Lecture 8

- HW 1 due tomorrow on gradescope

- Today: Intro to Context Free Languages and Pushdown Automata

Nondeterministic Finite Aut.

given some DFA M, want to construct new NFA M'

Say we want to accept w if either (1) or (2) holds:

(1) : string w has length exactly 2

(2) : w ends in one of the 1st 3 states q_1, q_2, q_3

$\underbrace{2a}_{q_1}$ $\underbrace{2b}_{q_2}$ $\underbrace{2c}_{q_3}$

Original

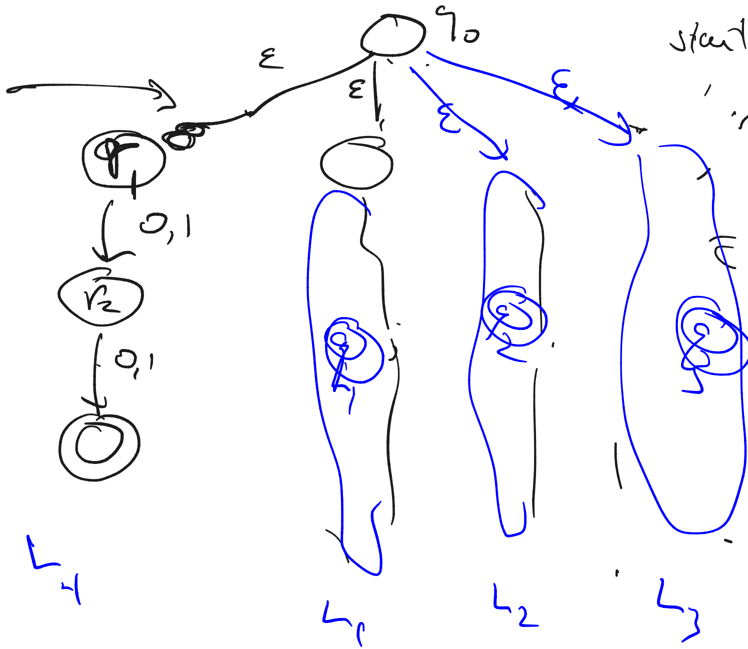
M: $\cancel{w} \in \Sigma^*$ $Q = q_0\ q_1\ q_2 \cdots q_{10}$

M': accept if: either $|w| = 2$

or w ends in one of the states q_1, q_2, q_3

Idea behind NFA is to guess one of these good possibilities & check.

checks if $|w| = 2$



start state of M, except only accept state is q_2

this path accepts iff w ends up in q_2

copy of M

$L_4 = \{w, \ |w| = 2\}$

$L_1 = \{ w \mid M(w) \text{ ends in state } q_1 \}$

$L_2 = \{ w \mid \quad \text{''} \quad q_2 \}$

$L_3 = \{ w \mid \quad \text{''} \quad q_3 \}$

Now spose we want to accept w iff (1') + (2')
both hold, where

L_1

M_1
for
$L_1 = \{w \mid |w| \text{ is odd}\}$

(1') : $|w|$ is odd

(2') : M on w ends in state q_1

M_2:
$L_2 = \{w \mid M(w) \text{ ends in state } q_1\}$

want to construct M accepts $\underbrace{L_1}_{M_1} \cap \underbrace{L_2}_{M_2}$

How to construct M from $\underbrace{M_1}_{L_1}$ & $\underbrace{M_2}_{L_2}$?

(1) ~~Use~~ Rewrite "intersection" in terms of $\underbrace{+}_{\text{union}}$ and negation

$$L = \underbrace{L_1}_{M_1} \cap \underbrace{L_2}_{M_2} = \underbrace{\overline{\overline{L_1} + \overline{L_2}}}$$

(2) "Cross Product" construction.

