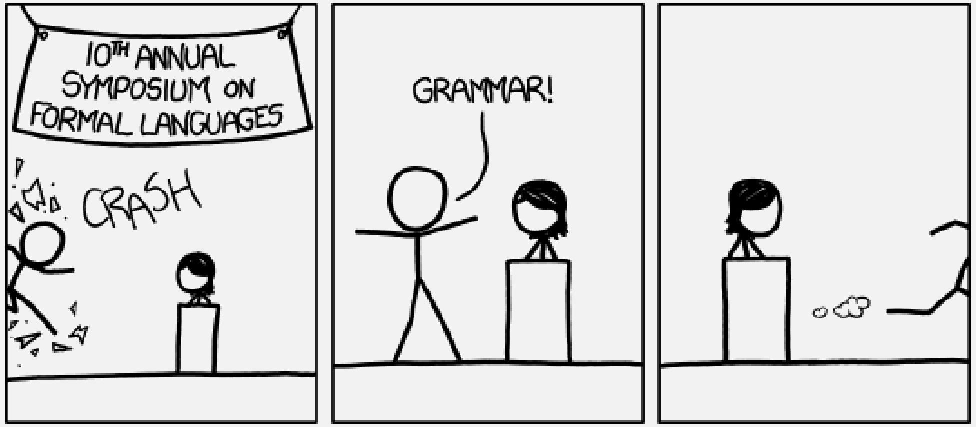
UMB CS 420

Pushdown Automata (PDAs)

Tuesday, October 18, 2022



Announcements

- HW 4 in
  - ~~Due Sun 10/16 11:59pm EST~~
- HW 5 out
  - Due Sun 10/23 11:59pm EST
- Reminder: Sean's Office Hours Monday in-person
  - McCormack 3rd floor 0139

HW2 Review

3. Come up with a procedure **DFA→NFA** that converts DFAs to equivalent NFAs.

   This means that given some DFA $M = (Q, \Sigma, \delta, q_{start}, F)$ that satisfies the formal definition of DFAs from class, **DFA→NFA**(M) should produce some equivalent NFA $N = (Q', \Sigma, \delta', q'_{start}, F')$ that satisfies the formal definition of NFAs.

- $Q' = Q$
- $q'_{start} = q_{start}$
- $F' = F$
- $\delta'(q, a) = \{ \delta(q, a) \}$
  - $\forall\, q \in Q, a \in \Sigma$
- $\delta'(q, \varepsilon) = \{\ \}$
  - $\forall\, q \in Q$

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

HW2 Review

3 Proving a Closed Operation For Regular Languages

Define the following operation, called OP, on languages:

$$\mathrm{OP}(B, C) = \{x \mid x \notin B \text{ or } x \notin C\}$$

Prove that OP is closed for regular languages.

Make sure your answer is in the form of a "Statements and Justifications" table, as explained in lecture (and also Hopcroft Chapter 1).

Proof:

Statement	Justification
1. Assume B and C are reg langs	1. Given, from def of closed operation
2. DFA M recognizes $OP(B, C)$	2. See M construction
3. $OP(B, C)$ is a regular language	3. (2) and Def of regular language
4. OP is closed for reg languages: i.e., if some B, C are reg langs, then $OP(B, C)$ is a reg lang	4. From (1) and (3)

(saying “modus ponens” is not a valid justification)

HW2 Review

3 Proving a Closed Operation For Regular Languages

Define the following operation, called OP, on languages:

$$\mathrm{OP}(B, C) = \{x \mid x \notin B \text{ or } x \notin C\}$$

Prove that OP is closed for regular languages.

Make sure your answer is in the form of a "Statements and Justifications" table, as explained in lecture (and also Hopcroft Chapter 1).

<u>Proof</u>:

Statement	**Justification**
1. Assume B and C are reg langs	1. Given, from def of closed operation
a) Let M_B = B lang DFA	a) Def of reg lang
b) Let M_B' recognize $\{ x \mid x \notin B \}$	b) construct M_B' : flip accept/non-accept states in M_B
c) Repeat for M_C = C lang DFA	
2. DFA M recognizes $OP(B, C)$	2. See M construction

Last Time: Generating Strings with a CFG

A CFG represents a context free language!

$G_1 =$

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

Strings in CFG's language
= all possible generated strings

$L(G_1)$ is $\{0^n\#1^n \mid n \geq 0\}$

Stop when string is all terminals

A CFG **generates** a string, by repeatedly applying substitution rules:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

Start variable

Last Time:

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:

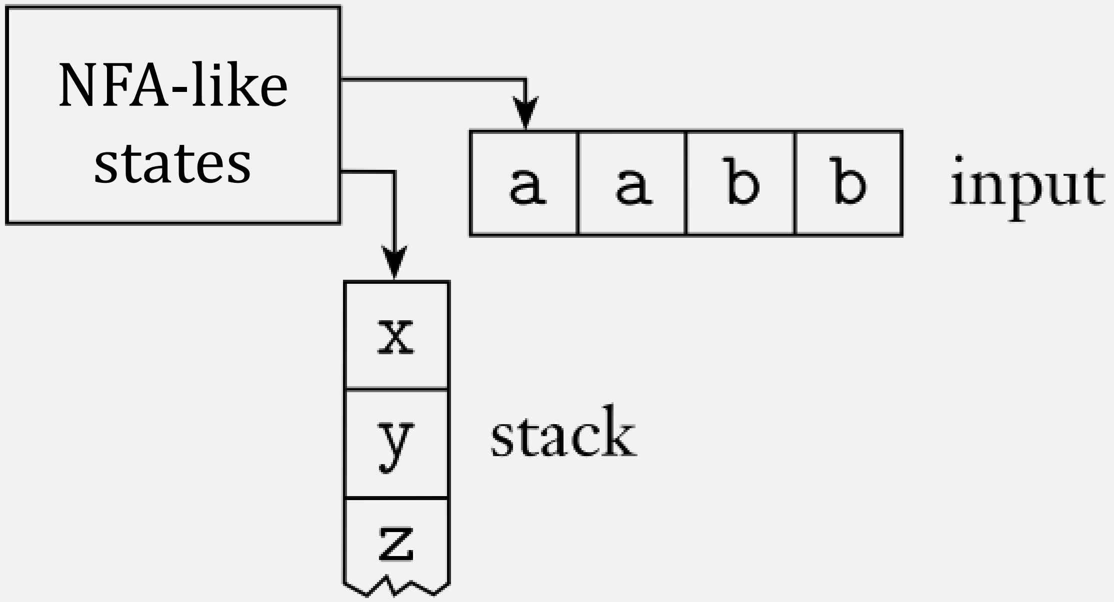
Regular Languages	**Context-Free** Languages (CFLs)
Regular Expression	Context-Free Grammar (CFG)
A Reg Expr describes a Regular lang	A CFG describes a CFL
TODAY:	
Finite Automaton (FSM)	Push-down automaton (PDA)
An FSM recognizes a Regular lang	A PDA recognizes a CFL

Today:

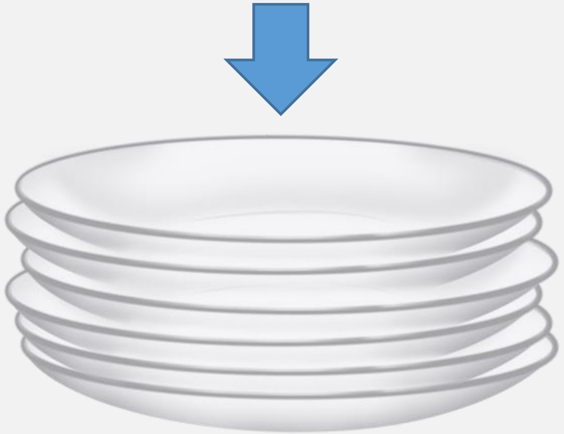
Regular Languages	**Context-Free** Languages (CFLs)
Regular Expression	Context-Free Grammar (CFG)
A Reg Expr describes a Regular lang	A CFG describes a CFL
TODAY:	
Finite Automaton (FSM)	Push-down automaton (PDA)
An FSM recognizes a Regular lang	A PDA recognizes a CFL
KEY DIFFERENCE:	
A Regular lang 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)

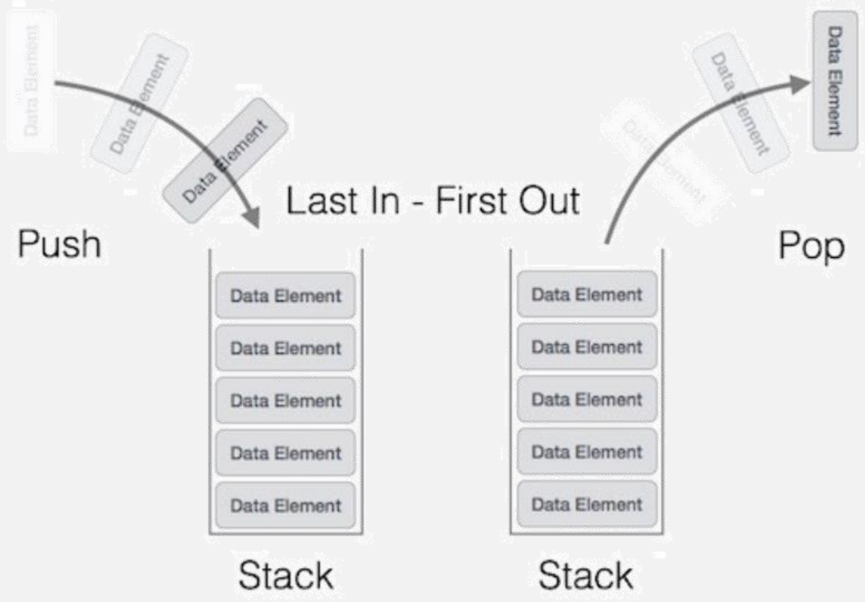
PDA = NFA + a stack



What is a Stack?

