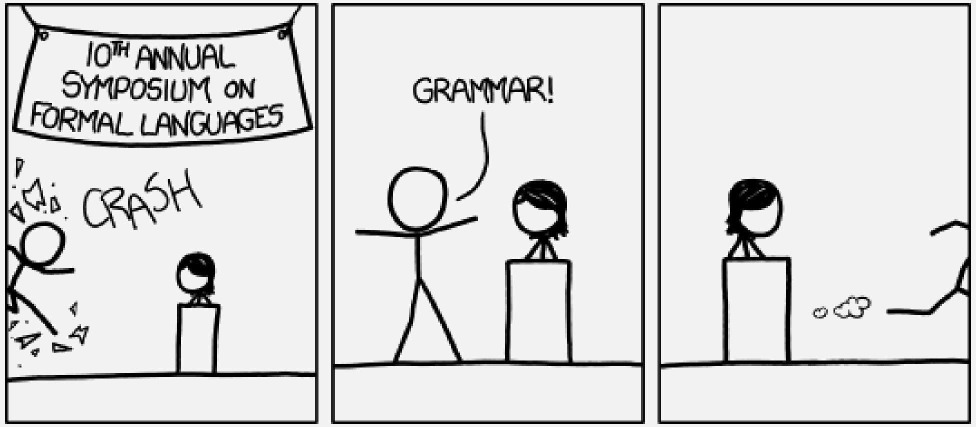
UMB CS 622

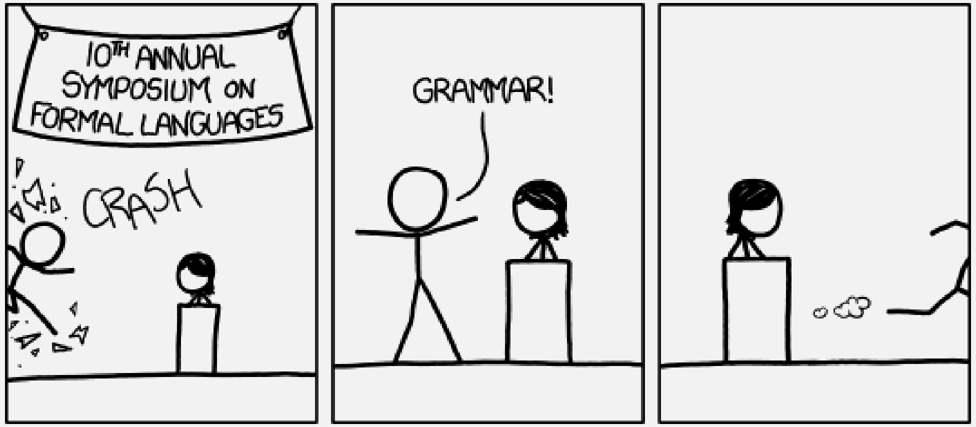
Pushdown Automata (PDAs)

Wednesday, March 20, 2024



Announcements

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



Last Time:

Context-Free Grammar (CFG)

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

terminals

Top variable is:
Start variable

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

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

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

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 <u>applying</u> 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:

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

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

Where:

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

$A \in V$ ← Variable

$A \rightarrow \gamma \in R$ ← 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

$A \rightarrow \gamma \in R$ ← Rule

Multi-step (recursively defined)

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

Recursive case: (> 0 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 (i.e., **strings**) …"

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

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

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

If a **CFG** <u>generates</u> all <u>strings</u> in a **language** L**,** then L is a **context-free language** (CFL)

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

2. **Start with small grammars** and **then combine**
   - just like with FSMs, and programming!

Designing Grammars: Building Up

- Start with small grammars and then combine (just like FSMs)
  - 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 $\{0^n 1^n | \, n \geq 0\}$:

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

  - Then create grammar for lang $\{1^n 0^n | \, n \geq 0\}$:

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

  - Then combine:

    $$S \rightarrow S_1 \mid S_2$$
    $$S_1 \rightarrow 0S_1 1 \mid \varepsilon$$
    $$S_2 \rightarrow 1S_2 0 \mid \varepsilon$$

New start variable and rule combines two smaller grammars

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

(Closed) Operations for CFLs?

- Start with small grammars and then combine (just like FSMs)
- "Or": $S \to S_1 \mid S_2$
- "Concatenate": $S \longrightarrow S_1 S_2$
- "Repetition": $S' \longrightarrow S' S_1 \mid \varepsilon$

Could you write out the full proof of these closure operations?

Example: Creating CFG

alphabet Σ is $\{0,1\}$

$$\{w \mid w \text{ starts and ends with the same symbol}\}$$

1) come up with examples: In the language: **010, 101, 11011** 1, 0 ? ☑
Not in the language: **10, 01, 110** ε ? ☒

2) Create CFG:

Needed Rules:

$S \rightarrow 0M0 \mid 1M1 \mid 0 \mid 1$ "start/end symbol are "linked" (ie, same); middle can be anything"

$M \rightarrow MT \mid \varepsilon$ "middle: all possible terminals, repeated (ie, all possible strings)"

$T \rightarrow 0 \mid 1$ "all possible terminals"

3) Check CFG: generates examples in the language; does not generate examples not in language

