Pushdown Automata(PDA)

18 February,2026

Recap

- To prove that a language L is **regular**, it is sufficient to show **any one** of the following:
  - Finite Automaton exists(DFA/NFA/ε-NFA)
  - Regular Expression exists
  - Regular Grammar exists (Type-3 grammar)
- **Regular Languages**=Languages **accepted by FA**=Languages generated by Regular Grammars=Languages described by Regular Expressions
- A language is regular **if and only if** it can be represented by a finite automaton, regular expression, or regular grammar.
- Goal: To classify the computational complexity of the language — i.e., to determine how powerful a machine is needed to recognize it.

Examples of RL: **Login Password Rules**

- **Rule:** Password must contain only letters and digits and be at least 6 characters.
  - This can be represented by a **regular expression**:
  - Regex:[a-zA-Z0-9]{6,}
  - Why regular?
    - We don’t need memory beyond counting a small number.
    - No complex dependency between characters.

Examples of RL: **Email / Pattern Validation (Simple Version)**

- **Rule:** String must follow pattern: letters + @ + letters + .com
  - This can be represented by a **regular expression**:
  - Regex: [a-z]+@[a-z]+\.com
  - No unbounded counting/memory needed → Finite states suffice

Examples of NON-RL: **Balanced Parentheses**

- Examples:
  - ((()))
  - (()())
- Can we say it is RL? - **NO**
  - You must remember how many “(” you saw.
  - Needs stack memory → **not regular**

Examples of NON-RL: **Equal Number of Events**

- Same number of login and logout events.
  - Example: $\mathbf{a^n b^n}$
- Can we say it is RL? - **NO**
  - Need to remember count → Not regular.
  - Needs stack memory → **not regular**

PDA - Introduction

- A language is context-free (CFL) if and only if it can be generated by a Context-Free Grammar (CFG) or accepted by a Pushdown Automaton (PDA).
- Formally:
  - **CFL** $\iff$ ***Generated by CFG*** $\iff$ ***Accepted by PDA***
- Why "**if and only if**" is Important - It means both directions are true:
  - If a language has a CFG → there exists a PDA that accepts it
  - If a language has a PDA → there exists a CFG that generates it
- PDA is an accepting device for CFL.
- PDA = Finite Automaton + unbounded stack memory(potentially infinite memory)

PDA - Introduction (Cont.)

- Proof Using Example: $L = \{a^n b^n \mid n \geq 0\}$
  - Why FA Cannot Recognize $a^n b^n$?
  - FA would need to remember how many $\boldsymbol{a}$ were seen.
  - But FA has limited memory(finite number of states)
  - No counting memory
  - But here n can be infinite. Therefore FA cannot store the count
  - So L id not **not regular**.
- How PDA Recognizes $a^n b^n$?
  - Discuss later.

PDA – Formal Definition

- A Pushdown Automaton (PDA) is formally defined as a 7-tuple: **PDA = (Q, Σ, Γ, δ, q_0, Z, F)** where:
  - **Q** is the finite set of states
  - **Σ** is the set of input symbols
  - **Γ** is the set of pushdown (stack) symbols
  - **q_0** is the initial state
  - **$Z_0 \in \Gamma$** is the initial stack symbol present in the stack(stack can not be empty)
  - **F** is the set of final states
  - **δ** is the transition function that maps
    $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$

Note: *In each transition, the PDA reads an input symbol (or ε i.e. reads nothing)[at most one symbol] and the symbol at the top of the stack[exactly one], moves to a new state, and updates the stack by pushing/popping symbols.*

DPDA – Formal Definition

A **Deterministic Pushdown Automaton (DPDA)** is defined as a 7-tuple:

$$M = \mathbf{(Q, \Sigma, \Gamma, \delta, q_0, Z, F)}$$

δ — Transition Function for DPDA

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Q \times \Gamma*$$

Determinism Rule in DPDA: For a DPDA, for any state $\boldsymbol{q}$ and stack symbol $\boldsymbol{X}$,

1. At most one transition is allowed.
2. If an ε-transition exists, then no input transition is allowed for that same configuration.

