Activity based

Project 3 Report on

THEORY OF COMPUTATION

Submitted to Vishwakarma University, Pune

Under the Initiative of

Contemporary Curriculum, Pedagogy, and Practice (C2P2 2)

By

ATHARVA SHEVATE

SRN No : 202201727

Roll No : 2

Div : E(E1)

Second Year Engineering

Department of Computer Engineering Faculty of Science and Technology

Introduction:

The Theory of Computation serves as the bedrock of computer science, delving into the fundamental principles that underpin the capabilities and limitations of computational systems. At its core lies the study of formal languages, automata, and computation, offering insights into the nature of computation and its theoretical boundaries.

WHAT IS PUSH DOWN AUTOMATA (PDA) :-
A push down automaton (PDA) is a type of abstract machine used in automata theory and formal language theory. It is an extension of a finite automaton, augmented with a stack memory. The stack is used for remembering and retrieving items, which makes PDAs suitable for recognizing and parsing context-free languages, a class of languages that cannot be recognized by finite automata alone.

A PDA consists of the following components:

1. Input Alphabet: A finite set of symbols that can be read from the input string.
2. Stack Alphabet: A finite set of symbols that can be stored on the stack.
3. Set of States: A finite set of states.
4. Transition Function: A function that determines the next state, the symbol to be popped from the stack, and the string of symbols to be pushed onto the stack, based on the current state, the current input symbol, and the symbol on top of the stack.
5. Start State: The initial state of the PDA.
6. Stack Start Symbol: The initial symbol on the stack.
7. Final States: A set of accepting states.

The PDA reads the input string symbol by symbol and performs transitions based on the transition function. During each transition, the PDA can push symbols onto the stack, pop symbols from the stack, or leave the stack unchanged. The PDA accepts the input string if it reaches a final state after processing the entire input string.

Example 1: A PDA that recognizes the language { a^n b^n | n >= 1 } (strings consisting of equal numbers of 'a' and 'b' symbols, with 'a's coming before 'b's).

This PDA has the following configuration:

- Input Alphabet: {a, b}
- Stack Alphabet: {$, X} ($ is the stack start symbol, and X is a marker) - States: {q0, q1, q2, q3} (q0 is the start state, and q3 is the final state) - Transition Function:
- δ(q0, a, $) = (q1, X$)
- δ(q1, a, X) = (q1, XX)
- δ(q1, b, X) = (q2, ε)
- δ(q2, b, X) = (q2, ε)
- δ(q2, ε, $) = (q3, $)

The PDA starts in state q0 with the stack containing only the start symbol $. It pushes X onto the stack for each 'a' read and transitions to state q1. When the first 'b' is read, it pops X from the stack and transitions to state q2. In state q2, it keeps popping X from the stack for each 'b' read. If the stack becomes empty (contains only $) after reading all 'b's, it transitions to the final state q3, accepting the input string.

Example 2: A PDA that recognizes the language { ww^R | w is a string over {0, 1} } (strings consisting of a word 'w' followed by its reverse 'w^R').

This PDA has the following configuration:

- Input Alphabet: {0, 1}
- Stack Alphabet: {0, 1, $}
- States: {q0, q1, q2, q3} (q0 is the start state, and q3 is the final state) - Transition Function:
- δ(q0, 0, $) = (q1, 0$)
- δ(q0, 1, $) = (q1, 1$)
- δ(q1, 0, $) = (q1, 0$)

- δ(q1, 1, $) = (q1, 1$)
- δ(q1, ε, $) = (q2, $)
- δ(q2, 0, 0) = (q2, ε)
- δ(q2, 1, 1) = (q2, ε)
- δ(q2, ε, $) = (q3, $)

The PDA starts in state q0 and pushes the input symbols onto the stack until it reads the end of the first part of the input string. It then transitions to state q2, where it compares the input symbols with the symbols on top of the stack. If they match, it pops the symbol from the stack. If the stack becomes empty after reading the second part of the input string, it transitions to the final state q3, accepting the input string.