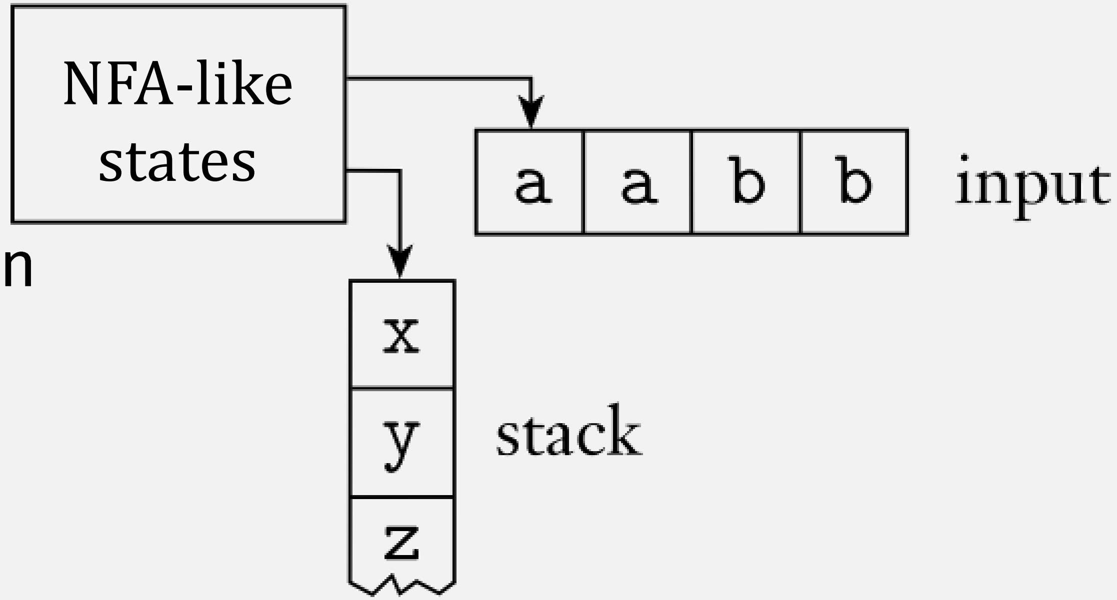


- A <u>restricted</u> kind of (infinite) memory
- Access to top element only
- 2 Operations only: **push, pop**



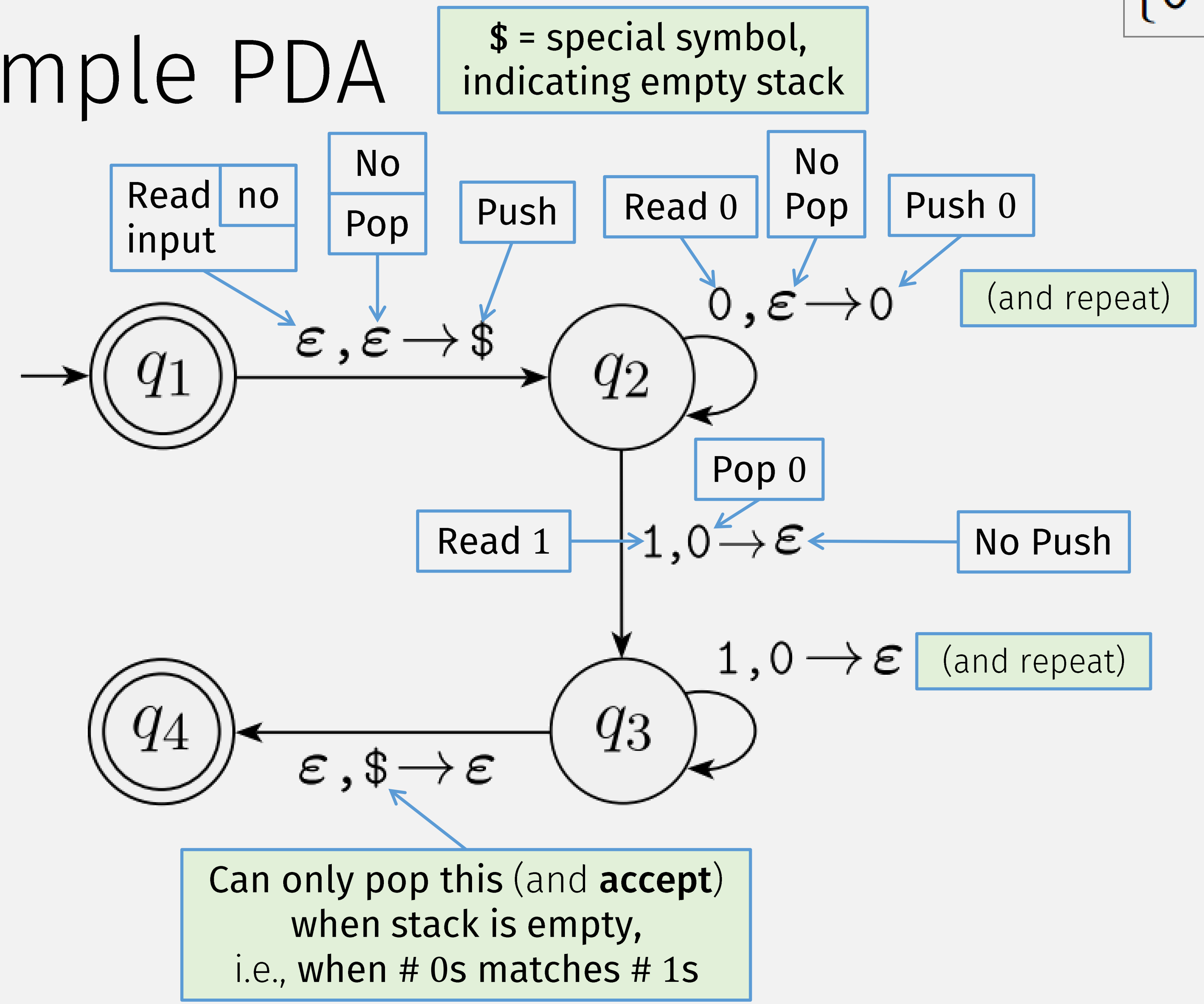
Pushdown Automata (PDA)

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



An Example PDA

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



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

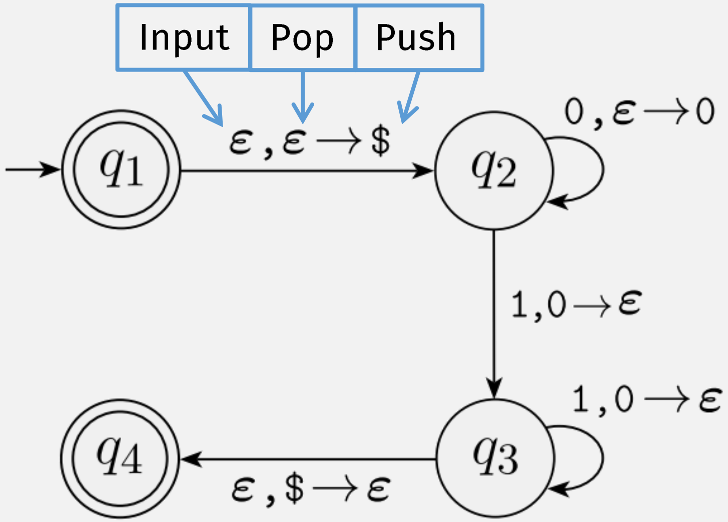
PDA Formal Definition Example

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

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

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

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



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

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

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

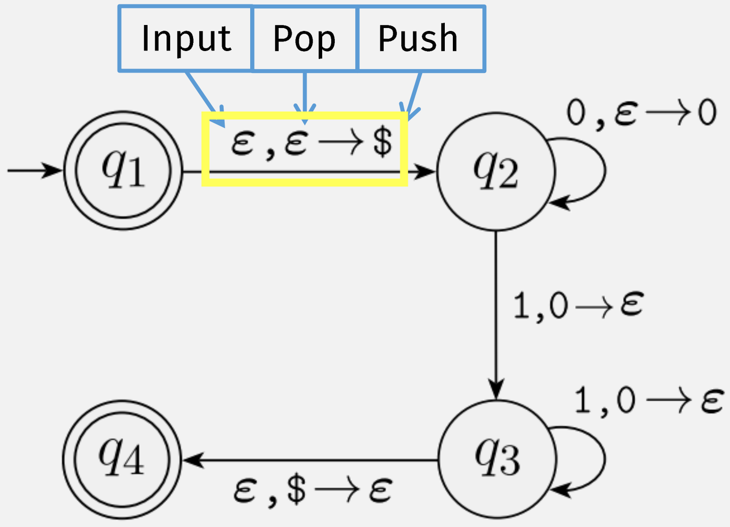
$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, 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									$\{(q_2, \$)\}$ **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

