FORMAL LANGUAGE AND AUTOMATA THEORY

UNIT – IV: Pushdown Automata

Pushdown Automata, Definition, Model, Graphical Notation, Instantaneous Description Language Acceptance of pushdown Automata, Design of Pushdown Automata, Deterministic and Non – Deterministic Pushdown Automata, Equivalence of Pushdown Automata and Context Free Grammars Conversion, Two Stack Pushdown Automata, Application of Pushdown Automata.

Pushdown Automata, Definition, Model, Graphical Notation

Basic Structure of PDA

A pushdown automaton is a way to implement a context- free grammar in a similar 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.

Basically a pushdown automaton is −

"Finite state machine" + "a stack"

A pushdown automaton has three components −

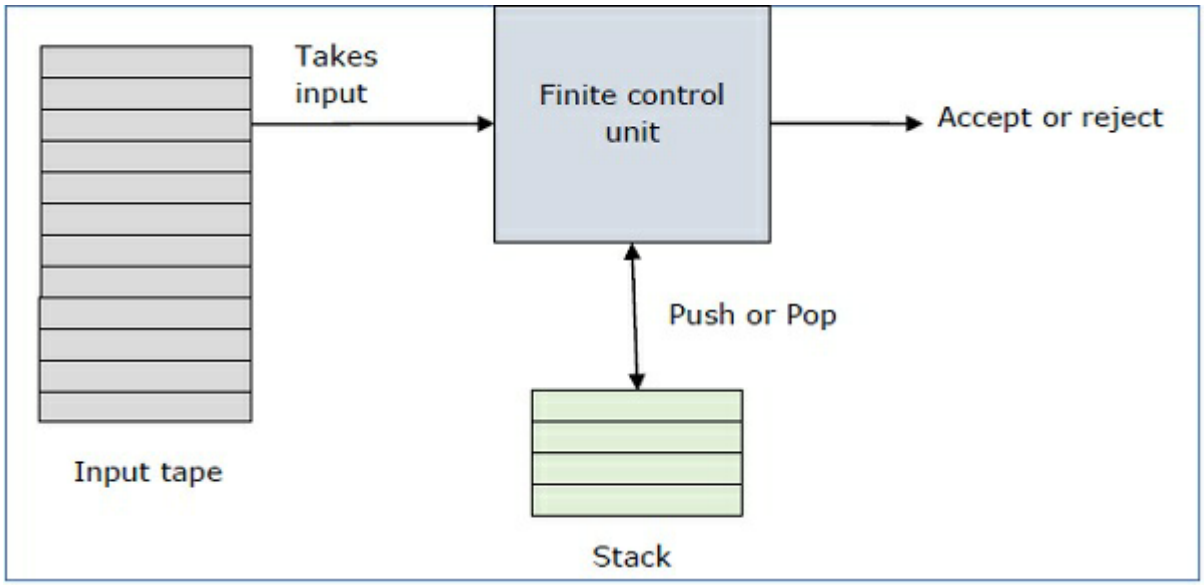
- an input tape,
- a control unit, and
- a stack with infinite size.

The stack head scans the top symbol of the stack.

A stack does two operations −

- **Push** − a new symbol is added at the top.
- **Pop** − the top symbol is read and removed.

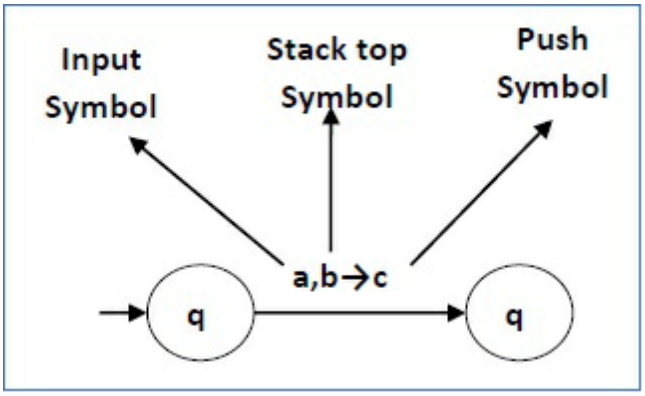
A PDA may or may not read an input symbol, but it has to read the top of the stack in every transition.