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

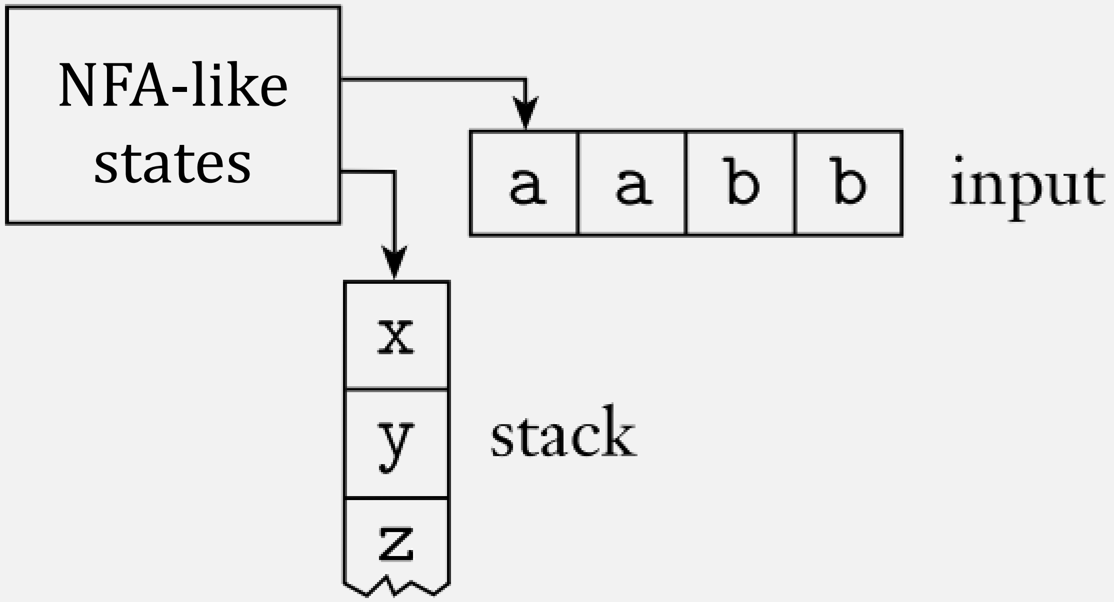
	Regular Languages	Context-Free Languages (CFLs)	
thm	Regular Expression	Context-Free Grammar (CFG)	def
	describes a Regular Lang	describes a CFL	
	Finite State Automaton (FSM)	**Push-down Automata** (PDA)	thm
def	recognizes a Regular Lang	recognizes a CFL	

Regular Language vs CFL Comparison

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

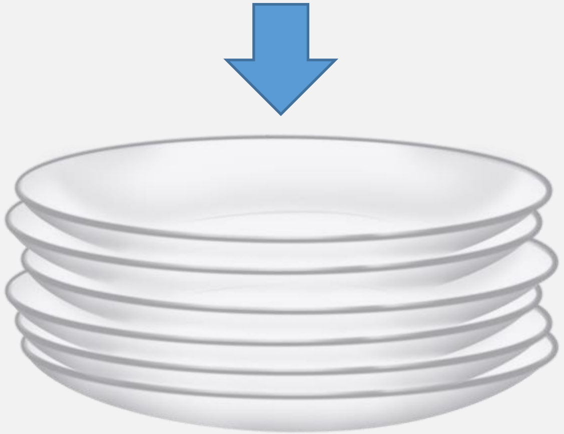
Pushdown Automata (PDA)

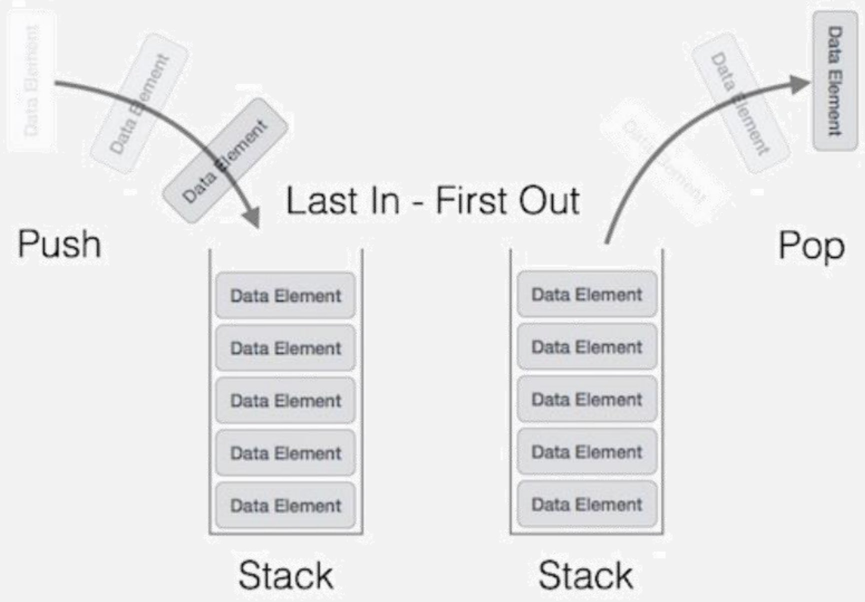
PDA = NFA + a <u>stack</u>



What is a Stack?

