Pushdown Automata (PDAs)

Monday, October 7, 2020

HW3 Questions?

HW4 out

- HW4 due in 2 weeks
- HW3 due Sunday 11:59pm EST

<u>Last time</u>: Designing Grammars

- 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": $S' \longrightarrow S' S_1 \mid \varepsilon$

In-class exercise: Designing grammars

alphabet Σ is $\{0,1\}$

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

- S -> 0C'0 | 1C'1 | ε
- C' -> C'C | ε
- C -> 0 | 1

"string starts/ends with same symbol, middle can be anything"

"all possible terminals, repeated (ie, all possible strings)"

"all possible terminals"

Analogies

Regular Language	Context-Free Language (CFL)
Regular Expression (Regexp)	Context-Free Grammar (CFG)
Regexp describes a Reg lang	CFG describes/generates a CFL

Analogies

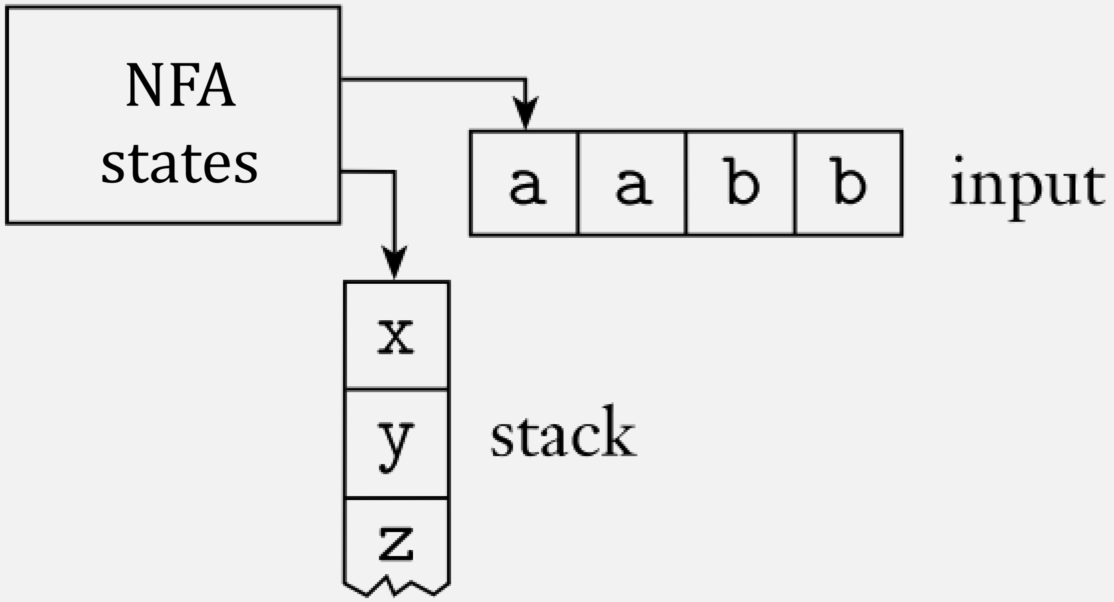
Regular Language	Context-Free Language (CFL)
Regular Expression (Regexp)	Context-Free Grammar (CFG)
Regexp <u>describes</u> a Reg lang	CFG <u>describes</u>/<u>generates</u> a CFL
	TODAY:
Finite automaton (FSM)	Push-down automaton (PDA)
FSM <u>recognizes</u> a regular lang	PDA <u>recognizes</u> a CFL

Analogies

Regular Language	Context-Free Language (CFL)
Regular Expression (Regexp)	Context-Free Grammar (CFG)
Regexp describes a Reg lang	CFG describes/generates a CFL
	TODAY:
Finite automaton (FSM)	Push-down automaton (PDA)
FSM recognizes a regular lang	PDA recognizes a CFL
DIFFERENCE:	**DIFFERENCE:**
Regular lang defined via FSM	CFL defined via CFG
Must prove Regexp ⇔ Reg lang	Must prove PDA ⇔ CFL

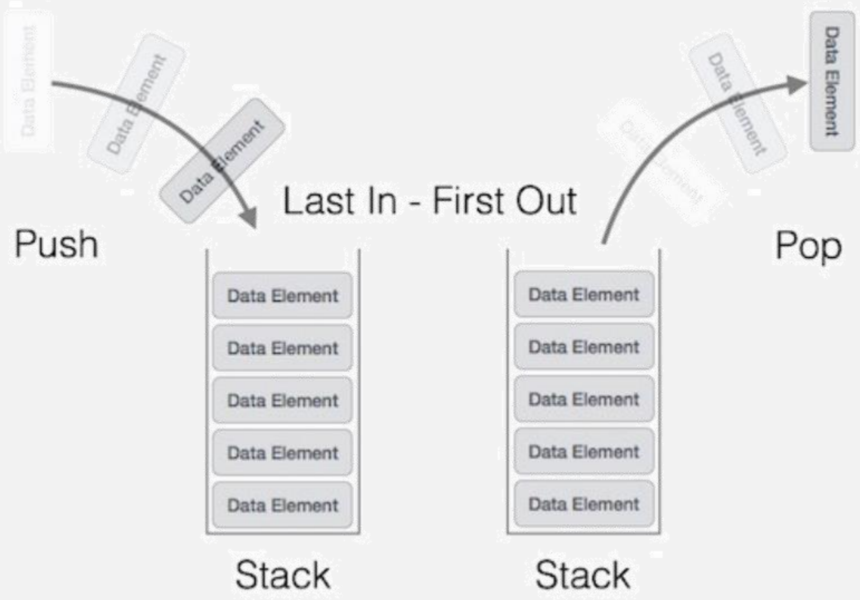
Pushdown Automata (PDA)