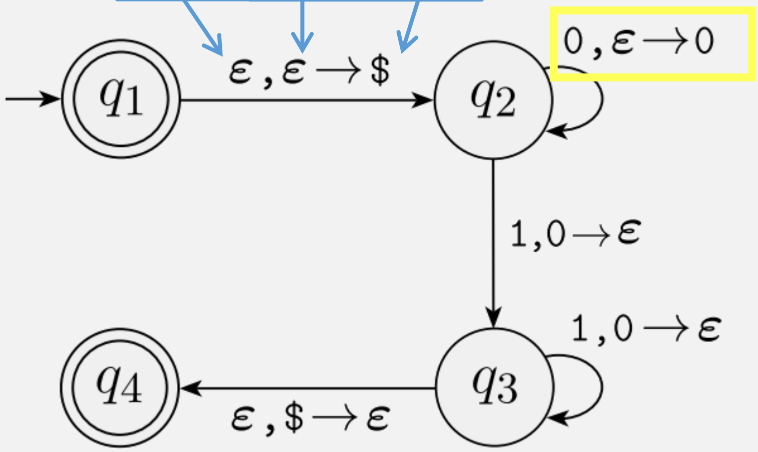
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

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

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

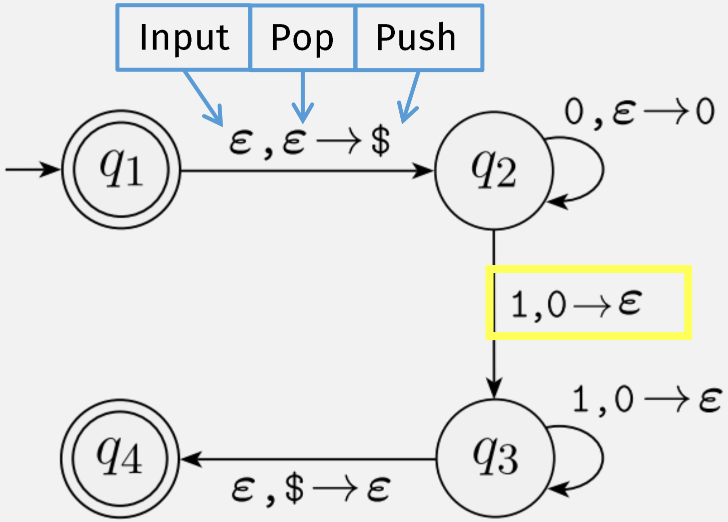
$\Gamma = \{0, \$\}$,

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

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

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

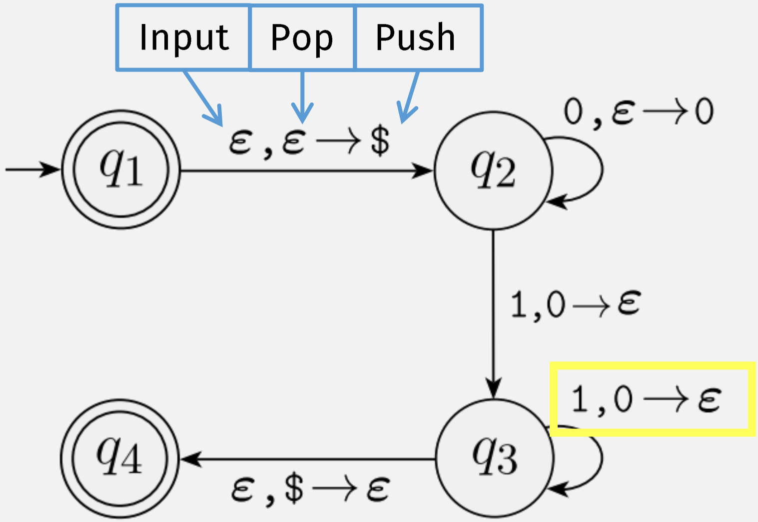
$\Gamma = \{0, \$\}$,

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

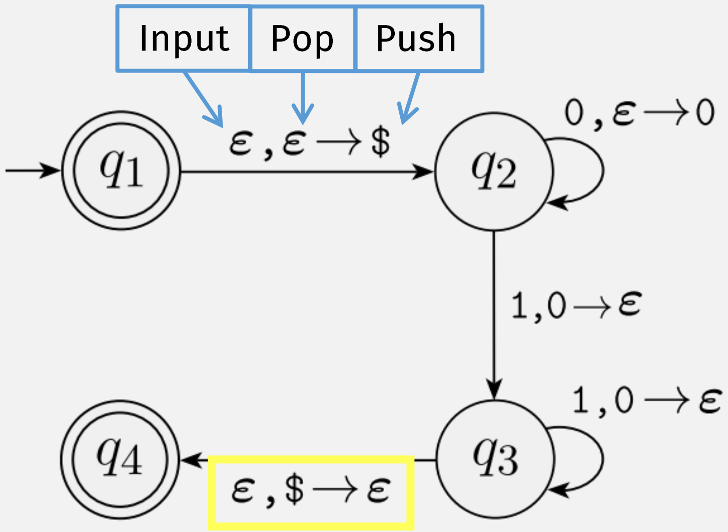
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

In-class exercise: Fill in the blanks

$Q =$

$\Sigma =$

$\Gamma =$

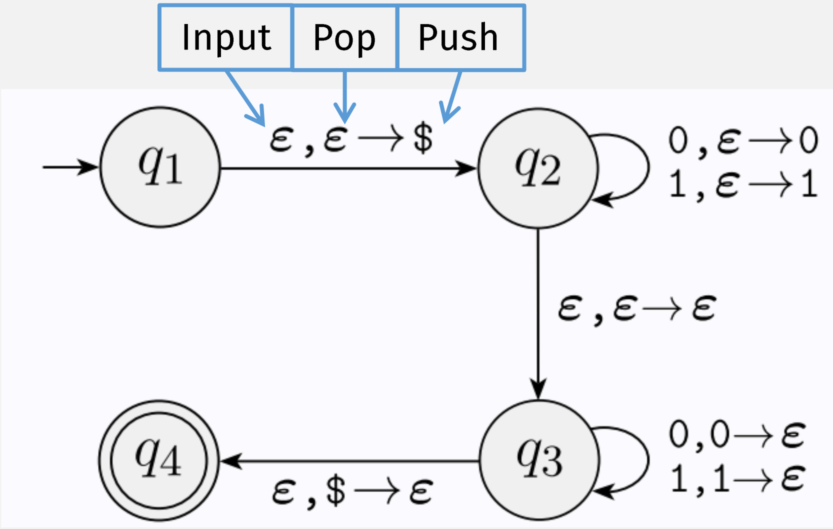
$F =$

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

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε

Input
Pop
State/ Push

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



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

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

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

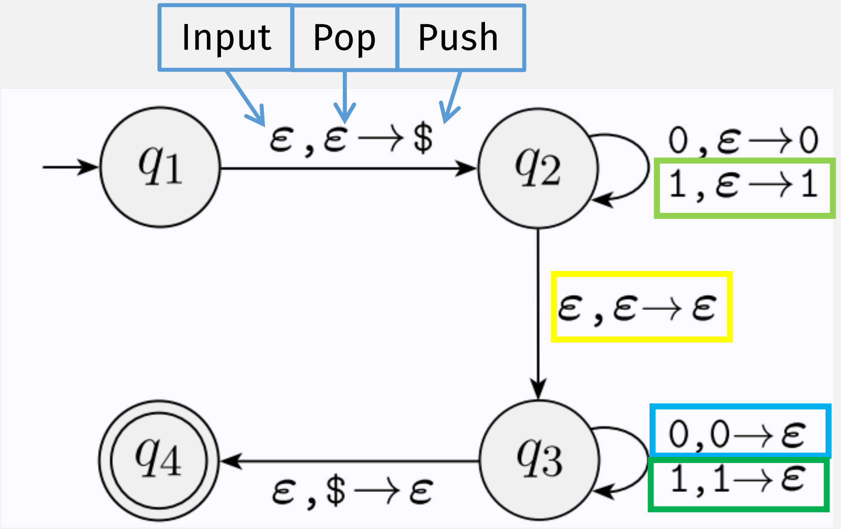
$\Gamma = \{0,1,\$\}$,

$F = \{q_4\}$

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 $\{ww^{\mathcal{R}} | w \in \{0,1\}^*\}$

Flashback: DFA Computation Model

Informally	*Formally (i.e., mathematically)*

- "Program" = a finite automata
- Input = string of chars, e.g. "`1101`"

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

To run a "program":

- Start in "start state"
- $r_0 = q_0$
- Repeat:
  - Read 1 char;
  - Change state according to the transition table
- $r_i = \delta(r_{i-1}, w_i)$, for $i = 1, \ldots, n$
- Result =
  - "**Accept**" if last state is "Accept" state
  - "**Reject**" otherwise
- M ***accepts*** w if sequence of states $r_0, r_1, \ldots, r_n$ in Q exists … with $r_n \in F$

A **sequence of states** represents a **DFA computation**

PDA Configurations (IDs)

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

- 3 components (q, w, γ) :
  q = the current state
  w = the remaining input string
  γ = the stack contents

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

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

Flashback: A DFA Extended Transition Fn

Define **extended transition function:** $\hat{\delta} : Q \times \Sigma^* \to Q$

- Domain:
  - Beginning state $q \in Q$ (not necessarily the start state)
  - Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range:
  - Ending state (not necessarily an accept state)

This specifies the **sequence of states** representing a **DFA** computation

(Defined recursively)

Empty string

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

nonEmpty string

First char

Remaining chars ("smaller argument")

- Recursive case: $\hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), w_2 \cdots w_n)$

Recursive call

Single transition step

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 | 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

