FORMAL LANGUAGE AND AUTOMATA THEORY

UNIT-IV

SYLLABUS

1. **Pushdown Automata**
2. **Definition**
3. **Model (or) Components of PDA**
4. **Graphical Notation**
5. **Instantaneous Description**
6. **Language Acceptance of Pushdown Automata**
7. **Design of Pushdown Automata**
8. **Deterministic and Non – Deterministic PDA**
9. **Equivalence of PDA to CFG**
10. **Conversion**
11. **Two Stack PDA**
12. **Application of Pushdown Automata**

UNIT-IV

PUSHDOWN AUTOMATA

- It is a **finite automata** with extra memory called **Stack (LIFO)** which helps pushdown automata to recognize context free language.
- PDA are used in theories about what can be computed by machine.
- They are **"more capable"** than **finite state machine** but **"less capable"** than **Turing Machine**.
- Pushdown automata is a way to implement a CFG in the same way we design DFA for a regular grammar.
- A **DFA** can remember a **"finite amount of information"**, but a **"PDA"** can remember an **"Infinite amount of information"**.
- A PDA is more powerful than FA. Any language which can be acceptable by FA can also be acceptable by PDA. PDA also accepts a class of language which even cannot be accepted by FA. Thus PDA is much more superior to FA.
- Basically, a pushdown automaton is:

"Finite state machine" + "a stack"

→ **DEFINITION: -**

PDA is a generalisation of FSA and a PDA changes from **"state to state"**, **"reading input symbols"** unlike FSA, **"transitions"** also update the stack either by popping symbols (or) pushing symbols them.

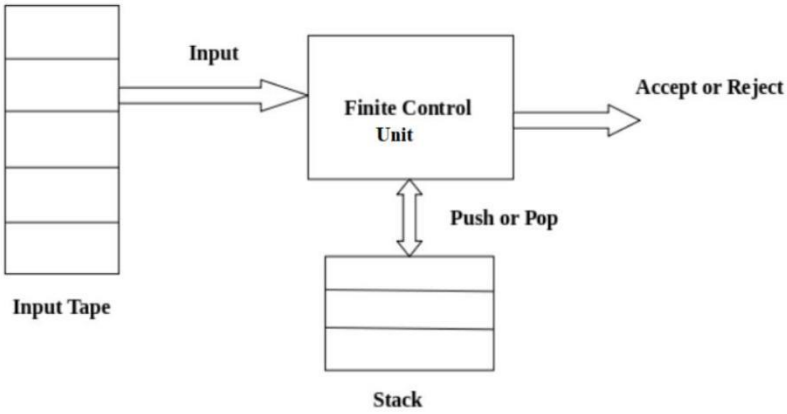
(or)

PDA is a way to represent the language class called **context free languages**. In other words PDA's are abstract devices that recognise context free languages.

→ **MODEL (OR) COMPONENTS OF PDA: -**

They are three (3) components are used in PDA

1. Input Tape
2. Finite Control Unit
3. Stack (Memory Unit)



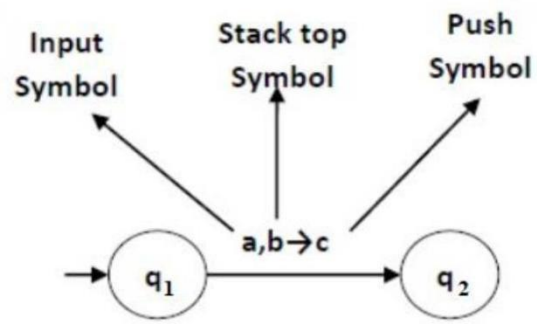
Input tape: The input tape is divided in many cells or symbols. The input head is read-only and may only move from left to right, one symbol at a time.

Finite control: The finite control has some pointer which points the current symbol which is to be read.

Stack: The stack is a structure in which we can push and remove the items from one end only. It has an infinite size. In PDA, the stack is used to store the items temporarily.

→ **GRAPHICAL NOTATION (or) TRANSITION DIAGRAM: -**

A transition in a PDA from a state **"q_1"** to **"q_2"** labeled as a, b → c



Here q_1 and q_2 is a States

At state **"q_1"**, if an input string **"a"** is encounter and top symbol of stack is **"b"**, push **"c"** on top pf the stack and move to state **"q_2"**.

→ **TRANSITION TABLE: -**

The description of operation of a PDA, for a given input string, can be represented in a tabular format called **"Transition Table"**.

Unread input	Transition	Stack	New State

→ ORDERED SEVEN (7) – TUPLES SPECIFICATION OF PDA: -

A PDA can be formally described as a 7-tuple $(Q, \sum, \delta, q_0, F, Z, \Gamma)$ −

- **Q** is the finite number of states
- $\sum$ is input alphabet
- **δ** is the transition function: $Q \times (\sum \cup \{\varepsilon\}) \times S \times Q \times S^*$
- **q_0** is the initial state ($q_0 \in Q$)
- **F** is a set of accepting states ($F \in Q$)
- **Z** is the initial stack symbol, placed on the TOS
- **Γ** is the final set of stack symbols

→ TRANSITION ON PDA: -

The transition of PDA can be represented in different ways, as follows:

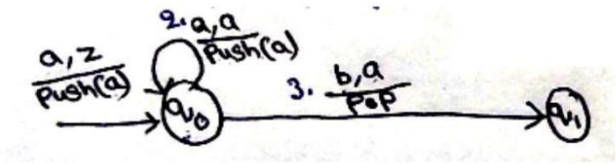
Form – 1:

δ (current state, current input symbol, current stack top) = (new state, new stack top)

Form – 2:

δ (current state, current input symbol, current stack top, operation on stack, new state)

Example: - 1). Consider a PDA, whose task is described in the transition diagram Show in below figure. The given input string (w) = aab, "z" initial symbol of stack.