- PDA = NFA + a stack



A (Mathematical) Stack Specification

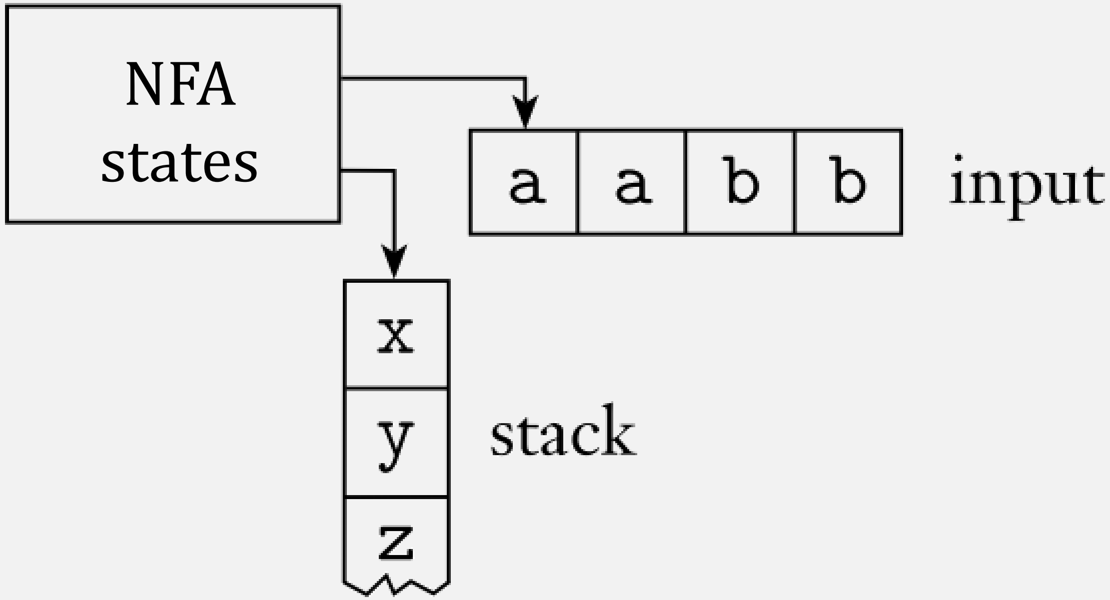
- Access to top element of stack only
- Operations: push, pop



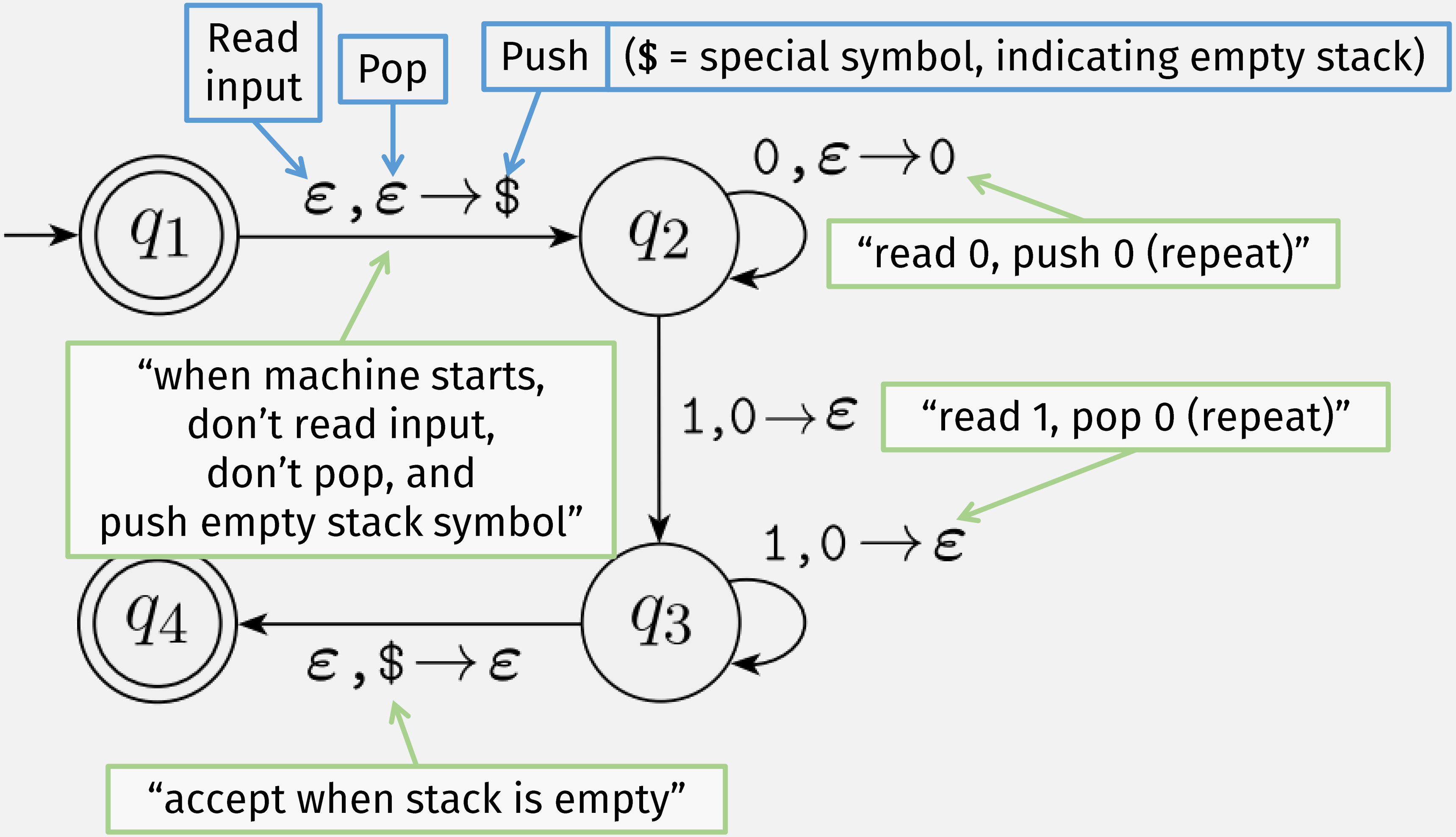
- (What could be a possible code representation?)