A PDA can be formally described as a 7- tuple (Q, ∑ , S, δ , q_0, I, F) –

- **Q** is the finite number of states
- **∑** is input alphabet
- **S** is stack symbols
- **δ** is the transition function: Q × (∑ ∪ {ε }) × S × Q × S*
- **q_0** is the initial state (q_0 ∈ Q)
- **I** is the initial stack top symbol (I ∈ S)
- **F** is a set of accepting states (F ∈ Q)

The following diagram shows a transition in a PDA from a state q_1 to state q_2, labeled as a,b → c –



This means at state **q_1**, if we encounter an input string **' a'** and top symbol of the

stack is **‘ b’** , then we pop **‘ b’** , push **‘ c’** on top of the stack and move to state q_2.

Terminologies Related to PDA

Instantaneous Description

The instantaneous description (ID) of a PDA is represented by a triplet (q, w, s) where

- **q** is the state
- **w** is unconsumed input
- **s** is the stack contents

Turnstile Notation

The "turnstile" notation is used for connecting pairs of ID's that represent one or many moves of a PDA. The process of transition is denoted by the turnstile symbol "⊢".

Consider a PDA (Q, ∑, S, δ, q_0, I, F). A transition can be mathematically represented by the following turnstile notation –

```
(p, aw, Tβ ) ⊢ (q, w, α b)
```

This implies that while taking a transition from state **p** to state **q**, the input symbol **‘ a’** is consumed, and the top of the stack **‘ T’** is replaced by a new string **‘ α ’** .

Note – If we want zero or more moves of a PDA, we have to use the symbol (⊢*) for it.

IMP Questions

1Q. Define PDA.

2Q. Define the transition function of a PDA.

3Q. Write down the basic structure of Push down Automata.

4Q. Discuss the PDA acceptable Language.

Important and Previous JNTU Questions:

Q.1. What is push down Automata? Show how context free language is accepted by push down automata. (8)

Instantaneous Description Language Acceptance of pushdown Automata, Design of Pushdown Automata

Final State Acceptability

In final state acceptability, a PDA accepts a string when, after reading the entire string, the PDA is in a final state. From the starting state, we can make moves that end up in a final state with any stack values. The stack values are irrelevant as long as we end up in a final state.

For a PDA (Q, ∑ , S, δ , q_0, I, F), the language accepted by the set of final states F is –

L(PDA) = {w | (q_0, w, I) ⊢* (q, ε , x), q ∈ F}

for any input stack string **x**.

Empty Stack Acceptability

Here a PDA accepts a string when, after reading the entire string, the PDA has emptied its stack.

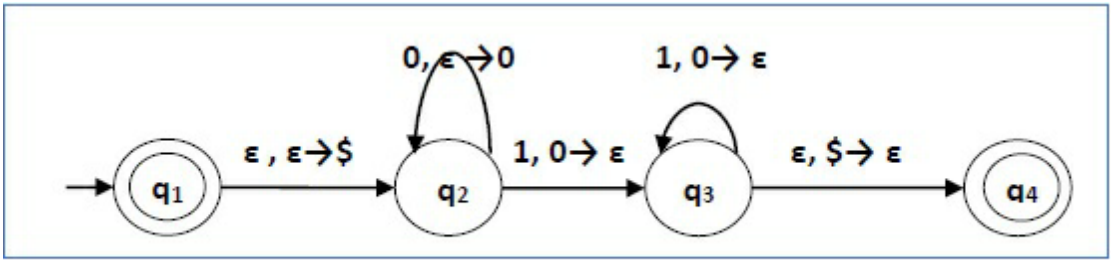
For a PDA (Q, ∑ , S, δ , q_0, I, F), the language accepted by the empty stack is –

L(PDA) = {w | (q_0, w, I) ⊢* (q, ε , ε), q ∈ Q}

Example

Construct a PDA that accepts **L = {$0^n 1^n$ | n ≥ 0}**

Solution



PDA for L= {$0^n 1^n$ | n≥0}

This language accepts L = {ε , 01, 0011, 000111, ... }

Here, in this example, the number of ' **a**' and ' **b**' have to be same.

