LOVELY PROFESSIONAL UNIVERSITY

UNIT V
PUSH DOWN AUTOMATA AND PARSING

UNIT V SYLLABUS

- PUSHDOWN AUTOMATA AND PARSING : Representation of Pushdown Automata, Acceptance by Pushdown Automata, Pushdown Automata: Deterministic Pushdown Automata and non-deterministic Pushdown Automata, Context free languages and Pushdown Automata, PARSING: Top-Down and Bottom-Up Parsing, Description and Model of Pushdown Automata, Pushdown Automata and ContextFree Languages, Comparison of deterministic and non-deterministic versions, closure properties, LL (k) Grammars and its Properties, LR(k) Grammars and its Properties

Pushdown Automata(PDA)

- **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 –

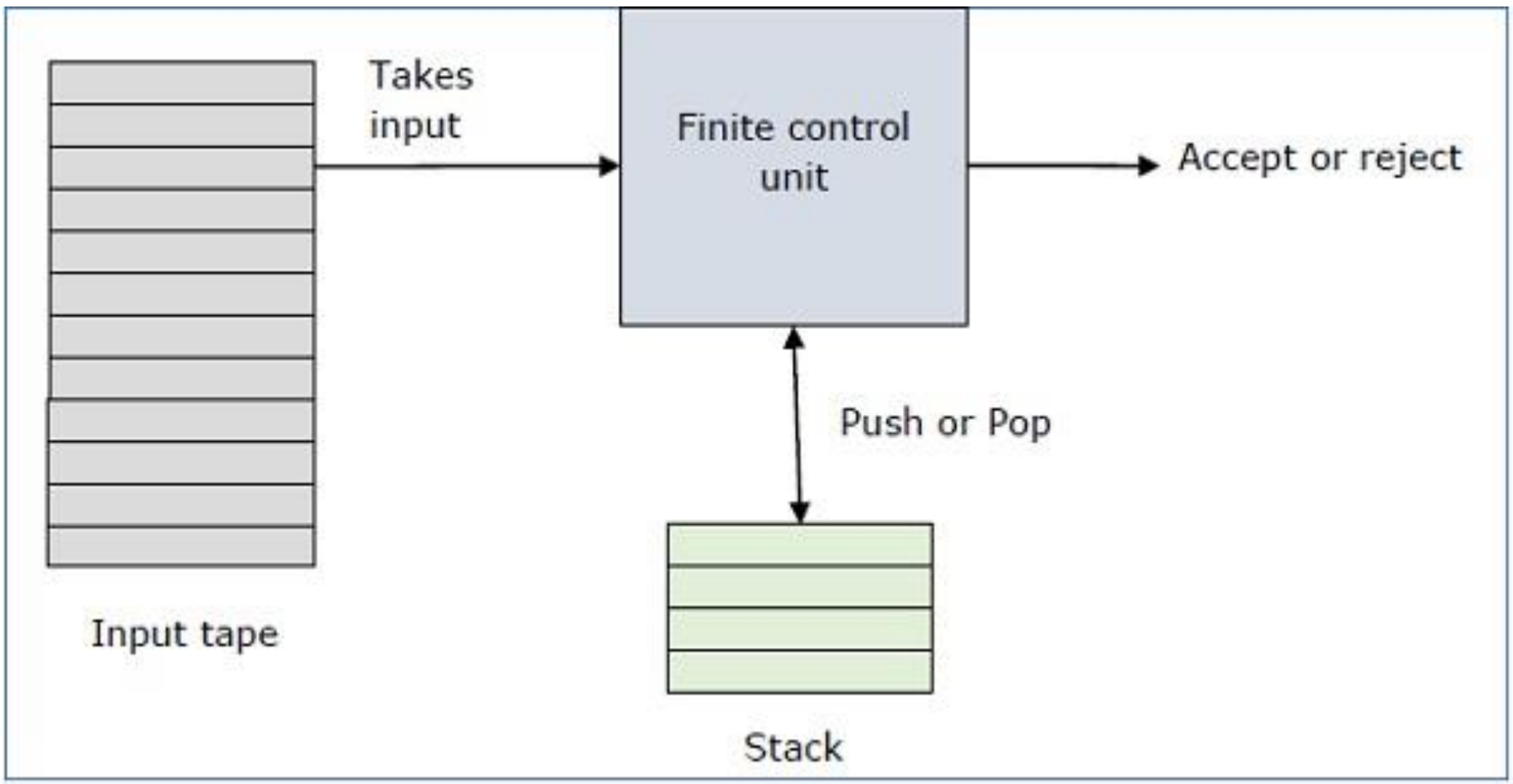