Pushdown Automata (PDA)

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



A Example PDA $\{0^n 1^n \mid n \geq 0\}$

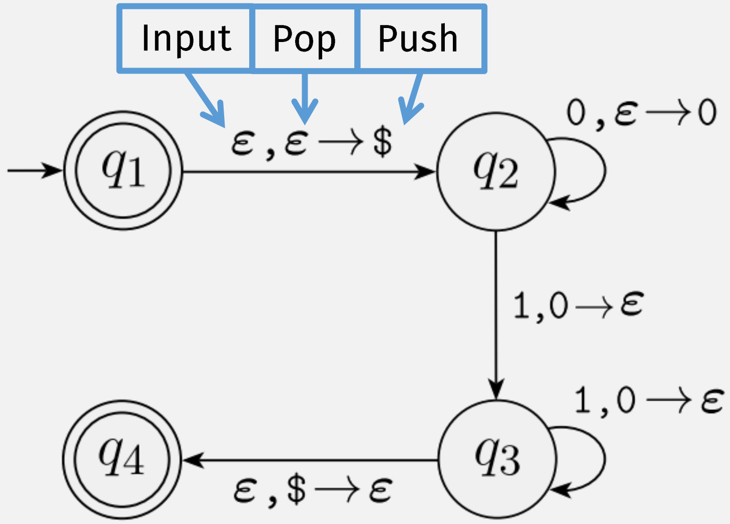


Formal Definition of PDA

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.

In-class example



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

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

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

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

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

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

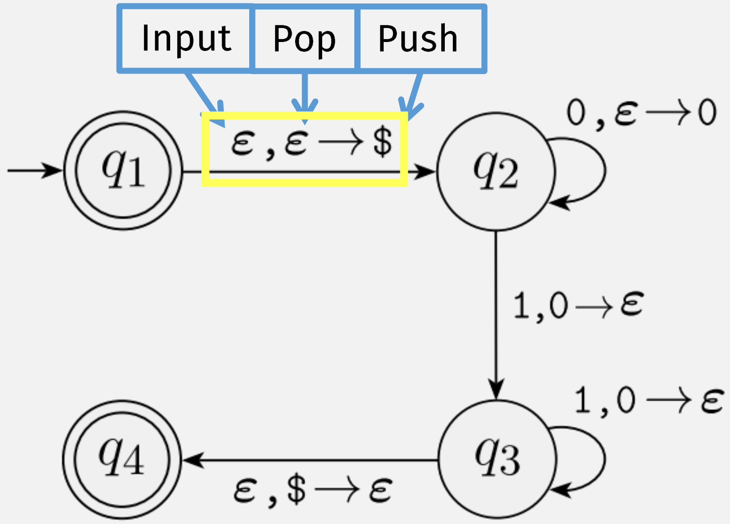
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