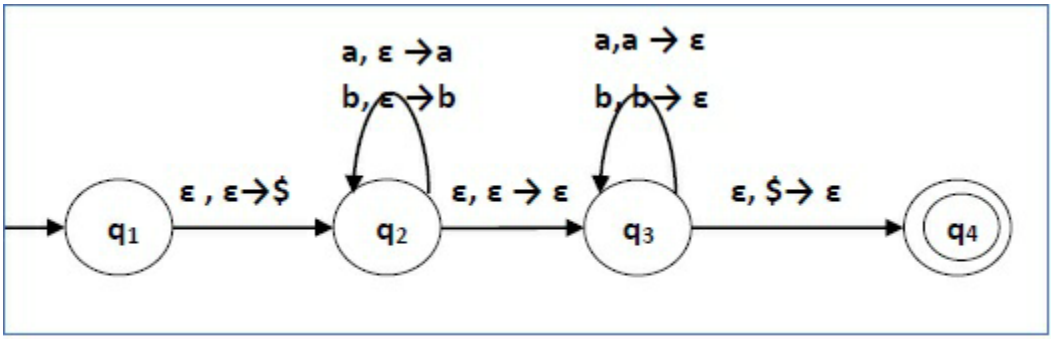
- Initially we put a special symbol ' **$**' into the empty stack.

- Then at state **q_2**, if we encounter input 0 and top is Null, we push 0 into stack. This may iterate. And if we encounter input 1 and top is 0, we pop this 0.
- Then at state **q_3**, if we encounter input 1 and top is 0, we pop this 0. This may also iterate. And if we encounter input 1 and top is 0, we pop the top element.
- If the special symbol ' $' is encountered at top of the stack, it is popped out and it finally goes to the accepting state q_4.

Example

Construct a PDA that accepts L = { ww^R | w = (a+b)* }

Solution



PDA for L= {ww^R | w = (a+b)*}

Initially we put a special symbol ' $' into the empty stack. At state **q_2**, the **w** is being read. In state **q_3**, each 0 or 1 is popped when it matches the input. If any other input is given, the PDA will go to a dead state. When we reach that special symbol ' $' , we go to the accepting state **q_4**.

IMP Questions

1Q. Define the acceptance of a PDA.

2Q. Define Final State Acceptance of a PDA.

3Q. Design a PDA For the language L = { ww^R | w = (ab) }

4Q. Discuss the PDA with a suitable Example.

Important and Previous JNTU Questions:

Q.1. Write a short note on PDA with an example. [Regular Examinations, May/June – 2015]

Deterministic and Non – Deterministic Pushdown Automata

A deterministic PDA must have only one transition for any given pair of input symbol/ stack symbol.
A non- deterministic PDA (NPDA) may have no transition or several transitions defined for a particular input symbol/stack symbol pair.
In an NPDA, there may be several paths to follow to process a given input string. Some of the paths may result in accepting the string. Other paths may end in a non- accepting state. An NPDA can " guess" which path to follow through the machine in order to accept a string.

Deterministic PDA

Let M = (Q, S, G, q_0, z, A, d), be a pushdown automaton. M is *deterministic* if there is no configuration for which M has a choice of more than one move. In other words, M is deterministic if it satisfies both of the following:

1. For any $q \in Q$, $a \in S \cup \{l\}$, and $X \in G$, the set d(q, a, X) has at most one element.

2. For any $q \in Q$ and $X \in G$, if $d(q, l, X) \neq \varnothing$, then $d(q, a, X) = \varnothing$ for every $a \in S$.

