Vidyavardhaka Sangha®, Mysore
VIDYAVARDHAKA COLLEGE OF ENGINEERING
Autonomous Institute, Affiliated to Visvesvaraya Technological University, Belagavi
(Approved by AICTE, New Delhi & Government of Karnataka)
Accredited by NBA (CV, CS, EE, EC, IS & ME) | NAAC with 'A' Grade
P.B. No. 206, Gokulam III Stage, Mysuru-570 002, Karnataka, India
Phone: +91 821 4276201 /202 /225, Fax: +91 824 2510677
Web: http://www.vvce.ac.in @vvceofficial

Automata Theory (BCSAT504)

Activity Based Assessment

On

"Pushdown Automata (PDA)"

Submitted By:

Yashwanth S	**4VV22CS189**
Venkatesha Y K	**4VV22CS181**
Yesh Gowda	**4VV22CS190**

Submitted To:

Prof. Harshitha K
Department of
Computer Science and Engineering
VVCE

Introduction:

In the theory of computation, automata play a fundamental role in understanding the behavior of various types of languages and computational processes. Among the various types of automata, **Pushdown Automata (PDA)** is a crucial concept for recognizing **context-free languages**, which are more complex than regular languages but are widely used in the parsing of programming languages and syntactic analysis.

In this report, we have implemented a **Pushdown Automaton (PDA)** to recognize strings with balanced parentheses. The language of balanced parentheses is a classic example of a context-free language, and the PDA uses a stack to help maintain the balance between opening and closing parentheses.

What We Have Implemented:

We have implemented a **Pushdown Automaton (PDA)** that recognizes the language of balanced parentheses. The PDA processes a string consisting of (and) and checks if:

1. Every opening parenthesis (has a corresponding closing parenthesis).
2. The parentheses are properly nested.

The PDA uses a stack to store unmatched opening parentheses, and each closing parenthesis pops an opening parenthesis from the stack. If, by the end of the input string, the stack is empty and the PDA ends in an accepting state, the string is accepted as balanced.

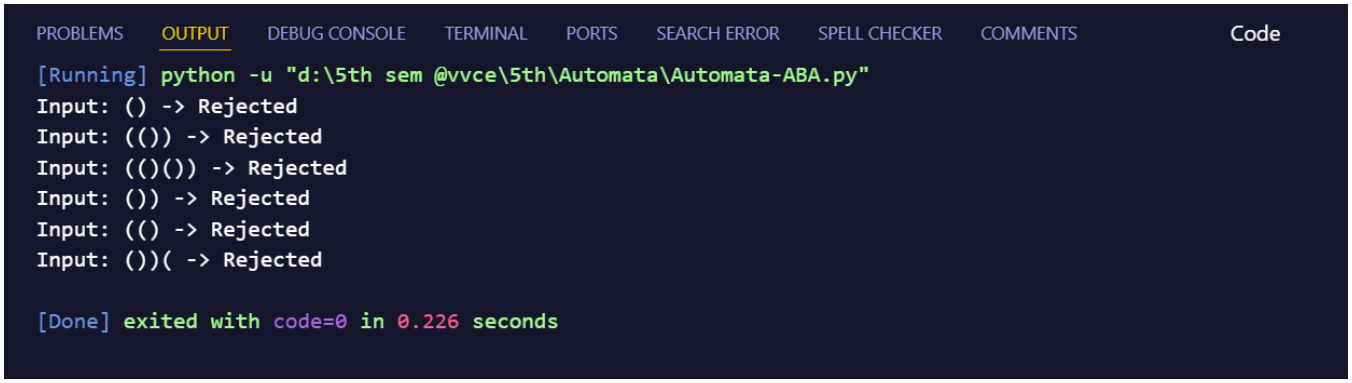
Code:

```
class PDA:
    def __init__(self):
        # Defining states and stack alphabet
        self.states = ['q0', 'q1', 'q2']
        self.start_state = 'q0'
        self.accepting_states = ['q0']
```

```
    # The stack that simulates the PDA's stack memory
    self.stack = []

def process_string(self, input_string):
    current_state = self.start_state
    self.stack = []  # Reset stack for each new string

    # Process each symbol in the input string
    for symbol in input_string:
        if symbol not in ['(', ')']:
            raise ValueError("Input string contains invalid symbols.")

        if current_state == 'q0':
            if symbol == '(':
                current_state = 'q1'
                self.stack.append('(')
            elif symbol == ')':
                if not self.stack:  # No '(' to match
                    return False
                current_state = 'q2'
                self.stack.pop()

        elif current_state == 'q1':
            if symbol == '(':
                self.stack.append('(')
            elif symbol == ')':
```

```
                if not self.stack:  # No '(' to match
                    return False
                current_state = 'q2'
                self.stack.pop()

            elif current_state == 'q2':
                if symbol == ')':
                    if not self.stack:  # No '(' to match
                        return False
                    self.stack.pop()

        # Check if the string ends in accepting state and the stack is empty
        return current_state in self.accepting_states and not self.stack
\

# Create an instance of PDA
pda = PDA()

# Test cases
test_strings = ["()", "(())", "(()())", "())", "(()", "())("]

# Evaluate each test string
for test_string in test_strings:
    result = pda.process_string(test_string)
    print(f"Input: {test_string} -> {'Accepted' if result else 'Rejected'}")
```

Output:



Methodology:

The **Pushdown Automaton (PDA)** was implemented using the following steps:

1. **States and Stack Setup**:
    - The PDA has three states: q0, q1, and q2. q0 is the starting state and also the accepting state. The PDA operates on a stack to track the balance of parentheses.
2. **Transition Function**:
    - The transition function is defined as follows:
        - From q0, if we encounter (, we push (onto the stack and transition to state q1.
        - From q1, if we encounter (, we continue pushing (onto the stack. If we encounter), we pop (from the stack and transition to state q2.
        - From q2, if we encounter), we pop (from the stack, staying in state q2 if parentheses are still unmatched.
3. **String Processing**:
    - For each input string, the PDA processes each character. If at any point, the stack cannot pop a parenthesis or there is an unmatched closing parenthesis, the string is rejected.
4. **Acceptance Criteria**:
    - After processing the string, the PDA accepts the string if the stack is empty and the PDA ends in the accepting state q0. If these conditions are not met, the string is rejected.

5. **Test Cases**:
    - We tested several input strings such as "()", "(())", "(()())", "))", and "((", to evaluate the PDA's correctness.

Conclusion:

The implemented **Pushdown Automaton (PDA)** successfully recognizes strings with balanced parentheses, demonstrating the utility of PDAs in recognizing **context-free languages**. By utilizing a stack, the PDA can handle the matching of parentheses, which is not possible with finite automata (DFA/NFA) alone. This implementation highlights the strength of PDAs in handling more complex language structures, as seen in programming languages and various other syntactical analysis tasks.