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

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

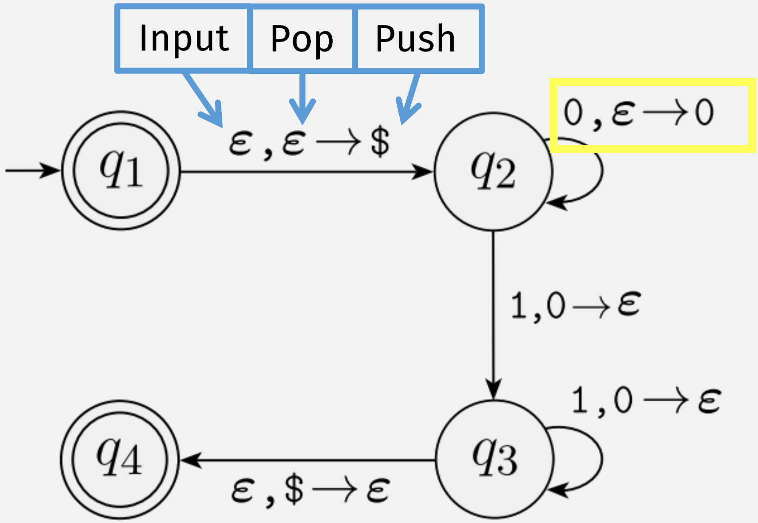
$\Gamma = \{0, \$\}$,

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

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.

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

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

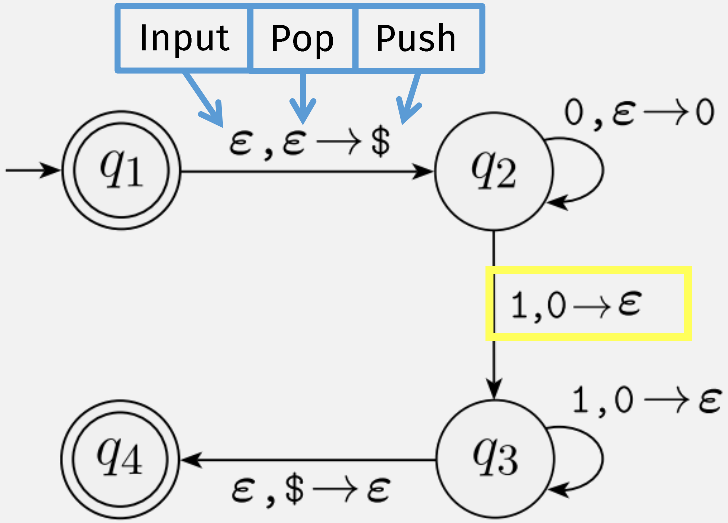
$\Gamma = \{0, \$\}$,

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

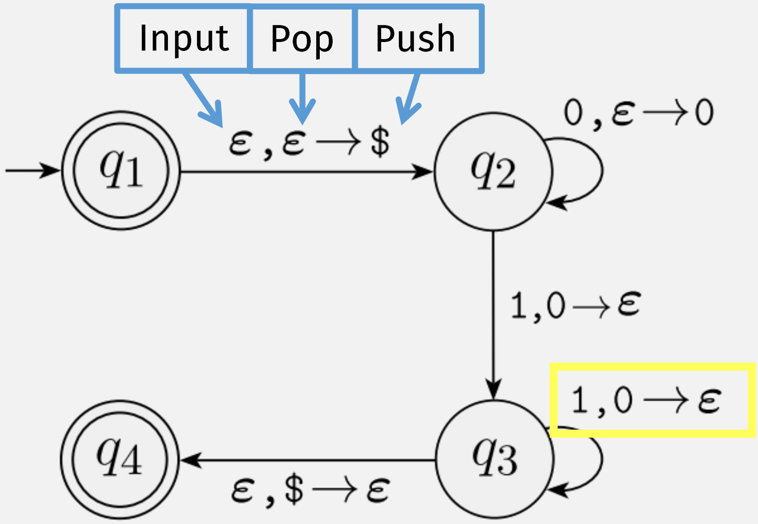
δ is given by the following table, wherein blank entries signify $\emptyset$.

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

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

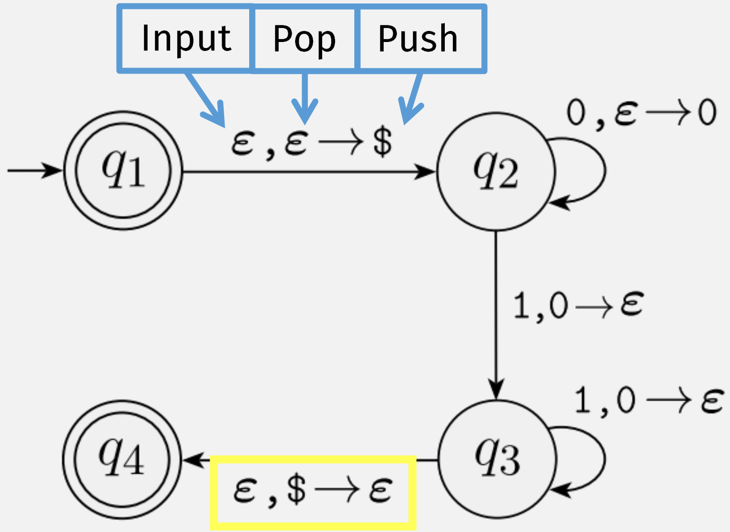
δ is given by the following table, wherein blank entries signify $\emptyset$.

