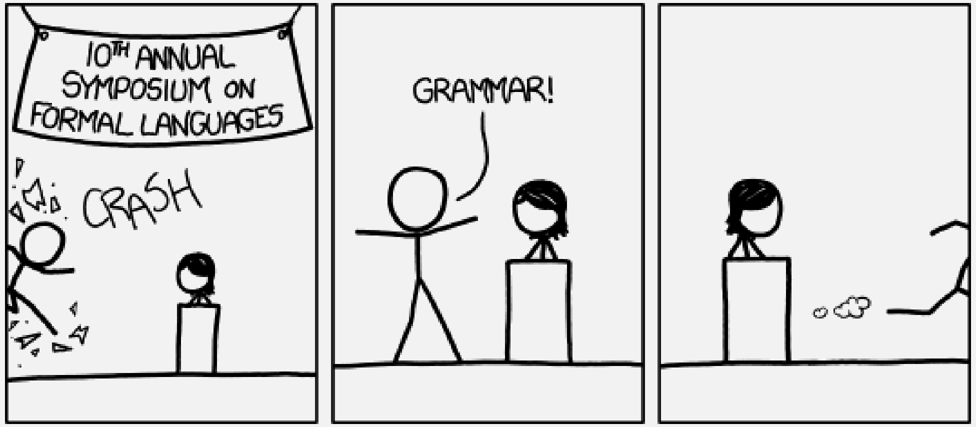
CS 420 / CS 620

Pushdown Automata (PDAs)

Wednesday October 22, 2025

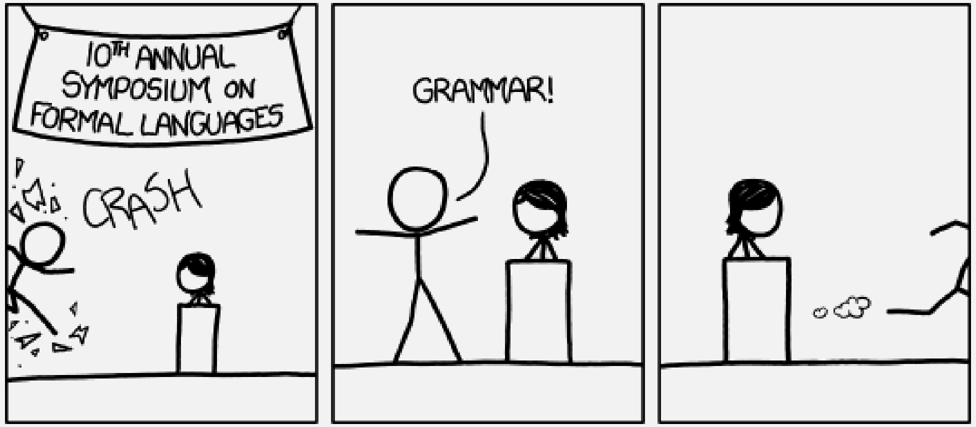
UMass Boston Computer Science



lecture14

Announcements

- HW 7
  - Out: **Mon 10/20 12pm (noon)**
  - Due: **Mon 10/27 12pm (noon)**



Last Time:

Context-Free Grammar (CFG)

Grammar $G_1 = (V, \Sigma, R, S)$

terminals

Top variable is:
Start variable

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

Variables
(a.k.a., **non-terminals**)

R is this set of rules (var-string pairs):

Substitution rules
(a.k.a., **productions**)

terminals (analogous to DFA's alphabet)

A ***context-free grammar*** is a 4-tuple $\boxed{(V, \Sigma, R, S)}$, where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

$V = \{A, B\}$,

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

$S = A$,

Last Time:

Generating Strings with a CFG

Grammar $G_1 = (V, \Sigma, R, S)$

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

Strings in CFG's language
= all possible **generated** / **derived** strings

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

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

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

This sequence of steps is called a **derivation**

Last Time:

A *context-free grammar* is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

Derivations: Formally

Let $G = (V, \Sigma, R, S)$

Single-step

$$\boxed{\alpha}\boxed{A}\boxed{\beta} \underset{G}{\Rightarrow} \boxed{\alpha}\boxed{\gamma}\boxed{\beta}$$

Where:

$\boxed{\alpha, \beta} \in (V \cup \Sigma)^*$ ← sequence of terminals or variables

$A \in V$ ← Variable

$\boxed{A} \rightarrow \boxed{\gamma} \in R$ ← Rule

Last Time:

Derivations: Formally

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

Let $G = (V, \Sigma, R, S)$

Single-step

$$\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$$

Where:

$\alpha, \beta \in (V \cup \Sigma)^*$ ← sequence of terminals or variables

$A \in V$ ← Variable

$A \to \gamma \in R$ ← Rule

Multi-step (recursively defined)

<u>Base case:</u> $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$ (0 steps)

<u>Recursive case:</u> (1 or more steps)

$$\alpha \overset{*}{\underset{G}{\Rightarrow}} \gamma$$

Where: $\alpha \underset{G}{\Rightarrow} \beta$ and $\beta \overset{*}{\underset{G}{\Rightarrow}} \gamma$

Single step

(smaller) Recursive "call"

Last Time:

Formal Definition of a CFL

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

$$G = (V, \Sigma, R, S)$$

"all possible sequences of terminal symbols …"

… "that can be **generated** with **rules of grammar** G"

"the **language** of a **grammar** G is …"

$$L(G) = \left\{ w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w \right\}$$

Any language that can be generated by some context-free grammar is called a ***context-free language***

Alternatively (an easier form to use in a proof is):

IF a **language** can be **generated by some CFG**, THEN that **language** is a **CFL**

Or: IF a **CFG describes** a **language**, THEN that **language** is a **CFL**

Last Time:

Designing Grammars : Basics

1. Think about what you want to “link” together

- E.g., $0^n 1^n$
  - $A \rightarrow 0A1$
  - **# 0s** and **# 1s** are “**linked**”

- E.g., **HTML**
  - $\mathrm{ELEMENT} \rightarrow \mathrm{<TAG>CONTENT</TAG>}$
  - **Start** and **end tags** are “**linked**”

2. **Start** with **small grammars** (computation) and **then combine**
   - just like with **DFA**s, **NFA**s, and **programming**!

Last Time:

Designing Grammars: Building Up

- **Start** with **small grammars** and then **combine** (just like **programming**)
  - To create a **grammar for the language** $\{0^n 1^n | n \geq 0\} \cup \{1^n 0^n | n \geq 0\}$
  - First create **grammar for lang** $\{0^n 1^n | \ n \geq 0\}$:

    $\boxed{S_1} \rightarrow 0S_1 1 \mid \varepsilon$
  - Then create **grammar for lang** $\{1^n 0^n | \ n \geq 0\}$:

    $\boxed{S_2} \rightarrow 1S_2 0 \mid \varepsilon$
  - Then **combine:** $S \rightarrow \boxed{S_1} \mid \boxed{S_2}$

    $S_1 \rightarrow 0S_1 1 \mid \varepsilon$

    $S_2 \rightarrow 1S_2 0 \mid \varepsilon$

New start variable and **rule** **combines two smaller grammars**

"|" = "or" = union (combines 2 rules with same left side)

Last Time:

(Closed) Operations for CFLs?

- **Start** with **small grammars** and then **combine** (just like **programming**)

- "Or": $S \to S_1 \mid S_2$

- "Concatenate": $S \longrightarrow S_1 S_2$

- "Star" (repetition): $S' \longrightarrow S' S_1 \mid \varepsilon$

Status check:
Could you write out the precise **Statement to Prove** and the **full proof?**

"The set of CFLs are closed under …"

"IF L_1 and L_2 are CFLs THEN … is a CFL"

Regular Language vs CFL Comparison

Regular Languages	Context-Free Languages (CFLs)
Regular Expression	Context-Free Grammar (CFG)
describes a Regular Lang	describes a CFL

Regular Language vs CFL Comparison

Regular Languages	**Context-Free Languages (CFLs)**
Regular Expression	Context-Free Grammar (CFG)
describes a **Regular Lang**	describes a **CFL**
Finite State Automaton (FSM)	???
recognizes a **Regular Lang**	recognizes a **CFL**

Regular Language vs CFL Comparison

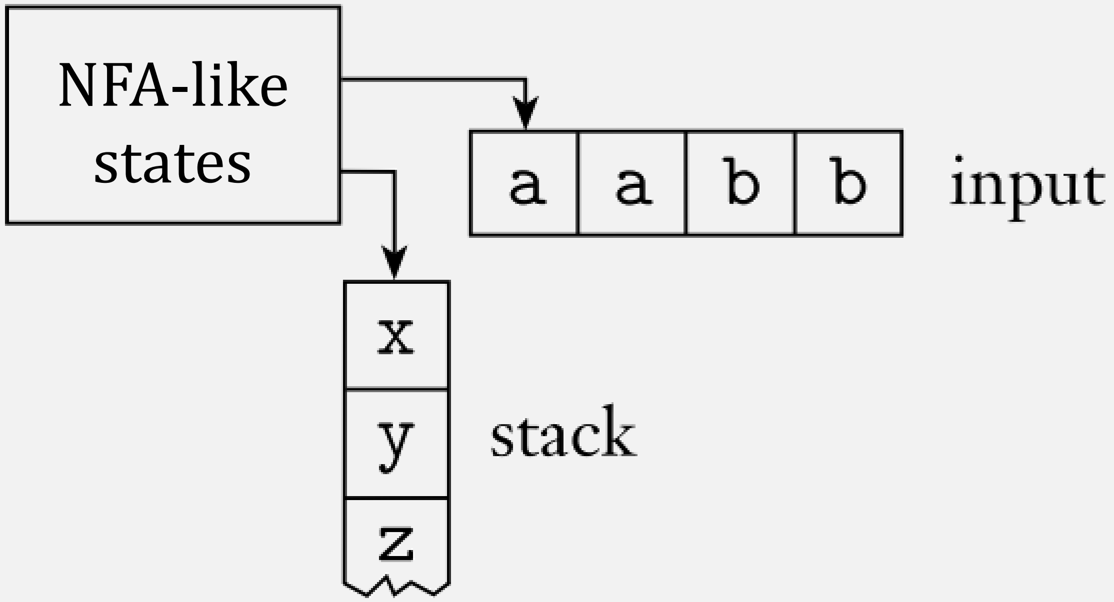
	Regular Languages	Context-Free Languages (CFLs)	
thm	Regular Expression	Context-Free Grammar (CFG)	def
	describes a Regular Lang	describes a CFL	
	Finite State Automaton (FSM)	**Push-down Automata** (PDA)	thm
def	recognizes a Regular Lang	recognizes a CFL	

Regular Language vs CFL Comparison

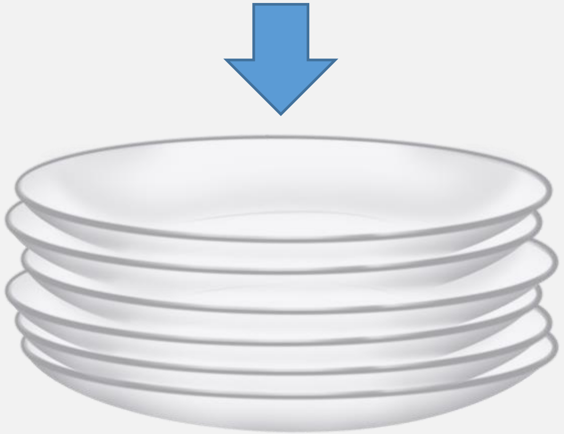
Regular Languages	Context-Free Languages (CFLs)
thm Regular Expression	Context-Free Grammar (CFG)
describes a Regular Lang	describes a CFL def
Finite State Automaton (FSM)	**Push-down Automata** (PDA) thm
def recognizes a Regular Lang	recognizes a CFL
Proved:	Must Prove:
Regular Lang ⇔ Regular Expr ☑	CFL ⇔ PDA ???