Formally,

If

$$\delta(q, \epsilon, X) \neq \emptyset$$

then for every input symbol $a \in \Sigma$:

$$\delta(q, a, X) = \emptyset$$

Examples of transitions(DPDA)

- Suppose we have configuration: State = q_1 and Top of stack = A
- Case 1: ε transition exists
  - If (q_1, ϵ, A) = (q_2, B) then Then transitions like: $\delta(q_1, a, A)$ are **NOT allowed**.
- Case 2: No ε transition
  - If: $\delta(q_1, \epsilon, A) = \emptyset$ then input transitions like δ(q1,a,A)=(q3,AA) are valid

Note: **ε exists → input forbidden**

ε absent → input allowed(may or may not exist)

NPDA – Formal Definition

A **Non-Deterministic Pushdown Automaton (NPDA)** is defined as a 7-tuple:

$$M = \mathbf{(Q, \Sigma, \Gamma, \delta, q_0, Z, F)}$$

δ — Transition Function for DPDA

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \text{finite subsets of } (Q \times \Gamma*)$$

Given: Current state, current input symbol (or ε) and top of stack

- The machine can move to **any one of several possible next configurations**.Because output is a **power set**, multiple moves are allowed → nondeterminism.

Examples of transitions(NPDA)

- Suppose we have configuration: State = q_1 and Top of stack = A
- Case 1: Multiple transitions possible **ALLOWED**
  - $\delta(q_1, a, A) = \{ (q_2, B), (q_3, \varepsilon), (q_4, AA) \}$
- Case 2: ε-transition + input transitions **ALLOWED**
  - $\delta(q_1, \varepsilon, A) = \{ (q_5, C) \}$
  - $\delta(q_1, a, A) = \{ (q_2, B) \}$ // BOTH coexist!
- Case 3: No restrictions on conflicts
  - $\delta(q_1, \varepsilon, A) = \{ (q_2, B) \}$
  - $\delta(q_1, a, A) = \{ (q_3, \varepsilon) \}$ // Valid in NPDA!

Key Difference Between NPDA and DPDA

Feature	**NPDA**	**DPDA**
Number of moves	Multiple allowed	At most one
ε & input together	Allowed	Forbidden
Determinism	No	Yes

Key Points

- In Theory of Computation: PDA usually refers to the **general (nondeterministic) pushdown automaton**.So:

$$\text{PDA} \equiv \textit{NPDA}$$

  unless the author explicitly says **DPDA.**

- A DPDA satisfies all NPDA properties **plus additional restrictions** (determinism).So mathematically:

$$\text{DPDA} \subseteq \textit{NPDA}$$

- Every DPDA is NPDA but vice versa is not true.
  - DPDA transitions are deterministic → allowed in NPDA.
  - NPDA may require nondeterministic choices → cannot always be converted to DPDA.

$$\text{DPDA} \subset \textit{NPDA}$$

- DPDA is less powerful than NPDA
  - Language classes:

$$\text{Languages } \textit{accepted by NPDA} = \textit{Context-Free Languages (CFL)}$$
$$\text{Languages } \textit{accepted by DPDA} = \textit{Deterministic CFL (DCFL)}$$

$\text{DCFL} \subset \textit{CFL}$. So NPDA recognizes **strictly more languages**.

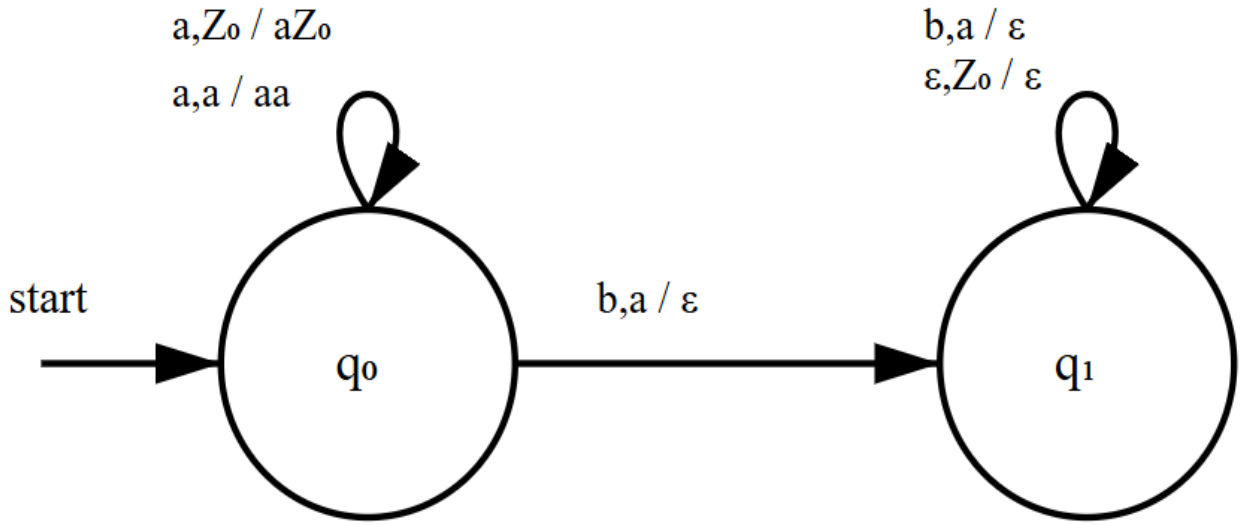
PDA String Acceptance

A PDA accepts string $w \in \Sigma^*$ if there exists some computation path that consumes all of w and satisfies one of these criteria:

