

Pushdown Automata

Chanathip Namprempre

Computer Science
Reed College

Outline

1 Pushdown Automata

2 $\text{PDA} \approx \text{CFG}$

Outline

1 Pushdown Automata

2 $\text{PDA} \approx \text{CFG}$

Pushdown Automata

A pushdown automata is a **non-deterministic**, **finite state** machine with access to a **stack**.

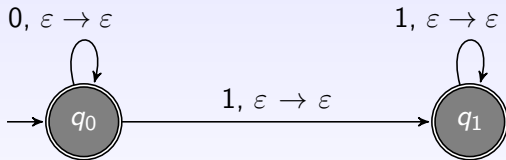
Definition

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 a set of states
- 2 Σ is the input alphabet
- 3 Γ is the **stack** alphabet
- 4 $\delta : Q \times \Sigma_{\epsilon} \times \Gamma_{\epsilon} \rightarrow \mathcal{P}(Q \times \Sigma_{\epsilon})$ is the transition function
- 5 $q_0 \in Q$ is the start state
- 6 $F \subseteq Q$ is the set of accept states

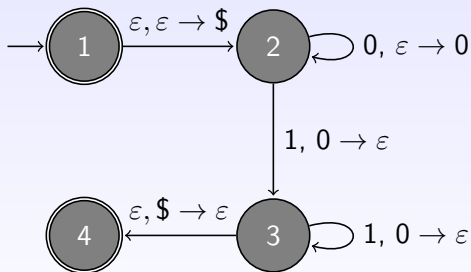
Example 1

$$L_1 = L(0^*1^*)$$



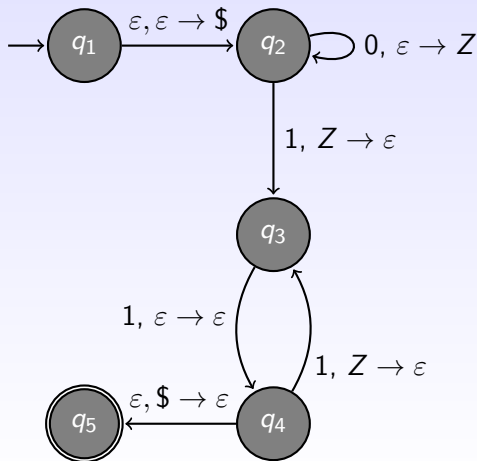
Example 2

$$L_2 = \{0^n 1^n \mid n \geq 0\}$$



Example 3

$$L_3 = \{0^n 1^{2n} \mid n \geq 0\}$$



Formal definition of computation

Let $m \in \mathbb{Z}^+$, $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a pushdown automaton.

Definition

M accepts w if w can be written as $y_1 y_2 \dots y_m$ where each $y_i \in \Sigma_\varepsilon$, and a sequence of states $r_0, r_1, \dots, r_m \in Q$ and string $s_0, s_1, \dots, s_m \in \Gamma^*$ exist with the following three conditions:

1. [M starts in the start state.] $r_0 = q_0$ and $s_0 = \varepsilon$,
2. [M moves according to δ , state, stack, and input symbol.] For $i = 0, \dots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, y_{i+1}, a)$ where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$,
3. [M ends up in an accept state.] $r_m \in F$.

Outline

1 Pushdown Automata

2 PDA \approx CFG

Equivalence of PDA and CFG

Theorem: PDA and CFG are equivalent

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

There are two directions that need to be proved:

[\Rightarrow]] If a language is a context free language, i.e., has a CFG generating it, then there is a PDA recognizing it.

(Easy)

[\Leftarrow]] If a language has a PDA recognizing it, then it has a CFG generating it.

(Hairy)

Constructing a PDA from a CFG

- ❶ Place the marker symbol \$ and the start variable on the stack
- ❷ Repeat forever
 - ❶ If the top of stack is a variable A , nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule, last symbol first.
 - ❷ 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. Otherwise, reject on this branch of nondeterminism.
 - ❸ If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if it has all been read.

Try it

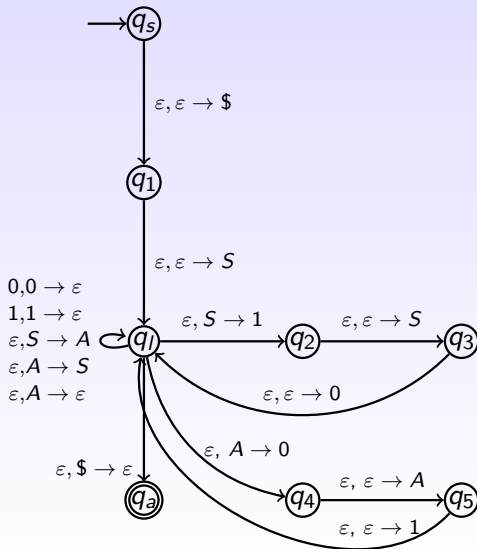
$$S \rightarrow 0S1 \mid A$$

$$A \rightarrow 1A0 \mid S \mid \varepsilon$$

Constructing a PDA from a CFG: Example

$S \rightarrow 0S1 \mid A$

$A \rightarrow 1A0 \mid S \mid \epsilon$



Constructing a PDA from a CFG: More examples

Once you write a PDA, try running it on some input strings. For each input string, compare the derivation from the grammar and the execution via the PDA.

1

$$S \rightarrow 0AA$$

$$A \rightarrow 0S \mid 1S \mid 0$$

2

$$S \rightarrow 0S1$$

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

Converting a PDA into a CFG

This is a bit hairy and not too interesting. We skip!