Pushdown Automata (PDA)

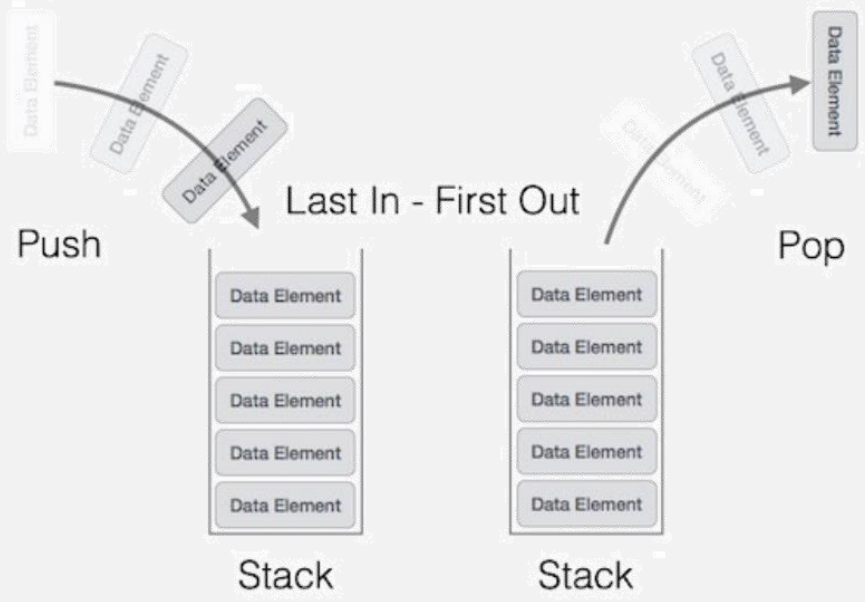
PDA = NFA + a <u>stack</u>



What is a Stack?

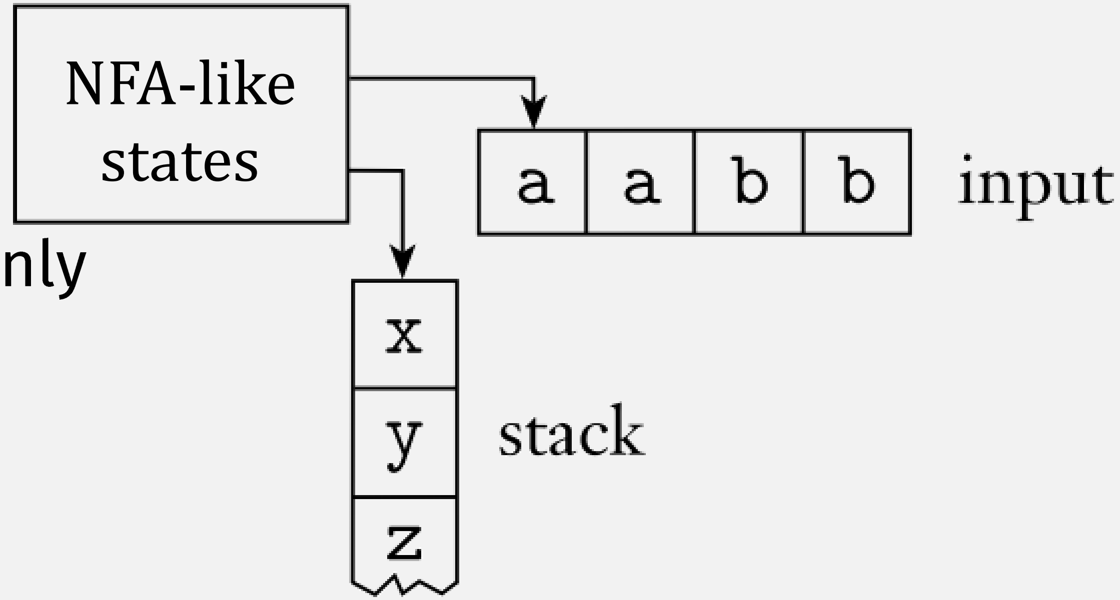


- A <u>**restricted**</u> kind of (**infinite**!) **memory**
- **Access** to **top element only**
- 2 Operations only: **push, pop**



Pushdown Automata (PDA)

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

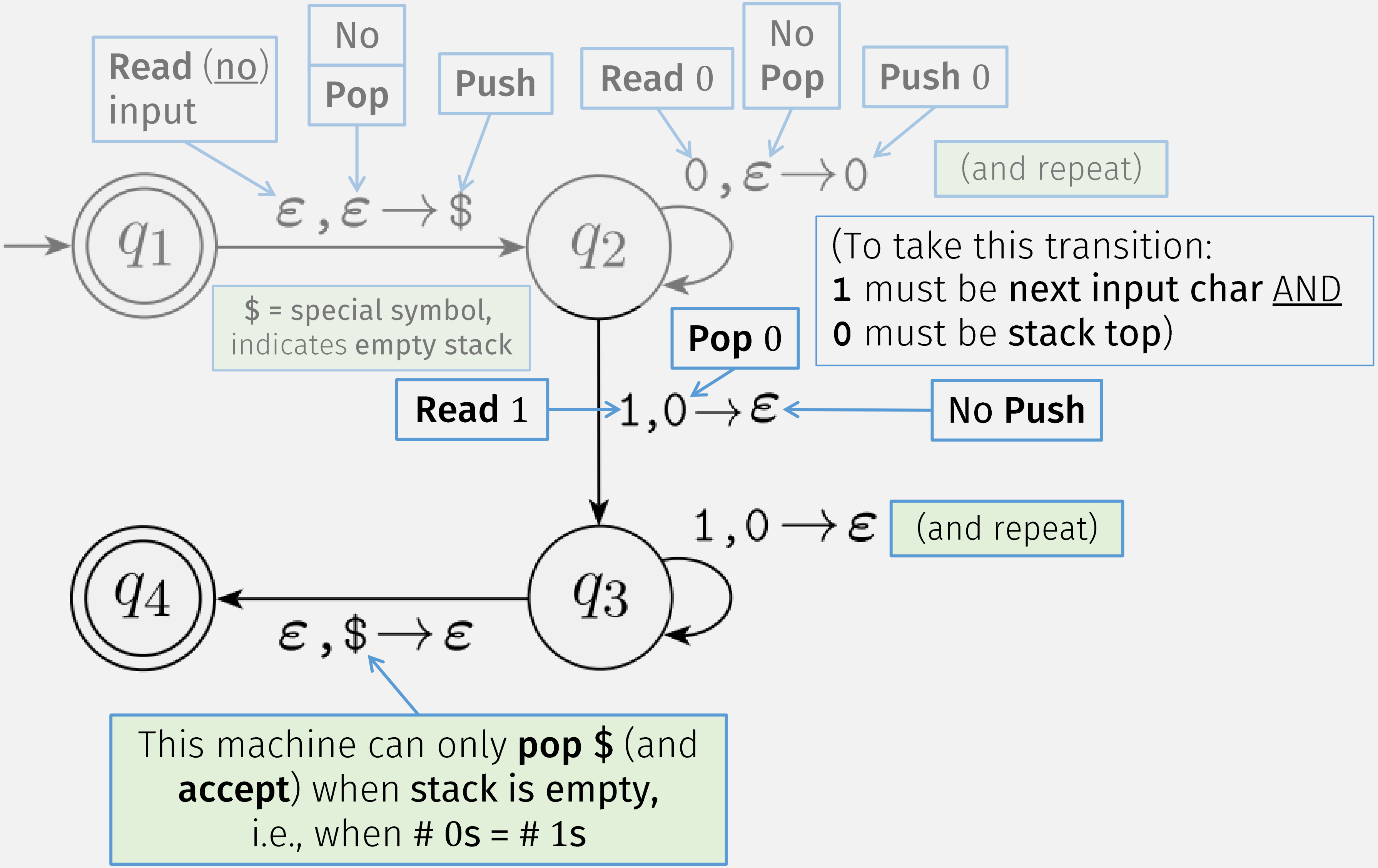


$$\{0^n 1^n \mid n \geq 0\}$$

An Example PDA

A **PDA transition** has 3 parts:

- **Read** (input)
- **Pop** (stack)
- **Push** (stack)



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. $\boxed{\Gamma \text{ is the stack alphabet,}}$

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

4. $\boxed{\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon) \text{ is the transition function,}}$

   Input Pop Push

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

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

PDA Formal Definition Example

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

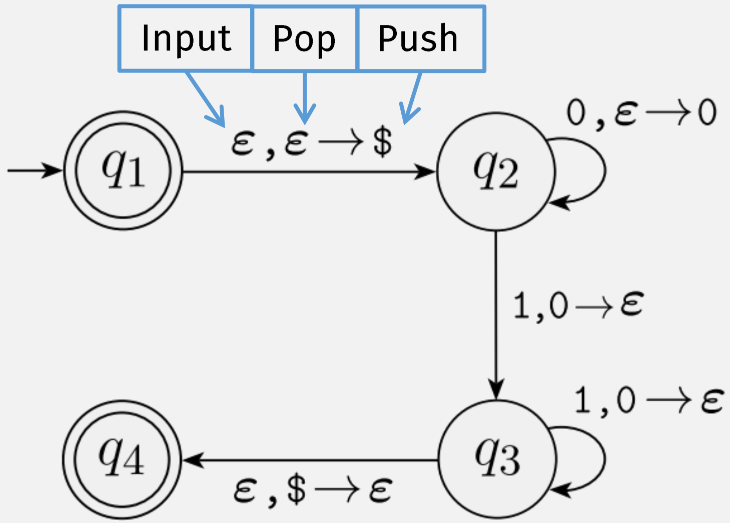
$$Q = \{q_1, q_2, q_3, q_4\},$$

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

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

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

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$,

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

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

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

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

Let's play a game:

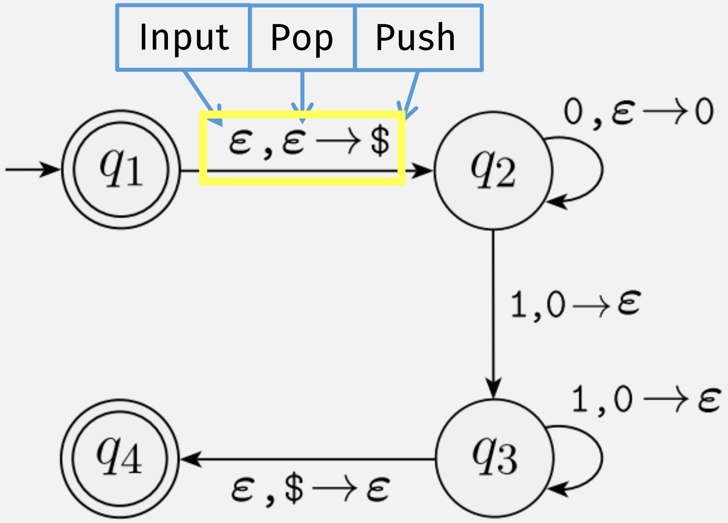
"Match transition to a number!"