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.

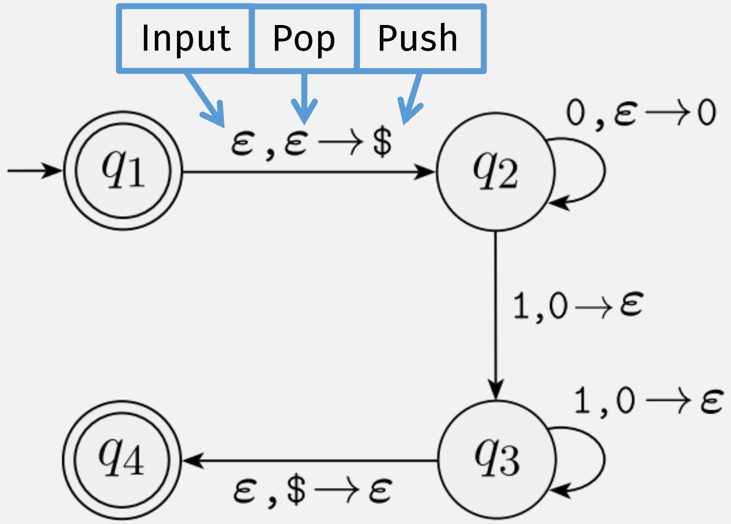
Input

Pop

Push

Pushdown Automata (PDA)

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



- Want to prove: PDA ⇔ CFG

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

CFL $\Leftrightarrow$ PDA

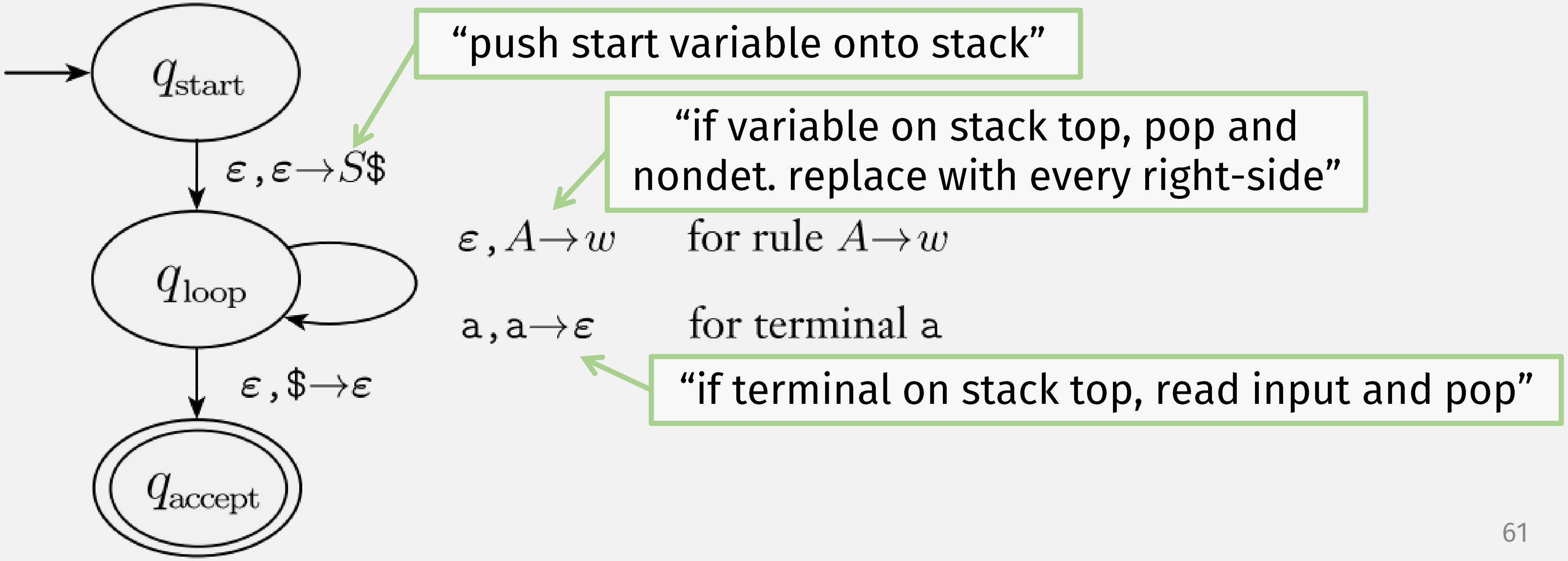
A lang is a CFL iff some PDA recognizes it

- => If CFL, then PDA recognizes it
  - (Easier)
  - All CFLs have CFG describing it (definition of CFL)
  - Convert CFG -> PDA
- <= If PDA recognizes, then CFL