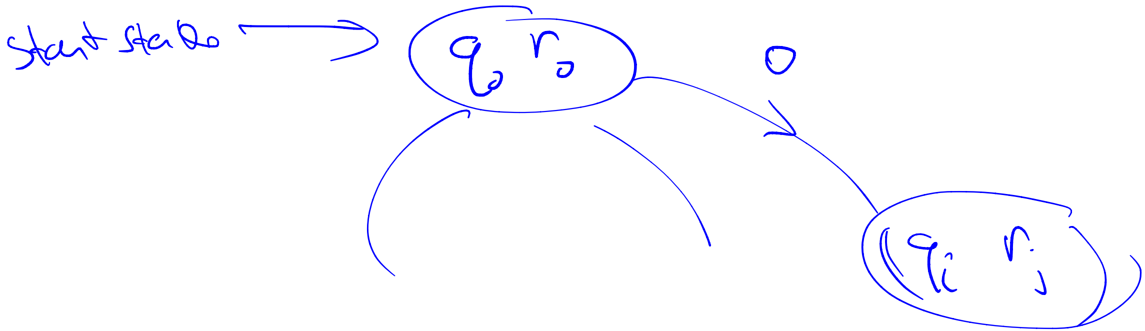
M_1 : $Q_1 = q_0 \cdots q_{10}$

M_2 : $Q_2 = r_0 \cdots r_{12}$

new M :

$$Q = \{ (q_i, r_j) \mid q_i \in Q_1, \ r_j \in Q_2 \}$$

$$\delta : (q_i, r_j), 0 \longrightarrow (q_i', r_j')$$



accept states : $\{ (q_i, r_j) \mid q_i \in$ accept state of M_1

$r_j \in$ " " " $M_2 \}$

r

Context-Free Languages + PushDown Automata

Recall what we did for the class of Regular Languages:

① For regular languages we first defined the regular languages to be the languages recognized by some DFA(≡NFA)

② We gave an alternative characterization of regular languages: Language/generation model: Regular Expressions

ⓩa We proved that these 2 characterizations are equivalent

$$\underbrace{\text{DFA / NFA}}_{\text{Machine Model}} \equiv \underbrace{\text{Regular Expressions}}_{\text{Language/generation Model}}$$

③ Pumping Lemma (for Reg. L's): Used to prove that some languages are not regular.

Context-Free Languages + PushDown Automata

Now we will define a larger class of languages that includes all regular languages plus new ones.

① We will first define CFL's to be those languages accepted by PUSHDOWN AUTOMATA (PDA)

② Then we give an alternative characterization of CFLs
Language/generation Model: Context Free grammars (CFGs)

②a We will prove these 2 characterizations are equivalent:

$$\underbrace{\text{PUSHDOWN AUTOMATA (PDA)}}_{\text{Machine Model}} \equiv \underbrace{\text{Context Free grammars (CFGs)}}_{\text{Language/generation Model}}$$

③ Pumping Lemma (for CFL's): used to prove that some languages are Not Context Free Languages

Context - Free Languages + PushDown Automata

Note: We will present in this order : 1, 2, 3, 2a

Book presents in this order : 2, 1, 2a, 3

1 We will first define CFL's to be those languages accepted by PUSHDOWN AUTOMATA (PDA) ← Machine Model

2 Then we give an alternative characterization of CFLs
Language/generation Model : Context Free grammars (CFGs)

2a We will prove these 2 characterizations are equivalent :

$\underbrace{\text{PUSHDOWN AUTOMATA (PDA)}}_{\text{Machine Model}} \equiv \underbrace{\text{Context Free grammars (CFGs)}}_{\text{Language/generation Model}}$

3 Pumping Lemma (for CFL's) : used to prove that
Some Languages are Not Context Free Languages

1

PushDown Automata

- Regular Languages/ NFA

  Languages recognizable by scanning input once from left → right, using a finite amount of memory

- We saw examples of languages that are <u>Not</u> regular:

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

- Pushdown Automata (PDAs) generalize NFAs to allow for a limited kind of (unbounded) memory: <u>a stack</u>