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\boxed{\{(q_2, \$)\}}$ 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

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$,

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

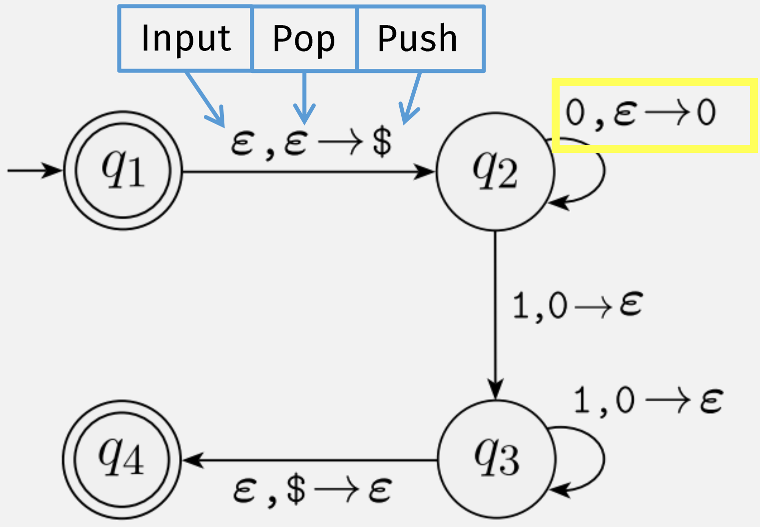
$\Gamma = \{0, \$\}$,

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

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$,

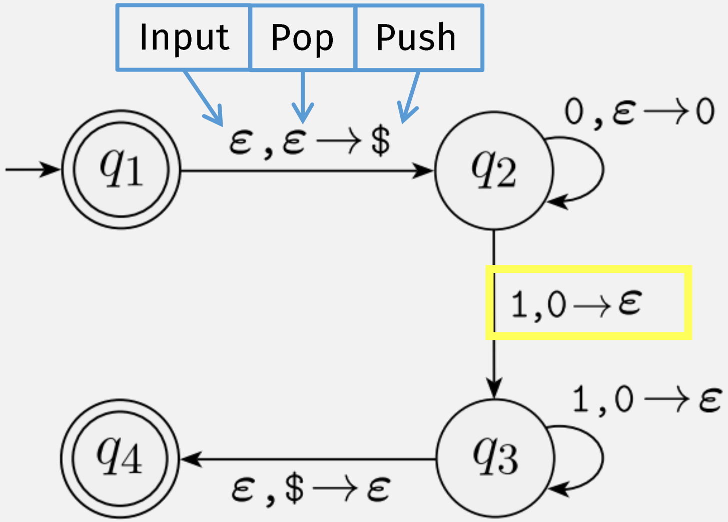
$\Sigma = \{0,1\}$,

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

$F = \{q_1, q_4\}$, 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			$\{(q_2, 0)\}$	$\boxed{\{(q_3, \varepsilon)\}}$ 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. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$,

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

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

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

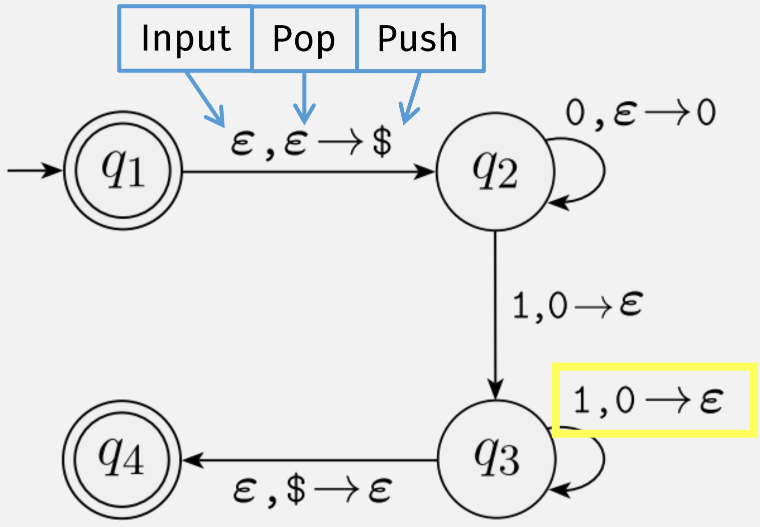
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

Input

Pop

Push

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.

Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$,

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

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

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

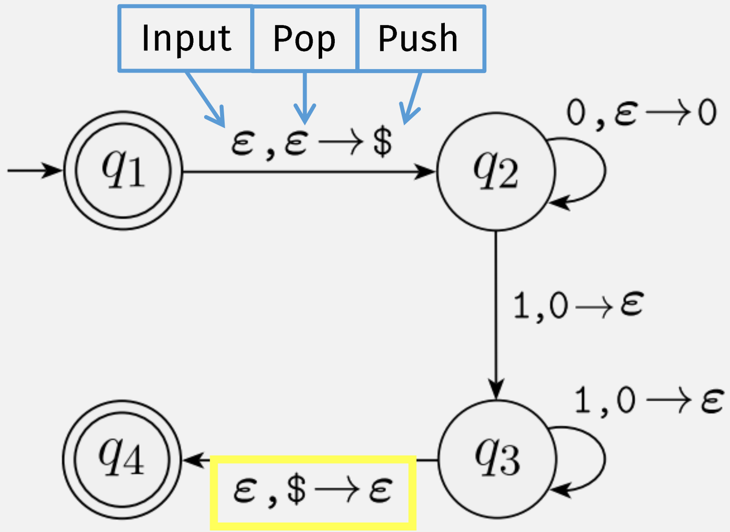
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

Let M_3 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where $Q =$

In-class exercise:
Fill in the blanks

$\Sigma =$

$\Gamma =$

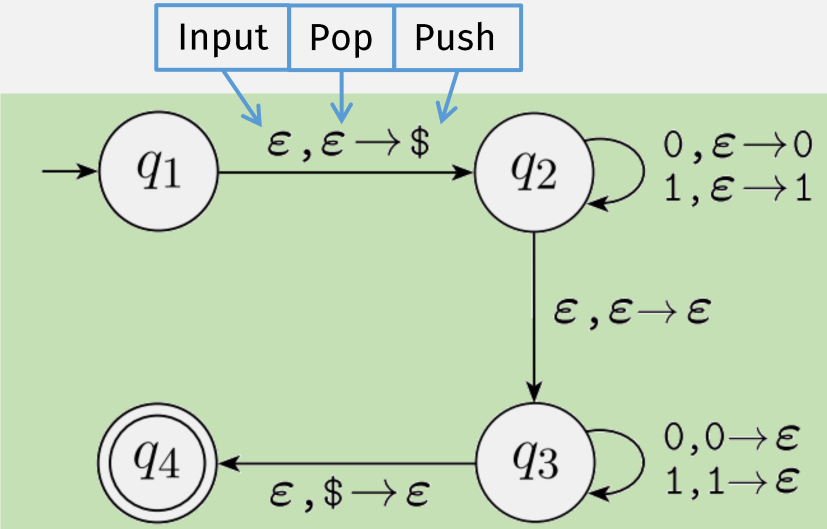
$F =$

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



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

Let M_3 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$,

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

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

$F = \{q_4\}$

In-class exercise:

Fill in the blanks

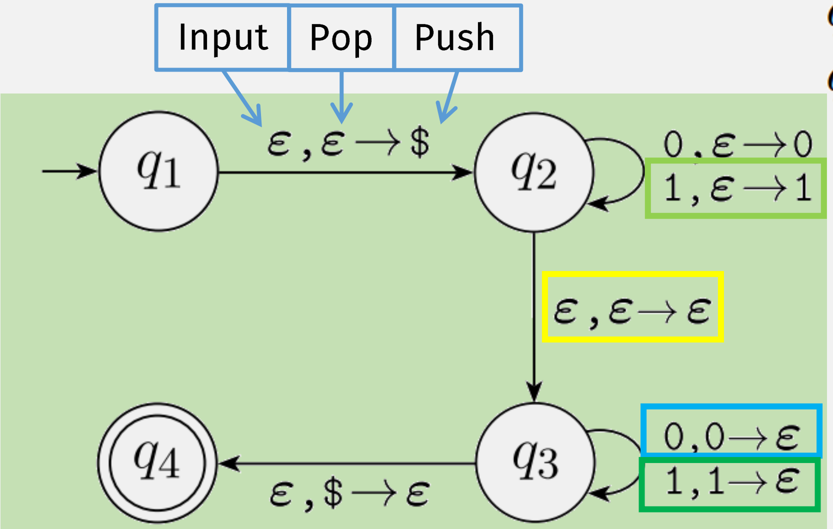
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

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

- Domain (inputs):
  - state $q \in Q$
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - state $q \in Q$

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

(Defined recursively)

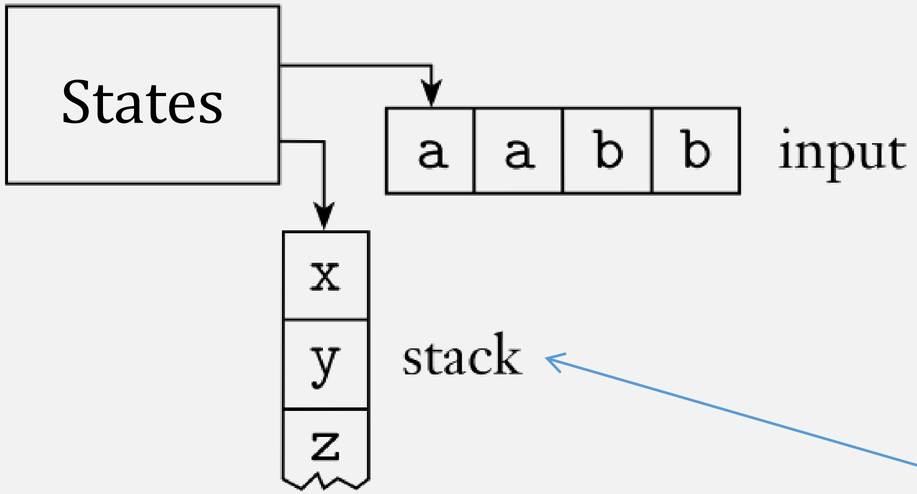
Base case $\hat{\delta}(q, \varepsilon) = q$

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

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

PDA Computation?

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



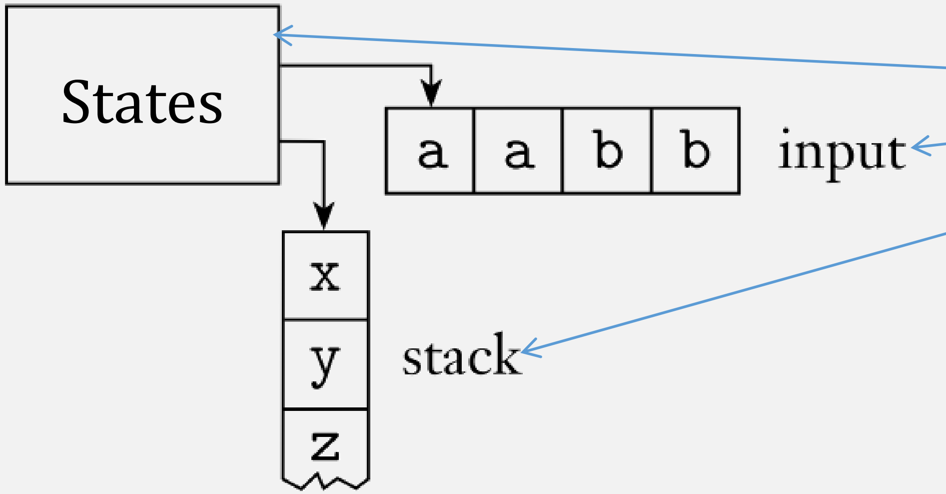
A DFA **computation** is a <u>sequence of states</u> ...