CFG -> PDA

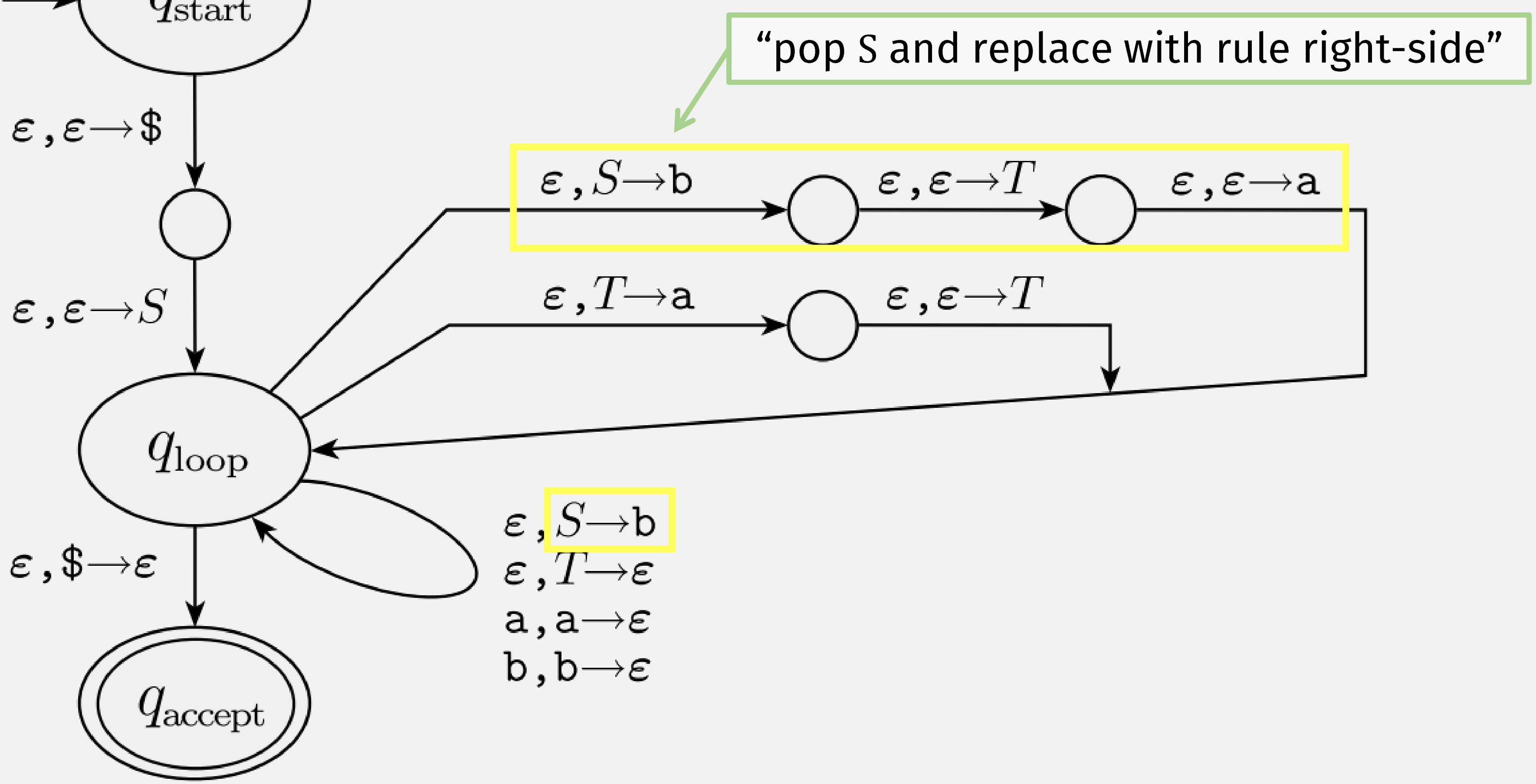
- Construct PDA from CFG such that:
  - PDA accepts input string only if the CFG can generate that string

