**Chapter 9: PUSHDOWN AUTOMATA**

**Topic – 1: Introduction To PDA**

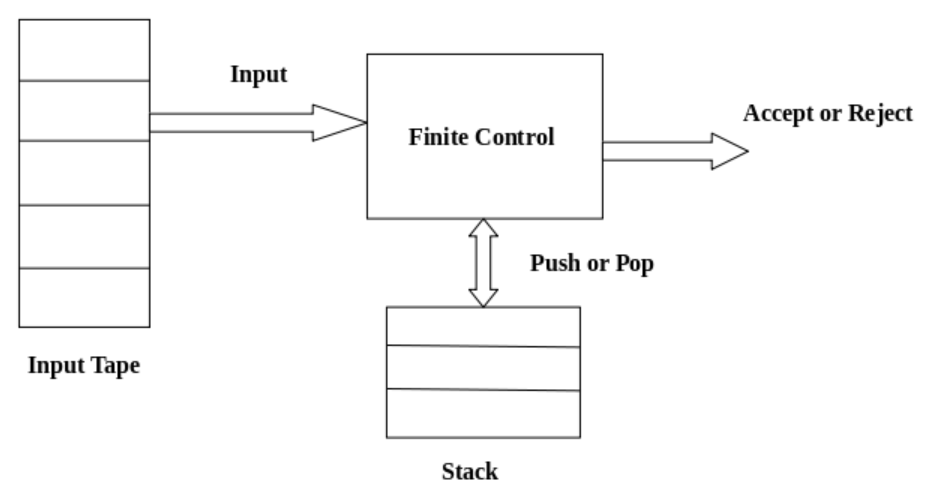
**Introduction**

* Making **PDA** **from** **CFG** is same as making **DFA** **from** **RE**.
* But unlike **DFA**, **PDA** can remember **infinite** amount of information.

**Working**

* Actually, PDA is a **NFA** with memory being managed in **stack** (**LIFO**).
* But the amount of memory that can be accessed is **limited**.
* When including a **new element** to stack, the top element has to be **popped**.
* **PDA** is more powerful than **FA** as **PDA** can accept more class of languages than **FA**, including those accepted by **FA**.

**PDA Components**



* **Finite control:** A **pointer** pointing to **one cell** at a time from input tape.
* Stack has capability to be of **infinite** in size, **doesn’t** mean there is **no** limit to the how much elements it can store.

**PDA Formal Definition**

**{Q, Σ, Γ, q0, Z, F, δ}**

**Γ = Set of stack symbols which can be pushed or popped from stack**

**Z = Start symbol present in Γ**

**Instantaneous Description (ID)**

* **ID** describes **how** an input string has to be computed.
* As per that, inputs are **pushed** or **popped** into stack.
* It is determined by 3 factors – **q**, **w**, **α**