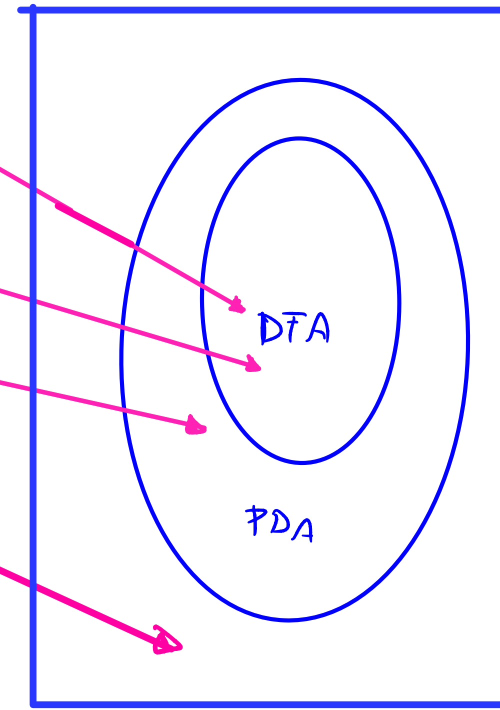
Examples of Languages

$L1 = \{ w \in \{0,1\}^* \mid w \text{ has an even number of 1's} \}$

$L2 = \{ w \in \{0,1\}^* \mid w \text{ ends with } 011 \}$

$L3 = \{ w \in \{0,1\}^* \mid w = 0^n 1^n,\ n \geq 1 \}$

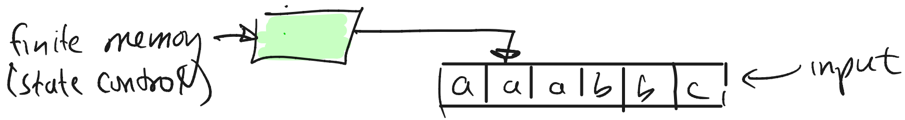
$L4 = \{ w \in \{0,1,2\}^* \mid w = 0^n 1^n 2^n,\ n \geq 1 \}$



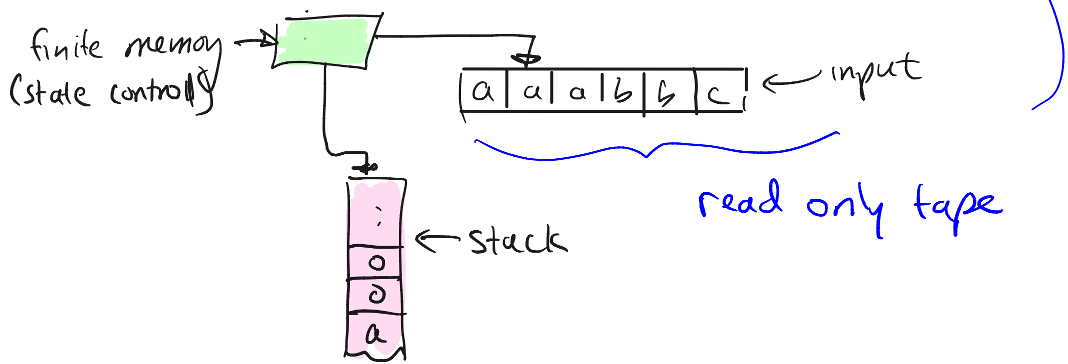
All Languages

$L \subseteq \Sigma^*$

NFA:



PDA: (Like NFAs, PDA is a nondeterministic model)



Example 1 $L = \{0^n 1^n \mid n \geq 0\}$

- PDAs are NFA's with extra stack
- In every step we can read next symbol (or ε-transition), move to a new state and push or pop or replace top symbol on stack

Idea:

- Start by pushing special "\$" symbol onto stack
- Read 0's and push them onto stack
- as soon as we see a 1, start popping a 0 off stack everytime we see a 1.
- Nondeterministically guess when we are at end of input. If there is the symbol "\$" on top of stack, go to accept state

Example 1

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

- PDAs are NFA's with extra stack
- In every step we can read next symbol (or ε-transition), move to a new state and push or pop or replace top symbol on stack

transition "$a, b \to c$" means when reading input symbol a, if b is symbol on top of stack, replace b by c

pop $\to$ "$a, b \to \varepsilon$" means if reading input symbol a, can pop b off stack

push $\to$ "$a, \varepsilon \to c$" means if reading symbol a, push c onto top of stack