A language L is a deterministic context- free language if there is a deterministic PDA (DPDA) accepting L

Important Questions

1Q. Define the Deterministic PDA.

2Q. Discuss how a Deterministic PDA is Different from a Non Deterministic PDA?

Important and Previous JNTU Questions:

Q.1. Consider a language L* , where L={ab, cd} with ∑ ={a,b,c,d}. (i) Write all words in L* that have six or less letters/symbols (ii) What is the shortest string in ∑ * that is not in the language L* ?

[Regular Examinations, May/June – 2015]

Equivalence of Pushdown Automata and Context Free Grammars Conversion

If a grammar **G** is context- free, we can build an equivalent nondeterministic PDA which accepts the language that is produced by the context- free grammar **G**. A

parser can be built for the grammar **G**.

Also, if **P** is a pushdown automaton, an equivalent context- free grammar G can be constructed where

L(G) = L(P)

In the next two topics, we will discuss how to convert from PDA to CFG and vice versa.

Algorithm to find PDA corresponding to a given CFG

Input – A CFG, G = (V, T, P, S)

Output – Equivalent PDA, P = (Q, ∑ , S, δ , q_0, I, F)

Step 1 – Convert the productions of the CFG into GNF.

Step 2 – The PDA will have only one state {q}.

Step 3 – The start symbol of CFG will be the start symbol in the PDA.

Step 4 – All non- terminals of the CFG will be the stack symbols of the PDA and all the terminals of the CFG will be the input symbols of the PDA.

Step 5 – For each production in the form **A → aX** where a is terminal and **A, X** are combination of terminal and non- terminals, make a transition **δ (q, a, A)**.

Problem

Construct a PDA from the following CFG.

G = ({S, X}, {a, b}, P, S)

where the productions are –

S → XS | ε , A → aXb | Ab | ab

Solution

Let the equivalent PDA,

P = ({q}, {a, b}, {a, b, X, S}, δ , q, S)

where δ –

δ (q, ε , S) = {(q, XS), (q, ε)}

δ (q, ε , S) = {(q, ε)}

δ (q, ε , X) = {(q, aXb), (q, Xb), (q, ab)}

δ (q, ε , X) = {(q, Xb)}

δ (q, ε , X) = {(q, ab)}

δ (q, a, a) = {(q, ε)}

δ (q, 1, 1) = {(q, ε)}

Algorithm to find CFG corresponding to a given PDA

Input – A CFG, G = (V, T, P, S)

Output – Equivalent PDA, P = (Q, $\sum$, S, δ , q_0, I, F) such that the non- terminals of the grammar G will be $\{X_{wx} \mid w,x \in Q\}$ and the start state will be $A_{q0,F}$.

Step 1 – For every w, x, y, z $\in$ Q, m $\in$ S and a, b $\in$ $\sum$, if δ (w, a, ε) contains (y, m) and (z, b, m) contains (x, ε), add the production rule $X_{wx} \rightarrow a X_{yz} b$ in grammar G.

Step 2 – For every w, x, y, z $\in$ Q, add the production rule $X_{wx} \rightarrow X_{wy} X_{yx}$ in grammar G.

Step 3 – For w $\in$ Q, add the production rule $X_{ww} \rightarrow \varepsilon$ in grammar G.

