Pushdown Automata

Theory, Examples, and Proofs

Sara Sorahi

Introduction to the Theory of Computation

Motivation

- Finite Automata recognize **regular languages**
- Finite memory $\Rightarrow$ no counting, no nesting
- Example not regular:

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

- Context-Free Grammars can generate such languages

Question: What machine model recognizes context-free languages?

Idea of Pushdown Automata

A **Pushdown Automaton (PDA)** is:

- A finite automaton
- Equipped with a **stack**

The stack provides:

- Unbounded memory
- Last-In, First-Out (LIFO) access

This enables:

- Counting
- Nesting
- Recursion

Formal Definition of a PDA

A Pushdown Automaton is a 6-tuple:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

- Q: finite set of states
- Σ: input alphabet
- Γ: stack alphabet
- δ: transition function
- q_0: start state
- F: accepting states

Transition Function

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

A transition:

- Reads an input symbol (or ε)
- Inspects the top of the stack
- Changes state
- Replaces the stack top with a string

Note: PDAs are inherently nondeterministic.

Configurations

A **configuration** of a PDA is:

$$(q, w, \gamma)$$

- q: current state
- w: remaining input
- γ: stack contents

A PDA accepts if it reaches:

- an accepting state, or
- an empty stack

These two acceptance modes are equivalent.

Example: Language $\{a^n b^n\}$

Idea:

- Read a symbols and push markers onto the stack
- Read b symbols and pop markers
- Accept if stack is empty at the end

PDA for $\{a^n b^n\}$

Transition notation: **input, pop** $\rightarrow$ **push**

- $a, Z \rightarrow AZ$
- $a, A \rightarrow AA$
- $b, A \rightarrow \varepsilon$
- $\varepsilon, Z \rightarrow Z$ (accept)

The PDA switches from pushing to popping when it sees the first b.

Trace Example

Input: `aaabbb`

Input	Stack
aaabbb	Z
aabbb	AZ
abbb	AAZ
bbb	AAAZ
bb	AAZ
b	AZ
ε	Z
accept	empty

Balanced Parentheses

Language:

$$\{w \in \{(,)\}^* \mid w \text{ is balanced}\}$$

Strategy:

- Push '(' onto stack
- Pop '(' when ')' is read
- Accept if stack is empty

Deterministic vs Nondeterministic PDA

- DFA = NFA
- DPDA $\subset$ NPDA (strict)

Some context-free languages **require nondeterminism**.

Example:

$$\{ww^R \mid w \in \{a, b\}^*\}$$

Fundamental Theorem

Theorem

A language is context-free if and only if it is recognized by a pushdown automaton.

Two directions:

- CFG $\Rightarrow$ PDA
- PDA $\Rightarrow$ CFG

CFG $\Rightarrow$ PDA: Idea

The PDA simulates a **leftmost derivation**:

- Stack holds grammar symbols
- Variables expanded nondeterministically
- Terminals matched against input

Acceptance occurs when input is consumed and stack is empty.

CFG $\Rightarrow$ PDA: Construction

Given CFG $G = (V, \Sigma, R, S)$, construct a PDA:

- One state q
- Stack alphabet $\Gamma = V \cup \Sigma \cup \{Z\}$
- Accept by empty stack

CFG $\Rightarrow$ PDA: Transitions

Initialization:

$$\delta(q, \varepsilon, Z) = (q, SZ)$$

Variable expansion:

$$\delta(q, \varepsilon, A) = (q, \alpha) \quad \text{for each } A \to \alpha$$

Terminal matching:

$$\delta(q, a, a) = (q, \varepsilon)$$

Finish:

$$\delta(q, \varepsilon, Z) = (q, \varepsilon)$$

Why the Construction Works

- PDA expansions simulate grammar derivations
- Terminal matching enforces correct order
- Empty stack implies full derivation

Therefore:

$$L(\mathrm{PDA}) = L(\mathrm{CFG})$$

PDA and Parsing

A PDA is an abstract model of a **parser**:

PDA	Parsing
Stack	Parse stack
Input	Token stream
Expand	Predict
Pop	Match

LL Parsing

LL parsing:

- Top-down
- Leftmost derivation
- Predictive

An LL(1) parser behaves like a **deterministic PDA**.

LR Parsing

LR parsing:

- Bottom-up
- Shift–reduce
- Viable prefixes

LR parsers are also deterministic PDAs with structured transitions.

Closure Properties of CFLs

Closed under:

- Union
- Concatenation
- Kleene star

Not closed under:

- Intersection
- Complement

Proof Sketch: CFG $\Rightarrow$ PDA

Let $G = (V, \Sigma, R, S)$ be a CFG.

Construct a PDA that:

- starts with S on the stack
- replaces variables using grammar rules
- matches terminals with the input

Acceptance occurs when:

- input is fully consumed
- stack becomes empty

Thus, every derivation in G corresponds to an accepting PDA computation.

Proof Sketch: PDA $\Rightarrow$ CFG

Let P be a PDA.

Idea:

- Grammar variables encode PDA state pairs
- A variable A_{pq} represents strings that take P from state p to q

Productions simulate:

- pushing symbols
- popping symbols

This construction generates exactly the language of the PDA.

Proof: CFLs Are Closed Under Union

Let L_1, L_2 be CFLs.

Let P_1, P_2 be PDAs recognizing them.

Construct a new PDA:

- Add a new start state
- Add ε-transitions to P_1 and P_2

The new PDA recognizes $L_1 \cup L_2$.

Proof: CFLs Are Closed Under Concatenation

To recognize $L_1 L_2$:

- Run the PDA for L_1
- When it accepts, switch to the PDA for L_2

The stack can be reused.

Thus, $L_1 L_2$ is context-free.

Why CFLs Are Not Closed Under Intersection

Consider:

$$L_1 = \{a^i b^i c^j\}, \quad L_2 = \{a^i b^j c^j\}$$

Both L_1 and L_2 are context-free.

But:

$$L_1 \cap L_2 = \{a^n b^n c^n\}$$

Which is not context-free.

Proof Sketch: NPDA $\neq$ DPDA

- Some CFLs require guessing (e.g. palindromes)
- Deterministic PDAs cannot guess midpoints

Therefore:

$$\text{DPDA} \subset \text{NPDA} \quad \text{(strict)}$$