  1)an input tape,

  2) a control unit, and

  3) 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.



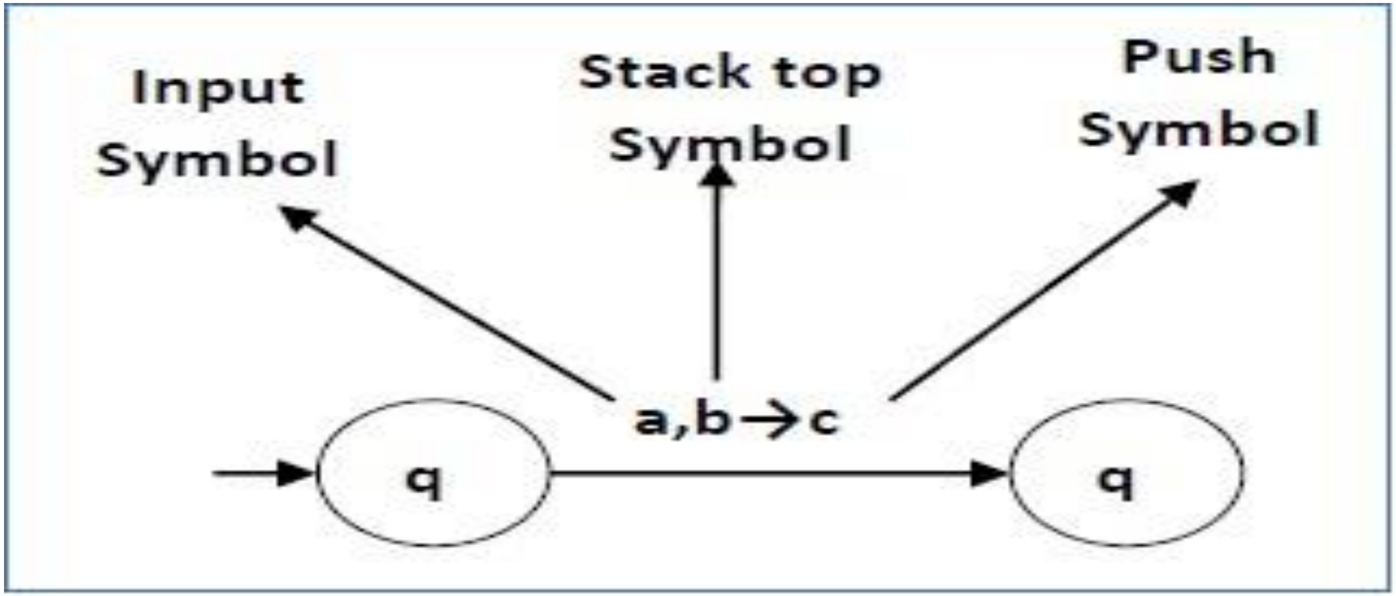
DEFINITION OF PDA

- A PDA can be formally described as a 7-tuple

  $(Q, \sum, S, \delta, q_0, I, F)$

- **Q** is the finite number of states
- **∑** is input alphabet
- **S** is stack symbols
- **δ** is the transition function: $Q \times (\sum \cup \{\varepsilon\}) \times S \times Q \times S^*$
- $\mathbf{q_0}$ is the initial state ($q_0 \in Q$)
- **I** is the initial stack top symbol ($I \in S$)
- **F** is a set of accepting states ($F \in Q$)

- 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 $\mathbf{q_1}$, 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 $\mathbf{q_2}$.