Solution: -

"Transition" of the above diagram are represented using different forms as:

I^st Transition

Form-1: $\delta(q_0, a, Z) = (q_0, a)$

This means for the current state **"q_0"**, current input symbol **"a"**, if the current stack top is **"Z"**, then new state and new stack top are represented as (q0, a).

Form-2: $\delta(q_0, a, Z, Push(a), q_0)$

This means for the current state **"q_0"**, current input symbol **"a"**, if the current stack top is **"Z"**, then **"Push a"** on to stack and remain the state **"q_0".**

2nd Transition

Form-1: $\delta(q_0, a, a) = (q_0, a)$

Form-2: $\delta(q_0, a, a, Push(a), q_0)$

3rd Transition

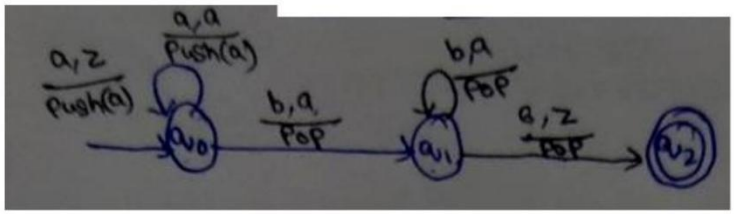
Form-1: $\delta(q_0, b, a) = (q_1, \varepsilon)$

Form-2: $\delta(q_0, b, a, pop(a), q_1)$

Transition Table: - Input String (w) = aab

Unread input	Transitions	Stack	New State
aab	------	Z	q_0
ab	(q_0, a, Z, push(a), q_0)	az	q_0
b	(q_0, a, a, push(a), q_0)	aaz	q_0
Ɛ	(q_0, b, a, pop(a), q_1)	az	q_1

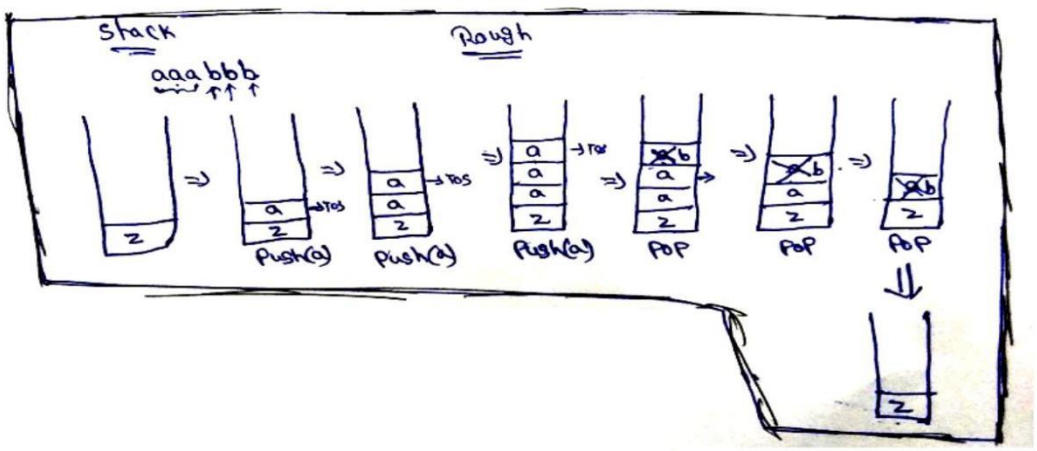
Example: -2). Consider the following transition diagram of a PDA. Input string (w) = aaabbb, **"z"** as the current stack top.



Solution: -

Transition Table: - **Input String (w) = aaabbb**

Unread input	Transitions	Stack	New State
aaabbb	------	Z	q_0
aabbb	(q_0, a, Z, push(a), q_0)	az	q_0
abbb	(q_0, a, a, push(a), q_0)	aaz	q_0
bbb	(q_0, b, a, push(a), q_0)	aaaz	q_0
bb	(q_0, b, a, pop, q_1)	aaz	q_1
b	(q_1, b, a, pop, q_1)	az	q_1
Ɛ	(q_1, b, a, pop, q_1)	z	q_1
--	(q_1, Ɛ, z, pop, q_2)	z	q_2



→ **INSTANTANEOUS DESCRIPTION (ID): -**

The ID of a PDA is represented by a **"triple"** (q, w, s).

Here q → States

w → Input string

s → Stack

"ID" is an informal notation of how PDA computes an **"input string"** and make a decision that string is accepted (or) Rejected.

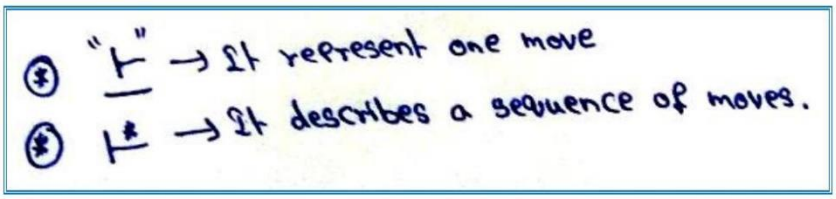
Example: - **1.** w = aabb

2. w = ababaab

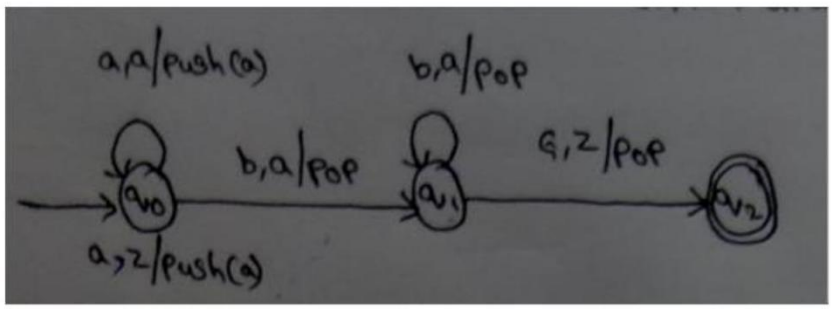
→ **TURNSTILE NOTATION: -**

It is used for connecting pairs of **ID's** that represent **"one or more moves"** of a PDA.

