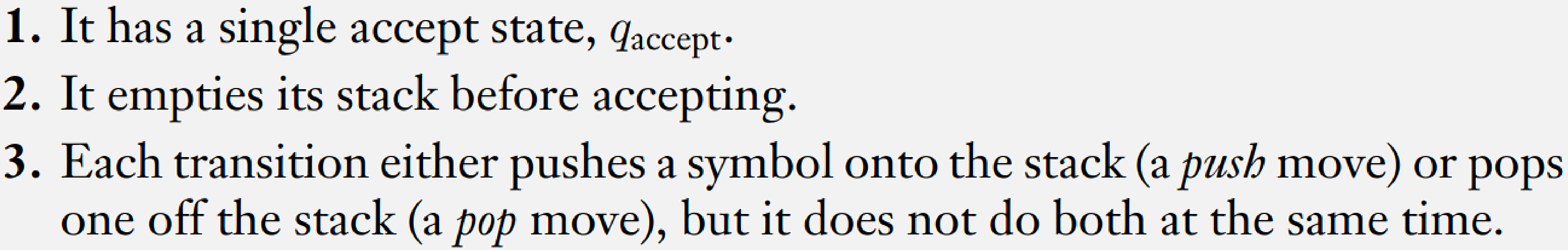
  - **Convert CFG→PDA**

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

  - To prove this part: show PDA has an equivalent 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

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)

The Key IDEA

- 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 $\{A_{pq} \mid p, q \in Q\}$

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

PDA P -> CFG G: Generating Strings

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

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

- 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**
  - Convert **CFG→PDA**

☑⇐ If **a PDA recognizes a language,** then **it's a CFL**
  - Convert **PDA→CFG**

■

Submit in-class work 3/20

On gradescope