$Q = \{q_0, q_1, q_2, q_3\}$ → states

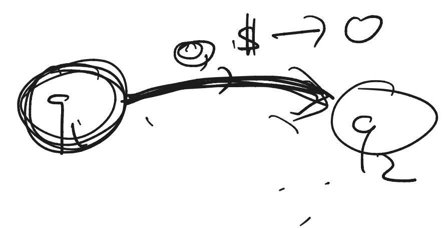
$\Sigma = \{0, 1\}$ → input alphabet

$\Gamma = \{0, \$\}$ → stack alphabet

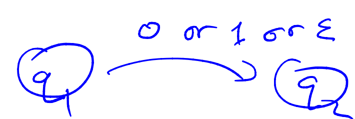
$F = \{q_0, q_4\}$ → accept states

q_0 → start state

δ



NFA:



Example 1

"$a, b \to c$" means when reading input symbol a, if b is symbol on top of stack, replace b by c

"$a, b \to \varepsilon$" means if reading input symbol a, can pop b off stack

"$a, \varepsilon \to c$" means if reading symbol a, push c onto top of stack

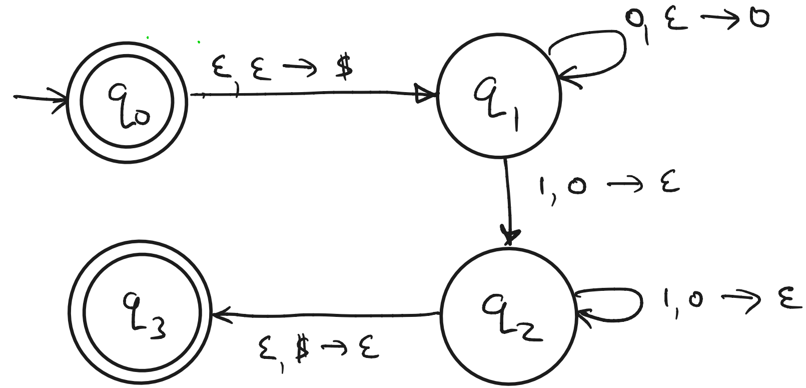
$Q = \{q_0, q_1, q_2, q_3\}$

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

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

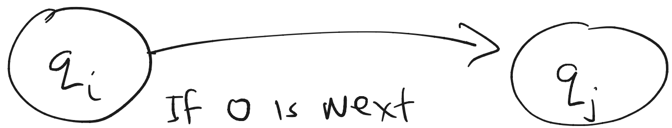
$F = \{q_0, q_3\}$

q_0



PDA accepts an input w if there exists a computation path starting in q_0 and ending in an accept state

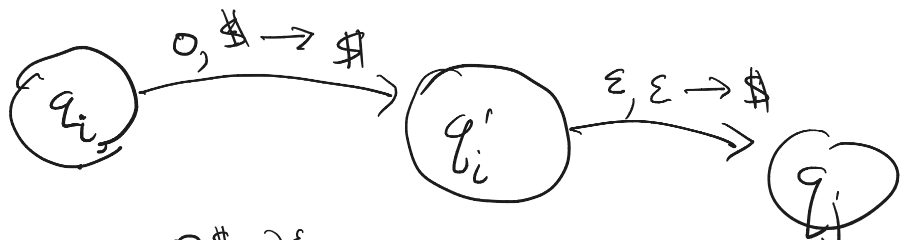
Say we want to simulate this transition:



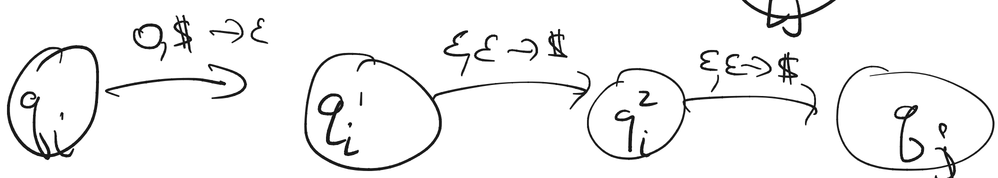
If 0 is next symbol read in input string <u>and</u> top of stack contains \$, then push another \$ and move to state q_j

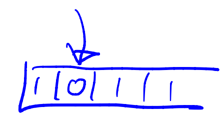
This can be simulated by either (a) or (b)

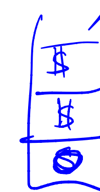
(a)



(b)