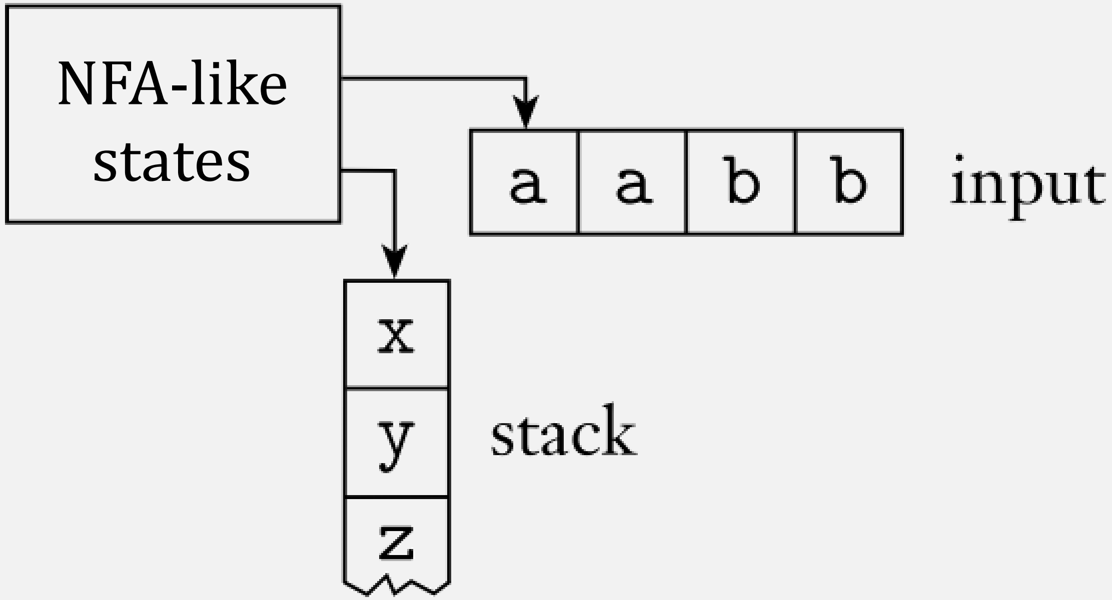
- A <u>**restricted**</u> kind of (**infinite**!) **memory**
- **Access** to **top element only**
- 2 Operations only: **push, pop**





Pushdown Automata (PDA)

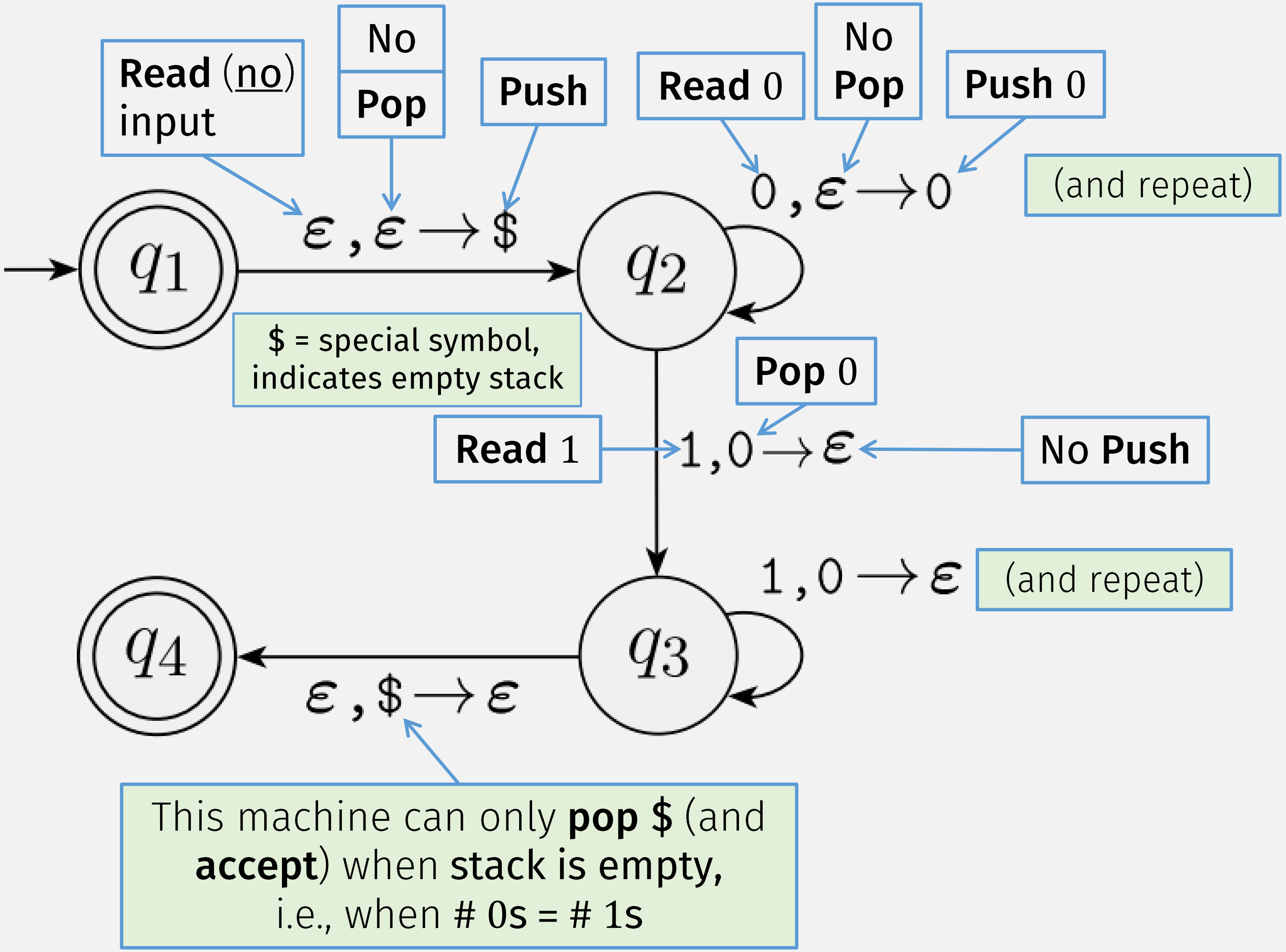
- **PDA** = NFA + a **stack**
  - Infinite memory
  - read/write top location only
    - Push/pop