The process of transition is denoted by the Turnstile symbol **" ⊢ "**.



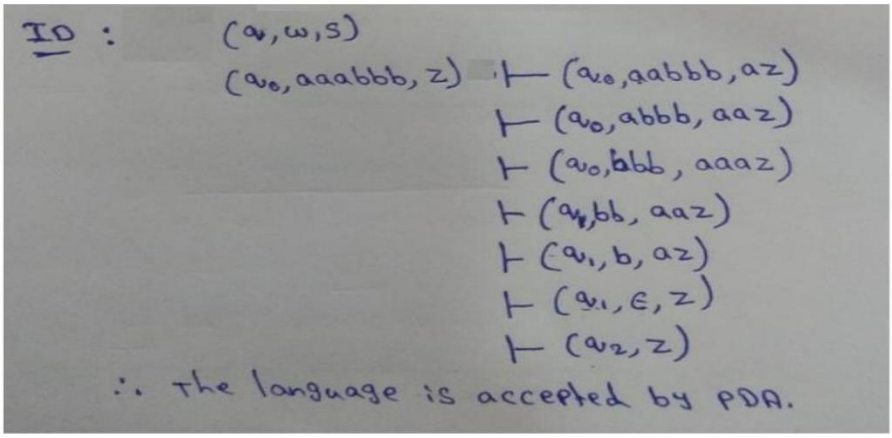
Example: -3). Consider the following transition diagram of a PDA.



Consider the input string (w) = aaabbb and **"z"** as the current stack top.

Solution: - Transition Table: - **Input String (w) = aaabbb**

Unread input	Transitions	Stack	New State
aaabbb	------	Z	q_0
aabbb	(q_0, a, Z, push(a), q_0)	az	q_0
abbb	(q_0, a, a, push(a), q_0)	aaz	q_0
bbb	(q_0, b, a, push(a), q_0)	aaaz	q_0
bb	(q_0, b, a, pop, q_1)	aaz	q_1
b	(q_1, b, a, pop, q_1)	az	q_1
Ɛ	(q_1, b, a, pop, q_1)	z	q_1
--	(q_1, Ɛ, z, pop, q_2)	z	q_2



→ LANGUAGE ACCEPTANCE OF PUSHDOWN AUTOMATA: -

A language can be accepted by Pushdown automata using two approaches:

1. **Acceptance by Final State**
2. **Acceptance by Empty Stack**

1) **Acceptance by Final State: -**

The PDA is said to accept its input by the final state if it enters any final state in zero or more moves after reading the entire input.

Let P = (Q, ∑, Γ, δ, q_0, Z, F) be a PDA. The language acceptable by the final state can be defined as:

L(PDA) = {w | (q_0, w, Z) ⊢* (p, ε, z), q ∈ F}

2) **Acceptance by Empty Stack: -**

On reading the input string from the initial configuration for some PDA, the stack of PDA gets empty.

Let P = (Q, ∑, Γ, δ, q_0, Z, F) be a PDA. The language acceptable by empty stack can be defined as:

N(PDA) = {w | (q_0, w, Z) ⊢* (p, ε, z), q ∈ Q}

Example: -4). Design a PDA for accepting a language L = {$a^n b^n$ | n>=1}

Solution: -

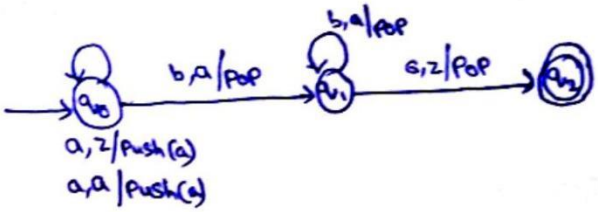
$a^n b^n$: Every String contains "n" number of a's followed by "n" number of b's.

n = 3, **$a^3 b^3$ → aaabbb**

L = {aabb, aaabbb, _ _ _ _ _ _}

Input String (w) = aaabbb

Transition Diagram: -



Transition on PDA: -

$\delta(q_0, a, z) = (q_0, a)$ (or) $(q_0, a, z, push(a), q_0)$

$(q_0, a, a, push(a), q_0)$

$(q_0, a, a, push(a), q_0)$

(q_0, b, a, pop, q_1)

(q_1, b, a, pop, q_1)

(q_1, b, a, pop, q_1)

$(q_1, \varepsilon, z, pop, q_2)$

Transition Table: - **Input String (w) = aaabbb**

Unread input	Transitions	Stack	New State
aaabbb	------	Z	q_0
aabbb	$(q_0, a, Z, push(a), q_0)$	az	q_0
abbb	$(q_0, a, a, push(a), q_0)$	aaz	q_0
bbb	$(q_0, b, a, push(a), q_0)$	aaaz	q_0
bb	(q_0, b, a, pop, q_1)	aaz	q_1
b	(q_1, b, a, pop, q_1)	az	q_1
ε	(q_1, b, a, pop, q_1)	z	q_1
--	$(q_1, \varepsilon, z, pop, q_2)$	z	q_2

PDA action for the Input String: -

a) Consider the input String (w) = aaabbb

ID: (q, w, s)

$(q_0, aaabbb, z) \vdash (q_0, aabbb, az)$

$\vdash (q_0, abbb, aaz)$

$\vdash (q_0, bbb, aaaz)$

$\vdash (q_1, bb, aaz)$

$\vdash (q_1, b, az)$

$\vdash (q_1, \varepsilon, z)$

$\vdash (q_2, z)$

So my string is accepted.

b) Consider the input string (w) = aabbb

ID: $(q_0, aabbb, z) \vdash (q_0, abbb, az)$

$\vdash (q_0, bbb, aaz)$

$\vdash (q_1, bb, az)$

$\vdash (q_1, b, z)$

$\vdash (q_1, z)$

∴ The String is not accepted, because **"q_1"** is not a final State.