These examples illustrate how PDAs can recognize context-free languages by utilizing a stack memory to keep track of the nested structures in the input strings.

PROBLEM STATEMENT :-

Q1.

Consider a PDA,

$P = (Q, \sum, \Gamma, \delta, q_0, F)$

where

$Q = \{s, f\}$

$\sum = \{a, b, c\}$

$\Gamma = \{a, b\}$

$q_0 = \{s\}$

$F = \{f\}$

δ is given as follpws:

1. $(s, a, \varepsilon) \rightarrow (s, a)$
2. $(s, b, \varepsilon) \rightarrow (s, b)$
3. $(s, c, \varepsilon) \rightarrow (f, \varepsilon)$
4. $(f, a, a) \rightarrow (f, \varepsilon)$
5. $(f, b, b) \rightarrow (f, \varepsilon)$

Check whether the string abacaba is accpeted by the above pushdown automation.

Name - Atharva Sanjay Shevate
Std - 20220727.
Roll no - 02.
Div - E (E1)

Q1] Write down the definition of PDA and compare it with FA

→ PDA (Push down automata) is a Finite automata with the Extra memory called Stack helps push down automata to recognize (CFL) (Context Free language).

PDA can be defined as
Q : set of states
Σ : Set of input symbols
Γ : set of push down Symbol.
q : - initial state
z : - initial pushdown symbol
δ :- Transition function which mean $Q \times \{\Sigma \cup E\} \times \Sigma$ into $Q \times \Gamma^*$

Push down automata (PDA)	Finite automata (FA)
1) Has the stack memory allow to store info	1) Has memory beyond current state
2) More powerful than FA as it recognise CFG including non regular language	2) Can only recognize regular language

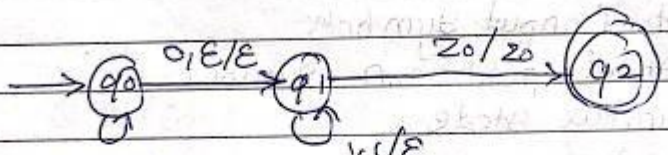
Page No.:
Date:

Q2. Given :-
$L = \{ w c w^3 / w = \{0,1\}^* \}$ w^3 = recursive of w

Steps :-
Step1 - on recuring 0 put it in stack
Step2 - on recuring c move to next step
Step 3 - check if symbol is scanned is '1' top of stack also contain '1'

OR

Step 4 :- Symbol 0 is scanned and on top also 0 is then pop element from stack else move to dead state continue this until the stack becomes empty.



0,0/00
1,1/11
0,1/0,1
1,Z0/1,Z0
0,Z0/Z0

Q = {S0 S1, S2
Σ = (0, 1 c)
Γ = {0, 1, Z0
S0 = initial state
Z0 = final st. special symbol
S2 = final state
δ(q1,0) = (q2,0)
δ(q1,c,1 (q2,1)
δ(q2,0, 0) (q3,ε)

q0 = S0, q1 = S1, q2 = S2
δ(q0,0,Z) = (q0,0,Z)
δ(q0,1,Z) = (q1,1,Z)
δ(q1,1,1 = (q1,11)

Q3 Given -

$P = \{Q, \Sigma, \Gamma, \delta, q_0, F\}$

$Q = \{q, r\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \{a, b\}$

$q_0 = \{S\}$

$F = \{f\}$

δ = 1. $(S, a, \varepsilon) \rightarrow (S, a)$

2. $(S, b, \varepsilon) \rightarrow (S, b)$

3. $(S, c, \varepsilon) \rightarrow (S, \varepsilon)$

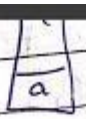
4. $(S, a, a) \rightarrow (r, \varepsilon)$