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 −

$$(p, aw, T\beta) \vdash (q, w, \alpha b)$$

- 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.

POLLING QUESTIONS

1. The transition a Push down automaton makes is additionally dependent upon the:

   a) stack
   b) input tape
   c) terminals
   d) none of the mentioned

2. With reference of a DPDA, which among the following do we perform from the start state with an empty stack?

a) process the whole string
b) end in final state
c) end with an empty stack
d) all of the mentioned

3. Halting states are of two types. They are:

a) Accept and Reject
b) Reject and Allow
c) Start and Reject
d) None of the mentioned

4. Which of the following correctly recognize the symbol ‘|-‘ in context to PDA?

a) Moves
b) transition function
c) or/not symbol
d) none of the mentioned

5. Which of the following automata takes stack as auxiliary storage?

a) Finite automata
b) Push down automata
c) Turing machine
d) All of the mentioned

Pushdown Automata Acceptance

- There are two different ways to define PDA acceptability.

  1. **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 −

$$L(PDA) = \{w \mid (q_0, w, I) \vdash^* (q, \varepsilon, x), q \in F\}$$

for any input stack string **x**.

2. 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 –

$$L(PDA) = \{w \mid (q_0, w, I) \vdash^* (q, \varepsilon, \varepsilon), q \in Q\}$$

Practice Questions

1. Construct a PDA that accepts

   $L = \{0^n 1^n \mid n \geq 0\}$

2. Construct a PDA that accepts even palindromes of the form:

   $L = \{ ww^R \mid w = (a+b)^* \}$

3. Construct a PDA that accepts

   L = {a^nb^nc |n>=1}

POLLING QUESTIONS

1. A string is accepted by a PDA when

a.Stack is empty

b.Acceptance state

c.both a and b

d.None of the mentioned

2. The following move of a PDA is on the basis of:

a.Present state

b.Input Symbol

c.both a and b

d.None of the mentioned

PDA & Context-Free Grammar

- 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

$$L(G) = L(P)$$

Algorithm to find PDA corresponding to a given CFG

- **Input** – A CFG, $G = (V, T, P, S)$
- **Output** – Equivalent PDA, $P = (Q, \sum, S, \delta, q_0, I, F)$

- Rule 1: $\delta(q, \varepsilon, A) = (q, \alpha) \mid A \rightarrow \alpha$ is in P
- Rule 2: $\delta(q,a,a)=(q, \varepsilon)$ for every $a \in \Sigma$

Practice Questions

1. Convert into PDA for the CFG given:

   S-> 0BB, B->0S|1S|0

2. Convert into PDA for the CFG given:

   S->aSA, S->bSb|c

POLLING QUESTIONS

LOVELY PROFESSIONAL UNIVERSITY

1. A PDA machine configuration (p, w, y) can be correctly represented as:

a) (current state, unprocessed input, stack content)

b) (unprocessed input, stack content, current state)

c) (current state, stack content, unprocessed input)

