Theory of Computation

Chapter 02
Pushdown Automata

Introduction to the Theory of Computation, 3rd Ed, Michael Sipser
Introduction to Automata Theory Languages and Computation, 2nd, Hopcroft, Motwani, and Ullman
Last modified 14/10/2020

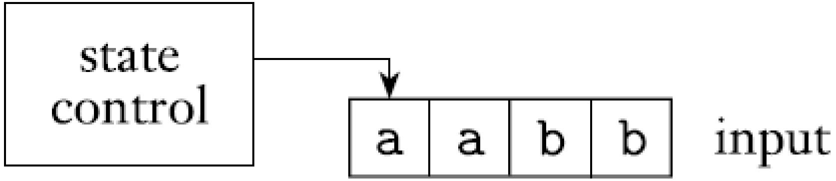
October, 2020
Zinia Sultana
Lecturer, CSE Dept, MIST

Pushdown Automata

- New type of computational model
- like nondeterministic finite automata but have an extra component called a stack.
- The stack provides additional memory beyond the finite amount available in the control.
- The stack allows pushdown automata to recognize some nonregular languages.
- Pushdown automata are equivalent in power to context-free grammars.

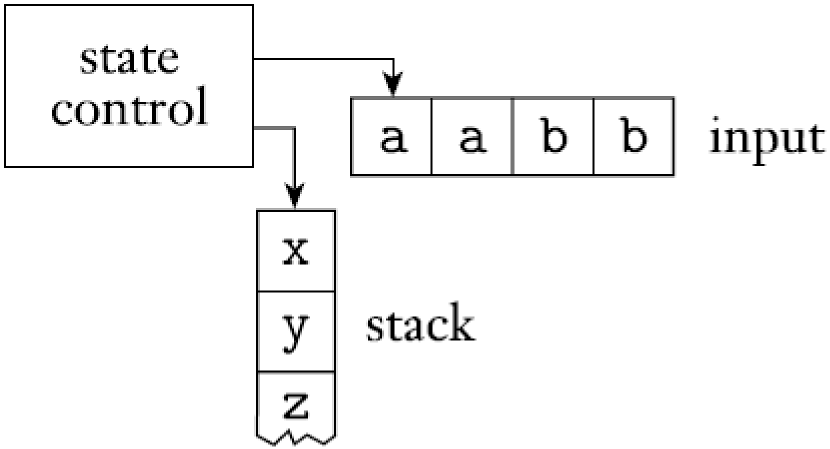
Pushdown Automata

- Schematic of a finite automaton



Pushdown Automata

- Schematic of a pushdown automaton



- Pushing
- Popping

Pushdown Automata

- A pushdown automaton **(PDA)** can write symbols on the stack and read them back later.
- Writing a symbol "pushes down" all the other symbols on the stack.
- At any time the symbol on the top of the stack can be read and removed

Formal Definition of a Pushdown Automaton

DEFINITION 2.13

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.

Computation of a Pushdown Automaton

A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows:

It accepts input w if w can be written as

$w = w_1 w_2 \cdots w_m$, where each $w_i \in \Sigma_\varepsilon$ and

sequences of states $r_0, r_1, \ldots, r_m \in Q$ and

strings $s_0, s_1, \ldots, s_m \in \Gamma$

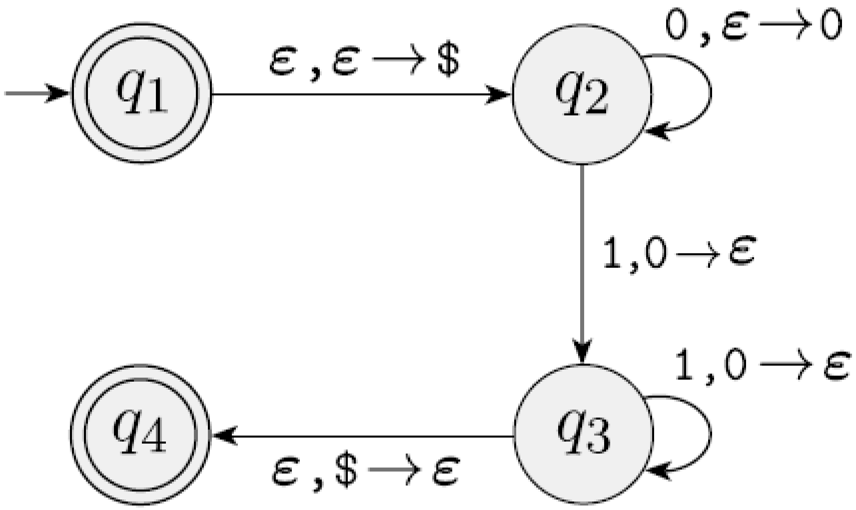
1. $r_0 = q_0$ and $s_0 = \varepsilon$. This condition signifies that M starts out properly, in the start state and with an empty stack.
2. For $i = 0, \ldots, m - 1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma$. This condition states that M moves properly according to the state, stack, and next input symbol.
3. $r_m \in F$. This condition states that an accept state occurs at the input end.