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

An Example PDA

A PDA transition has <u>3 parts:</u>
- **Read**
- **Pop**
- **Push**



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,

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

4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ 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

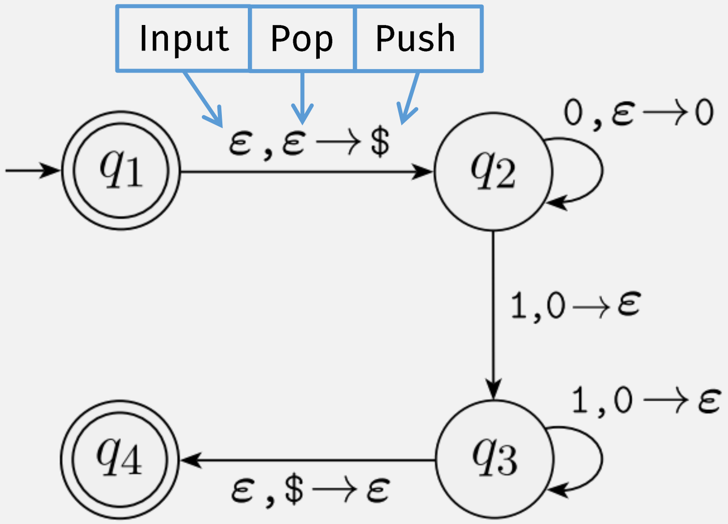
$$Q = \{q_1, q_2, q_3, q_4\},$$

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

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

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

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

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

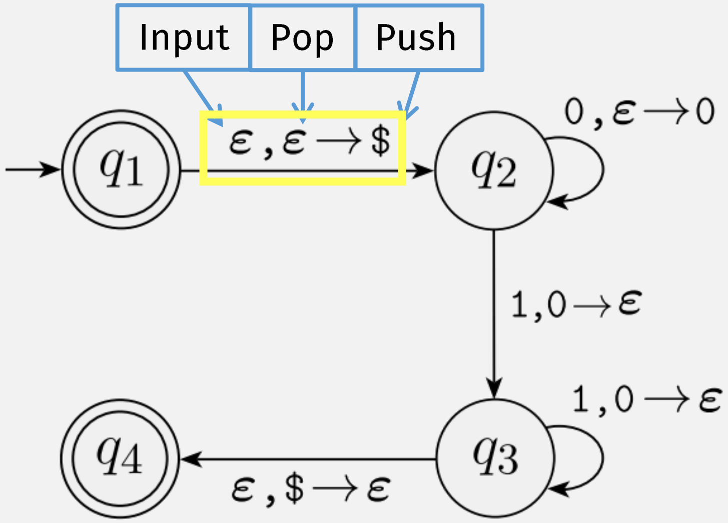
$\Gamma = \{0, \$\}$,

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

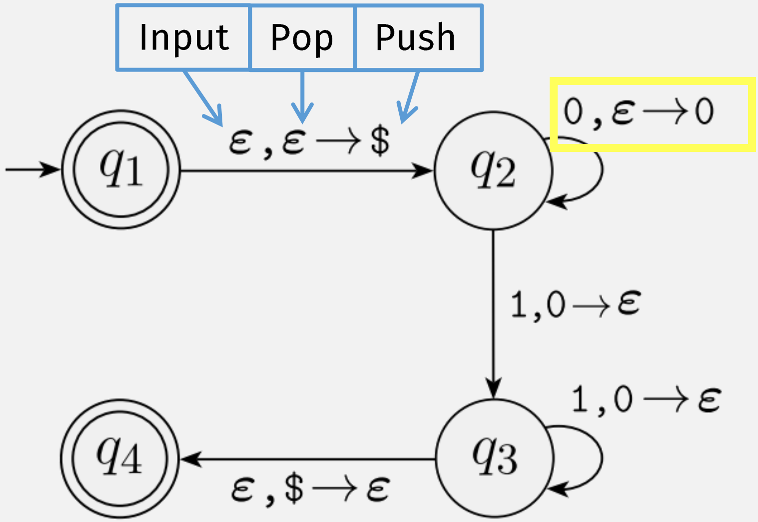
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