- Intuitively, PDA <u>nondeterministically</u> tries all rules



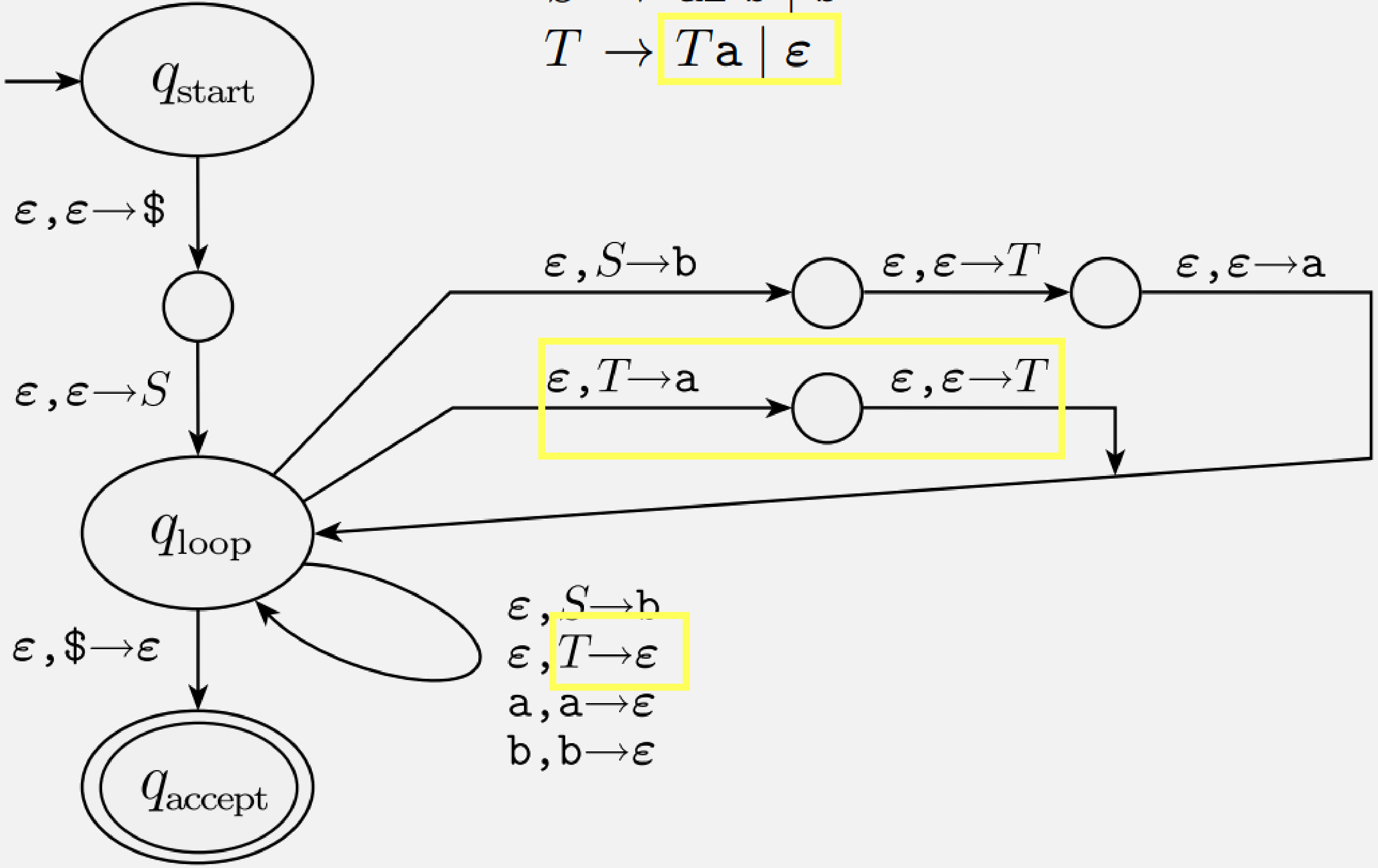
Example CFG -> PDA

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



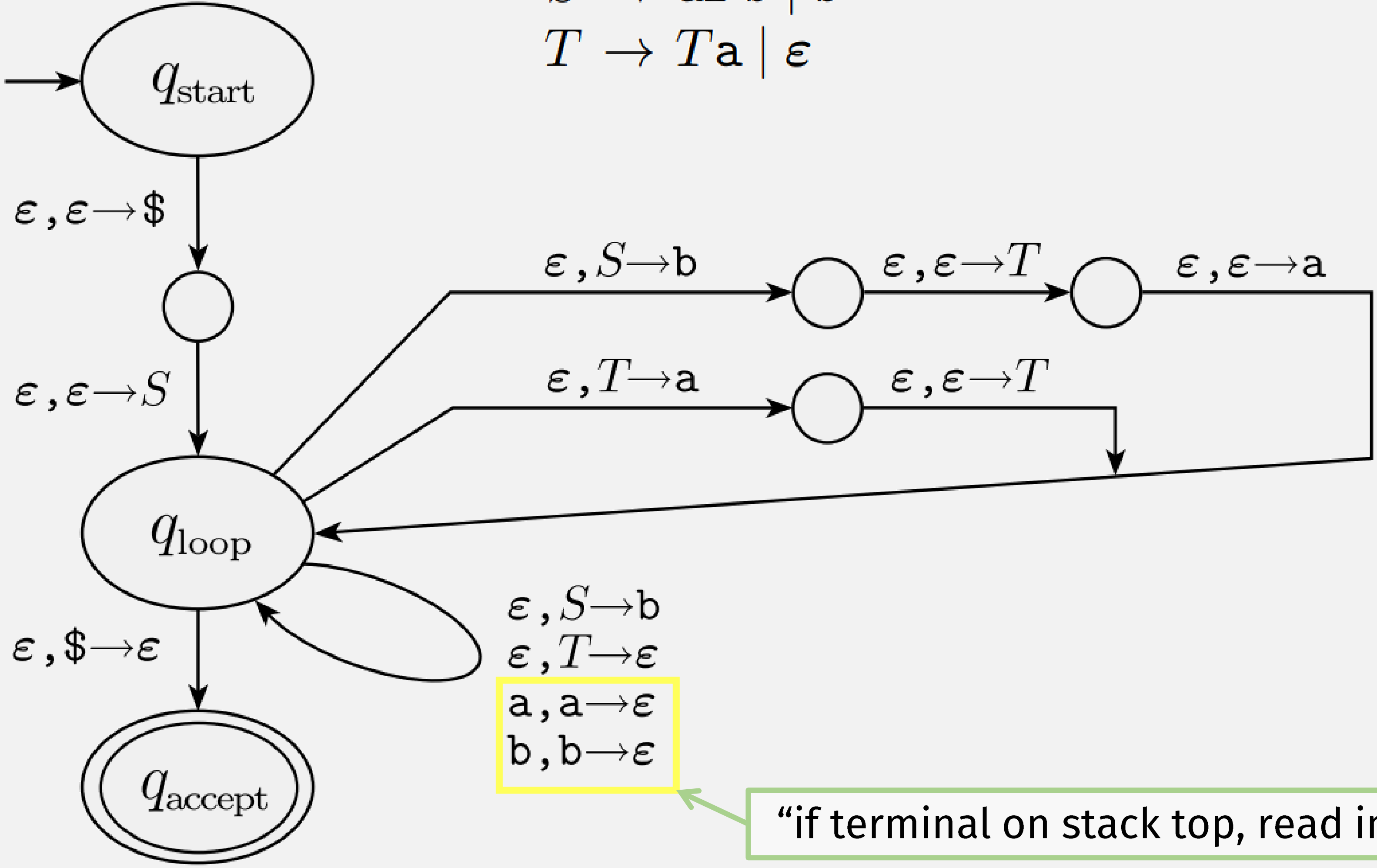
Example CFG -> PDA

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



Example CFG -> PDA

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



"if terminal on stack top, read input and pop"

A lang is a CFL iff some PDA recognizes it

- => If CFL, then PDA recognizes it
  - (Easier)
  - All CFLs have CFG describing it (definition of CFL)
  - Convert CFG -> PDA (**DONE**!)
- <= If PDA recognizes, then CFL
  - (Harder)
  - Need PDA -> CFG

PDA -> CFG: Step 1

Before converting PDA to CFG, modify it (without changing its lang) 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.

(confirm this to yourselves)

PDA $\mathbf{P}$ -> CFG $\mathbf{G}$: substitution rules

- For every pair of states p, q: add grammar variable A_{pq}

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

  - Apq represents “all possible inputs read going from state p to q”
- Add rules: $\mathrm{A}_{\mathrm{pq}} \rightarrow \mathrm{A}_{\mathrm{pr}}\mathrm{A}_{\mathrm{rq}}$, for every state r