Pushdown Automata -1

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

Read symbols from the input. As each 0 is read, push it onto the stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read. If reading the input is finished exactly when the stack becomes empty of 0s, accept the input. If the stack becomes empty while 1s remain or if the 1s are finished while the stack still contains 0s or if any 0s appear in the input following 1s, reject the input.

Pushdown Automaton – M1



State diagram for the PDA M1 that recognizes $\{0^n 1^n \mid n \geq 0\}$

Formal definition of a PDA M1

The following is the formal description of the PDA (page 112) that recognizes the language $\{0^n1^n|\, n \geq 0\}$. 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 $\emptyset$.

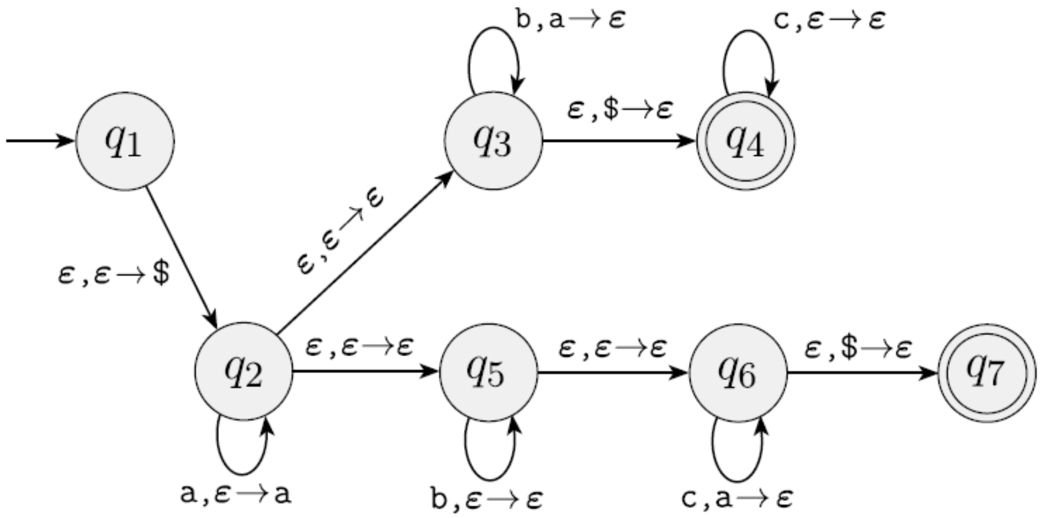
Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

PDA - Practice

- L = {w | w is binary string containing equal number of zeros and ones}

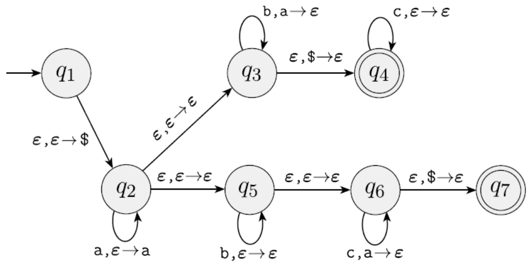
PDA M2

- $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$



PDA M2

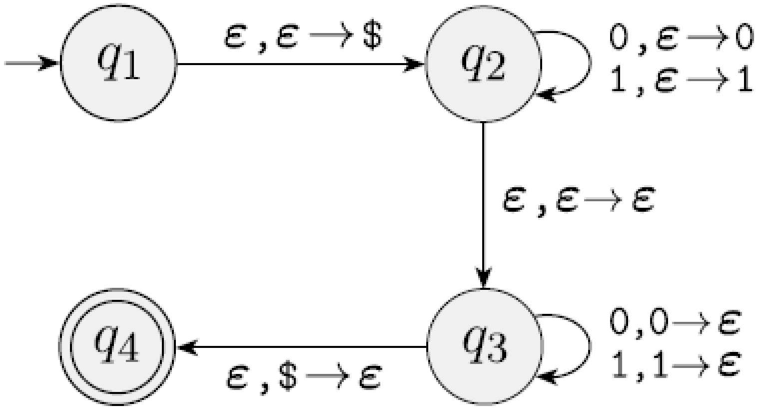
- $L = \{a^i b^j c^k \mid i, j, k \geq 0 \; and \; i = j \; or \; i = k\}$