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

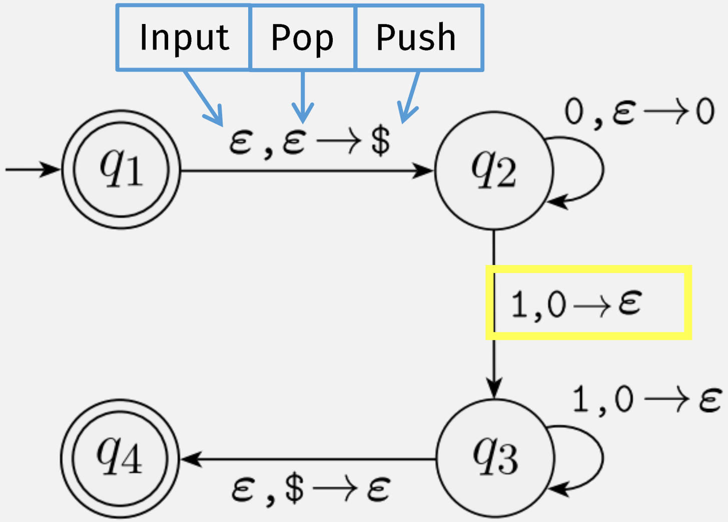
$\Sigma = \{0,1\}$,

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

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

Input Pop Push

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

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

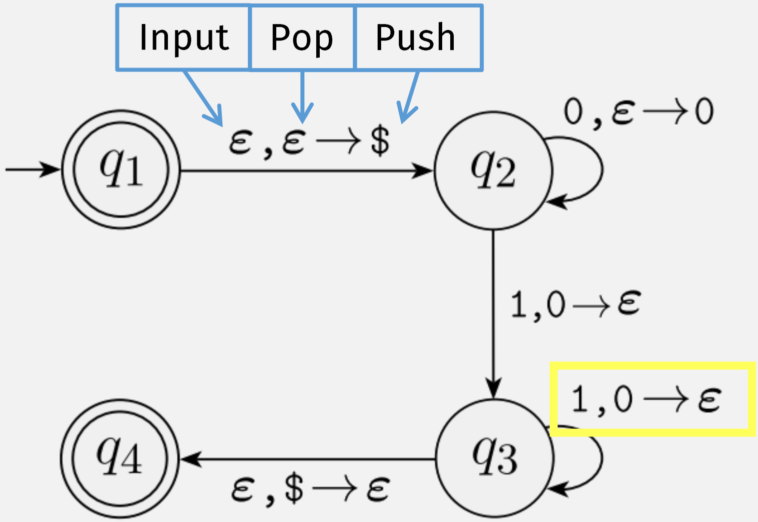
$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, 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				$\boxed{\{(q_3, \varepsilon)\}}$ **3**				$\{(q_4, \varepsilon)\}$	
q_4									

Input Pop Push



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

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

Input Pop Push

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

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

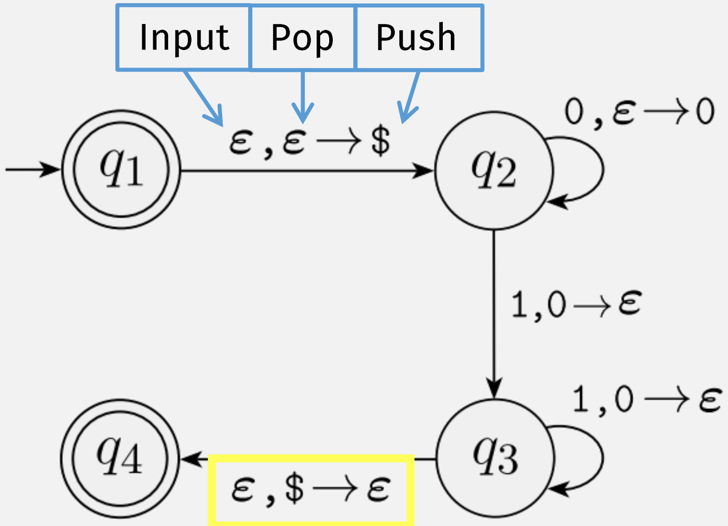
$\Gamma = \{0, \$\}$,

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

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

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

Input Pop Push



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

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

Input Pop Push

In-class exercise: Fill in the blanks

$Q =$

$\Sigma =$

$\Gamma =$

$F =$

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

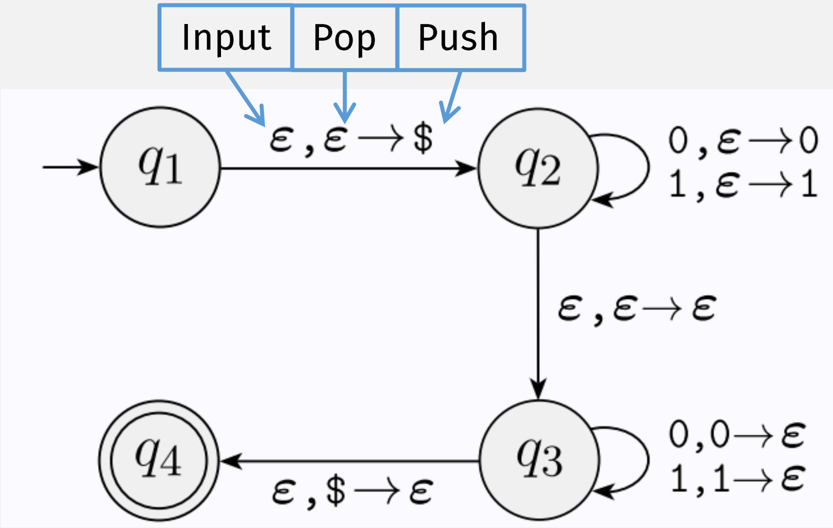
Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε

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

<u>In-class exercise:</u> Fill in the blanks

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

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

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

$F = \{q_4\}$

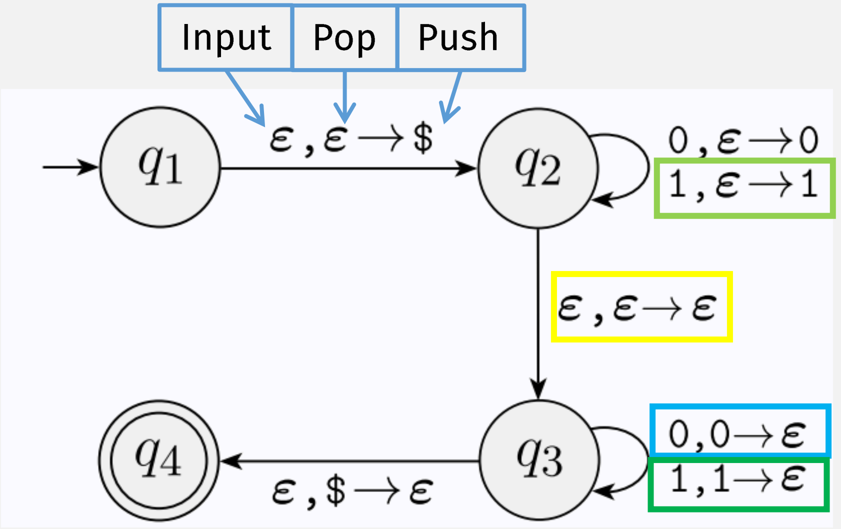
δ is given by the following table, wherein blank entries signify $\emptyset$.

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	*Formally (i.e., mathematically)*

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

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

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 $w = w_1 w_2 \cdots w_n$ 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)

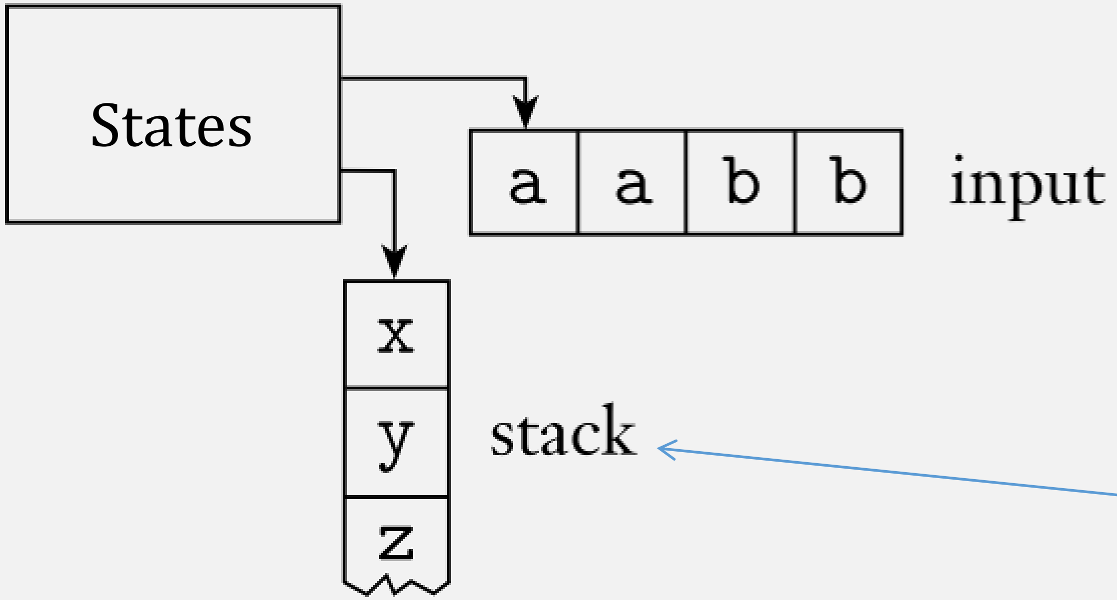
Base case $\hat{\delta}(q, \varepsilon) = q$