5 $(r, b, b) \rightarrow (r, \varepsilon)$

String given: abacaba

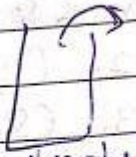
Step	Tape	Stack	Rule No
		Empty	-
1.	0: |a|b|a|c|a|b|a|		
2.	02 |a|b|a|c|a|b|a|	a	01
	03 |a|b|a|c|a|b|a|	b a	02
	04 |a|b|a|c|a|b|a|	a b a	01
	05 |a|b|a|c|a|b|a|	a b a	03

07 |a|b|a|c|a|b|a)
↑



05

08 |a|b|a|c|a|b|a|
↑



04

No more symbols left in stack. The given str is excepted by PDA (a b a c a b a)

Q4. Given - CFG - $S \to aA$

$A \to aABc \mid bB \mid a$

$B \to b$

$C \to c$

S = Start Symbol

Step 01 :- Initially

$\delta(p, \in, \in) \to (q, S)$

Step 2 - $S \to aS$

$\delta(q, a, S) \to (q, a, A)$

$A \to aABc \mid bB \mid a$

$\delta(q, \varepsilon, A) = (q, q, ABc)$

$\delta(q, \varepsilon, A) \to (q, b, B)$

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

$B \to B$

$\delta(q, \varepsilon, B) \to (q, B)$

$C \to C$

$\delta(q, \varepsilon, C) \to (q, C)$

Terminal Symbol in CFG (are in a, b c)

$S \to$

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

$q(q, b, b) \to (q, \varepsilon)$

$q(q, c, c) \to (q, \varepsilon)$

$\therefore Q = \{p, q\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \{S, A, B, C, a, b, c\}$

$q_0 = p$

Inverse function:-

$\delta(p, \varepsilon, \varepsilon) \rightarrow (q, S)$

$\delta(q, \varepsilon, S) \rightarrow (q, aA)$

$\delta(q, \varepsilon, A) \rightarrow (q, aABC)$

$\delta(q, \varepsilon, A) \rightarrow (q, b, B)$

$\delta(q, \varepsilon, aA) \rightarrow (q, a)$

$\delta(q, \varepsilon, B) \rightarrow (q, c)$

$\delta(q, \varepsilon, C) \rightarrow (q, C)$

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

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

$\delta(q, c, c) \rightarrow (q, \varepsilon)$

Conclusion:-

In conclusion, push down automata (PDAs) are a powerful computational model that extends the capabilities of finite automata by introducing a stack memory. This additional memory allows PDAs to recognize and process context-free languages, a class of languages that cannot be recognized by finite automata alone.

The key features of PDAs are:

1. Input Alphabet: A finite set of symbols that can be read from the input string.
2. Stack Alphabet: A finite set of symbols that can be stored on the stack.
3. Set of States: A finite set of states that the PDA can be in.
4. Transition Function: A function that determines the next state, the symbol to be popped from the stack, and the string of symbols to be pushed onto the stack, based on the current state, the current input symbol, and the symbol on top of the stack.
5. Start State: The initial state of the PDA.
6. Stack Start Symbol: The initial symbol on the stack.
7. Final States: A set of accepting states.

PDAs operate by reading the input string symbol by symbol and performing transitions based on the transition function. During each transition, the PDA can push symbols onto the stack, pop symbols from the stack, or leave the stack unchanged. The PDA accepts the input string if it reaches a final state after processing the entire input string.

PDAs are widely used in various areas of computer science, including compiler design, natural language processing, and formal language theory. They provide a formal framework for recognizing and parsing context-free languages, which are crucial for programming languages, markup languages (e.g., HTML, XML), and many other applications.

Despite their power, PDAs have limitations. They cannot recognize or generate all contextsensitive languages, a broader class of languages that includes context-free languages.

For languages beyond the context-free class, more powerful computational models, such as linear bounded automata or Turing machines, are required.

Overall, push down automata are an important theoretical concept in automata theory and formal language theory, and they have practical applications in areas where context-free languages play a significant role.