Example 1

$Q = \{q_0, q_1, q_2, q_3\}$

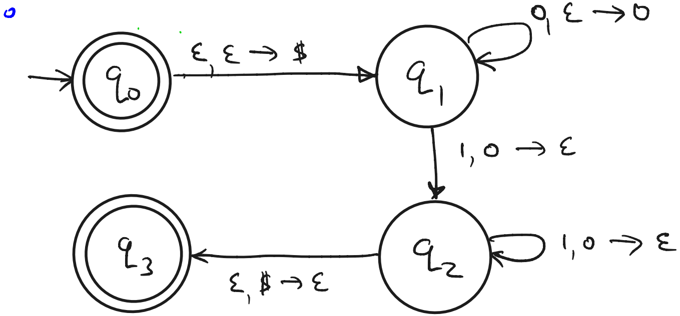
$\Sigma = \{0, 1\}$

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

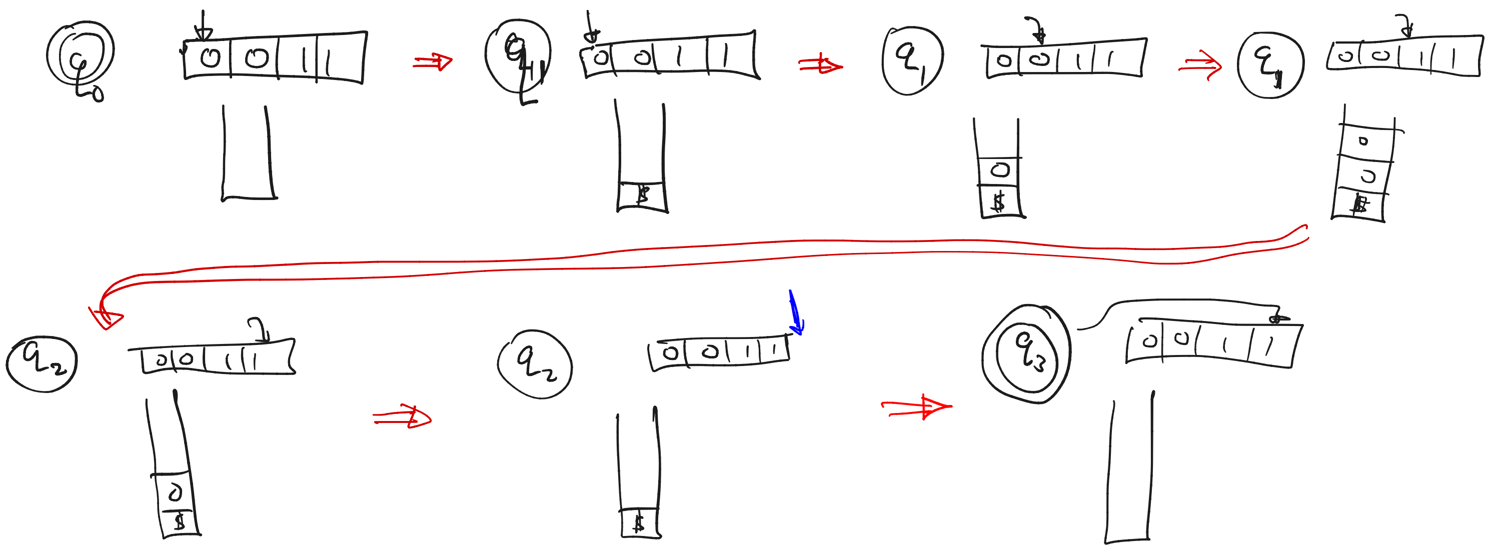
$F = \{q_0, q_3\}$

q_0

M:



on input w = 0011 : (so M accepts w)