→ DESIGN OF PDA: -

The basic design strategy for PDA as follows:

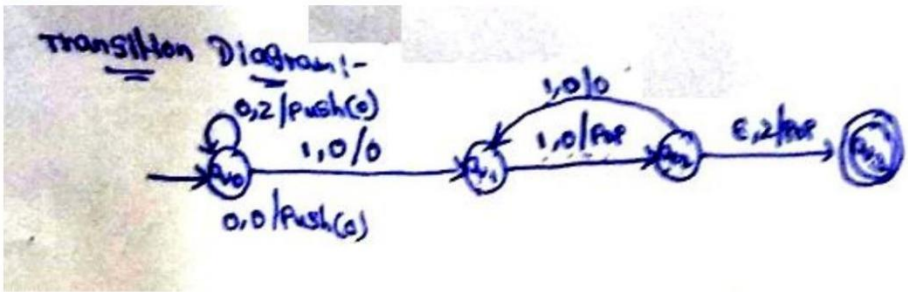
1. Understand the language properties, for which the PDA has to be design.
2. Determine the state and alphabet set required.
3. Identify the initial, accepting (final) and dead states of PDA.
4. Decide on the stack symbols required.
5. Determine the initial stack symbols from the stack symbol set.
6. For each state, decide on the transition to be made for each character of the input string.
7. For each state transition, decide on the stack operation to be performed.
8. Obtain the transition diagram and table for PDA.
9. Test, the PDA obtained on short string.

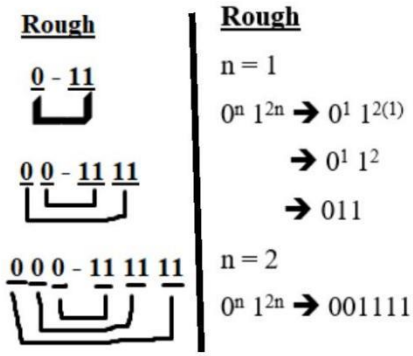
Example: -4). Design a PDA for accepting a language L = $\{0^n 1^{2n} \mid n>1\}$

Solution: - In this language **"n"** number of **0's** should be followed by **"2n"** number of **1's**.

*** **Logic: -** One Zero is corresponding two one's.

Input String (w) = 001111





This is Logic

Transition on PDA: -

$(q_0, 0, z, push(0), q_0)$

$(q_0, 0, 0, push(0), q_0)$

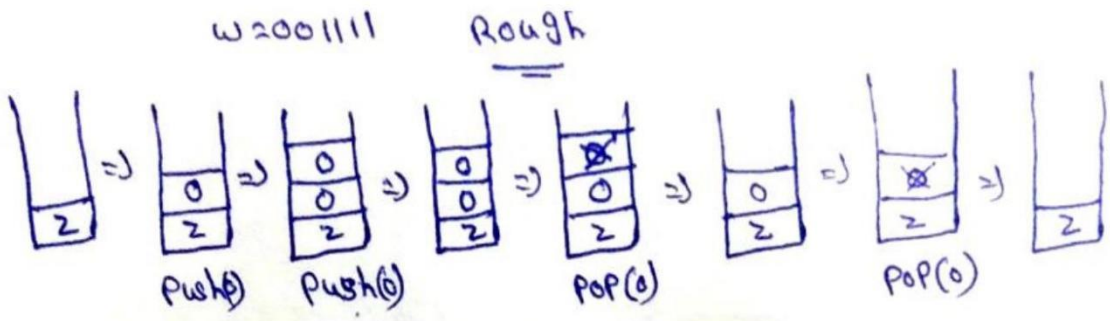
$(q_0, 1, 0, 0, q_1)$

$(q_1, 1, 0, pop(0), q_2)$

$(q_2, 1, 0, 0, q_1)$

$(q_1, 1, 0, pop(0), q_2)$

$(q_2, \varepsilon, z, pop, q_3)$



Transition Table: - **Input String (w) = 001111**

Unread input	Transitions	Stack	New State
001111	------	Z	q_0
01111	$(q_0, 0, Z, push(0), q_0)$	0z	q_0
1111	$(q_0, 0, 0, push(0), q_0)$	00z	q_0
111	$(q_0, 1, 0, 0, q_1)$	00z	q_1
11	$(q_1, 1, 0, pop(0), q_2)$	0z	q_2
1	$(q_2, 1, 0, 0, q_1)$	0z	q_1
Ɛ	$(q_1, 1, 0, pop(0), q_2)$	z	q_2
--	$(q_2, Ɛ, z, pop, q_3)$	z	q_3

PDA action for the Input String: -

a) Consider the input String (w) = 001111

ID: (q, w, s)

$(q_0, 001111, z) \vdash (q_0, 01111, 0z)$

$\vdash (q_0, 1111, 00z)$

$\vdash (q_1, 111, 00z)$

$\vdash (q_2, 11, 0z)$

$\vdash (q_1, 1, 0z)$

$\vdash (q_2, Ɛ, z)$

$\vdash (q_3, z)$

∴ So, the String is accepted.

b) To show that input String (w) = 001

ID: (q, w, s)

$(q_0, 001, z) \vdash (q_0, 11, 0z)$

$\vdash (q_0, 1, 00z)$

$\vdash (q_1, Ɛ, 00z)$

∴ So, this String is Rejected.

