1 Computing Using a Stack

Beyond Finite Memory: The Stack

- So far we considered automata with finite memory
- Today: automata with access to an infinite *stack*
- The stack can contain an unlimited number of characters. But
  - can read/erase only the top of the stack: *pop*
  - can add to only the top of the stack: *push*
- On longer inputs, automaton may have more items in the stack

---

Keeping Count Using the Stack

- An automaton can use the stack to recognize $\{0^n 1^n \mid n \geq 0\}$
  - On reading a 0, push it into the stack
  - After the 0s, on reading each 1, pop a 0
  - (If a 0 comes after a 1, reject)
  - If attempt to pop an empty stack, reject
  - If stack not empty at the end, reject
  - Else accept

Matching Parenthesis Using the Stack

- An automaton can use the stack to recognize balanced parenthesis
- e.g. (())() is balanced, but ())() and (() are not
  - On seeing a (push it on the stack
  - On seeing a) pop a (from the stack
  - If attempt to pop an empty stack, reject
  - If stack not empty at the end, reject
  - Else accept

---

2 Definition of Pushdown Automata

Pushdown Automata (PDA)

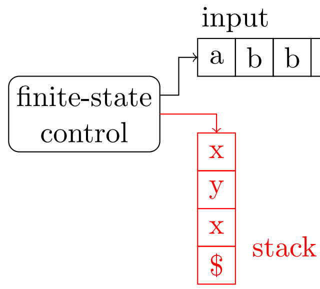
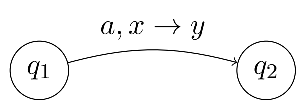


Figure 1: A Pushdown Automaton

- Like an *NFA with ϵ-transitions*, but with a stack
  - Stack depth unlimited: not a finite-state machine
  - Non-deterministic: accepts if any thread of execution accepts
- Has a non-deterministic finite-state control
- At every step:
  - Consume next input symbol (or none) and pop the *top symbol on stack* (or none)
  - Based on current state, consumed input symbol and popped stack symbol, do (non-deterministically):
    1. push a symbol onto stack (or push none)
    2. change to a new state



If at q_1, with next input symbol a and top of stack x, then *can* consume a, pop x, push y onto stack and move to q_2 (any of a, x, y may be ϵ)

Pushdown Automata (PDA): Formal Definition

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

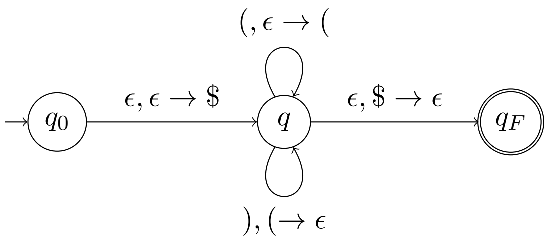
- Q = Finite set of states
- Σ = Finite input alphabet

- Γ = Finite stack alphabet
- q_0 = Start state
- $F \subseteq Q$ = Accepting/final states
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}))$

---

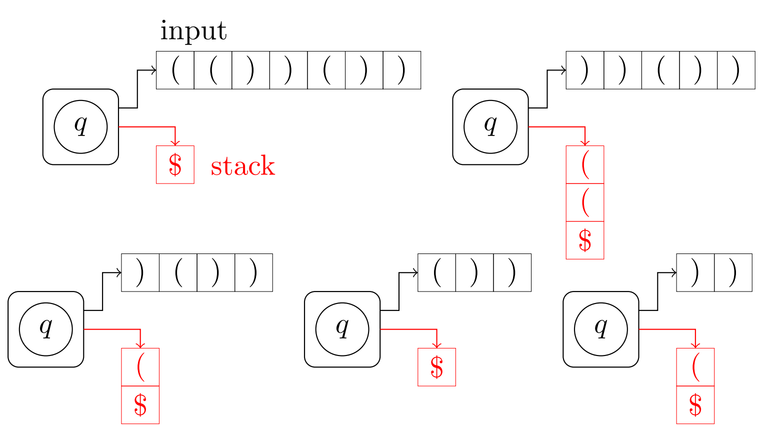
3 Examples of Pushdown Automata

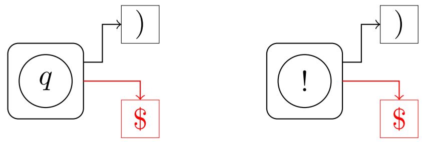
Matching Parenthesis: PDA construction



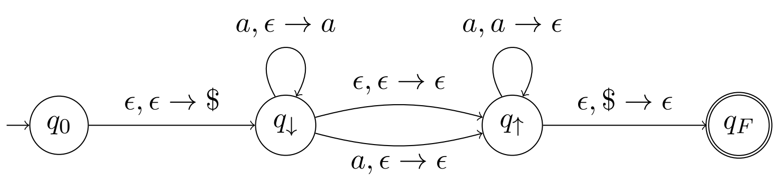
- First push a "bottom-of-the-stack" symbol \$ and move to q
- On seeing a (push it onto the stack
- On seeing a) pop if a (is in the stack
- Pop \$ and move to final state q_F