Example 1

$Q = \{q_0, q_1, q_2, q_3\}$

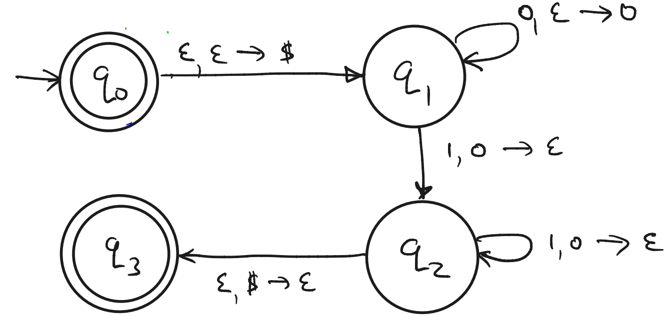
$\Sigma = \{0, 1\}$

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

$F = \{q_0, q_3\}$

q_0

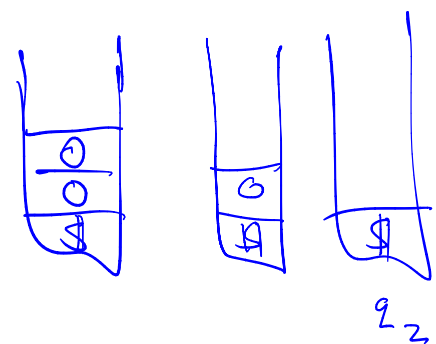
M:



on input $w = 00111$:

rejects since no sequence of moves exists that agrees with transition function and ends in accept state

$\mathcal{L}(M) = \{0^n 1^n \mid n \geq 0\}$



PDA (Formal Description)

A PDA is described by a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

states, input alphabet, stack alphabet, start state, accept states

$$\delta : Q \times \{\Sigma \cup \varepsilon\} \times \{\Gamma \cup \{\varepsilon\}\} \to \mathcal{P}(Q \times \{\Gamma \cup \varepsilon\})$$

M <u>accepts</u> w if w can be written as $w = w_1 w_2 w_3 \dots w_m$, where each $w_i \in \{\Sigma \cup \varepsilon\}$, and $\exists$ a sequence of states $r_0, r_1, \dots, r_m \in Q$ and $\exists$ sequence of strings $\underbrace{s_0, s_1, \dots, s_m}_{s_i = \text{contents of stack at time } i} \in \Gamma^*$ satisfying:

1. $r_0 = q_0$, $s_0 = \varepsilon$ (start state is q_0, stack initially empty)
2. for all $i = 0, 1, \dots, m-1$ $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = at$, $s_{i+1} = bt$, $a, b \in \Gamma \cup \varepsilon$, $t \in \Gamma^*$
   (M moves according to transition function δ)
3. $r_m \in F$ (final state is an accept state)

original $w = 0011$

$w = \varepsilon 0 \varepsilon \varepsilon 011$

PDA (Formal Description)

A PDA is described by a 6-tuple $M = (\underset{\text{states}}{Q}, \underset{\text{input alphabet}}{\Sigma}, \underset{\text{tape alphabet}}{\Gamma}, \delta, \underset{\text{start state}}{q_0}, \underset{\text{accept states}}{F})$

$$\delta : Q \times \{\Sigma \cup \varepsilon\} \times \{\Gamma \cup \varepsilon\} \rightarrow \mathcal{P}(Q \times \{\Gamma \cup \varepsilon\})$$

M <u>accepts</u> w if w can be written as $w = w_1 w_2 w_3 \ldots w_m$, where each $w_i \in \{\Sigma \cup \varepsilon\}$, and $\exists$ a sequence of states $r_0, r_1, \ldots, r_m \in Q$ and $\exists$ sequence of strings $\underbrace{s_0, s_1, \ldots, s_m}_{s_i \,=\, \text{contents of stack at time } i} \in \Gamma^*$ satisfying:

① $r_0 = q_0$, $s_0 = \varepsilon$ (start state is q_0, stack initially empty)

② for all $i = 0, 1, \ldots, m-1$ $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = at$, $s_{i+1} = bt$, $a, b \in \Gamma \cup \varepsilon$, $t \in \Gamma^*$

(M moves according to transition function δ)

③ $r_m \in F$ (final state is an accept state)

$$\mathcal{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

A language is a CFL if some PDA accepts it

PDA (Formal Description)

A PDA is described by a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

states, input alphabet, tape alphabet, start state, accept states

$$\delta : Q \times \{\Sigma \cup \varepsilon\} \times \{\Gamma \cup \varepsilon\} \rightarrow P(Q \times \{\Gamma \cup \varepsilon\})$$

Notes: We only accept if we are in an accept state when all of w is processed.

Note that we can accept a string w even if stack is not empty at end of processing w.