Class Work:
1Q. Write an Algorithm to find CFG corresponding to a given PDA

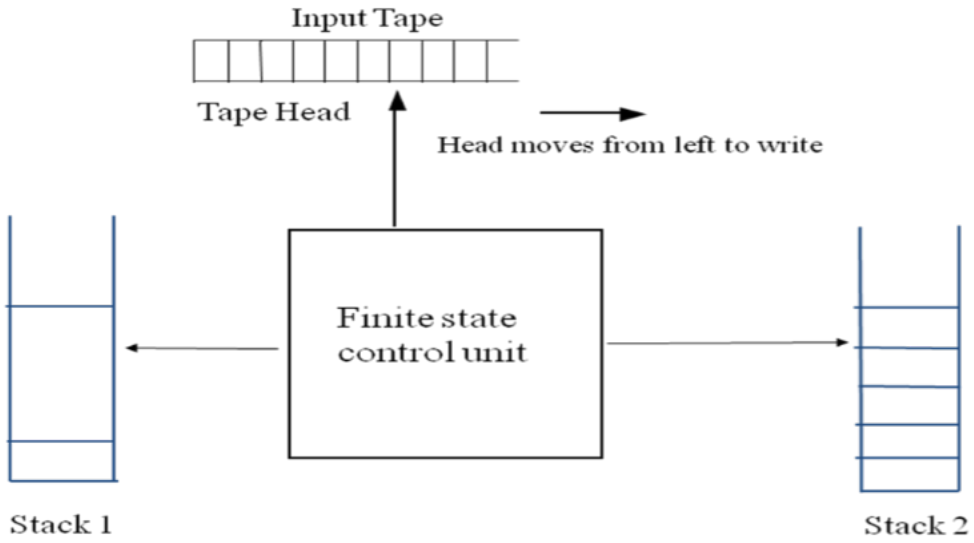
Homework:
1Q. Write an Algorithm to find PDA corresponding to a given CFG

Important and Previous JNTU Questions:
Q.1. What is a context free Language? Give examples? Write about the properties of context free languages? [Regular Examinations, May/June – 2015]

Two Stack Pushdown Automata

- A Turing machine can accept languages not accepted by any PDA with one stack.
- The strength of pushdown automata can be increased by adding additional (extra) stacks.

- Actually, a PDA with two stacks has the same computation power as a Turing Machine.



Two- Stack PDA is a computational model based on the generalization of Pushdown Automata (PDA).

Non- deterministic Two- Stack PDA is equivalent to a deterministic Two- Stack PDA.

The move of the Two- Stack PDA is based on

The state of the finite control.

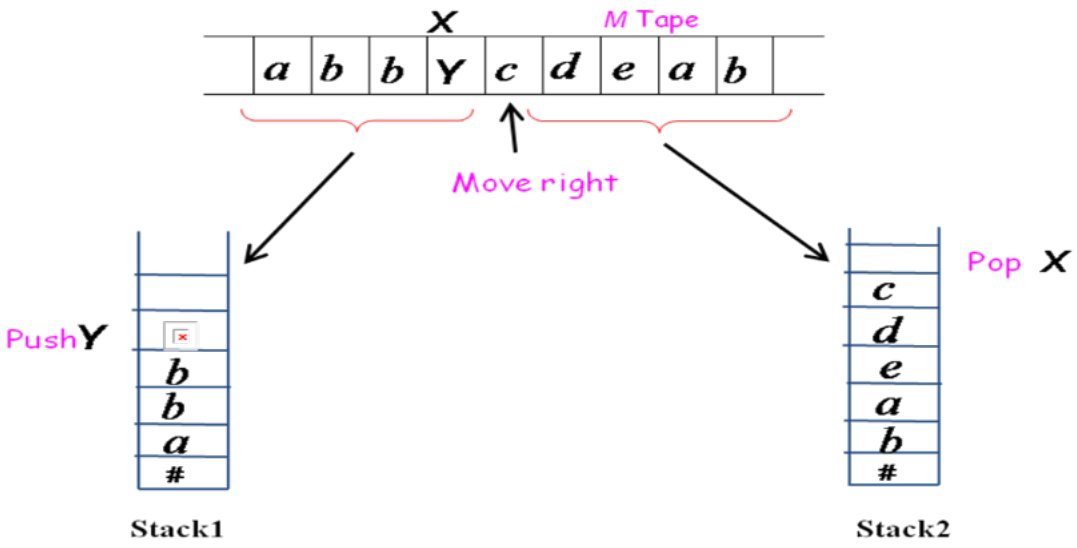
The input symbol read.

The top stack symbol on each of its stacks.