Transition table:

PDA M3

- $L = \{ww^R | \ w \in \{0,1\}^*\}$



PDA – Practice

- $L = \{a^n b^{3n} \mid n \geq 0\}$

Equivalence with Context-free Grammars

THEOREM 2.20

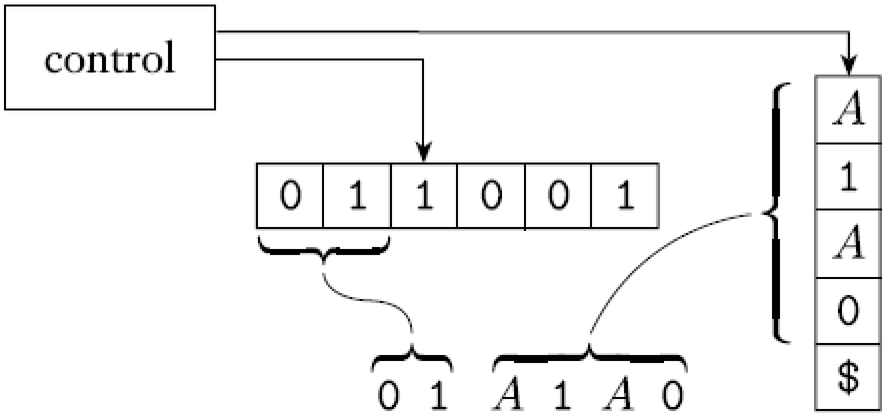
A language is context free if and only if some pushdown automaton recognizes it.

Equivalence with Context-free Grammars

LEMMA 2.21

If a language is context free, then some pushdown automaton recognizes it.

Equivalence with Context-free Grammars



Equivalence with Context-free Grammars

The following is an informal description of P.

1. Place the marker symbol \$ and the start variable on the stack.

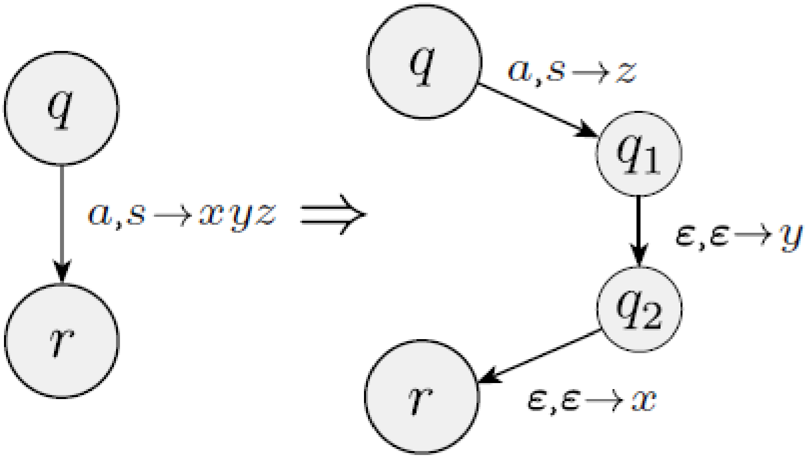
2. Repeat the following steps forever.

   a. If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.