Matching Parenthesis: PDA execution



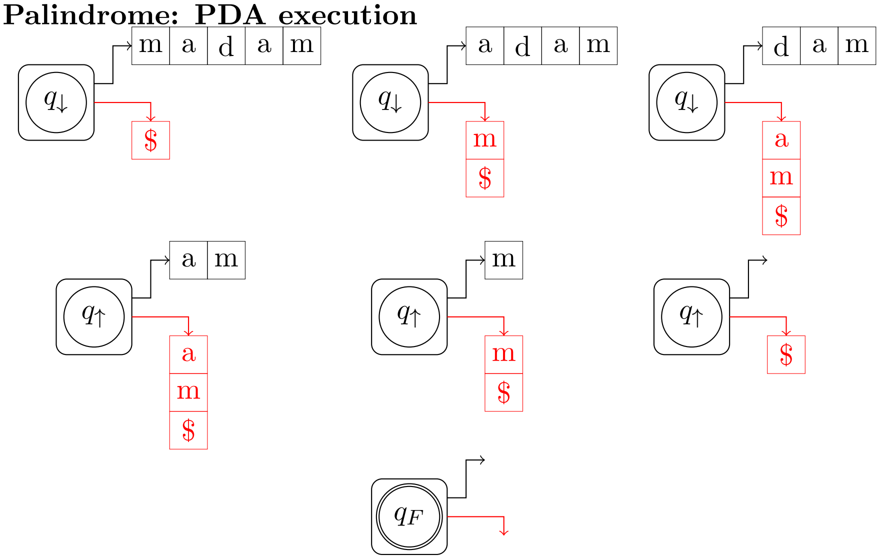


Palindrome: PDA construction



- First push a "bottom-of-the-stack" symbol \$ and move to a pushing state
- Push input symbols onto the stack
- Non-deterministically move to a popping state (with or without consuming a single input symbol)
- If next input symbol is same as top of stack, pop
- If \$ on top of stack move to accept state

Palindrome: PDA execution



4 Semantics of a PDA

4.1 Computation

Instantaneous Description

In order to describe a machine's execution, we need to capture a "snapshot" of the machine that completely determines future behavior

- In the case of an NFA (or DFA), it is the state
- In the case of a PDA, it is the state + *stack contents*

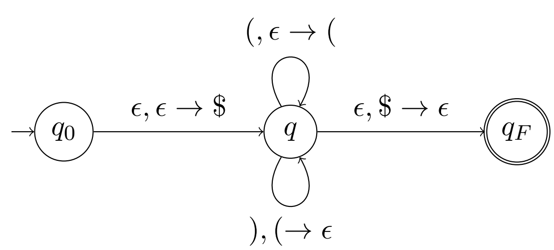
Definition 1. An *instantaneous description* of a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is a pair $\langle q, \sigma \rangle$, where $q \in Q$ and $\sigma \in \Gamma^*$

Computation

Definition 2. For a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, string $w \in \Sigma^*$, and instantaneous descriptions $\langle q_1, \sigma_1 \rangle$ and $\langle q_2, \sigma_2 \rangle$, we say $\langle q_1, \sigma_1 \rangle \xrightarrow{w}_P \langle q_2, \sigma_2 \rangle$ iff there is a sequence of instanteous descriptions $\langle r_0, s_0 \rangle, \langle r_1, s_1 \rangle, \ldots \langle r_k, s_k \rangle$ and a sequence $x_1, x_2, \ldots x_k$, where for each i, $x_i \in \Sigma \cup \{\epsilon\}$, such that

- $w = x_1 x_2 \cdots x_k$,
- $r_0 = q_1$, and $s_0 = \sigma_1$,
- $r_k = q_2$, and $s_k = \sigma_2$,
- for every i, $(r_{i+1}, b) \in \delta(r_i, x_{i+1}, a)$ such that $s_i = as$ and $s_{i+1} = bs$, where $a, b \in \Gamma \cup \{\epsilon\}$ and $s \in \Gamma^*$

Example of Computation



Example 3.

$\langle q_0, \epsilon \rangle \xrightarrow{(()(} \langle q, (($\rangle$ because

$$\langle q_0, \epsilon \rangle \xrightarrow{x_1 = \epsilon} \langle q, \$ \rangle \xrightarrow{x_2 = (} \langle q, (\$ \rangle \xrightarrow{x_3 = (} \langle q, ((\$ \rangle \xrightarrow{x_4 =)} \langle q, (\$ \rangle \xrightarrow{x_5 = (} \langle q, ((\$ \rangle$$

4.2 Language Recognized

Acceptance/Recognition

Definition 4. A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ *accepts* a string $w \in \Sigma^*$ iff for some $q \in F$ and $\sigma \in \Gamma^*$, $\langle q_0, \epsilon \rangle \xrightarrow{w}_P \langle q, \sigma \rangle$

Definition 5. The *language recognized/accepted* by a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is $\mathbf{L}(P) = \{w \in \Sigma^* \mid P \text{ accepts } w\}$. A language L is said to be *accepted/recognized* by P if $L = \mathbf{L}(P)$.

4.3 Expressive Power

Expressive Power of CFGs and PDAs

CFGs and PDAs have equivalent expressive powers. More formally, ...

Theorem 6. *For every CFG G, there is a PDA P such that $\mathbf{L}(G) = \mathbf{L}(P)$. In addition, for every PDA P, there is a CFG G such that $\mathbf{L}(P) = \mathbf{L}(G)$. Thus, L is context-free iff there is a PDA P such that $L = \mathbf{L}(P)$.*

Proof. Skipped. □