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

(one path in computation tree)

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

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

1 or more 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

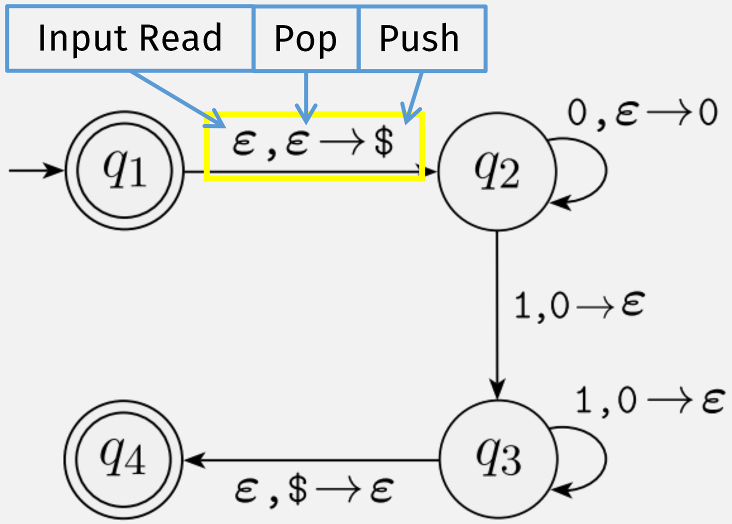
Recursive "call"

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

PDA Running Input String Example

State | Remaining Input | Stack

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



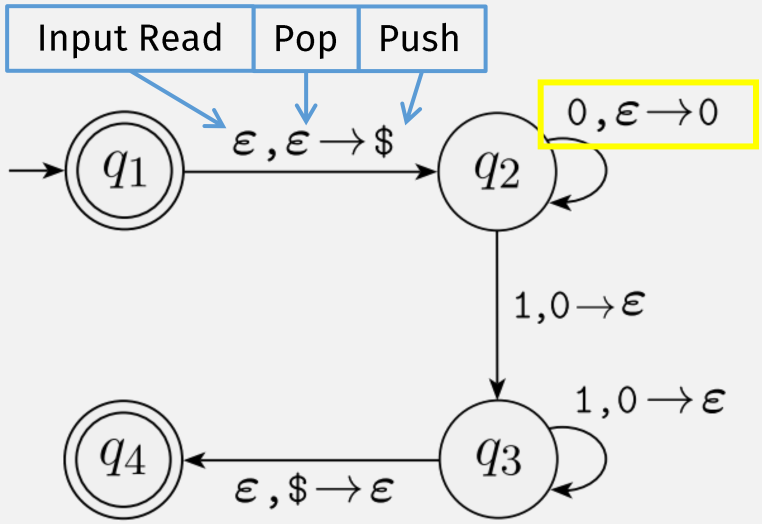
PDA Running Input String Example

State | Remaining Input | Stack

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

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

Read 0, push 0

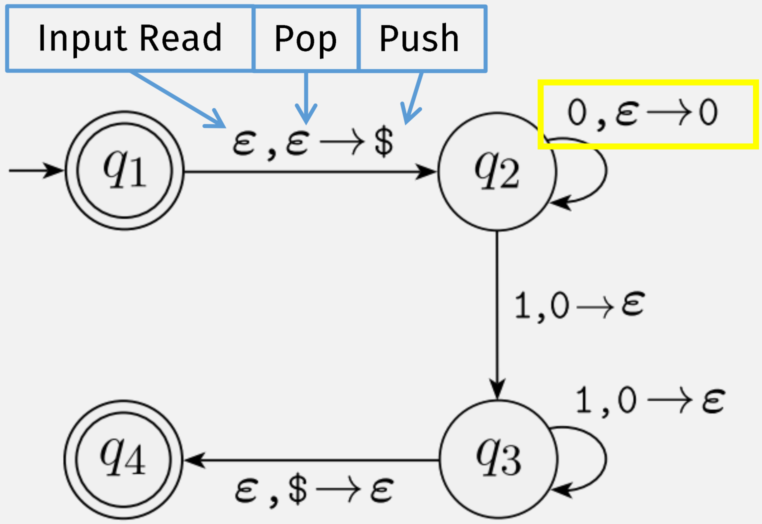


PDA Running Input String Example

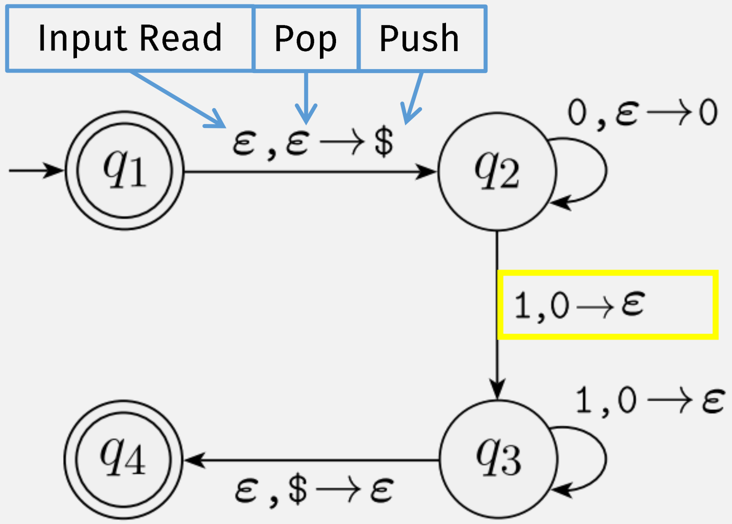
State | Remaining Input | Stack

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

Read 0, push 0



PDA Running Input String Example



State | Remaining Input | Stack

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

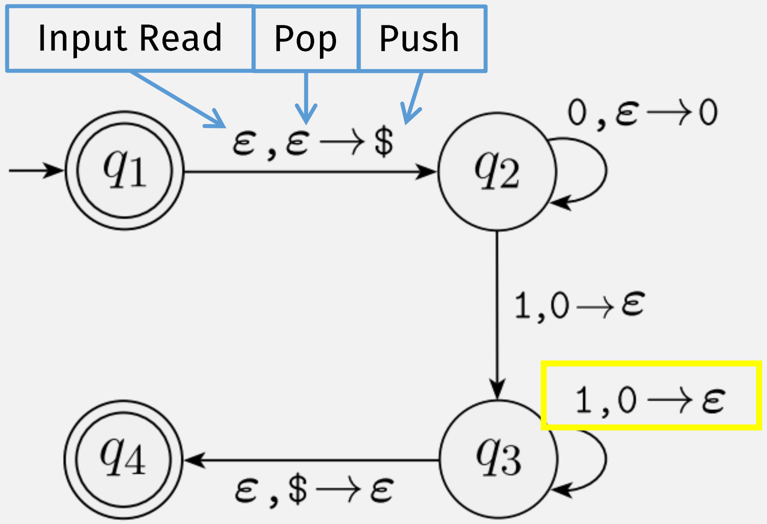
Read 1, pop 0

PDA Running Input String Example

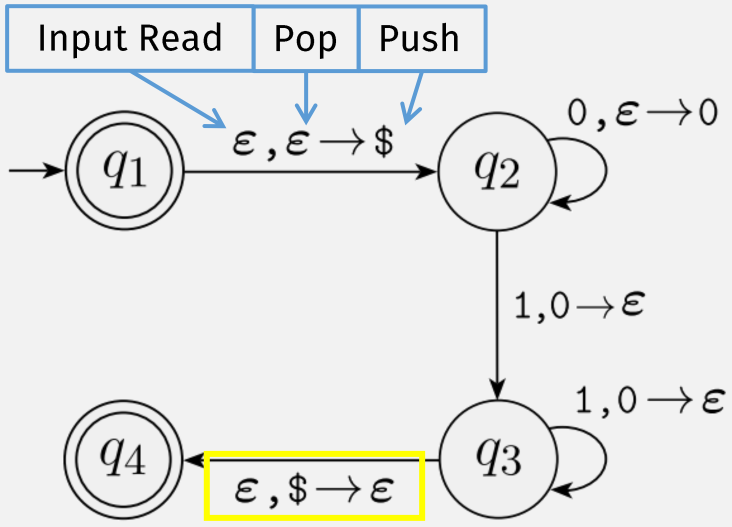
State | Remaining Input | Stack

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

Read 1, pop 0



PDA Running Input String Example



State | Remaining Input | Stack

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

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 $M = L(M) = \{ w \mid M$ **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

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

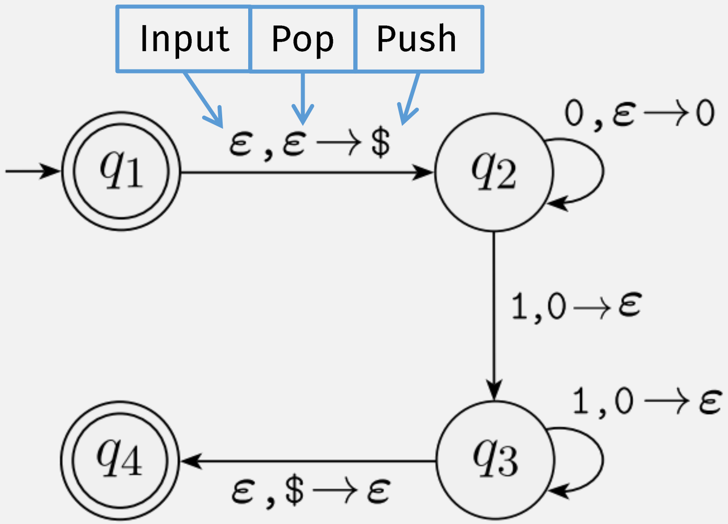
Machine accepts if final state is accept state

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

Regular Language vs CFL Comparison

Regular Languages	Context-Free Languages (CFLs)
Regular Expression	Context-Free Grammar (CFG)
describes a Regular Lang	describes a CFL
Finite State Automaton (FSM)	**Push-down Automata** (PDA)
recognizes a Regular Lang	recognizes a CFL
Proved:	Must Prove:
Regular Lang ⇔ Regular Expr ☑	CFL ⇔ PDA ???

thm

def

def

thm

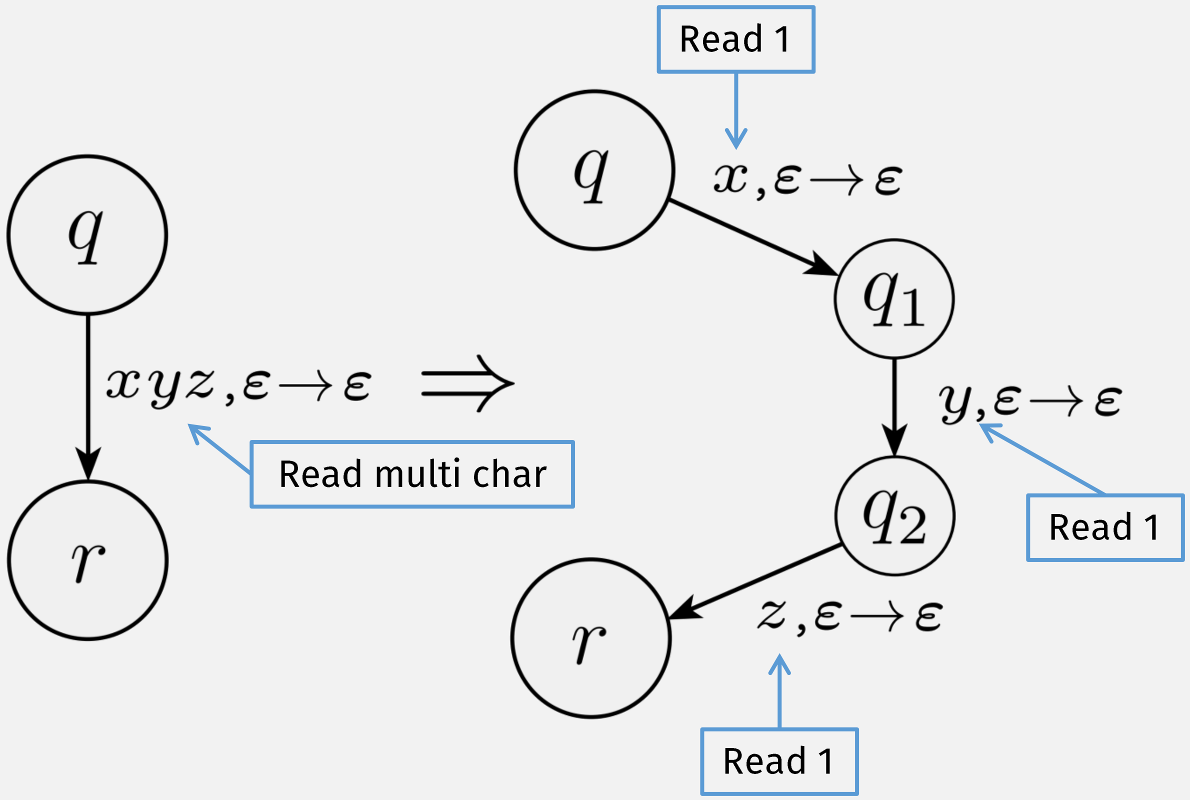
A lang is a CFL **iff** some PDA recognizes it