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

- Recursive Case

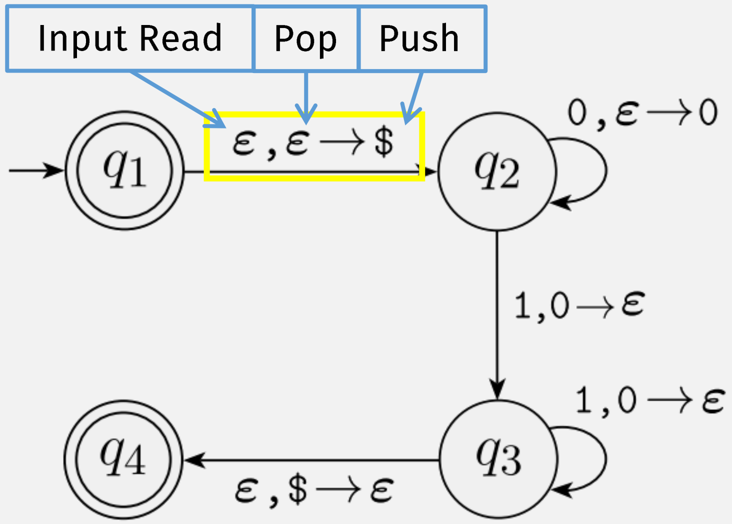
$$I \overset{*}{\vdash} J \text{ if there exists some ID } K$$

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

PDA Running Input String Example

State | Remaining Input | Stack

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



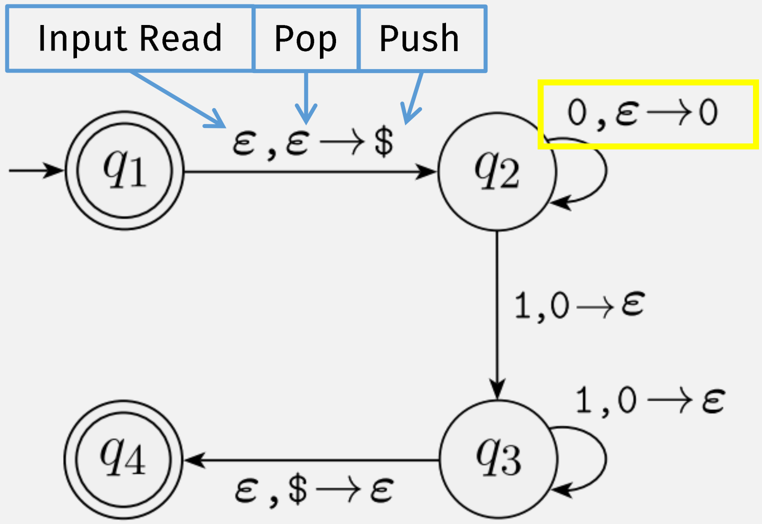
PDA Running Input String Example

State | Remaining Input | Stack

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

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

Read 0, push 0

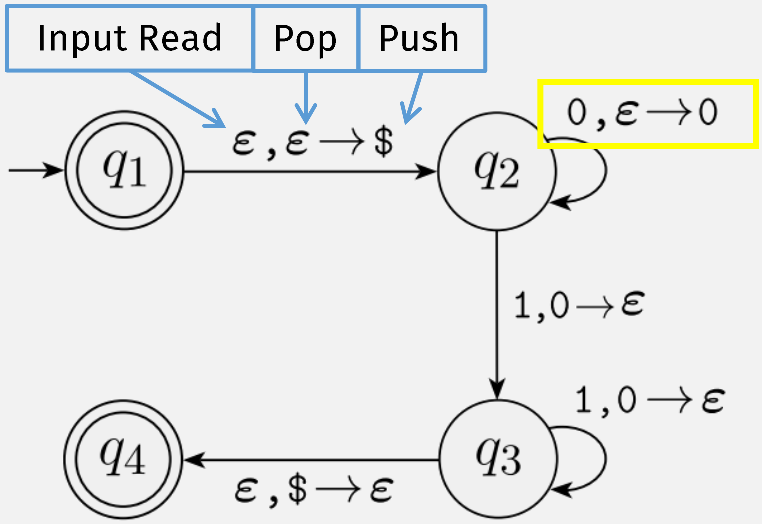


PDA Running Input String Example

State | Remaining Input | Stack

$$(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$)$$
$$\vdash (q_2, 011, 0\$)$$
$$\vdash (q_2, 11, 00\$)$$

Read 0, push 0

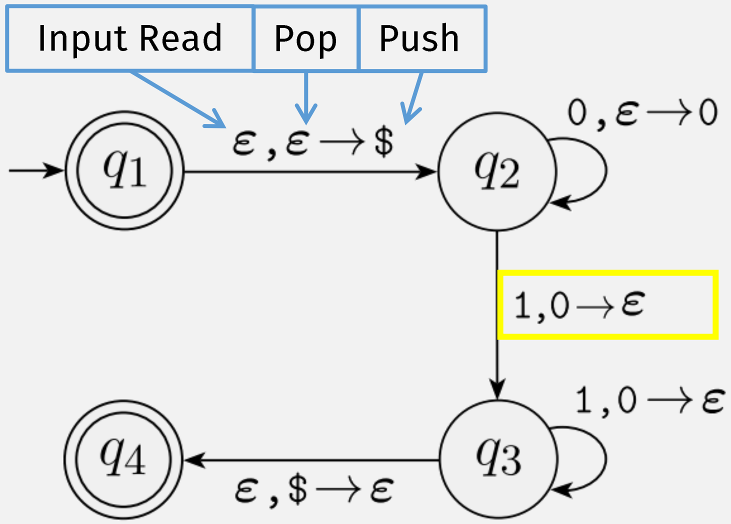


PDA Running Input String Example

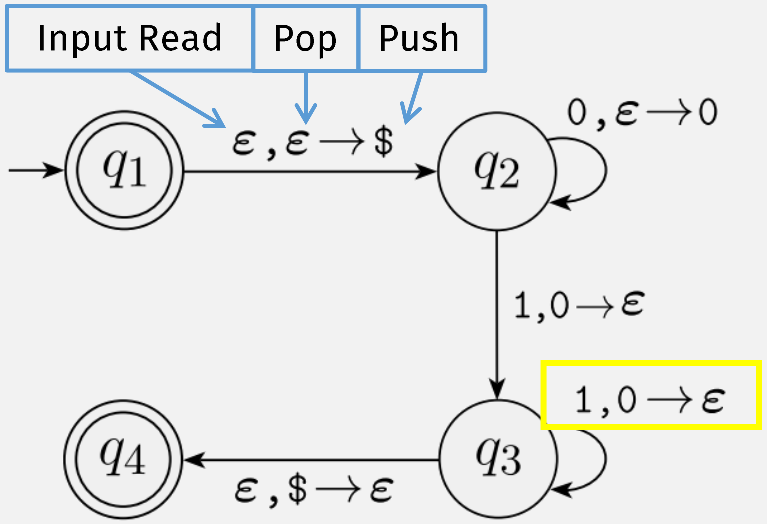
State | Remaining Input | Stack

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

Read 1, pop 0



PDA Running Input String Example

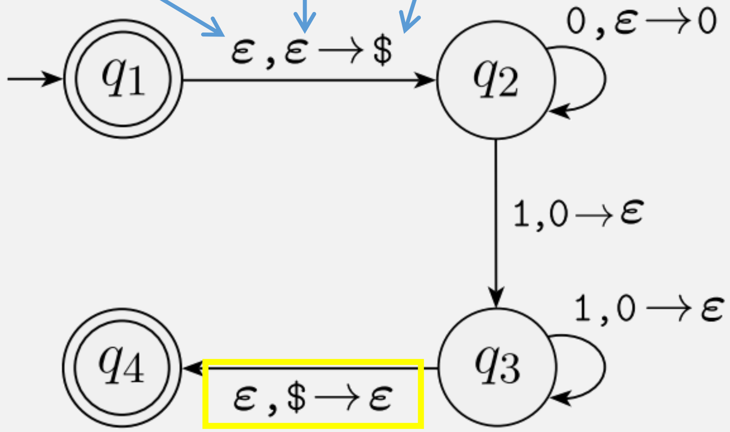


State | Remaining Input | Stack

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

Read 1, pop 0

PDA Running Input String Example



State | Remaining Input | Stack

$$(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$)$$
$$\vdash (q_2, 011, 0\$)$$
$$\vdash (q_2, 11, 00\$)$$
$$\vdash (q_3, 1, 0\$)$$
$$\vdash (q_3, \varepsilon, \$)$$
$$\vdash (q_4, \varepsilon, \varepsilon)$$

pop empty stack symbol

Flashback: Computation and Languages

- The **language** of a **machine** is the **set of all strings that it accepts**
- E.g., An FSM 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

Pushdown Automata (PDA)

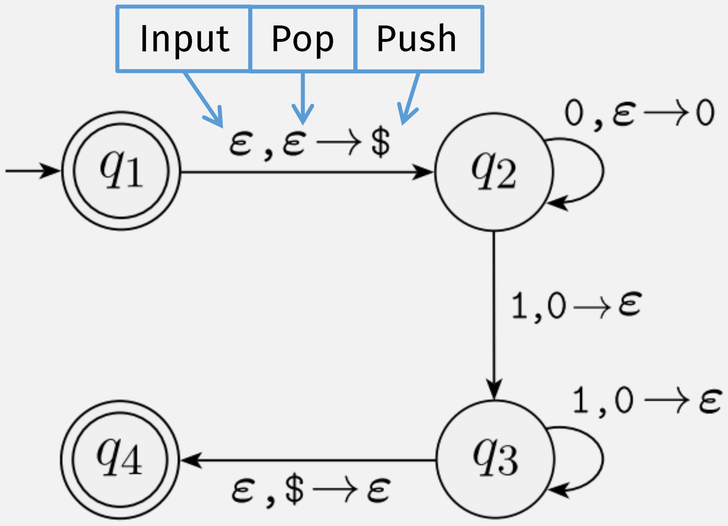
- **PDA** = NFA + a stack
  - Infinite memory
  - Can only read/write top location: Push/pop