Example: -5). Design a PDA for accepting a language $L = \{ww^R \mid w \in (a+b)^*\}$

Solution: -

w → words with any combination of **"a"** and **"b"** including null string.

w^R → words having reverse of **"w"**.

$(a+b)^* = a^* + b^*$

$= a^* \cup b^*$

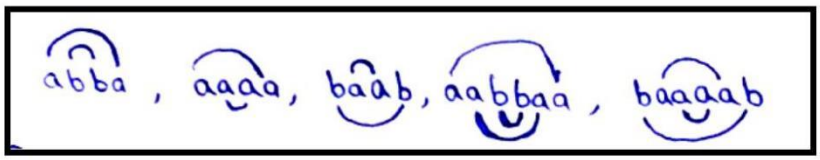
$= \{Ɛ, a, aa, aaa, \_\_\_\_\_\} \cup \{Ɛ, b, bb, bbb, \_\_\_\_\_\}$

$= \{Ɛ, a, b, aa, bb, aaa, bbb, \_\_\_\_\_\}$

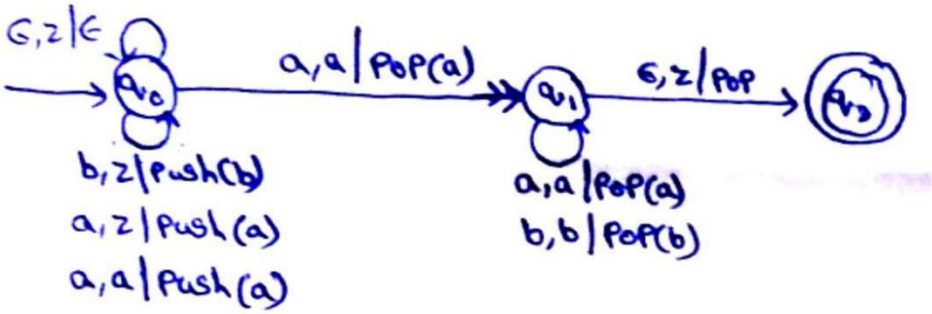
$L = \{Ɛ, aa, bb, abba, aaaa, baab, aabbaa, baaaab, \_\_\_\_\_\}$

Input String (w) = baaaab

***Logic: -



Transition Diagram: -



Transition Table: - **Input String (w) = baaaab**

Unread Input	Transition	Stack	New States
baaaab	-	z	q_0
aaaab	$(q_0, b, z, Push(b), q_0)$	bz	q_0
aaab	$(q_0, a, z, Push(a), q_0)$	abz	q_0
aab	$(q_0, a, a, Push(a), q_0)$	aabz	q_0
ab	$(q_0, a, a, Pop(a), q_1)$	abz	q_1
b	$(q_1, a, a, Pop(a), q_1)$	bz	q_1
ϵ	$(q_1, b, b, Pop(b), q_1)$	z	q_1
-	$(q_1, \epsilon, z, Pop, q_2)$	z	q_2

PDA action for the Input String: -

a) Consider the input String (w) = baaaab

ID: (q, w, s)

$(q_0, baaaab, z) \vdash (q_0, aaaab, bz)$

$\vdash (q_0, aaab, abz)$

$\vdash (q_0, aab, aabz)$

$\vdash (q_0, ab, abz)$

$\vdash (q_1, b, bz)$

$\vdash (q_1, \epsilon, z)$

$\vdash (q_2, z)$

∴ So, the String is accepted.

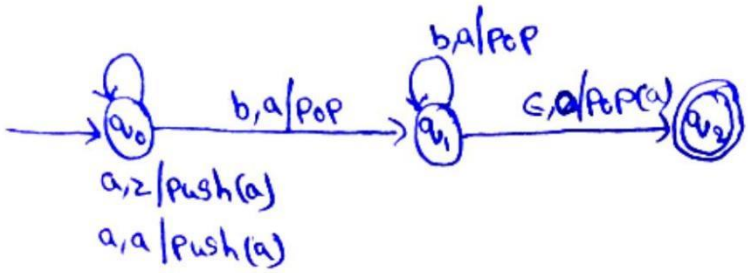
Example: -6). Design a PDA for accepting a language L = {$a^n b^m$ | n>=m, m>=1}

Solution: - At least **"1b"** is there and **"2a's"** should be there.

L = {aab, aaabbb, aaaabbbb, _ _ _ _ _ }

Input String (w) = aaabbb

Transition Diagram: -



Transition Table: -

Unread input	Transitions	Stack	New State
aaabbb	------	Z	q_0
aabbb	(q_0, a, Z, push(a), q_0)	az	q_0
abbb	(q_0, a, a, push(a), q_0)	aaz	q_0
bbb	(q_0, b, a, push(a), q_0)	aaaz	q_0
bb	(q_0, b, a, pop, q_1)	aaz	q_1
b	(q_1, b, a, pop, q_1)	az	q_1
Ɛ	(q_1, b, a, pop, q_1)	z	q_1
--	(q_1, Ɛ, z, pop, q_2)	z	q_2

PDA action for the Input String: -

a) Consider the input String (w) = aaabbb

ID: (q, w, s)

(q_0, aaabbb, z) ⊢ (q_0, aabbb, az)

⊢ (q_0, abbb, aaz)

⊢ (q_0, bbb, aaaz)

⊢ (q_1, bb, aaz)

⊢ (q_1, b, az)

⊢ (q_1, Ɛ, z)

⊢ (q_2, z)

So my string is accepted.

b) Consider the input string (w) = aabbb

ID: (q_0, aabbb, z) ⊢ (q_0, abbb, az)

⊢ (q_0, bbb, aaz)

⊢ (q_1, bb, az)

$\vdash (q_1, b, z)$

$\vdash (q_1, z)$

∴ The String is not accepted, because **"q_1"** is not a final State.