⇒ If **a language is a** $\boxed{\text{CFL}}$, then **a PDA recognizes it**

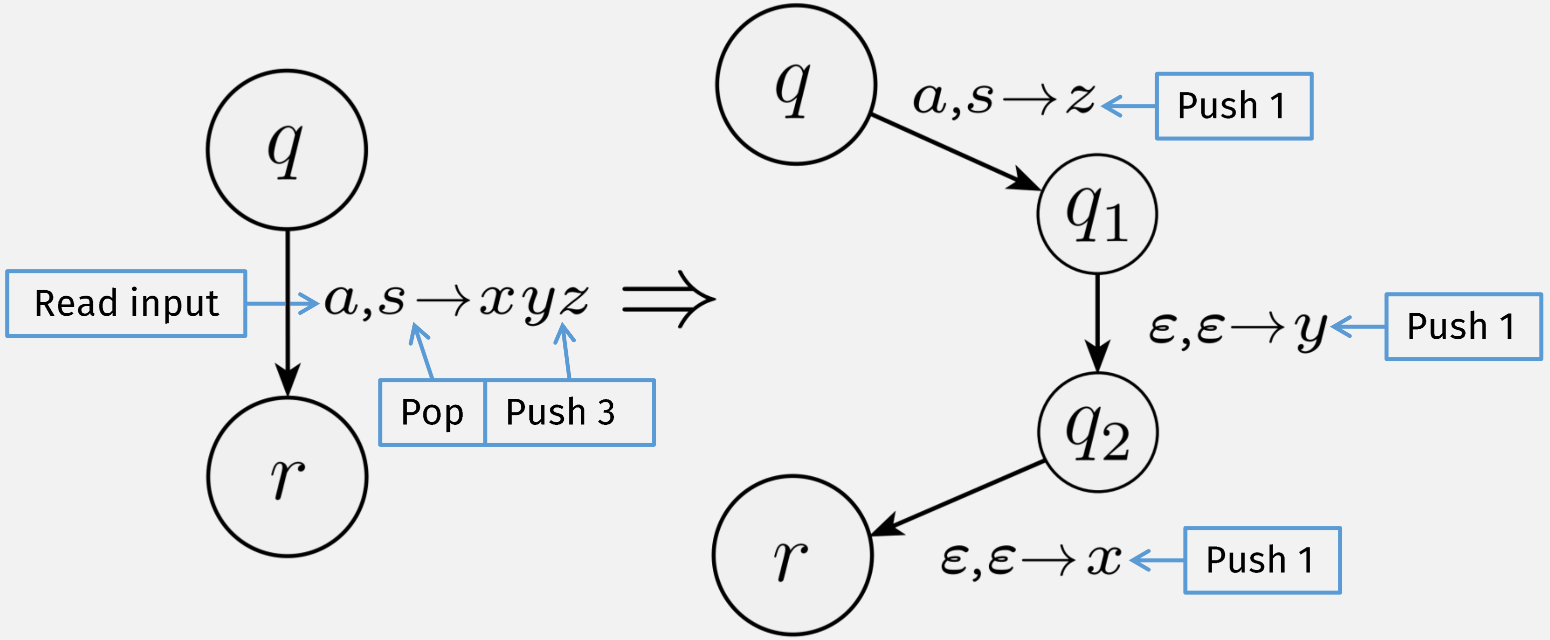
- We know: A **CFL** has a **CFG describing it** (definition of CFL)
- To prove this part, show: the **CFG** has an equivalent **PDA**

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

Shorthand: Multi-Symbol Read Transition



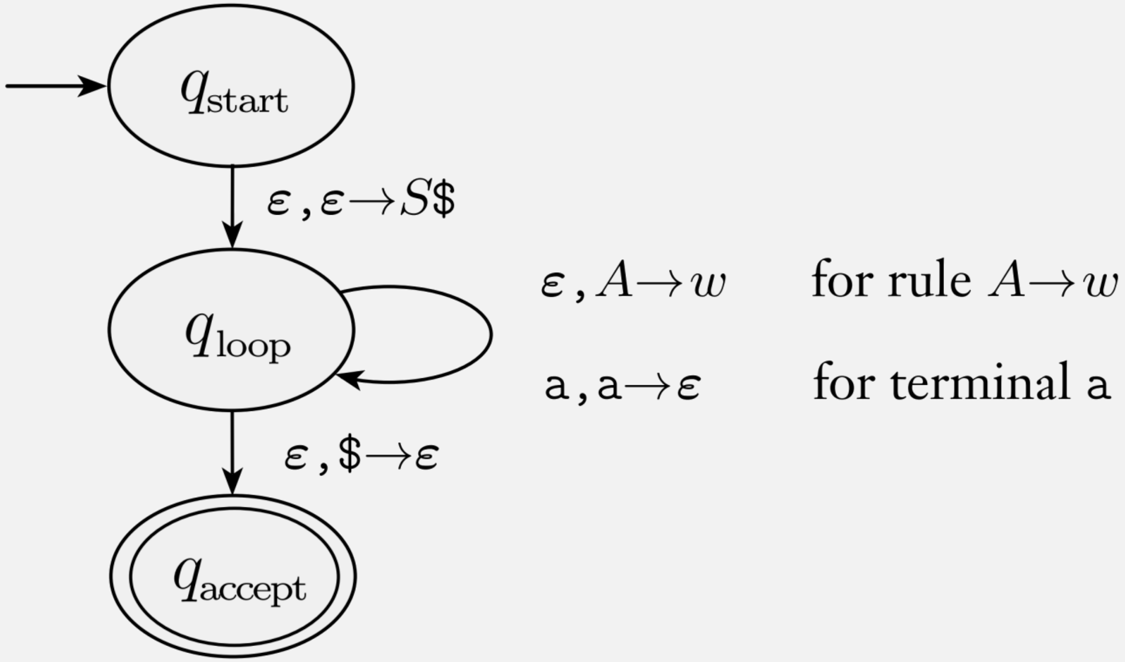
Shorthand: Multi-Stack Push Transition



Note the reverse order of pushes

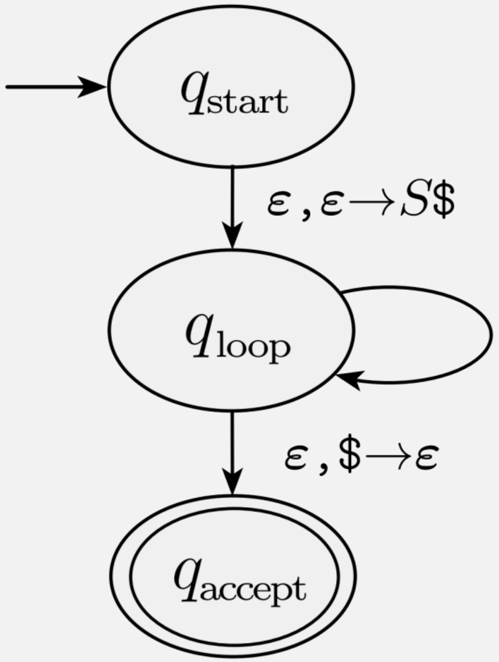
CFG→PDA (sketch)

- **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



Convert: **every CFG rule** to **PDA "loop" transition** that:
- Pops LHS variable
- Pushes RHS

(Stack is "workspace" containing intermediate string of vars + terminals)

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

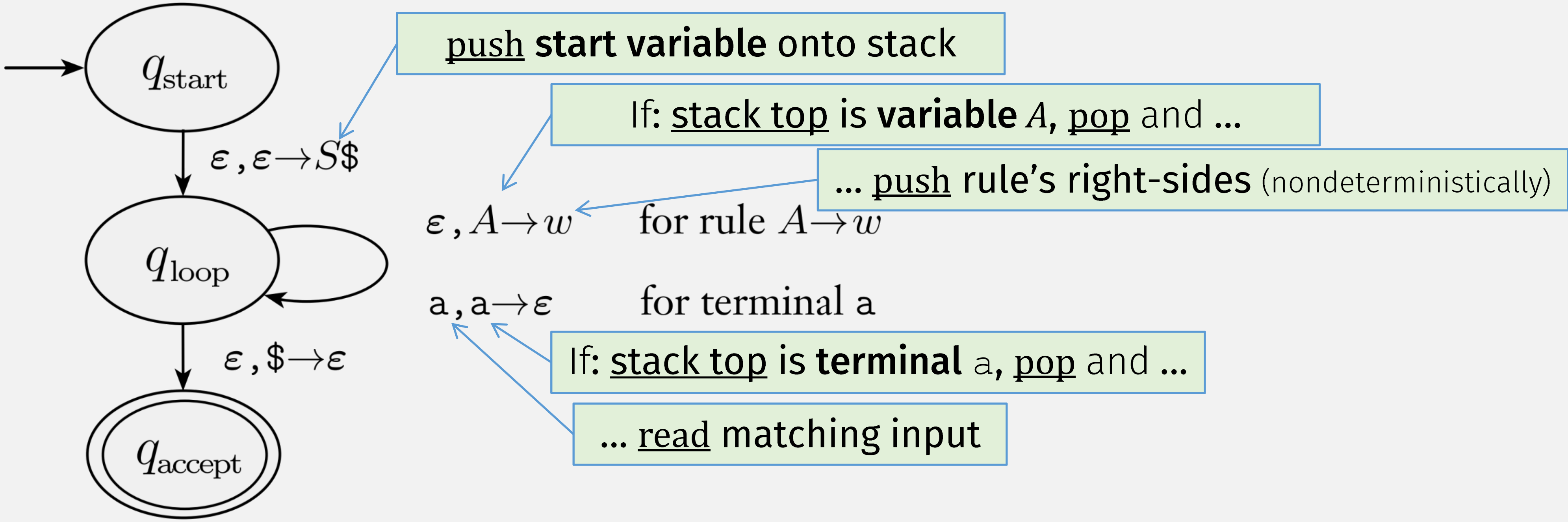
$\mathtt{a}, \mathtt{a} \to \varepsilon$ for terminal a

Convert: **every terminal** to **"loop" transition** that:
- Reads input char
- Pops matching char on stack

(Read the terminals as they become known)