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

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> …

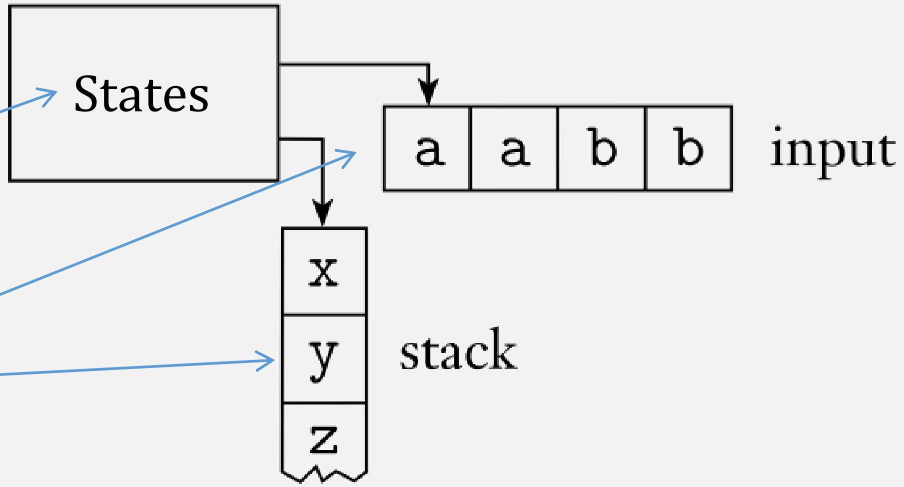
A PDA **computation** is a <u>not</u> just a <u>sequence of states</u> …

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

PDA Configurations (IDs)

- A **configuration** (or **ID**) is a "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**

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

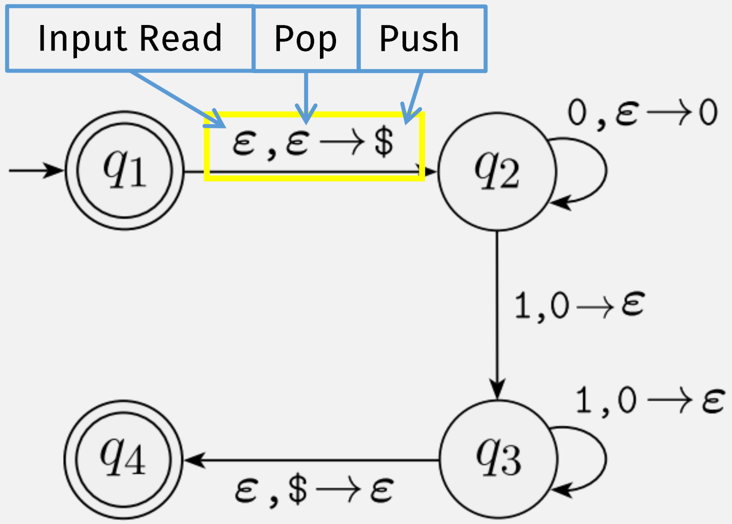
Recursive "call"