**{q, w, α}**

**q = Current state**

**w = Remaining inputs**

**α = Stack contents**

**Turnstile Notations**

**⊢ = Turnstile notation (one move)**

**⊢\* = Sequence of moves**

**(p, b, T) ⊢ (q, w, α)**

**p = Previous state**

**b = Input symbol consumed at state q**

**T = Character on top of stack which was replaced by α**

**(State, Input, Top)**

**Example**

**Ques: Design PDA for accepting language {anb2n | n ≥ 1}. Also simulate it for the string "aaabbbbbb".**

**Ans: Means it’s a string where n number of a’s get followed by 2n number of b’s.**

**Algorithm we will apply:**

**When 'a' is read, we will insert 2 a’s.**

**When 'b' is read, we will insert 2 b’s & pop 1 'a' out.**

**ID construction:**

**δ(q0, a, Z) = δ(q0, aaZ) [Reading first 'a']**

**δ(q0, a, a) = δ(q0, aaa) [Reading non-first 'a']**

**δ(q0, b, a) = δ(q1, Ԑ) [Reading first 'b']**

**δ(q1, b, a) = δ(q1, Ԑ) [Reading string with only 'b' from state q1]**

**δ(q1, Ԑ, Z) = δ(q2, Ԑ) [Reading null string]**

**Notice how next state varies when input varies.**

**Z shows end of string.**

**PDA = {{q0, q1, q2}, {a, b}, {a, Z}, δ, q0, Z, {q2}}**

**PDA simulation for "aaabbbbbb":**

**δ(q0, aaabbbbbb, Z)**

**⊢ δ(q0, aabbbbbb, aaZ)**

**⊢ δ(q0, abbbbbb, aaaaZ)**

**⊢ δ(q0, bbbbbb, aaaaaaZ)**

**⊢ δ(q1, bbbbb, aaaaaZ)**

**⊢ δ(q1, bbbb, aaaaZ)**

**…**

**⊢ δ(q1, b, aZ)**

**⊢ δ(q1, Ԑ, Z)**

**⊢ δ(q2, Ԑ)**

**ACCEPT**

**We are not using stack for storage, but for processing relationship between various input symbols.**

**Example – II**

**Ques: Design PDA for accepting language {0n1m0n | m,n ≥ 1}**

**Ans:**

**PDA working here:**

**When the first series of 0n are read, we push a 0 to stack.**

**For 1m, we don’t affect the stack.**

**And for second series of 0n, we pop a 0 from stack.**

**δ(q0, 0, Z) = δ(q0, 0Z)**

**δ(q0, 0, 0) = δ(q0, 00)**

**δ(q0, 1, 0) = δ(q1, 0)**

**δ(q1, 1, 0) = δ(q1, 0)**

**δ(q1, 0, 0) = δ(q1, Ԑ)**

**δ(q0, Ԑ, Z) = δ(q2, Z) [Accept state]**

**As one might have noticed, that we write Z where the symbols along it show the right quantity.**

**Topic – 2: PDA Acceptance**

**Acceptance By Final State**

* **PDA** might be accepted by a **final state** after having read the whole input.

**L(PDA) = {w | (q0, w, Z) ⊢\* (p, Ԑ, Ԑ), q Є F}**

**Acceptance By Empty Stack**

* The stack might get empty for reading **starting** of some strings.

**N(PDA) = {w | (q0, w, Z) ⊢\* (p, Ԑ, Ԑ), q Є Q}**

**Acceptance By Both Final State & Empty Stack**

* If there exists a language for which **N(P1)** & **L(P2)**, that means the language accepted by **empty stack** **PDA** will be accepted by **final state** **PDA** too.
* And if there exists a language for which **L(P1)** & **N(P2)**, that means the language accepted by **final state PDA** will be accepted by **empty stack PDA** too.

**Example**

**Ques: Create PDA which accepts L over {0,1} by empty stack. It accepts string of 0s & 1s where 0s are twice as number of 1s.**

**Ans: In this, either the 0s come before 1s or 1s come before 0s. Construct the logic accordingly.**

**Topic – 3: Non-Deterministic Pushdown Automata (NPDA)**

**Introduction**

* **NPDA** is even **more powerful** than **DPDA**.
* **CFGs** accepting **DPDAs** will accept **NPDAs** too.
* But **CFGs** accepting **NPDAs** might **not** always accept **NPDAs**.

**Example**

**Ques: Design PDA for palindrome strings.**

**Ans: We will construct IDs for accepting palindrome string.**

**For pushing:**

**δ(q0, a, Z) = δ(q0, aZ)**

**δ(q0, b, Z) = δ(q0, bZ)**

**δ(q0, a, a) = δ(q0, aa)**

**δ(q0, b, a) = δ(q0, ba)**

**δ(q0, a, b) = δ(q0, ab)**

**δ(q0, b, b) = δ(q0, bb)**

**For popping:**

**δ(q0, a, a) = δ(q1, Ԑ)**

**δ(q0, b, b) = δ(q1, Ԑ)**

**δ(q1, a, a) = δ(q1, Ԑ)**

**δ(q1, b, b) = δ(q1, Ԑ)**

**δ(q1, Ԑ, Z) = δ(q2, Ԑ)**