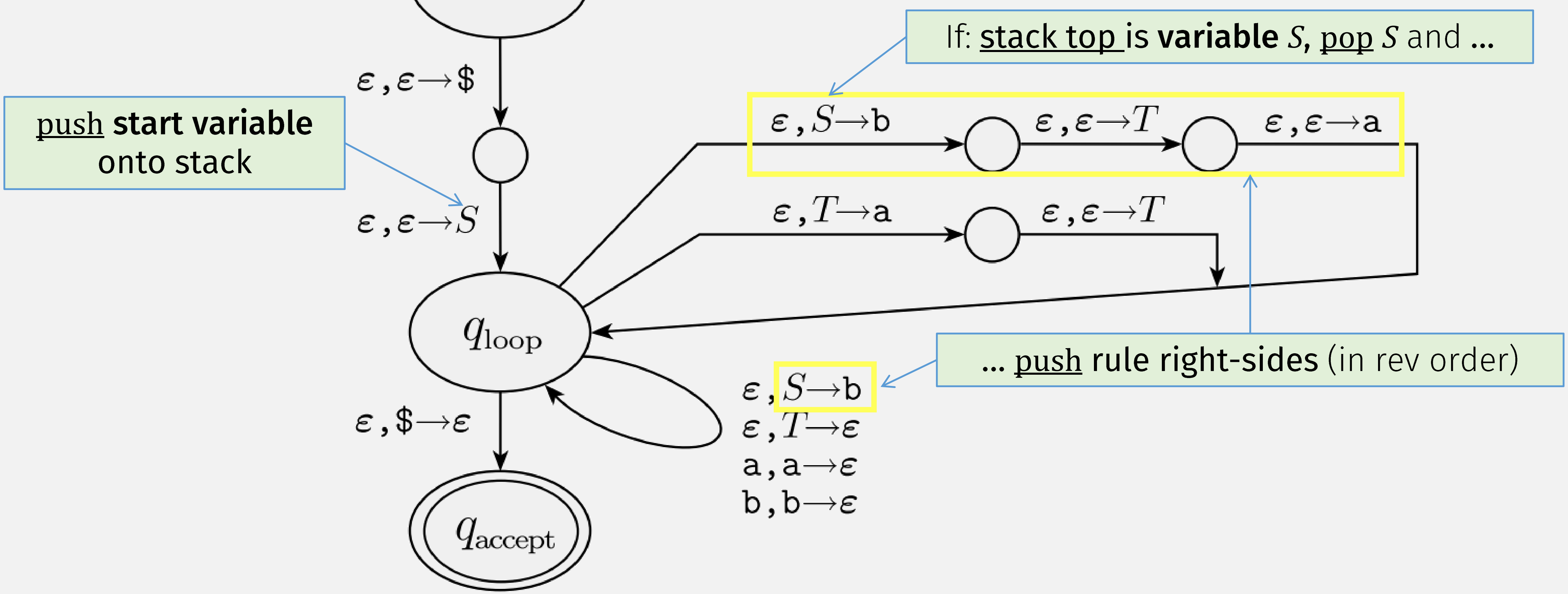
CFG→PDA (sketch)

- **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



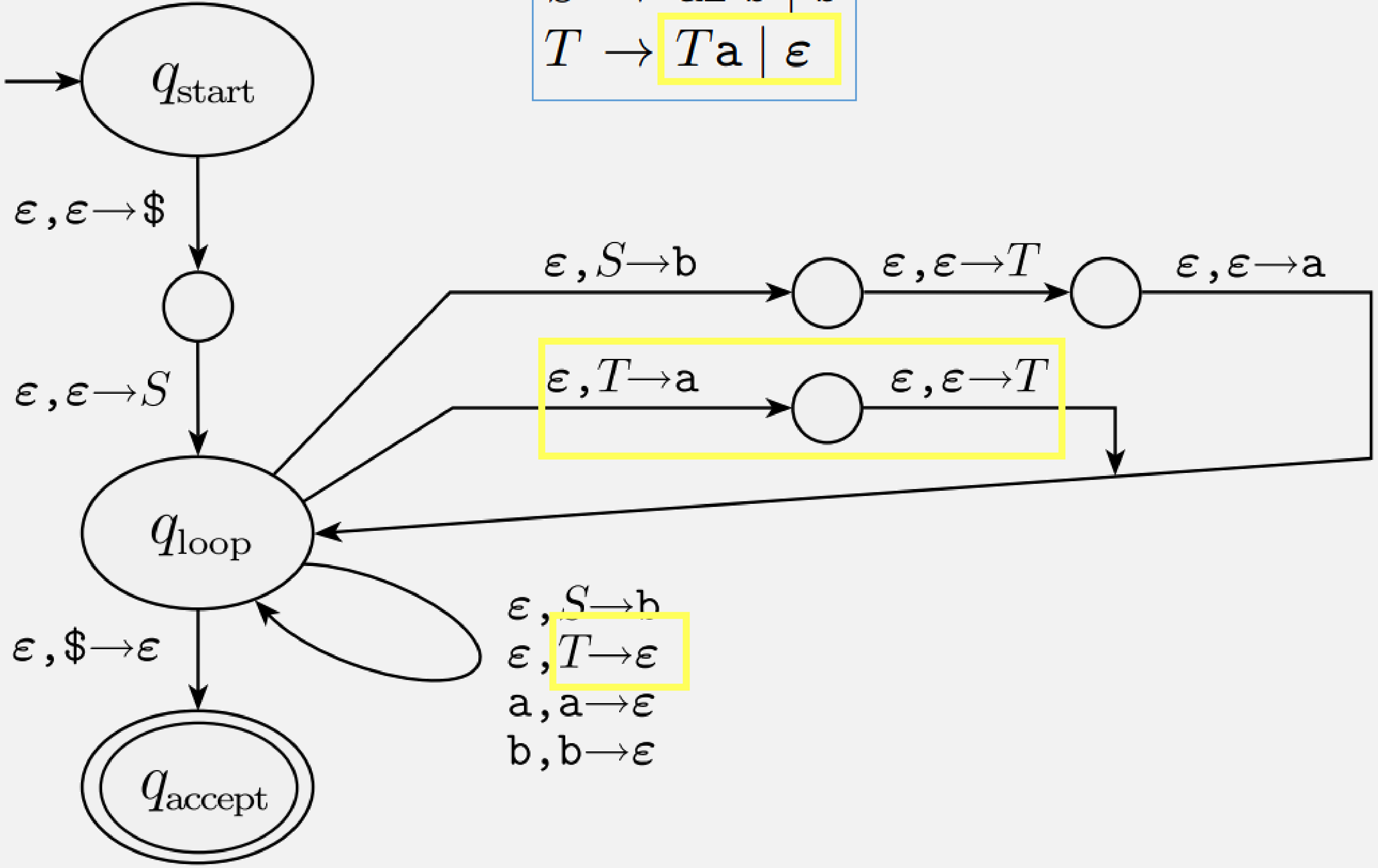
Example **CFG→PDA**

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



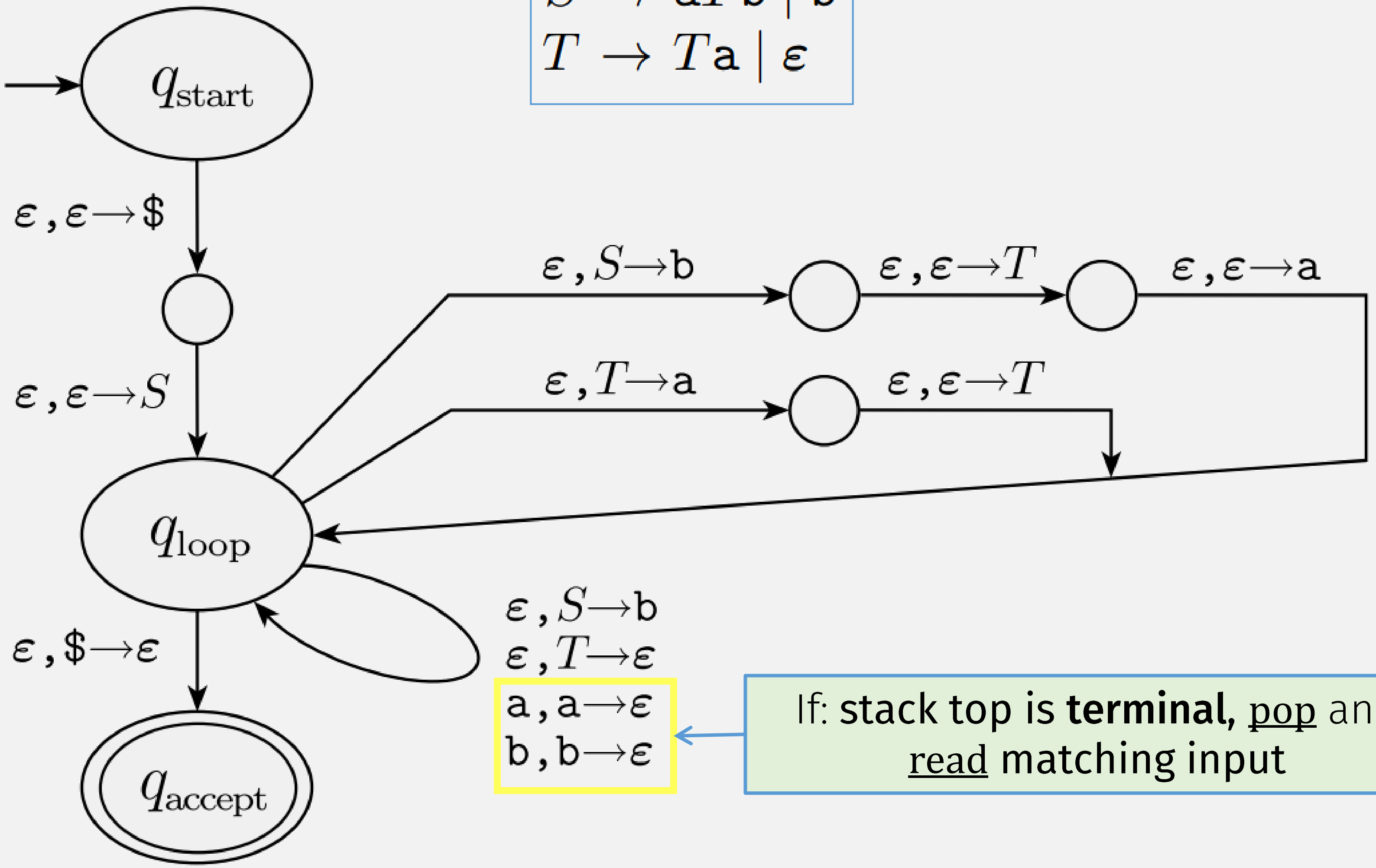
Example **CFG→PDA**

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



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:

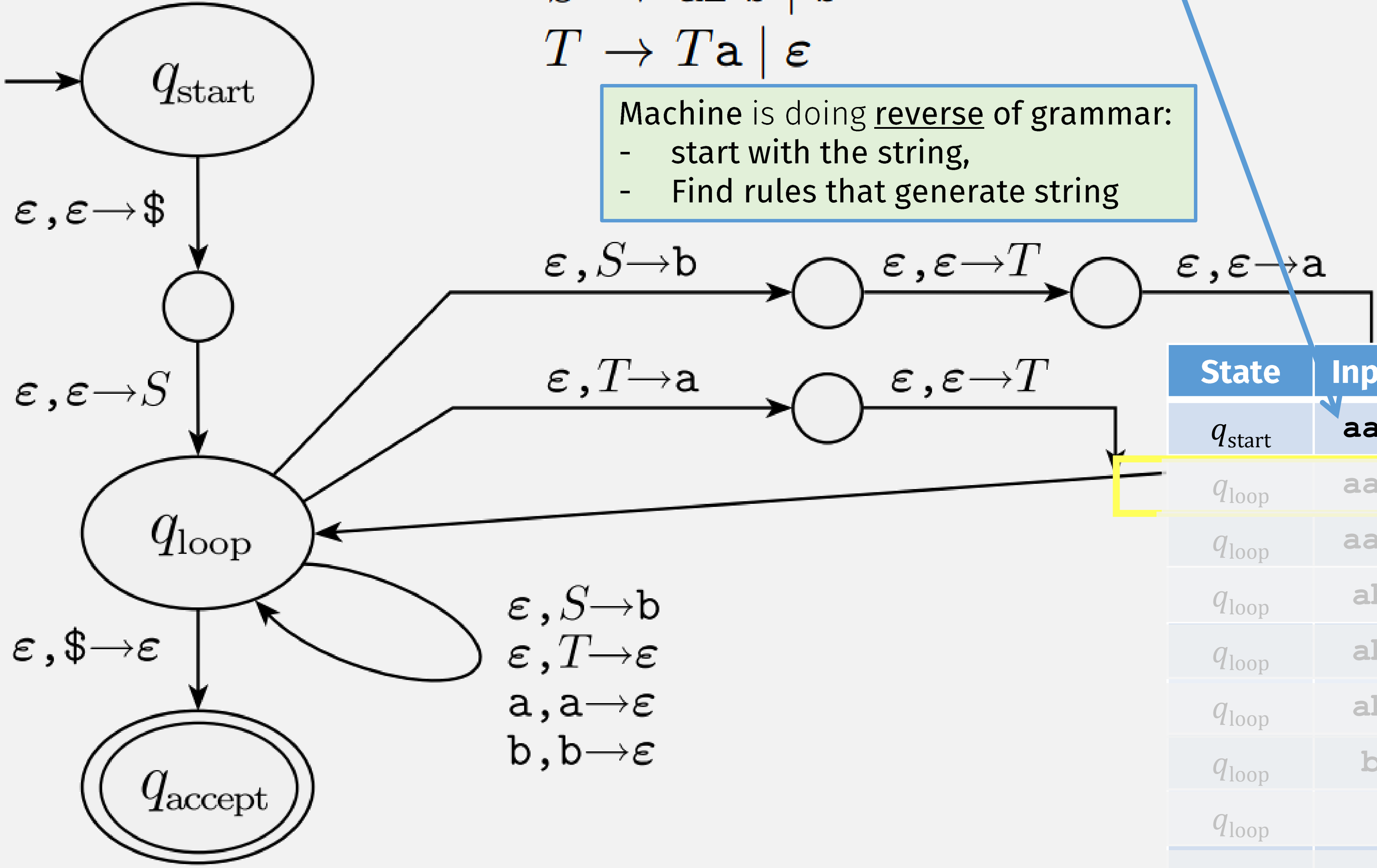
$S \Rightarrow \mathtt{a}T\mathtt{b}$ (using rule $S \rightarrow \mathtt{a}T\mathtt{b}$)