d) none of the mentioned

2. Let $\sum=\{0,1\}^*$ and the grammar G be:

$S \to \varepsilon$

$S \to SS$

$S \to 0S1 \mid 1S0$

State which of the following is true for the given

a) Language of all and only Balanced strings

b) It contains equal number of 0's and 1's

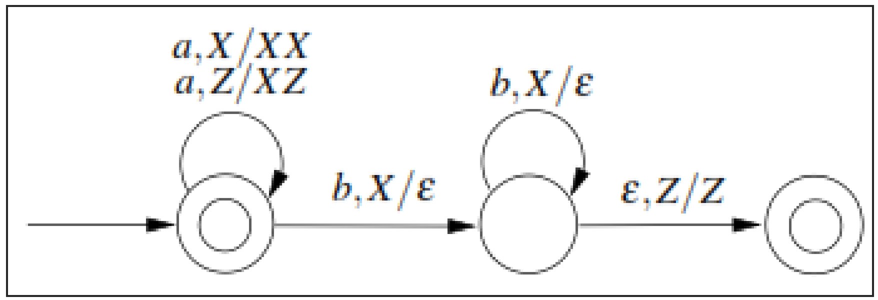
c) Ambiguous Grammar

d) All of the mentioned

3. A push down automata is said to be __________ if it has atmost one transition around all configurations.

a) Finite
b) Non regular
c) Non-deterministic
d) Deterministic

4. Consider the transition diagram of a PDA given below with input alphabet Σ = {a,b} and stack alphabet Γ ={X,Z} .Z is the initial stack symbol. Let *L* denote the language accepted by the PDA.



Which one of the following is **TRUE**?

- (A) L ={anbn|n≥0} and is not accepted by any finite automata
- (B) L ={an|n≥0}U{anbn|n≥0} and is not accepted by any deterministic PDA
- (C) L is not accepted by any Turing machine that halts on every input
- (D) L ={an|n≥0}U{anbn|n≥0} and is deterministic context-free

Conversion of PDA to CFG

- The productions in P are induced by move of PDA as follows:

1. S productions are given by:

   S->[q0 Z0 q]   for every $q \in Q$

2. For every popping move:

δ(q ,a, Z)=(q’, ε) induces production [q Z q’]->a

3. For each push move:

$\delta(q, a, Z)=(q1, Z1,Z2,\ldots.Zm)$ induces many productions $[q\ Z\ q'] \rightarrow a\ [q1\ Z1\ q2]\ [q2\ Z2\ q3] \ldots..\ [qm\ Zm\ q']$

where each state q1,q2,...qm can be any state in PDA.

Practice Questions

1. Generate CFG for given PDA,
   $M=\{\{q0,q1\},\{0,1\},\{x,Z0\}, \delta, q0, Z0,q1\}$ where δ is given as follows:

   $\delta(q0,1,Z0)=(q0,xZ0)$

   $\delta(q0,1,x)=(q0,xx)$

   $\delta(q0,0,x)=(q0,x)$

   $\delta(q0, \varepsilon,x)=(q1,\varepsilon)$

   $\delta(q1, \varepsilon,x)=(q1,\varepsilon)$

   $\delta(q1,0,x)=(q1,xx)$

   $\delta(q1,0,Z0)=(q1,\varepsilon)$

Pushdown Automata & 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’.

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’.

1. Which of the following derivations does a top-down parser use while parsing an input string?
a) Leftmost derivation
b) Leftmost derivation in reverse
c) Rightmost derivation
d) Rightmost derivation in reverse

2. A bottom up parser generates ___________

a) Rightmost Derivation

b) Right most derivation in reverse

c) Left most derivation

d) Left most derivation in reverse

Top Down Parser
LL(k) Grammar

- An LL parser (Left-to-right, Leftmost derivation) is a top-down parser for a subset of context-free languages . It parses the input from Left to right, performing Leftmost derivation of the sentence.

- An LL parser is called an LL(k) parser if it uses k tokens of lookahead when parsing a sentence.

- A grammar is called an LL(k) grammar if an LL(k) parser can be constructed from it.

- LLR grammars are a proper superset of LL(k) grammars for any k. For every LLR grammar there exists an LLR parser that parses the grammar in linear time.

- The LL(k) parser is a deterministic pushdown automaton with the ability to peek on the next k input symbols without reading.