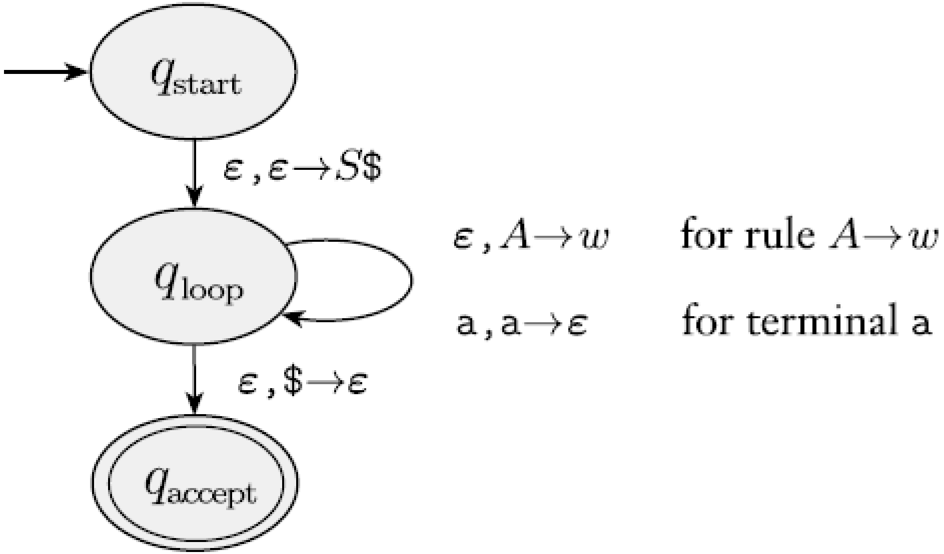
   b. If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.

   c. If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if it has all been read.

Equivalence with Context-free Grammars



Equivalence with Context-free Grammars



Equivalence with CFG - Example

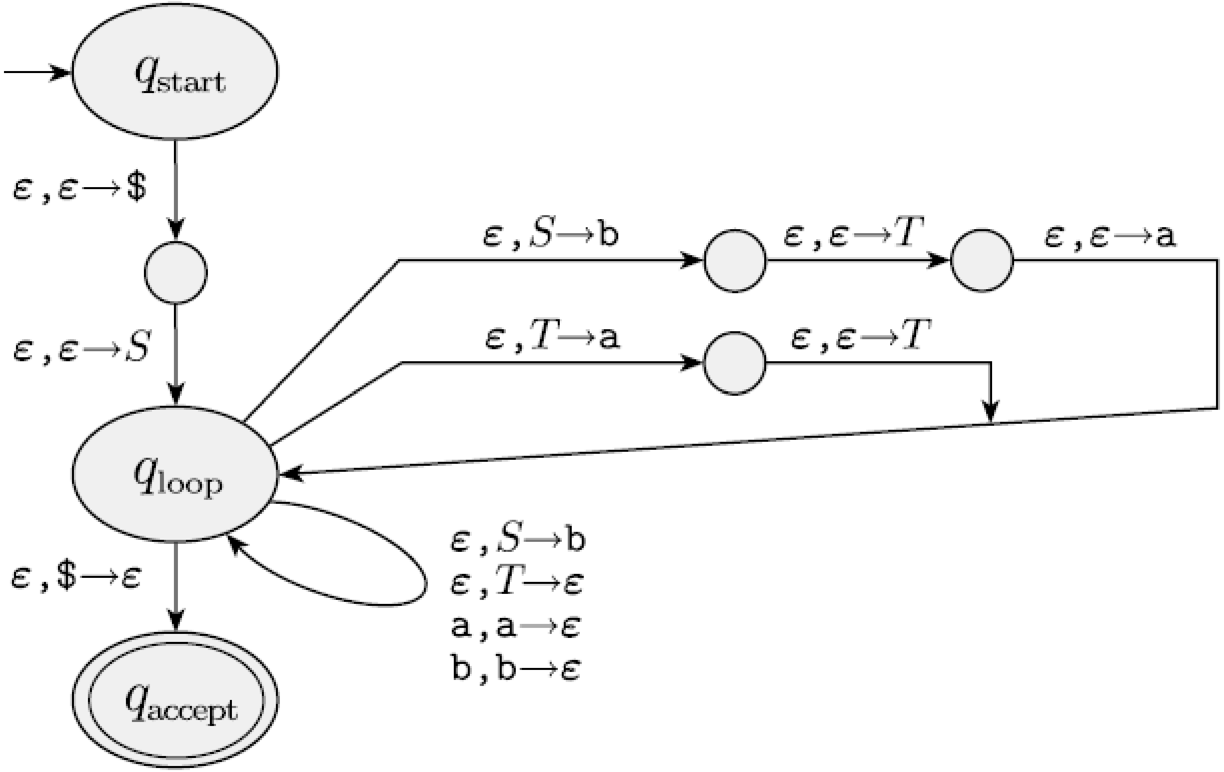
Construct a PDA for the following CFG G:

$S \rightarrow aT\,b \mid b$

$T \rightarrow T\,a \mid \varepsilon$

Equivalence with CFG - Example

$S \to aT\,b \mid b$
$T \to T\,a \mid \varepsilon$



Equivalence with CFG - Example

Try yourself:

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

Equivalence with CFG

LEMMA 2.27

If a pushdown automaton recognizes some language, then it is context free.