  - “All possible strings when going from p to q =
    - All possible strings going from p to r, concatentated with
    - All possible strings going from r to q”
- We still need rules that produce terminals!
- The key: pair up stack pushes and pops (essence of 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 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 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 $\mathbf{P}$ -> CFG $\mathbf{G}$: 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 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

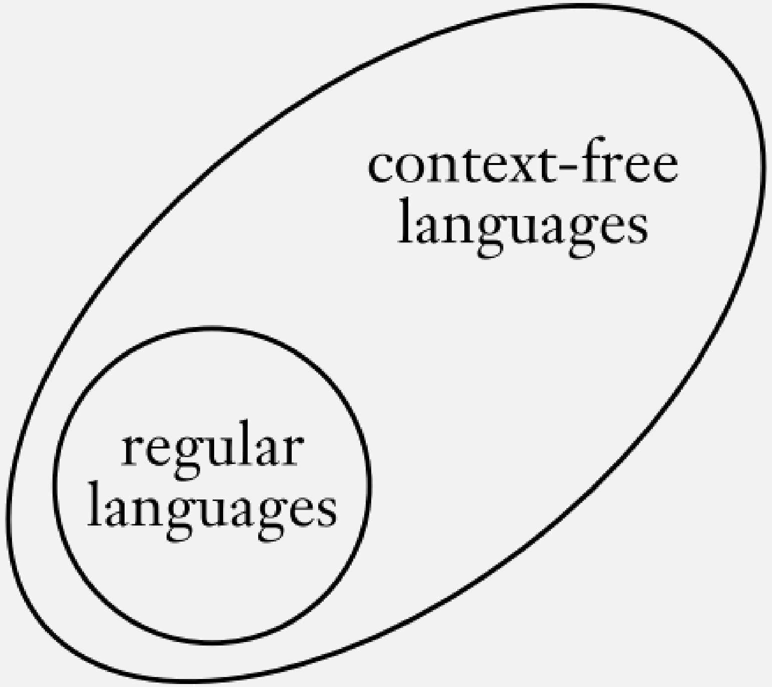
A lang is a CFL ⇔ A PDA recognizes it

- => If CFL, then PDA recognizes it
  - All CFLs have CFG describing it (definition of CFL)
  - Convert CFG -> PDA
- <= If PDA recognizes, then CFL
  - Convert PDA -> CFG

$\blacksquare$

Regular languages are CFLs, prove 3 ways

- NFA -> PDA (with no stack moves) -> CFG
  - Just now
- DFA -> CFG
  - Textbook page 107
- Regexp -> CFG
  - HW4



Check-in Quiz 10/7

On Gradescope

End of Class Survey 10/7

See course website

Remember, no class next Monday!