Example: -7). Design a PDA for accepting a language L = {wCw^R | w Ɛ (0+1)*}

Solution: - w → words with any combination of **"0"** & **"1"** including null string.

w^R → words having reverse of "w".

(0+1)* = 0* + 1*

= 0* U 1*

= {Ɛ, 0, 00, 000, _ _ _ _ _ _} U {Ɛ, 1, 11, 111, _ _ _ _ _ _}

= {Ɛ, 0, 1, 00, 11, 000, 111, _ _ _ _ _ _}

L = {Ɛ, 0C0, 1C1, 00C00, 001C100, 110C011,_ _ _ _ _ __}

Input String (w) = 001C100.

Now Construct Transition Diagram, Transition table and PDA action for the Input String.

→ **DETERMINISTIC AND NON – DETERMINISTIC PDA: -**

DETERMINISTIC PDA: -

PDA is deterministic, if each input string can only be processed by the machine in **only one way**, i.e., for the **"same input symbol"** and **"same stack symbol"**, there must be only one choice.

Formally, a PDA P = (Q, Σ, δ, q_0, F, Z, Γ) is deterministic if

i. δ (q, a, z) has only one element.

ii. δ (q, Ɛ, z) is not empty, then δ (q, a, z) should be empty.

If conditions (i) & (ii) are satisfied, then the PDA is deterministic, otherwise PDA is nondeterministic.

Example: - The PDA for L = {$a^n b^n$ | n>=1} is deterministic.

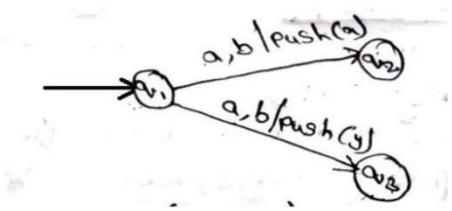
NON – DETERMINISTIC PDA: -

PDA is nondeterministic, if there is same string that can be processed by it in **"more than one way"**.

There are two (2) types of nondeterministic PDA (it occurs two types of moves):

i. When a state emits **two (or) more edges** labelled with the "**same input symbol"** and "**same stack symbol**".

Example: -



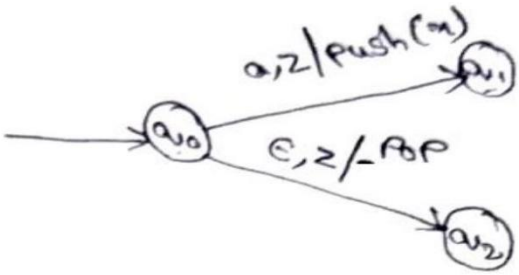
Solution: -

$\delta (q_1, a, b) = (q_2, x)$ (or) $(q_1, a, b, push(x), q_2)$

$\delta (q_1, a, b) = (q_3, y)$ (or) $(q_1, a, b, push(y), q_3)$

ii. When a state emits **two edges** labelled with the "**same stack symbol"**, where one input symbol is **'Ɛ'** and the other input symbol is not.

Example: -



Solution: -

$(q_0, a, z, push(x), q_1)$

$(q_0, Ɛ, z, pop, q_2)$

Example: -

1. Design a PDA for accepting a language $L = \{w \in \{a, b\}^* \mid n_a = n_b\}$

2. Design a PDA for accepting a language $L = \{ww^R \mid w \in (a, b)^*\}$ a language of palindrome with even length of words..

→ **EQUIVALENCE OF PDA TO CFG: -**

The Context free grammar **"G"** and pushdown automata **"P"** are said to be equivalent.

L(G) = L(P)

In other words, equivalence of PDA and CFG means:

1. The class of language accepted by context free grammar is exactly the same as the class of languages accepted by PDA. It is possible to convert any context free grammar to PDA, Such that L(G) = L(P).
2. The language accepted by PDA is exactly the same as the language accepted by context free grammar, it is possible to convert any PDA to context free grammar, such that L(P) = L(G).

1) Conversion from Context Free Grammar to PDA: -

There are set of rules:

Step: - 1). We have to convert the given production of CFG into GNF.

Step: - 2). The PDA will only have one state {q}.

Step: - 3). The initial symbol of CFG will be initial symbol in the PDA.

Step: - 4). For **"Non-Terminal Symbol"** we have to add the following rule:

A → a

δ (q, Ɛ, A) = (q, a)

So, we have to Convert

Step: - 5). For each **"Terminal Symbol"** we have to add the following rule:

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

Example: -1. Construct a PDA equivalent to following CFG productions:

S → aAA

A → aS | bS | a

Solution: -

Step: - 1) The given CFG is in GNF form

Step: - 4) S → aAA

$\delta (q, \varepsilon, S) = (q, aAA)$

A → aS | bS | a

$\delta (q, \varepsilon, A) = \{(q, aS) | (q, bS) | (q, a)\}$

Step: - 5) There are two (2) Terminal Symbols **"a"**, **"b"**.

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

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

So, this is a equivalent PDA, for given CFG.

Example: -2. Construct a PDA equivalent to following CFG productions:

S → aBB

B → 0S | 1S | 0

Solution: -

Step: - 1) The given CFG is in GNF form

Step: - 4) S → 0BB

$\delta (q, \varepsilon, S) = (q, 0BB)$

B → 0S | 1S | 0

$\delta (q, \varepsilon, B) = \{(q, 0S) | (q, 1S) | (q, 0)\}$

Step: - 5) There are two (2) Terminal Symbols **"0"**, **"1"**.