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

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

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

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

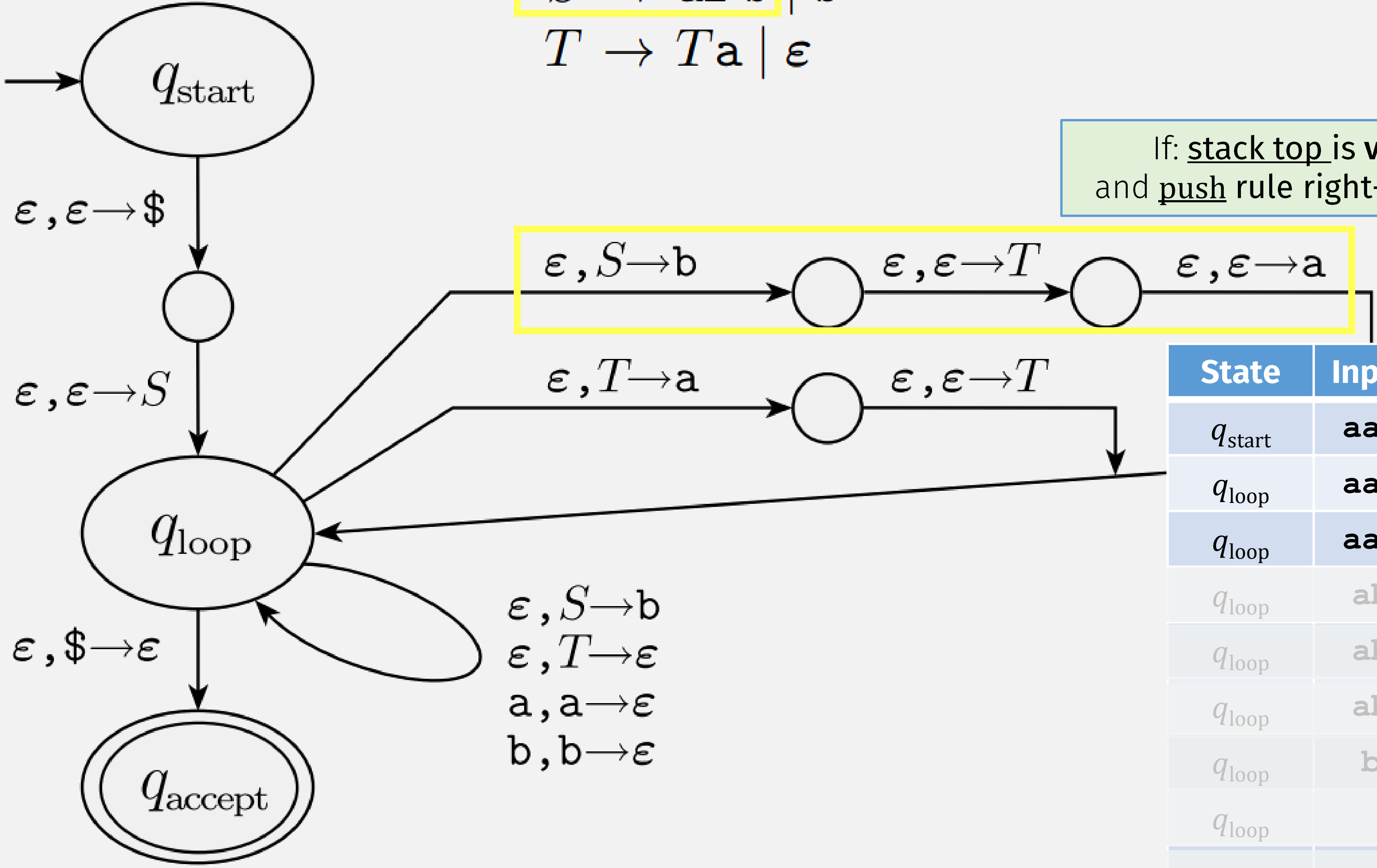
Example Derivation using CFG:

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

$\Rightarrow \mathtt{a}T\mathtt{ab}$ (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**

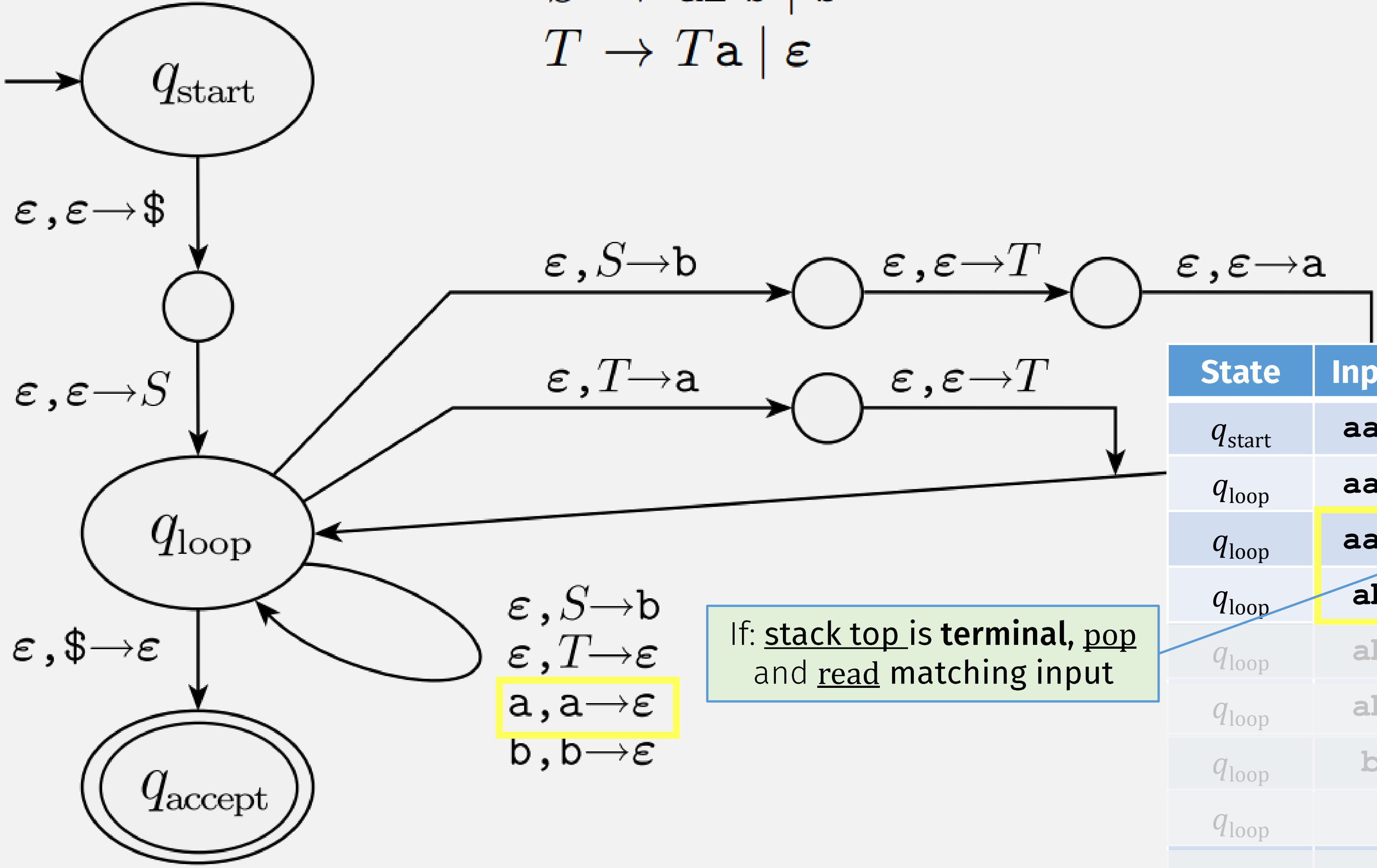
Example Derivation using CFG:

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

$\Rightarrow \boxed{\mathbf{a}}T\mathbf{ab}$ (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 **terminal**, pop and read matching input

PDA Example

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

Example **CFG→PDA**

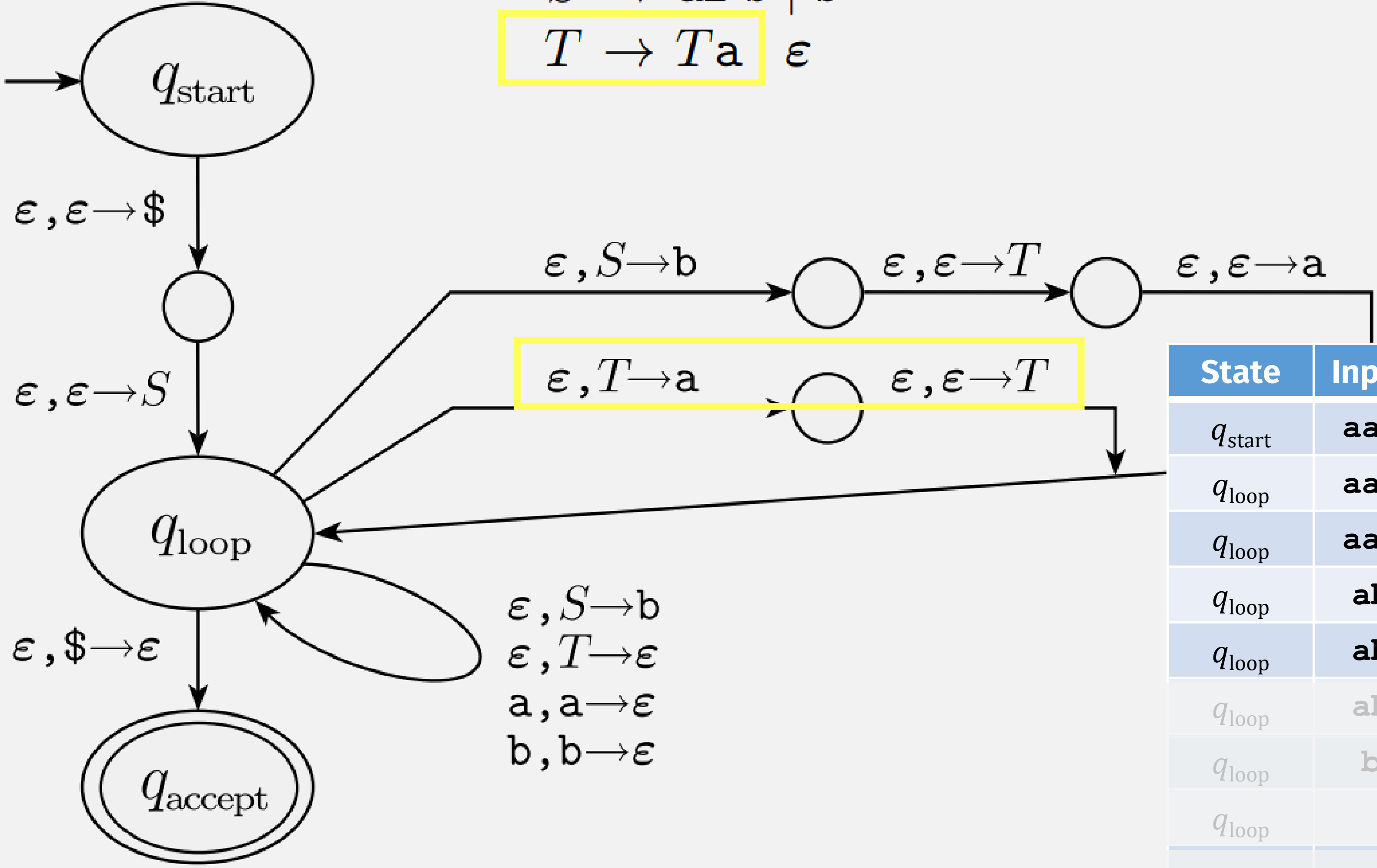
Example Derivation using CFG:

