Pushdown Automata and CFLs Are Equivalent

Properties of Context-Free Languages

Theorem. The context-free languages are closed under union, concatenation, and star.

Proof Let S_1 and S_2 be the start symbols of two CFG. Let S_0 be the new start symbol of the new CFG we are to create.

Adding $S_0 \Rightarrow S_1 \mid S_2$ works for union.

Adding $S_0 \Rightarrow S_1 S_2$ works for concatenation.

Adding $S_0 \Rightarrow \epsilon \mid S_0 S_1$ works for star. ∎

CFLs Capture PDA

Theorem. Every language recognized by PDA is context-free.

Let L be a language recognized by a PDA $M = (Q, \Sigma, \Gamma, \delta, p_0, F)$.

We can modify M so that:

(*) M **has a unique accept state and, when it enters the state, the stack is empty.**

(**) **In a single move** M **may not both pop and push.**

Unique accept state and Empty Stack Acceptance

Modify M to create an equivalent PDA $N = (Q', \Sigma, \Gamma', \delta', q_0', \{q_f\})$.

Unique accept state and Empty Stack Acceptance

Modify M to create an equivalent PDA $N = (Q', \Sigma, \Gamma', \delta', q_0', \{q_f\})$.

N simulates M after adding a new special symbol $\perp$ to the stack. If M enters a accept state, N may choose to empty the stack until it encounters $\perp$, when N may accept.

Unique Accept State and Empty Stack Acceptance

Modify M to create an equivalent PDA $N = (Q', \Sigma, \Gamma', \delta', q_0', \{q_f\})$.

- $\Gamma' = \Gamma \cup \{\perp\}$.
- Q' consists of:
  - Q,
  - a new initial state I,
  - a new, unique accept state q_f,
  - a clean-up state C,
  - some additional states for achieving the "not both push and pop" requirement.

The use of q_0' and C as a Bottom Marker

There is just one move in state q_0: $\delta'(q_0', \epsilon, \epsilon) = \{(p_0, \perp)\}$.

The transition mean: **place a $\perp$ on stack** and then proceed to the initial state of M.

The Role of $\perp$ and C

In each accept state p of M, we add (C, ϵ) to $\delta'(p, \epsilon, \epsilon)$.

The transition means: **from any accept state of F, you may proceed to C.**

The Role of $\perp$ and C

We have

- $\delta'(C, \epsilon, \perp) = \{(q_f, \epsilon)\}$ and
- for each $a \in \Gamma$, $\delta'(C, \epsilon, a) = \{(C, \epsilon)\}$.

These transitions allow emptying stack and then entering q_f.

No Pop and Push at the Same Time

Suppose we have a permissible transition (q, c) for $\delta(p, a, b)$, where $a \in \Sigma_\epsilon$ and $b, c \in \Gamma$. Then we add a new state q' exclusively for this particular transition and replace this transition with two transitions:

- (q', ϵ) in $\delta(p, a, b)$ and
- (q, c) in $\delta(q', \epsilon, \epsilon)$.

Construction of Grammar

We say that M can *transition from state* p *to state* q *on input* w *while maintaining the minimum stack height* if it is possible for M to transition from p to q by processing w so that

- the stack height before reading w is the same as the stack height after finishing to read w and then entering q,
- during these two events the stack height never goes below the stack level at the time M starts processing w.

Construction of Grammar

Construct a CFG (V, Σ, P, S): $V = \{A_{pq} \mid p, q \in Q\}$ and $S = A_{q_0 q_f}$, where q_0 is the initial state of M and q_f is the unique accept state of M.

A_{pq} is the variable corresponding to the set of all strings w that M can process and transition from p to q while maintaining the minimum stack height.

Production rules:

- For every $p \in Q$, $A_{pp} \to \epsilon$.

Production rules:

- For every $p \in Q$, $A_{pp} \to \epsilon$.
- For all $p, q, r \in Q$, $A_{pq} \to A_{pr}A_{rq}$.

Production rules:

- For every $p \in Q$, $A_{pp} \to \epsilon$.
- For all $p, q, r \in Q$, $A_{pq} \to A_{pr} A_{rq}$.
- For all $p, q, r, s \in Q$, $b, c \in \Sigma_\epsilon$, and $d \in \Gamma_\epsilon$,

  if $(r, d) \in \delta(p, b, \epsilon)$ **and** $(q, \epsilon) \in \delta(s, c, d)$**, then** $A_{pq} \to bA_{rs}c$**.**

  This means: one possibility for transition from p to q while maintaining the stack height is to:
  - transition from p to r after adding d on top of stack,
  - transition from r to s while maintaining the stack height, and
  - transition from s to q while popping the d.

PDAs Recognize CFLs

Theorem. Each context-free language is recognized by a PDA.

Given an arbitrary CFL L and want to construct a PDA for L.

We can assume that L is given by a CNF grammar $G = (V, \Sigma, R, S)$.

We will design a PDA that simulates a leftmost derivation with respect to G.

Simulating Leftmost Derivation

Use symbol $\perp$ to mark the bottom of stack.

After placing a string $S\perp$ (read from top to bottom) on top of stack, repeat the following:

- Pop one symbol X from stack.
- If $X = \perp$ enter a accept state.
- Otherwise, nondeterministically select a rule $X \to w$.
  - If $w = a$ for some terminal a, read one input symbol; if the symbol is a, continue; otherwise, stop.
  - If $w = AB$, place AB on top of stack and continue.

Do It in a PDA Way

- If there is a pending job from the previous step, do it and continue.
- If at the initial state, place $\perp$ and continue.
- Choose whether to read input or not; $a \in \Sigma_\epsilon$.
- Choose whether to read from stack or not; $X \in \Gamma_\epsilon$.
- If $a \neq \epsilon$ and X is a variable with rule $X \to a$, continue.
- If $a = \epsilon$ and $X = \perp$, enter the accept state.
- If $a = \epsilon$, X is a variable, and there is a rule of the form $X \to BC$, select one rule place C on top of stack and in the next step place B.