1. Acceptance by Final State
   - Input fully consumed
   - PDA reaches final state $\mathbf{q_f} \in F$
   - Stack contents irrelevant
2. Acceptance by Empty Stack
   - Input fully consumed
   - Stack completely empty (no symbols left, not even Z_0)
   - State irrelevant (***q*** can be anything)

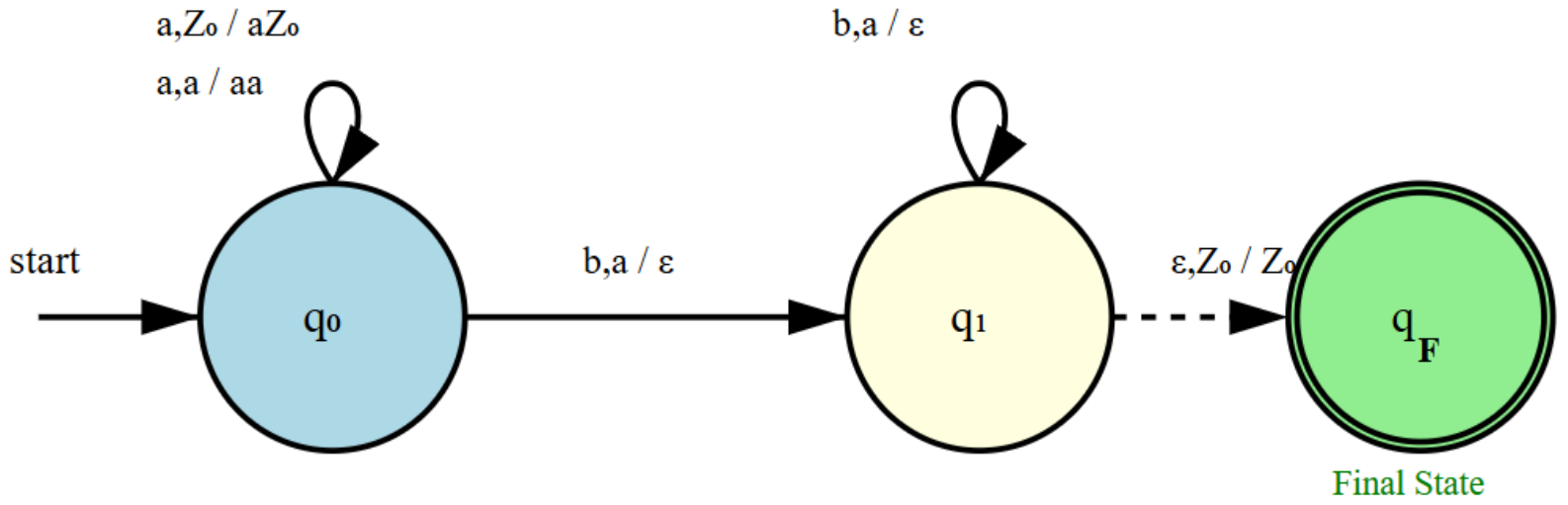
Construction of PDA

- Design PDA accepting $\{a^n b^n : n >= 1\}$ by **empty stack.**
  - $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
  - $\delta(q_0, a, a) = (q_0, aa)$
  - $\delta(q_0, b, a) = (q_1, \varepsilon)$
  - $\delta(q_1, b, a) = (q_1, \varepsilon)$
  - $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$



Construction of PDA

- Design PDA accepting $\{a^n b^n : n >= 1\}$ by **final state.**
  - $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
  - $\delta(q_0, a, a) = (q_0, aa)$
  - $\delta(q_0, b, a) = (q_1, \varepsilon)$
  - $\delta(q_1, b, a) = (q_1, \varepsilon)$
  - $\delta(q_1, \varepsilon, Z_0) = (q_f, Z_0)$
  - Final state = $\{q_f\}$



THANK YOU