- 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



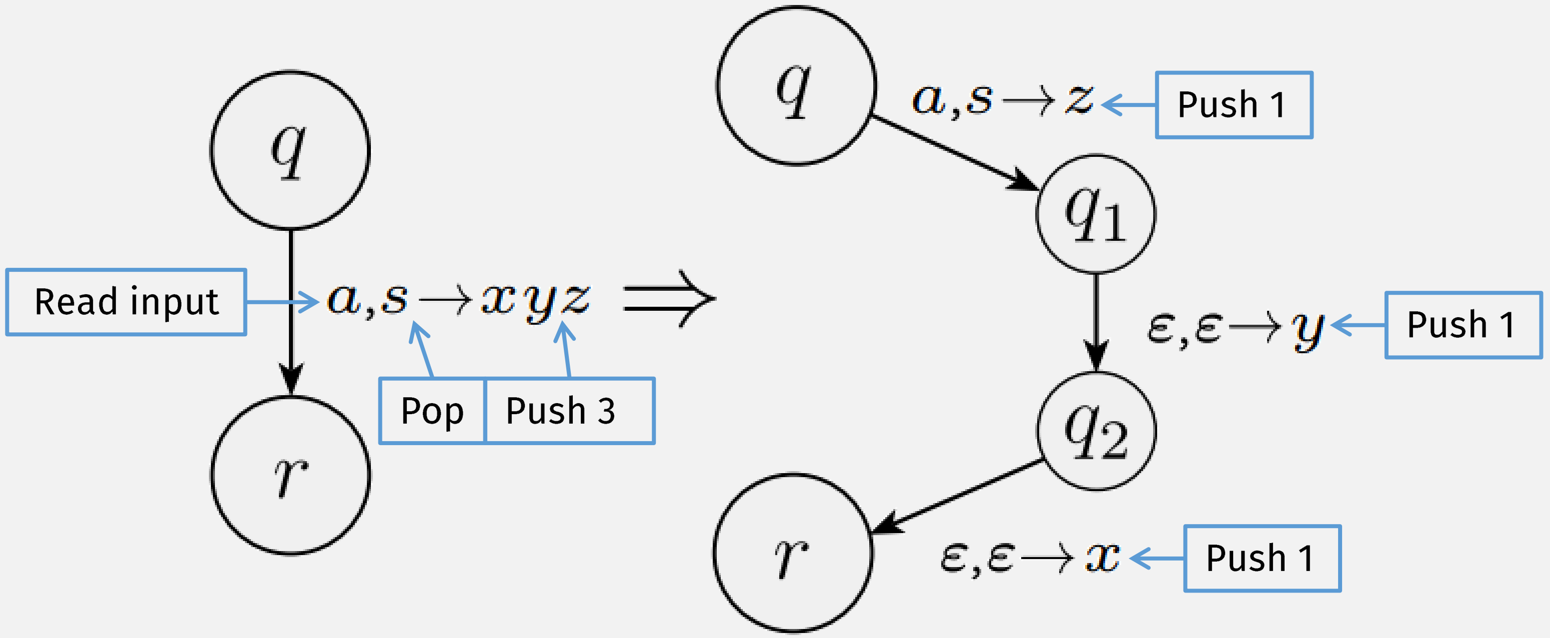
A lang is a CFL **iff** some PDA recognizes it

$\Rightarrow$ If **a language is a CFL,** then **a PDA recognizes it**

- (Easier)
- We know: **A CFL has a CFG describing it** (definition of CFL)
- Must show: **the CFG has an equivalent PDA**

$\Leftarrow$ If **a PDA recognizes a language,** then **it's a CFL**

Shorthand: Multi-Symbol Stack Pushes

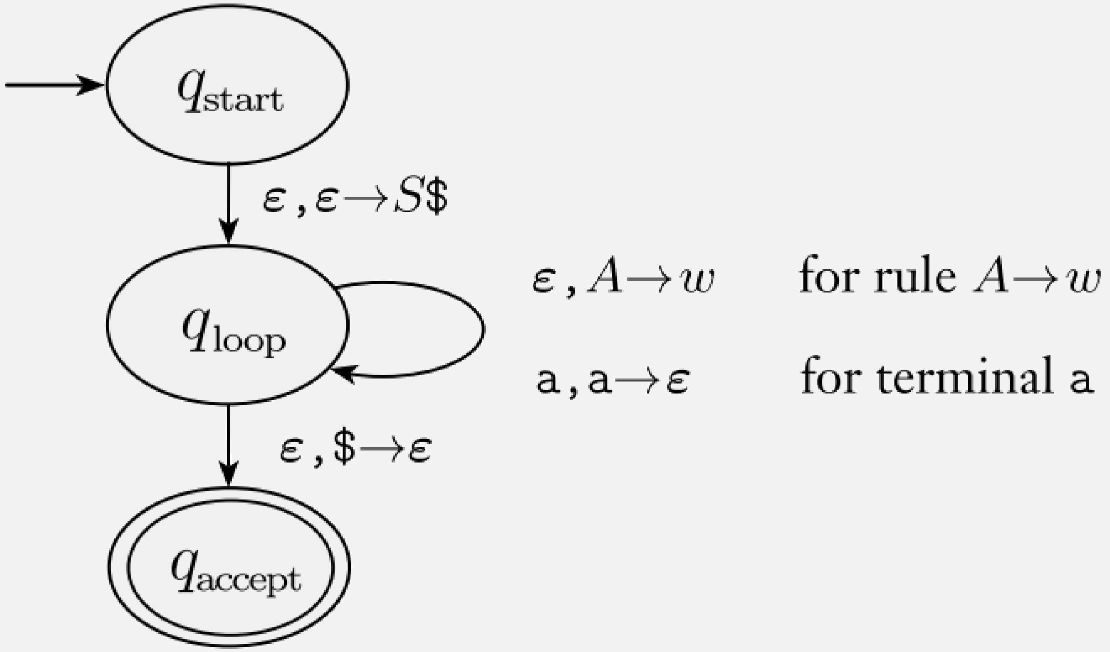


Note the <u>reverse</u> order of pushes

CFG→PDA (sketch)

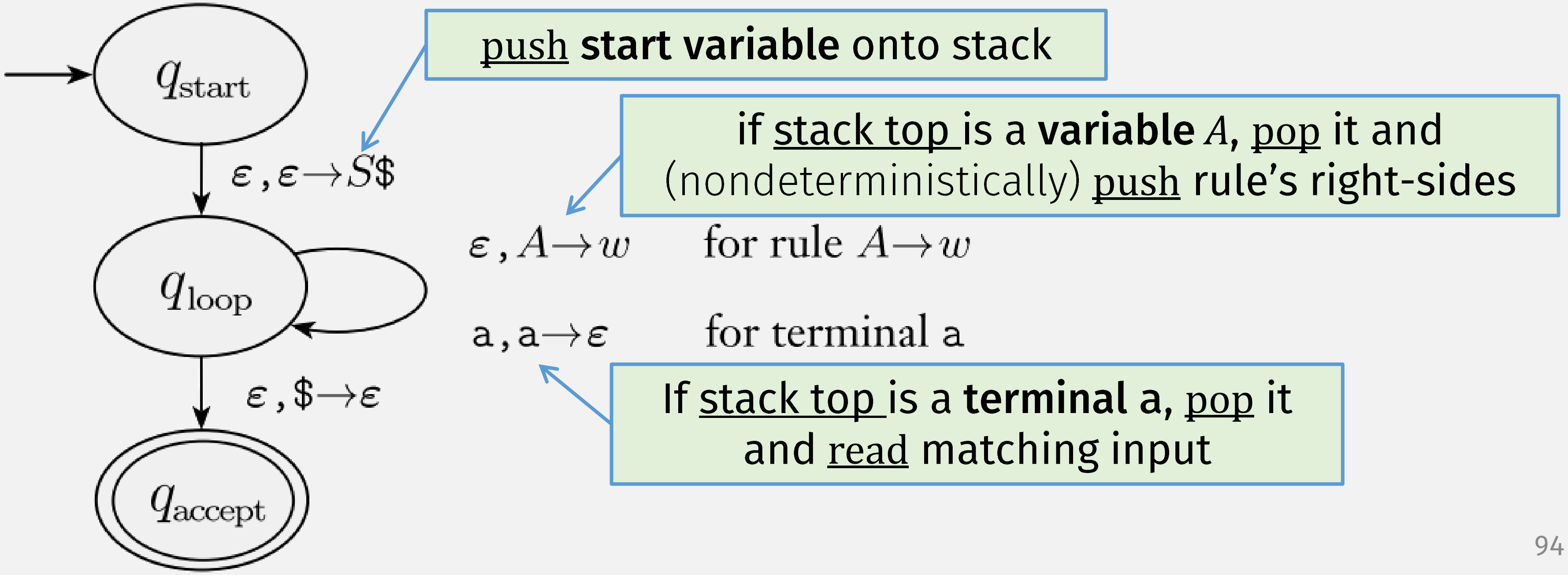
- Construct a PDA from CFG such that:
  - PDA accepts input string only if the CFG can generate that string