The idea is that the first stack can hold what is to the left of the head, while the second stack holds what is to the right of the head.

When we want to get a symbol which is in the first stack , we will pop all the content in the first stack above that symbol and push these content in the second stack.

In this way we can access any symbol stored in the stack without losing any content.



IMP Questions

1Q. Write Short Notes on Two Stack PDA

2Q. Write an application and advantages of Two Stack PDA

Important and Previous JNTU Questions:

Q.1. How Two Stack PDA is powerful than a PDA and why Two Stack PDA have the Equivalent power of a Turing Machine.

Application of Pushdown Automata

PDA & PARSING

Parsing is used to derive a string using the production rules of a grammar. It is used to check the acceptability of a string. Compiler is used to check whether or not a string is syntactically correct. A parser takes the inputs and builds a parse tree.

A parser can be of two types −

- **Top- Down Parser** − Top- down parsing starts from the top with the start- symbol and derives a string using a parse tree.
- **Bottom- Up Parser** − Bottom- up parsing starts from the bottom with the string and comes to the start symbol using a parse tree.

Design of Top- Down Parser

For top- down parsing, a PDA has the following four types of transitions −

- Pop the non- terminal on the left hand side of the production at the top of the stack and

push its right- hand side string.

- If the top symbol of the stack matches with the input symbol being read, pop it.
- Push the start symbol ‘ S’ into the stack.
- If the input string is fully read and the stack is empty, go to the final state ‘ F’ .

Example

Design a top- down parser for the expression "x+y*z" for the grammar G with the following production rules −

P: S → S+X | X, X → X*Y | Y, Y → (S) | id

Solution

If the PDA is (Q, ∑ , S, δ , q_0, I, F), then the top- down parsing is −

(x+y*z, I) ⊢(x +y*z, SI) ⊢ (x+y*z, S+XI) ⊢(x+y*z, X+XI)

⊢(x+y*z, Y+X I) ⊢(x+y*z, x+XI) ⊢(+y*z, +XI) ⊢ (y*z, XI)

⊢(y*z, X*YI) ⊢(y*z, y*YI) ⊢(*z,*YI) ⊢(z, YI) ⊢(z, zI) ⊢(ε , I)

Design of a Bottom- Up Parser

For bottom- up parsing, a PDA has the following four types of transitions −

- Push the current input symbol into the stack.
- Replace the right- hand side of a production at the top of the stack with its left- hand side.
- If the top of the stack element matches with the current input symbol, pop it.
- If the input string is fully read and only if the start symbol ‘ S’ remains in the stack, pop it and go to the final state ‘ F’ .

Example

Design a top- down parser for the expression "x+y*z" for the grammar G with the following production rules −

P: S → S+X | X, X → X*Y | Y, Y → (S) | id

Solution

If the PDA is (Q, ∑ , S, δ , q_0, I, F), then the bottom- up parsing is −

(x+y*z, I) ⊢ (+y*z, xI) ⊢ (+y*z, YI) ⊢ (+y*z, XI) ⊢ (+y*z, SI)

$\vdash$(y*z, +SI) $\vdash$ (*z, y+SI) $\vdash$ (*z, Y+SI) $\vdash$ (*z, X+SI) $\vdash$ (z, *X+SI)

$\vdash$ (ε , z*X+SI) $\vdash$ (ε , Y*X+SI) $\vdash$ (ε , X+SI) $\vdash$ (ε , SI)

Applications of PDA

implementing Online Transaction process system
Implementing Tower of Hanoi (Recursive Solution)
Implementing Timed Automata Model
Implementing Deterministic Top Down Parsing LL Grammar
Implementing Context free Language
Implementing Predictive Bottom up Parsing LR Grammar
Converting a PDA to Context free Language

IMP Questions

1Q. How a PDA helpful to Design a parser?
2Q. Write the applications of PDA

Important and Previous JNTU Questions:

Q.1. How to design a PARSER using PDA?
Q.2. Implement a Two Stack PDA to get an even palindrome from a given string.