$\delta (q, 0, 0) = (q, \varepsilon)$

$\delta (q, 1, 1) = (q, \varepsilon)$

So, this is a equivalent PDA, for given CFG.

Example: -3. Construct a PDA for the grammar

S → aA

A →aABD | bB | a

B → b

D → d

Solution: -

Step: - 1) The given CFG is in GNF form

Step: - 4) S → aA

δ (q, Ɛ, S) = (q, aA)

A → aABD | bB | a

δ (q, Ɛ, A) = {(q, aABD) | (q, bB) | (q, a)}

B → b

δ (q, Ɛ, B) = (q, b)

D → d

δ (q, Ɛ, D) = (q, d)

Step: - 5) There are two (2) Terminal Symbols **"a"**, **"b"**,**"d"**.

δ (q, a, a) = (q, Ɛ)

δ (q, b, b) = (q, Ɛ)

δ (q, d, d) = (q, Ɛ)

So, this is a equivalent PDA, for given CFG.

Example: - 4. Construct PDA for the following CFG

S → aABB | aAA

A → aBB | a

B → bBB | A

Solution: - Elimination of Unit Production: -

B → A < aBB / a

B → aBB | a

∴ After eliminating Unit Production B → A

CFG in GNF is S → aABB | aAA

A → aBB | a

B → bBB | aBB | a

Step: - 1. The given CFG is in GNG form

Step: - 4. δ (q, Ɛ, S) = (q, aABB)

δ (q, Ɛ, S) = (q, aAA)

δ (q, Ɛ, A) = (q, aBB)

δ (q, Ɛ, A) = (q, a)

δ (q, Ɛ, B) = (q, bBB)

δ (q, Ɛ, B) = (q, aBB)

δ (q, Ɛ, B) = (q, a)

Step: - 5. There are two (2) terminal symbols **"a"**, **"b"**.

δ (q, a, a) = (q, Ɛ)

δ (q, b, b) = (q, Ɛ)

∴ The Pushdown Automata Transitions are:

δ (q, Ɛ, S) = (q, aABB)

δ (q, Ɛ, S) = (q, aAA)

δ (q, Ɛ, A) = (q, aBB)

δ (q, Ɛ, A) = (q, a)

δ (q, Ɛ, B) = (q, bBB)

δ (q, Ɛ, B) = (q, aBB)

δ (q, Ɛ, B) = (q, a)

δ (q, a, a) = (q, Ɛ)

δ (q, b, b) = (q, Ɛ)

Example!- 5) construct PDA for the following CFG.

$S \to 0S1$

$S \to A$

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

Solution!-

First of all

Elimination of ϵ-Production!-

Nullable variable : {A}

Direct | Indirect

$A \to \epsilon$ × (direct)

$S \to A \Rightarrow \epsilon$ (Indirect)

$S \to 0S1 \Rightarrow 0\epsilon1 \Rightarrow 01$ (Indirect)

$A \to 1A0 \Rightarrow 1\epsilon0 \Rightarrow 10$ (Indirect)

$A \to S \Rightarrow A \Rightarrow \epsilon$ (Indirect)

CFG = $S \to 0S1 \mid 01$

$S \to A$

$A \to 1A0 \mid 10 \mid S$

Elimination of unit Production!-

$S \to A \to 1A0 \mid 10$

$A \to S \to 0S1 \mid 01$

CFG = $S \to 0S1 \mid 01$ ✓

$S \to 1A0 \mid 10$ ✓

$A \to 1A0 \mid 10$ ✓

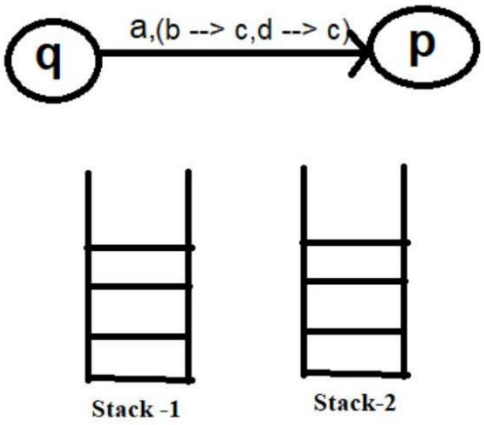
$A \to 0S1 \mid 01$ ✓

→ **TWO STACK PDA: -**

- It is similar to a PDA, but it has two stack instead of one stack.
- A machine using two pushdown automata accept the recursively enumerable language (or) Turing machine.

Definition: - A Two – Stack Pushdown Automata (2-stack PDA) is similar to a PDA, but it has two stacks instead of one. In each transition, we must denote the POP and PUSH action on both stacks.

Example:



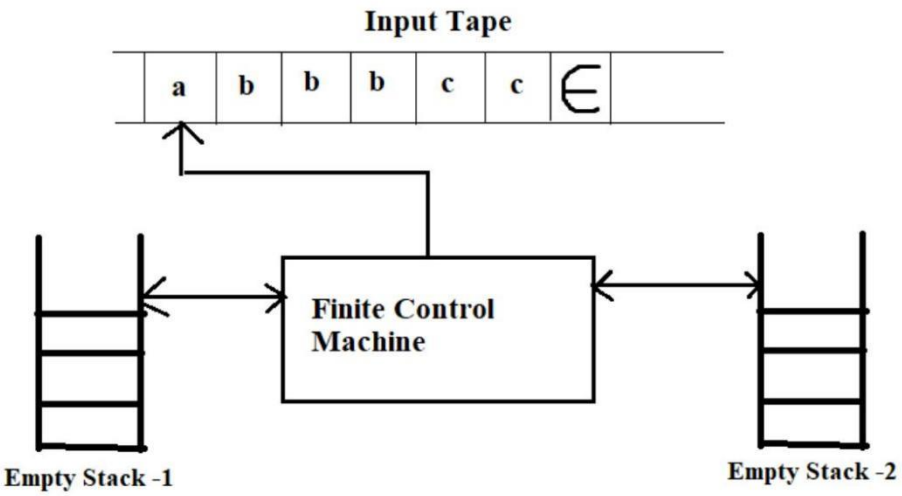
The above diagram means State **"q"** & State **"p"** from **"q to p"** we read input string **"a"** is encounter with condition **b → c, d → c**.