Equivalence with CFG

- ✓ Modify the PDA
- ✓ Describe rules for CFG

Equivalence with CFG - Modify the PDA

Modifying PDA:

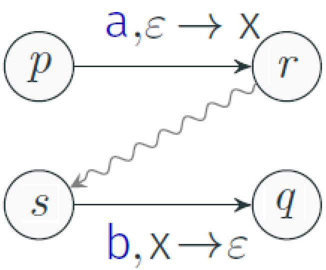
1. It has a single accept state, q_{accept}.
2. It empties its stack before accepting.
3. Each transition either pushes a symbol onto the stack (a *push* move) or pops one off the stack (a *pop* move), but it does not do both at the same time.

Equivalence with CFG - Add G's rules

Let, $P = (Q, \Sigma, \Gamma, \delta, q0, \{qaccept\})$. The variables of G are $\{Apq|\ p, q \in Q\}$.
The start variable is $A_{q_0,q_{accept}}$.
Now we describe G's rules in three parts.

1. For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, 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.
2. For each $p, q, r \in Q$, put the rule $Apq \to A_{pr}A_{rq}$ in G.
3. Finally, for each $p \in Q$, put the rule $A_{pp} \to \varepsilon$ in G.

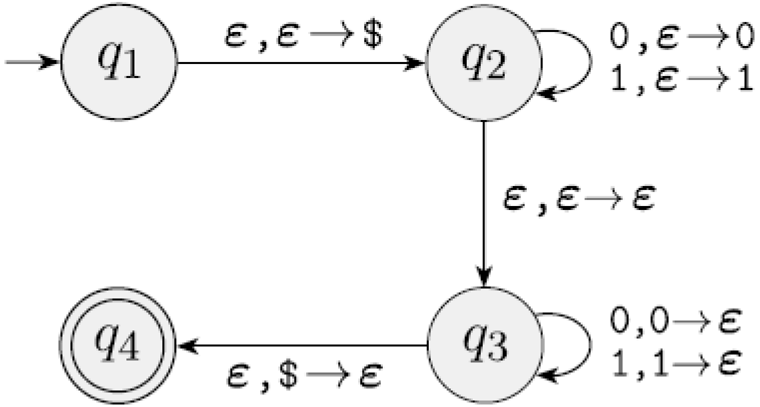
Equivalence with CFG - Add G's rules

PDA	CFG
	$A_{pp} \to \varepsilon$
	$A_{pq} \to A_{pr} A_{rq}$
	$A_{pq} \to \text{a} A_{rs} \text{b}$ $\text{a} = \varepsilon$ or $\text{b} = \varepsilon$ allowed

Start variable: A_{pq}
Here, start state – p
accepting state – q

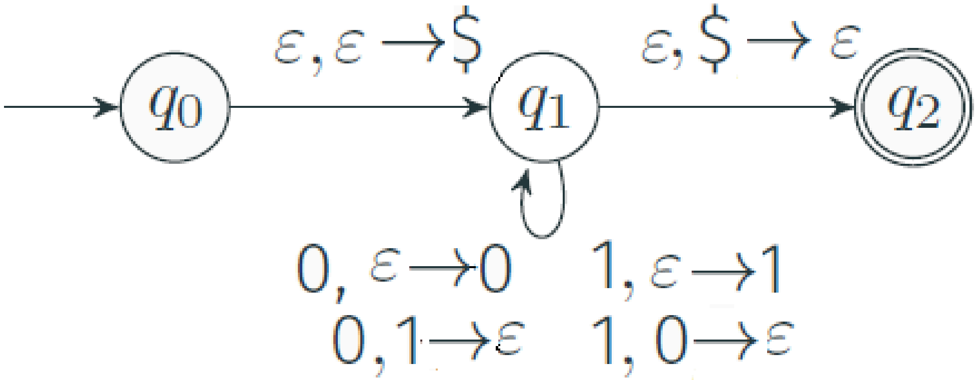
Equivalence with CFG - Example

Find CFG for the following PDA:



Equivalence with CFG – Try yourself

Find CFG for the following PDA:



END