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

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



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	ab$	$T \rightarrow \varepsilon$
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

- To prove this part: show PDA has an equivalent CFG

PDA→CFG: Prelims

Before converting PDA to CFG, modify it so :

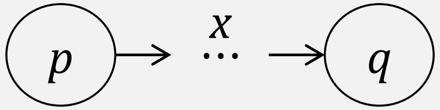
1. It has a single accept state, q_{accept}.
2. It empties its stack before accepting.
3. Each transition either pushes a symbol onto the stack (a *push* move) or pops one off the stack (a *pop* move), but it does not do both at the same time.

Important:
This doesn't change the language recognized by the PDA

PDA P -> CFG G: Transitions and 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)

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

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

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

A language is a CFL ⇔ A PDA recognizes it

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

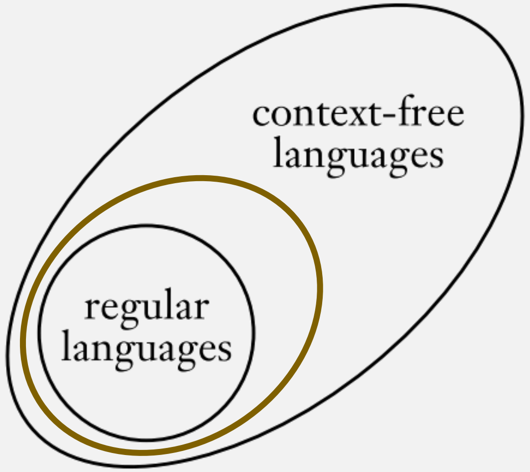
- Convert CFG→PDA

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

- Convert PDA→CFG

■

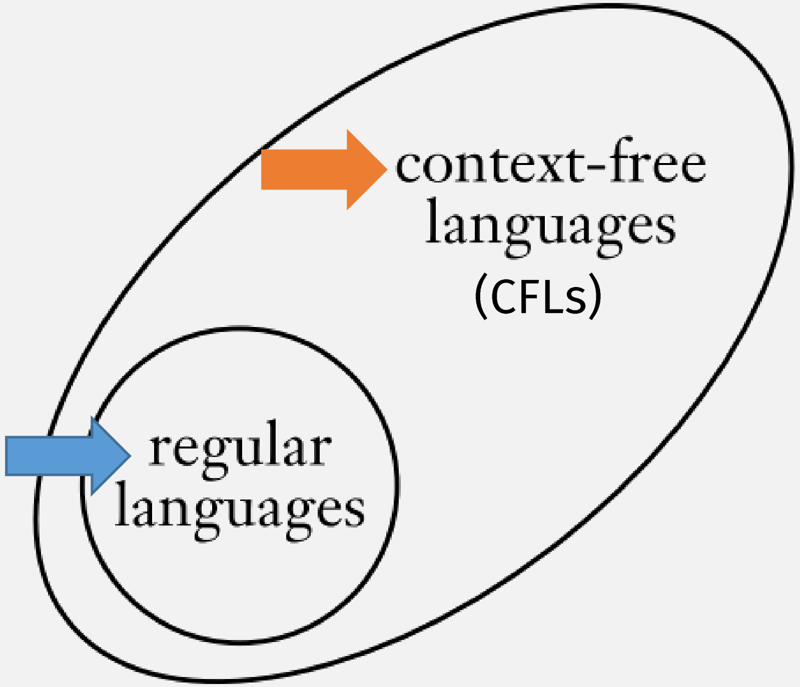
Regular vs Context-Free Languages
(and others?)



Is This Diagram "Correct"?

(What are the statements implied by this diagram?)

1. Every regular language is a CFL
2. Not every CFL is a regular language



How to Prove This Diagram "Correct"?

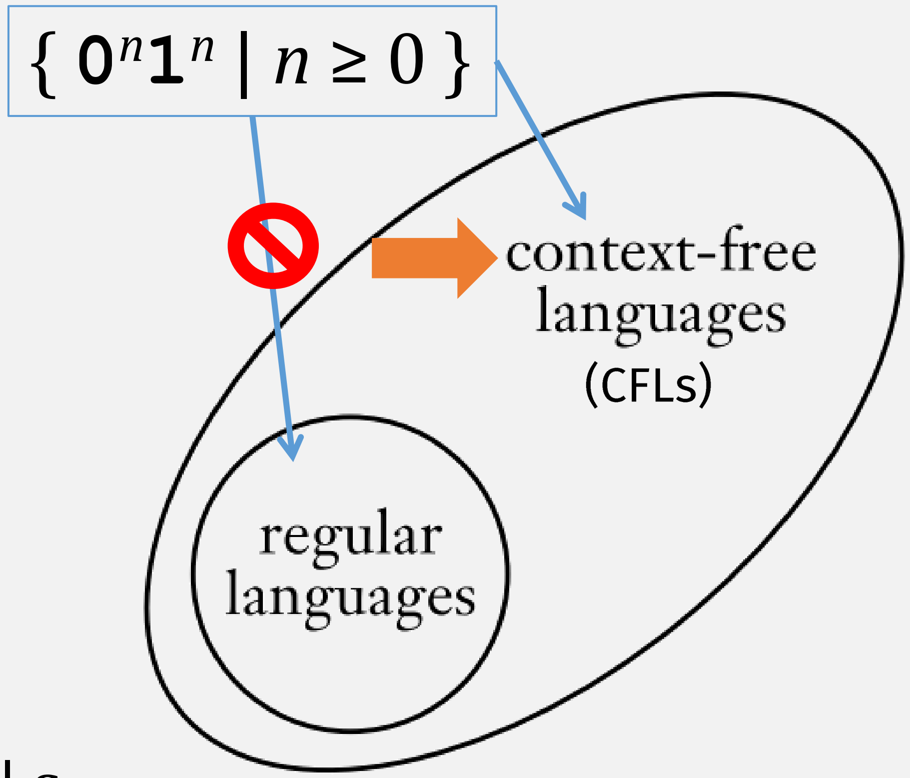
1. Every regular language is a CFL

2. Not every CFL is a regular language

Find a CFL that is not regular

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

- It's a CFL
  - *Proof*: CFG $S \to \mathbf{0}S\mathbf{1} \mid \varepsilon$
- It's not regular
  - *Proof*: by contradiction using the Pumping Lemma



How to Prove This Diagram "Correct"?

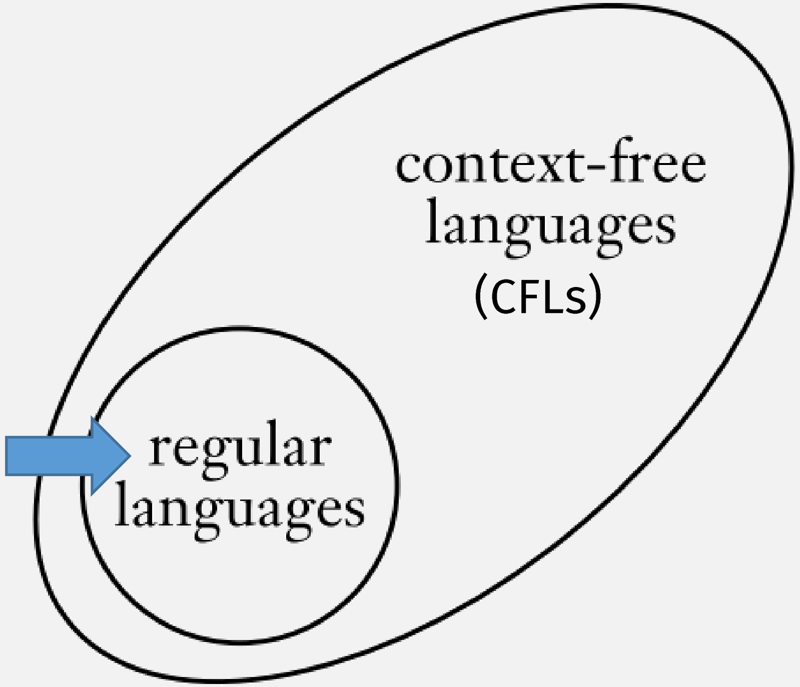
1. Every regular language is a CFL

For any regular language A, show …

… it has a CFG or PDA

☑ 2. Not every CFL is a regular language

A regular language is represented by a:
- DFA
- NFA
- Regular Expression



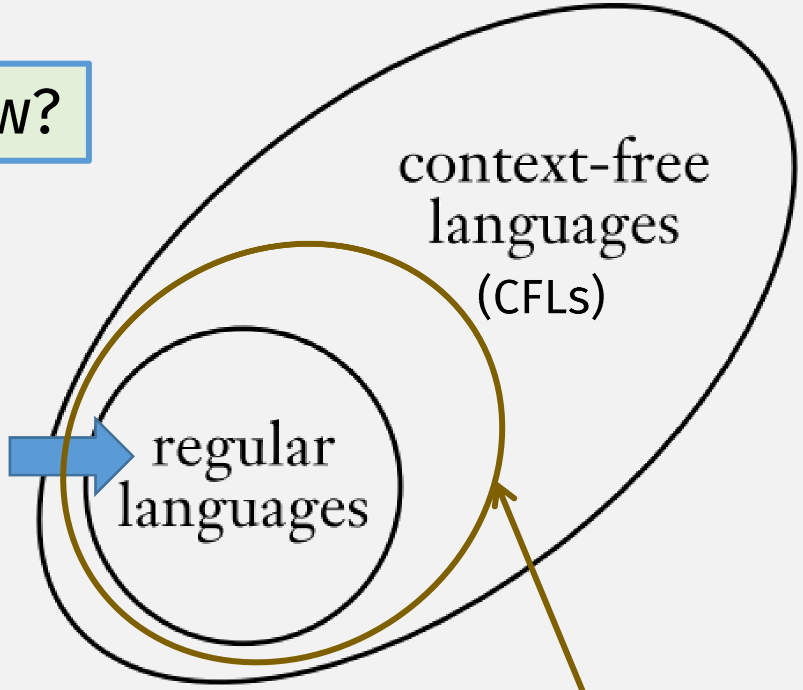
Regular Languages are CFLs: 3 Ways to Prove

- DFA → CFG or PDA

Coming soon to a future hw?

- NFA → CFG or PDA

- Regular expression → CFG or PDA



Are there other interesting <u>subsets of CFLs</u>?