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



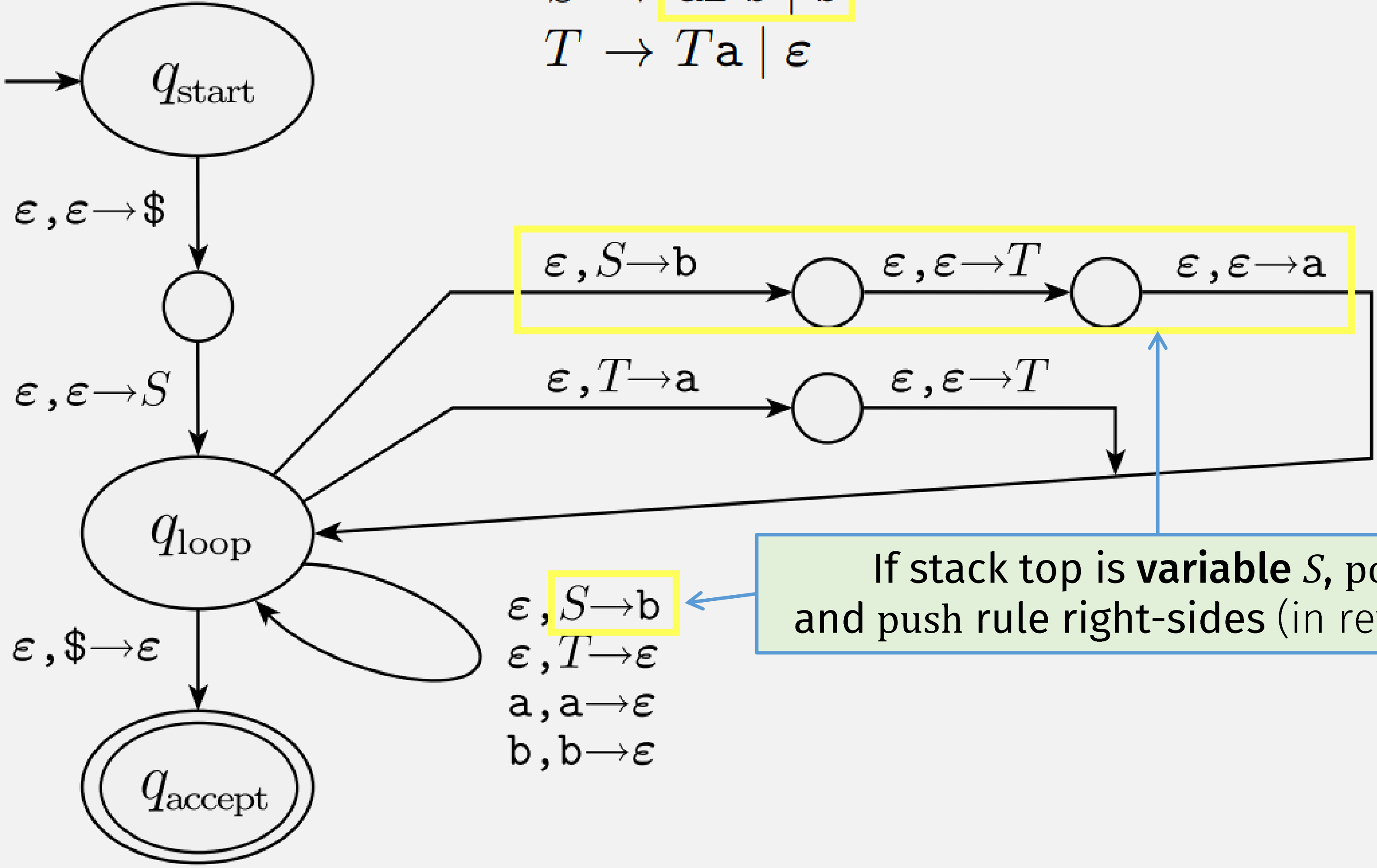
CFG→PDA (sketch)

- 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



Example **CFG→PDA**

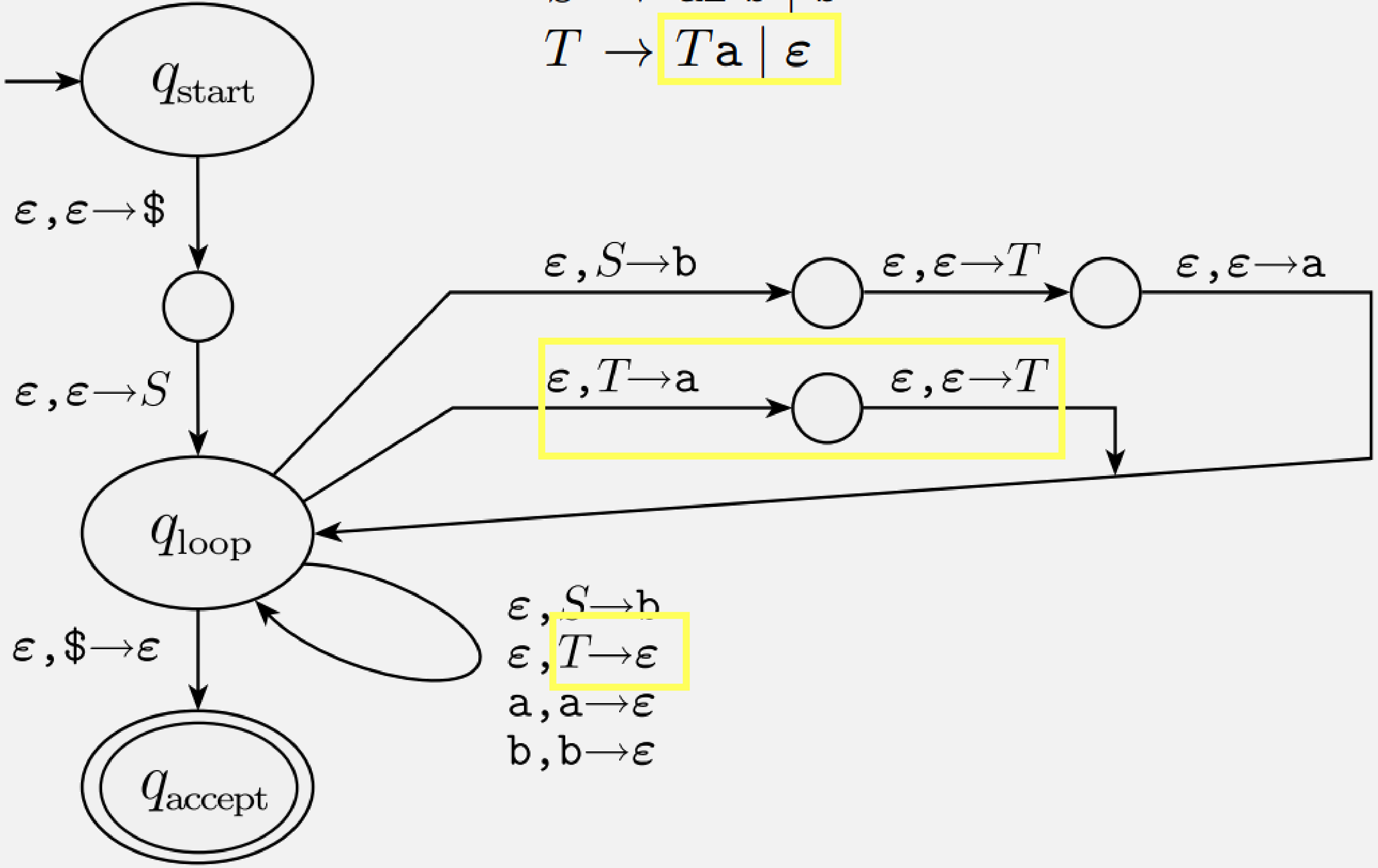
$$S \to \text{a}T\text{b} \mid \text{b}$$
$$T \to T\text{a} \mid \varepsilon$$