Example

- To explain an LL(1) parser's workings we will consider the following small LL(1) grammar:
- S → F
- S → (S + F)
- F → a

and parse the following input:

- **(a + a)**

- An LL(1) parsing table for a grammar has a row for each of the non-terminals and a column for each terminal (including the special terminal, represented here as **$**, that is used to indicate the end of the input stream).
- A list of rules for a leftmost derivation of the input string, which is:
- S $\rightarrow$ **(** S **+** F **)** $\rightarrow$ **(** F **+** F **)** $\rightarrow$ **(a +** F **)** $\rightarrow$ **(a + a)**

Properties of LL(k) Grammar

PROPERTIES

- If G is LL(k) grammar, then it is unambiguous for all k≥1
- Free from left recursion
- If G is LL(k) grammar then G is LL(k+1) grammar for all k≥1
- For every production A → α|ß, where α != ß
  - FIRST(α) ∩ FIRST (ß) = ϕ
  - Atmost one of α or ß can derive ε
  - If α derives ε, then FIRST(ß) ∩ FOLLOW(A) = ϕ

Bottom Up Parser
LR(k) Grammar

- An **LR** parser (**L**eft-to-right, **R**ightmost derivation in reverse) reads input text from left to right without backing up (this is true for most parsers), and produces a rightmost derivation in reverse: it does a bottom-up parse – not a top-down LL parse or ad-hoc parse.

- Avoid backtracking or guessing, the LR parser is allowed to peek ahead at *k* look ahead input symbols before deciding how to parse earlier symbols.

- LR parsers are deterministic; they produce a single correct parse without guesswork or backtracking, in linear time.

- Start at the leaves and grow toward root
- As input is consumed, encode possibilities in an internal state
- A powerful parsing technology
- LR grammars
  - Construct right-most derivation of program
  - Left-recursive grammar, virtually all programming language are left-recursive
  - Easier to express syntax

- Right-most derivation
  - Start with the tokens
  - End with the start symbol
  - Match substring on RHS of production, replace by LHS
  - Shift-reduce parsers
    - Parsers for LR grammars
    - Automatic parser generators (yacc, bison)

- Example Bottom-Up Parsing

$S \rightarrow S + E \mid E$

$E \rightarrow num \mid (S)$

(1+2+(3+4))+5 ← (E+2+(3+4))+5 ←(S+2+(3+4))+5

←(S+E+(3+4))+5 ←(S+(3+4))+5 ←(S+(E+4))+5

←(S+(S+4))+5 ←(S+(S+E))+5 ←(S+(S))+5

←(S+E)+5 ←(S)+5 ←E+5

←S+5 ←S+E ←S

Properties of LR(K)

Property 1: Every LR(k) grammar *G* *is unambiguouS*

Properties of LR(K)

Property 2 If G is an LR(k) grammar, there exists a deterministic pushdown automaton A accepting $L(G)$.

Property 3 If A is a deterministic pushdown automaton A, there exists an LR(1) grammar G such that $L(G) = N(A)$.

Property 4 If G is an LR(k) grammar, where $k > 1$, then there exists an equivalent grammar G_1 which is LR(1). In so far as languages are concerned, it is enough to study the languages generated by LR(0) grammars and LR(1) grammars.

Definition 8.2 A context-free language is said to be deterministic if it is accepted by a deterministic pushdown automaton.

Property 5 The class of deterministic languages is a proper subclass of the class of context-free languages.

POLLING QUESTIONS

1. Which of these is true about LR parsing?

   a) Is most general non-backtracking shift-reduce parsing
   b) It is still efficient
   c) Is most general non-backtracking shift-reduce parsing & It is still efficient
   d) None of the mentioned

2. Which of the following is incorrect for the actions of A LR-Parser I) shift s ii) reduce A->ß iii) Accept iv) reject?

a) Only I)
b) I) and ii)
c) I), ii) and iii)
d) I), ii) , iii) and iv)

3. When ß (in the LR(1) item A -> ß.a,a) is not empty, the look-head __________

a) Will be affecting
b) Does not have any affect
c) Shift will take place
d) Reduction will take place