1. **"b"** is popped off the first stack and **"c"** is pushed on the first stack.
2. **"d"** is popped off the second stack and **"c"** is pushed on the second stack.

→ **Model (or) Components of 2 stack PDA (or) Block Diagram: -**

There are three (3) components are used 2 stack pushdown Automata:

1. The state of finite control.
2. The input symbol read.
3. The TOP stack symbol on each of its stack.



→ ORDERED NINE (9) – TUPLES SPECIFICATION OF PDA: -

A 2-stack PDA can be formally described as a 9-tuple.

$M = (Q, \sum, \delta, q_0, F, \Gamma_1, \Gamma_2, Z_0, Z_1)$

- **Q** is the finite number of states
- $\sum$ is input alphabet
- **δ** is the transition function

  δ: Q x $\sum$ x {ε}) x Γ_1 x Γ_2 → Q × Γ_1* x Γ_2*
- **q_0** is the initial state
- **F** is a set of accepting states
- **Z** is the initial stack symbol, placed on the TOS
- **Γ_1** is the final set of stack1 symbols
- **Γ_2** is the final set of stack2 symbols
- **Z_0** is the initial stack1 symbol
- **Z_1** is the initial stack2 symbol

Example: - Construct a 2 stack PDA which accepts the following language $L = \{a^n b^n c^n \mid n>=1\}$

Solution: - $L = \{a^n b^n c^n \mid n>=1\}$

n = 1, 2, 3, 4, _ _ _ _ _ __

n =1; $a^n b^n c^n$ = abc

n =2; $a^n b^n c^n$ = aabbcc

n =3; $a^n b^n c^n$ = aaabbbccc

L = {abc, aabbcc, aaabbbccc, _ _ _ _ _ _ _}

Input string (w) = aabbcc

Transitions: -

$\delta (q_0, a, z_0, z_1) = (q_0, a, z_1)$

$\delta (q_0, a, a, z_1) = (q_0, a, z_1)$

$\delta (q_0, b, a, z_1) = (q_1, a, b)$

$\delta (q_1, b, a, b) = (q_1, a, b)$

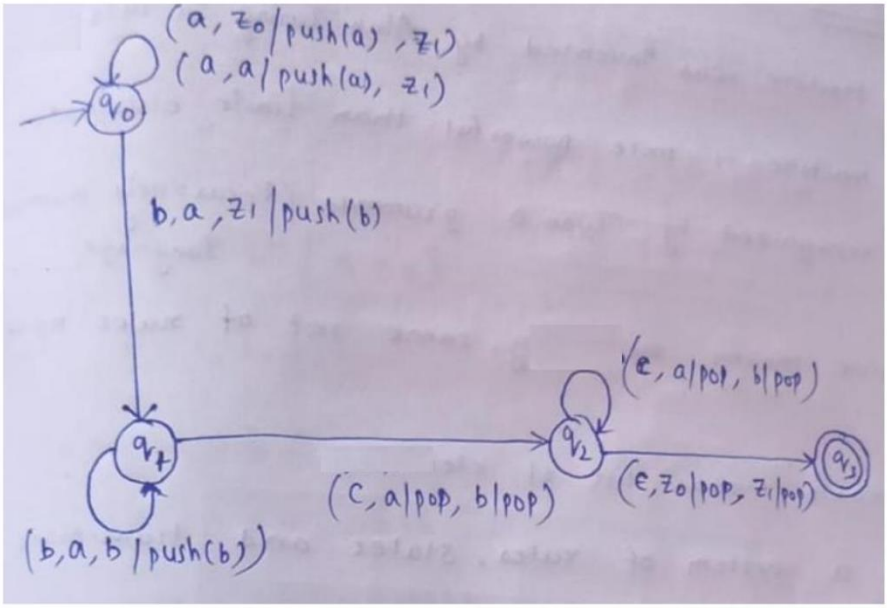
$\delta (q_1, c, a, b) = (q_2, a, b)$

$\delta (q_1, c, a, b) = (q_2, a, b)$

$\delta (q_2, c, a, b) = (q_2, z_0, z_1)$

$\delta (q_2, \varepsilon, z_0, z_1) = (q_3, \varepsilon, \varepsilon)$

Transition Diagram: -



2 stack PDA action for the Input String:

Consider the input string (w) = aabbcc

ID: (q, w, s)

$(q_0, aabbcc, s) = \vdash (q_0, abbcc, az_0, z_1)$

$= \vdash (q_0, bbcc, aaz_0, z_1)$

$= \vdash (q_1, bcc, aaz_0, bz_1)$

$= \vdash (q_1, cc, aaz_0, bbz_1)$

$= \vdash (q_2, c, az_0, bz_1)$

$= \vdash (q_2, \varepsilon, z_0, z_1)$

$= \vdash (q_3, \varepsilon)$

∴ The input string is accepted the final state.

→ **APPLICATION OF PUSHDOWN AUTOMATA: -**

- Used for deriving a string from the grammar.
- Used for designing Top-down parser and Bottom-up parser in compiler design.
- Used in evaluation of the arithmetic expressions.
- Used for solving the Tower of Hanoi problem.
- It works on regular grammar and context free grammar.
- It accepts regular language and context free language.
- It has remembering capability by maintaining a stack.
- It is more powerful than Finite Automata.