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

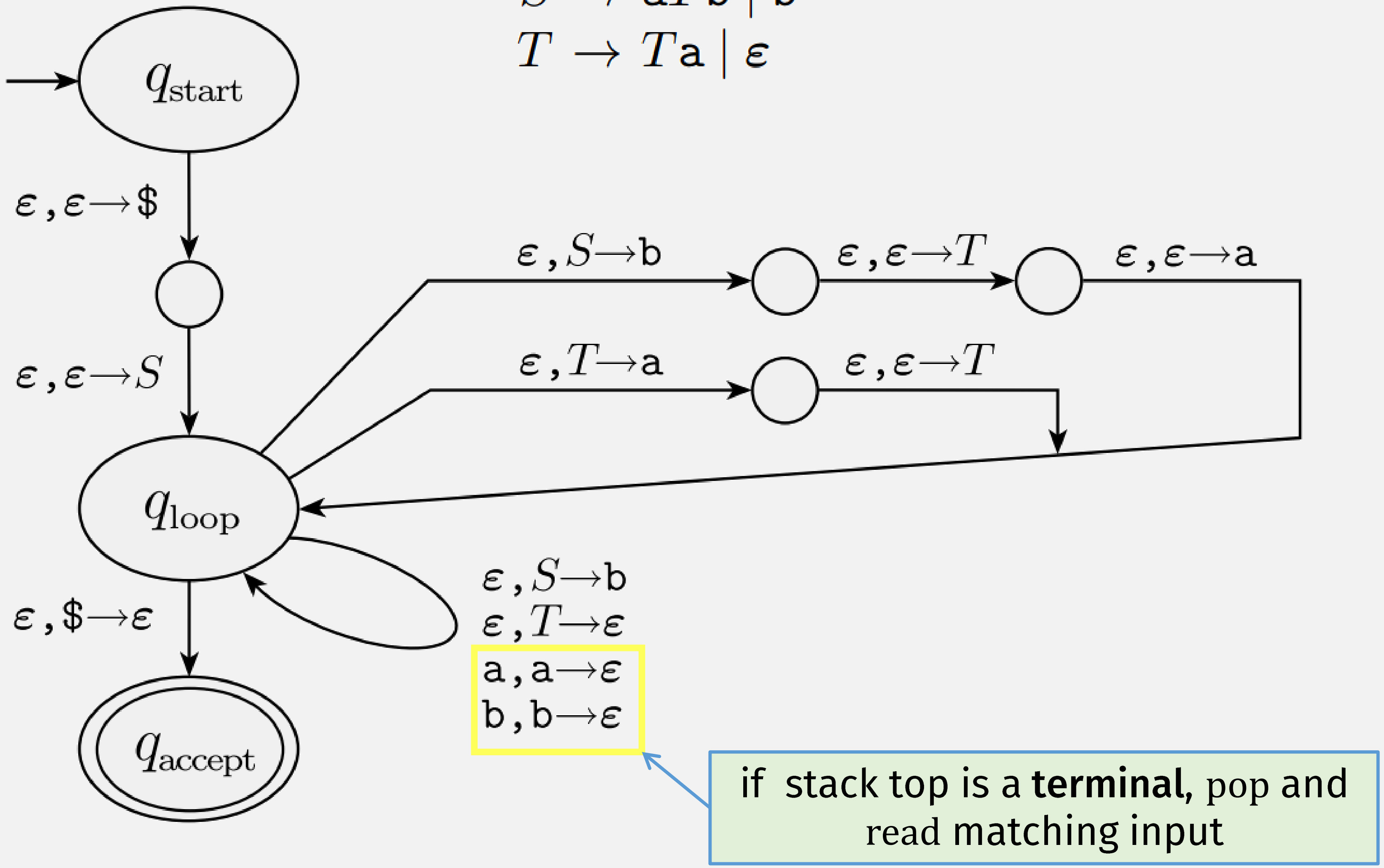
Example **CFG→PDA**

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



Example CFG→PDA

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



Example **CFG→PDA**

Example Derivation using CFG:

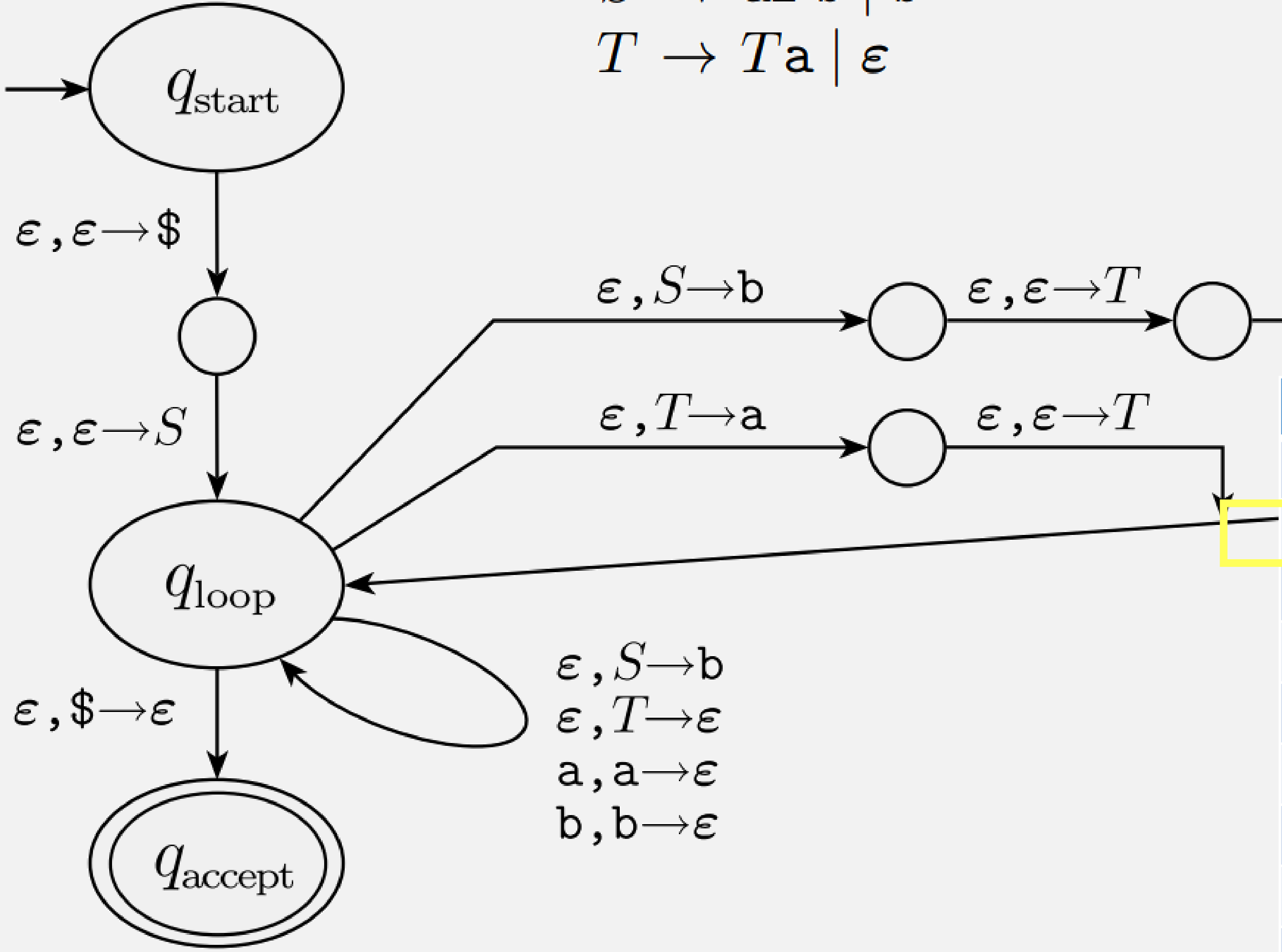
$S \Rightarrow \mathbf{a}T\mathbf{b}$ (using rule $S \rightarrow \mathbf{a}T\mathbf{b}$)

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

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

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

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



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:

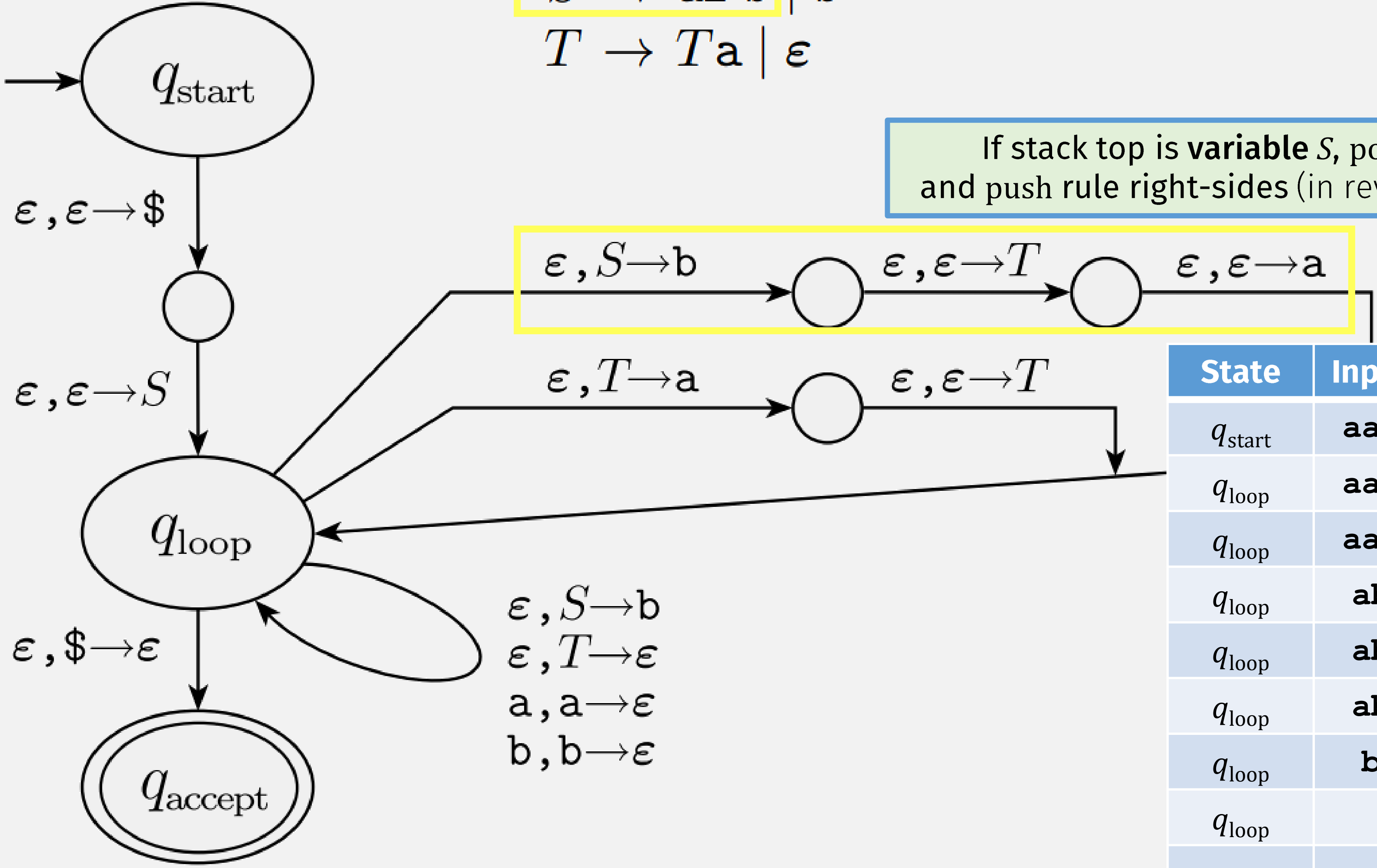
$S \Rightarrow \mathtt{a}T\mathtt{b}$ (using rule $S \rightarrow \mathtt{a}T\mathtt{b}$)