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

PDA Running Input String Example

State | Remaining Input | Stack

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



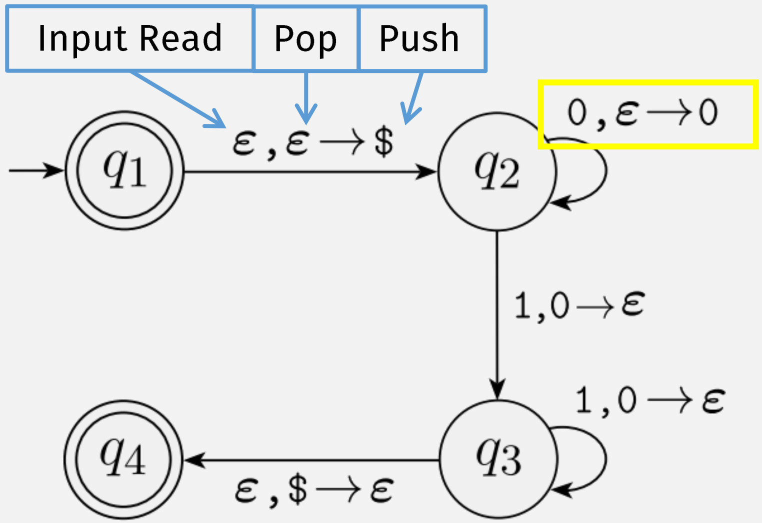
PDA Running Input String Example

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

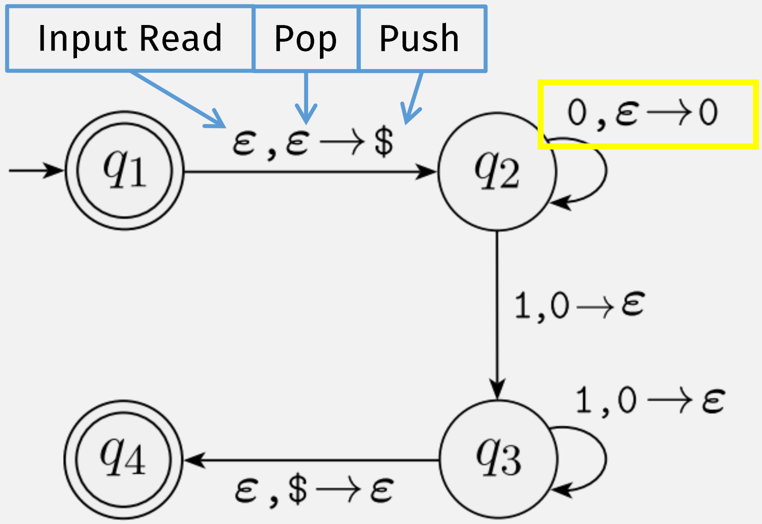
State | Remaining Input | Stack

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

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

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

Read 0, push 0

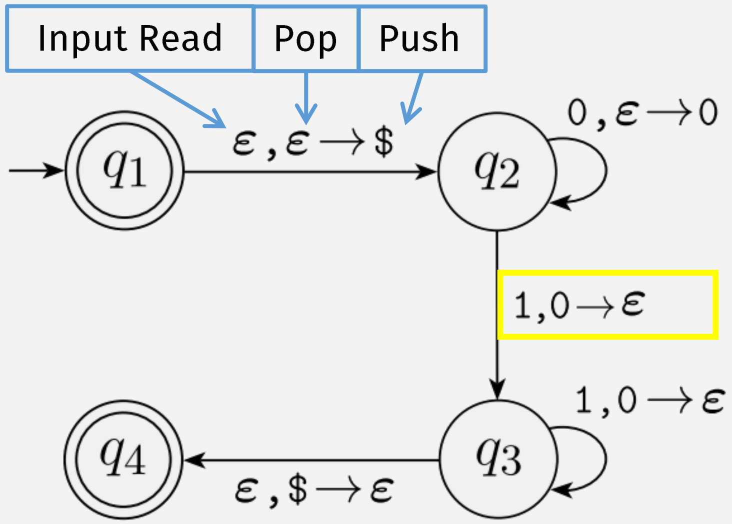


PDA Running Input String Example

State | Remaining Input | Stack

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

Read 1, pop 0

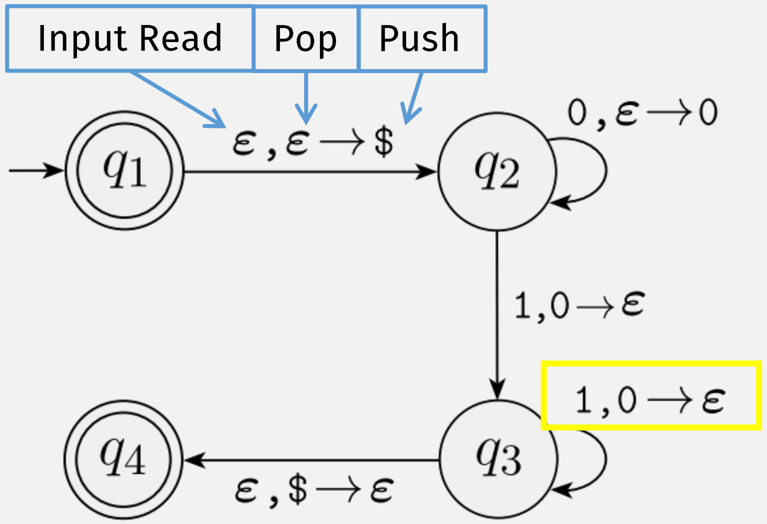


PDA Running Input String Example

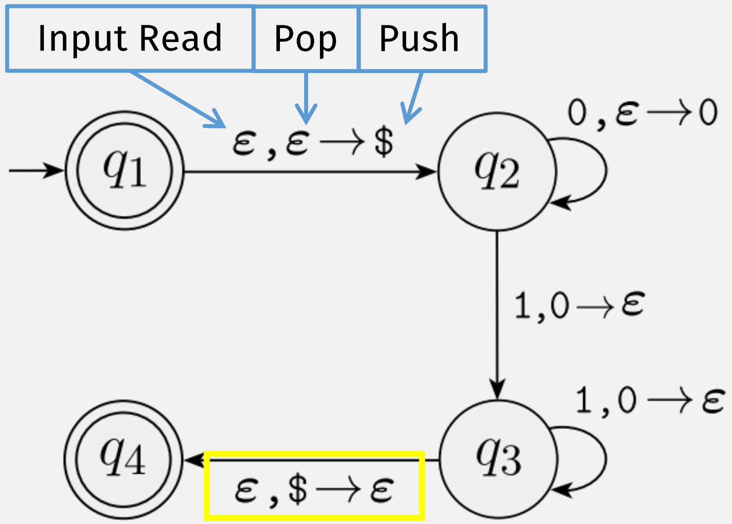
State | Remaining Input | Stack

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

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 | 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

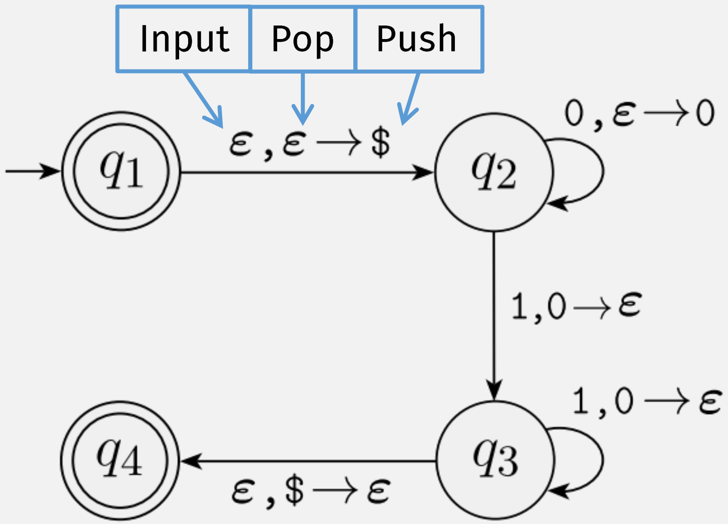
A **configuration** (q, w, γ) has three components

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** $\boxed{\text{CFL}}$, 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**

Submit in-class work 3/20

On gradescope