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

Example **CFG→PDA**

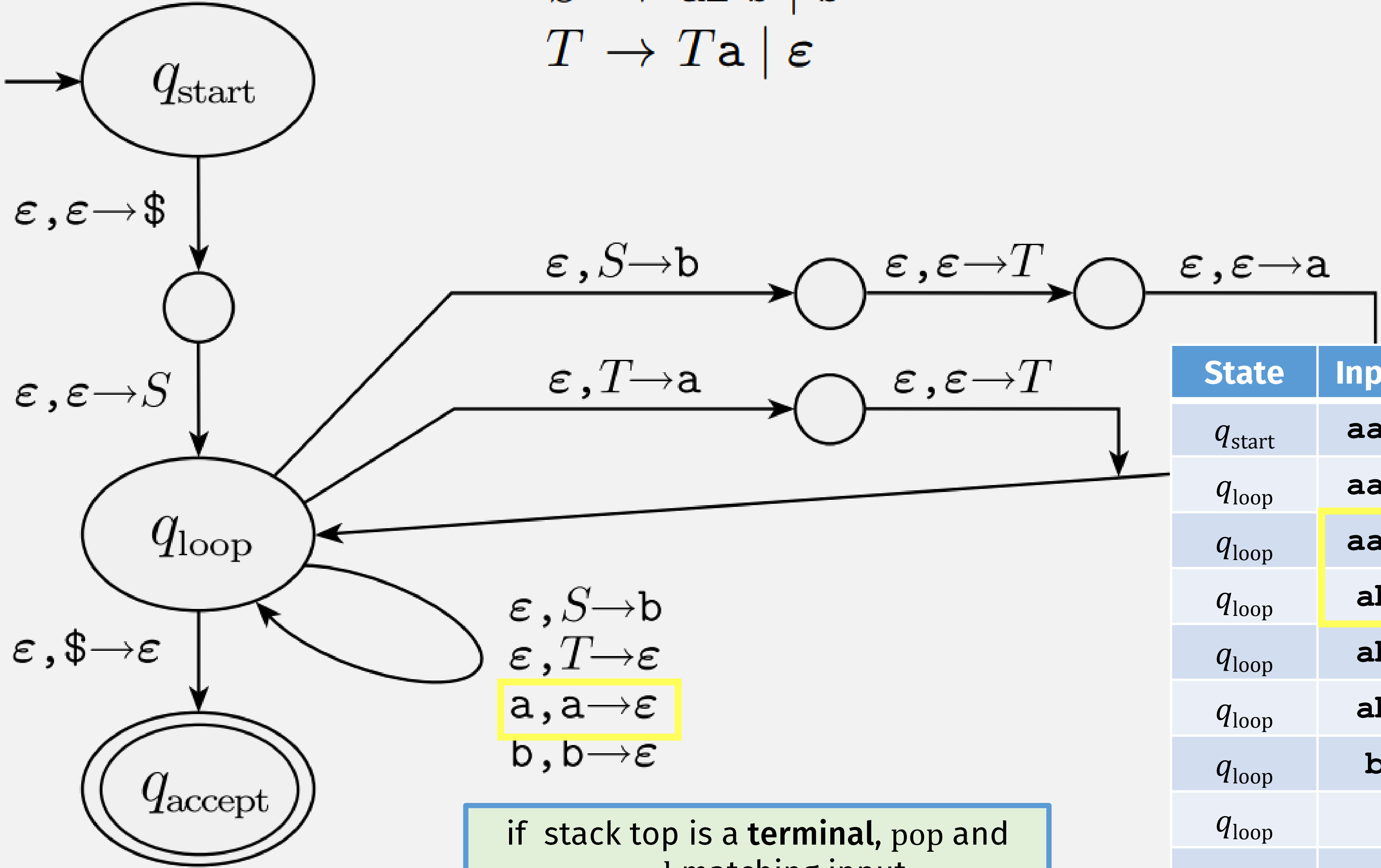
<u>Example Derivation using CFG:</u>

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

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



<u>PDA Example</u>

State	Input	Stack	Equiv Rule
q_{start}	aab		
q_{loop}	aab	$S\$$	
q_{loop}	aab	aTb$	$S \rightarrow \mathbf{a}T\mathbf{b}$
q_{loop}	ab	Tb$	
q_{loop}	ab	Tab$	$T \rightarrow T\mathbf{a}$
q_{loop}	ab	ab$	$T \rightarrow \varepsilon$
q_{loop}	b	b$	
q_{loop}		$	
q_{accept}			

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

Example CFG→PDA

Example Derivation using CFG:

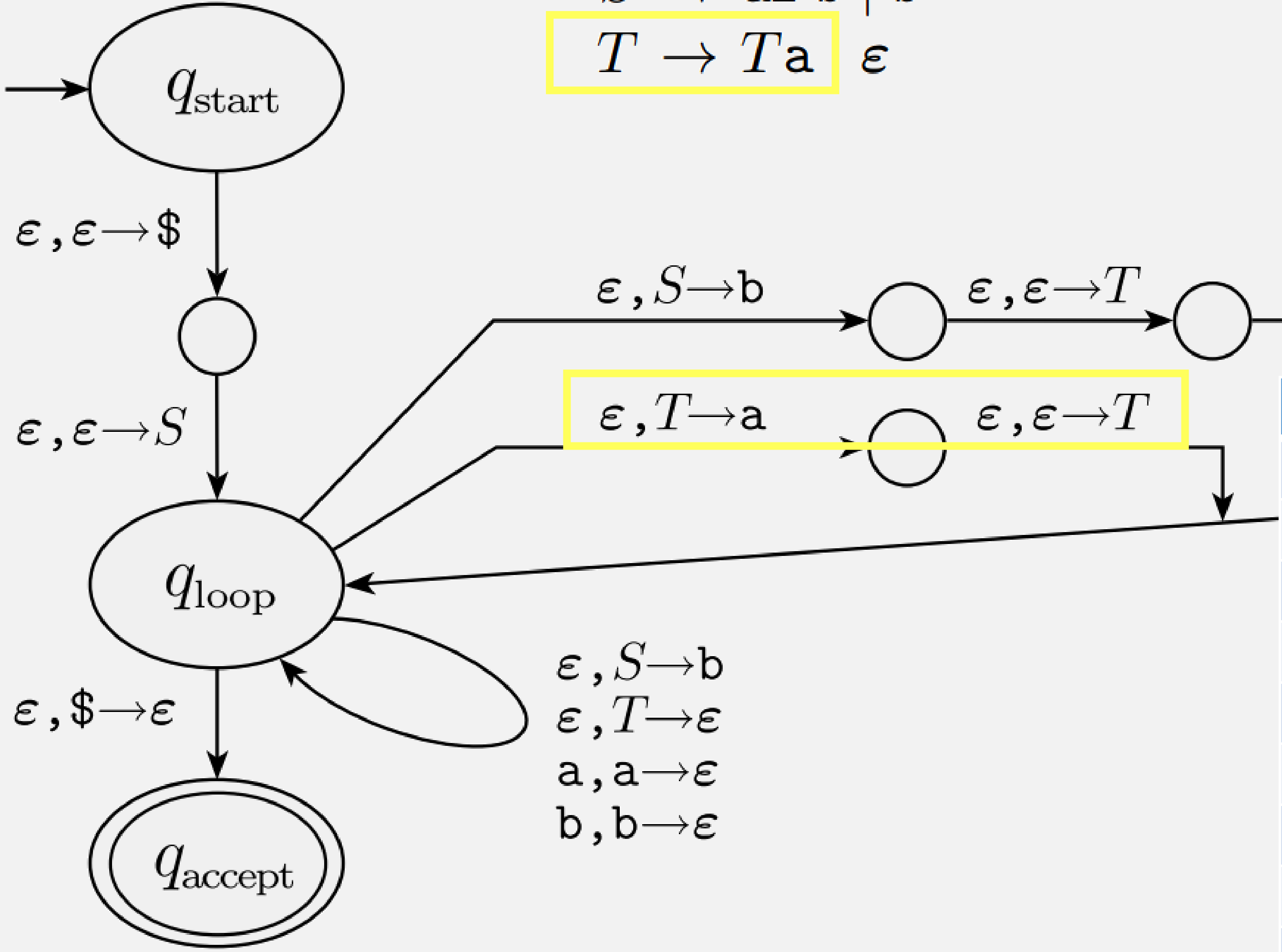
$S \Rightarrow \mathtt{a}T\mathtt{b}$ (using rule $S \to \mathtt{a}T\mathtt{b}$)

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

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

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

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



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}			

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)
- Must Show: PDA has an equivalent CFG

PDA→CFG: Prelims

Before converting PDA to CFG, <u>modify</u> 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

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

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

- <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

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

- <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**

∎

Check-in Quiz 10/18

On Gradescope