NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

(KARACHI CAMPUS)

FAST School of Computing Spring 2024

# PROJECT REPORT

MEMBERS:

- Ayesha Ansari (22K-4453)
- Mishkaat Yousuf (22K-4624)
- Jahantaab Kulsoom (22K-4214)

COURSE INSTRUCTOR:

 MISS Zain Noreen

# Project Topic:

Development of a Pushdown Automata (PDA) Simulator

## Introduction:

The Development of Pushdown Automaton (PDA) Simulator project aims to create a software tool that allows users to simulate the behavior of Pushdown Automata. This simulator will provide an interactive environment for users to input PDAs, test them with various input strings, and visualize their transitions, aiding in the understanding and exploration of context-free languages and automata theory.

## Description:

1. PDA Definition Input: The simulator will allow users to input the definition of a PDA, including states, transitions, input symbols, stack symbols, initial state, initial stack symbol, and final states. This input can be provided manually or loaded from a file.

2. Simulation Interface: The simulator will provide a user-friendly interface for interacting with the PDA. Users can input strings to be processed by the PDA and observe the transitions and stack changes step by step. Visualizations such as transition table will aid in understanding the computation.

3. Step-by-Step Execution: Users can choose to execute the PDA in a step-by-step manner, advancing through each transition one at a time. This feature allows for detailed analysis of the PDA's behavior and helps users understand how it processes input strings.

4. Input Validation: The simulator will validate user input to ensure that it adheres to the PDA definition format. This includes checking for valid state transitions, input symbols, stack symbols, and other constraints defined by PDA theory.

5. Performance Optimization: The simulator will be designed to efficiently handle large PDA definitions and input strings, ensuring smooth execution even for complex simulations.

# Project Video Link:

https://drive.google.com/file/d/1FUX-nTZfqVNyiaFJifwLGQojploZY2ru/view?usp=sharing

# Project Code:

```
#include <iostream>

#include <fstream>

#include <vector>

#include <sstream>

#include <map>

#include <algorithm> // Include for std::find

#include <stdexcept>

#include <unistd.h>

#include <iterator> // Include for std::istream_iterator



using namespace std;



class FileHandler {

public:

    vector<string> readFile(const string& filePath) {

        vector<string> lines;

        ifstream file(filePath);

        if (file.is_open()) {

            string line;
```

```
        while (getline(file, line)) {
            lines.push_back(line);
        }
        file.close();
    } else {
        throw invalid_argument("File could not be opened.");
    }
    return lines;
}

map<string, vector<string>> parseFile(const vector<string>& lines) {
    map<string, vector<string>> parsedLines;
    parsedLines["states"] = splitString(lines[0]);
    parsedLines["input_symbols"] = splitString(lines[1]);
    parsedLines["stack_symbols"] = splitString(lines[2]);
    parsedLines["initial_state"] = {lines[3]};
    parsedLines["initial_stack"] = {lines[4]};
    parsedLines["final_states"] = splitString(lines[5]);
    vector<string> productions;
    for (size_t i = 6; i < lines.size(); ++i) {
        productions.push_back(lines[i]);
    }
    parsedLines["productions"] = productions;
```

```
        return parsedLines;

    }



private:

    vector<string> splitString(const string& str) {
        istringstream iss(str);

        vector<string> tokens;

        string token;

        while (iss >> token) {

            tokens.push_back(token);

        }

        return tokens;

    }

};



class PDA {

public:

    void compute(const string& inputString, const map<string, vector<string>>& parsedLines) {

        string modifiedInputString = inputString + "e";

        string initStackSymbol = parsedLines.at("initial_stack")[0];

        stack.push_back(initStackSymbol);

        const vector<string>& finalStates = parsedLines.at("final_states");

        const string& initialState = parsedLines.at("initial_state")[0];
```

```
    const vector<string>& stackSymbols = parsedLines.at("stack_symbols");
    const vector<string>& productions = parsedLines.at("productions");

    string currentStackSymbol = initStackSymbol;
    string currentState = initialState;

    cout << "State\tInput\tStack\tMove\n";
    cout << currentState << "\t_\tZ\t(" << currentStackSymbol << ", " << stackToString(stack) << ")\n";

    for (char c : modifiedInputString) {
       for (const string& production : productions) {
          istringstream iss(production);
          vector<string> tokens(istream_iterator<string>{iss}, istream_iterator<string>());
          if (tokens[0] == currentState && tokens[1] == string(1, c) && tokens[2] == currentStackSymbol) {
             currentState = tokens[3];
             if (tokens[4].size() == 2) {
                stack.push_back(string(1, c));
} else if (tokens[4].size() == 3) {
                stack.push_back(string(1, c));
                stack.push_back(string(1, c));
             } else if (tokens[4] == "e" && stack.size() != 1) {
                stack.pop_back();
                break;
             }
```

```
            }

        }

        string previousStackSymbol = currentStackSymbol;

        currentStackSymbol = stack.back();

        cout << currentState << "\t" << c << "\t" << previousStackSymbol << "\t(" << currentStackSymbol << ", " <<
stackToString(stack) << ")\n";

        usleep(1000000); // Sleep for 1 second
    }



    if (find(finalStates.begin(), finalStates.end(), currentState) != finalStates.end()) {

        cout << "String accepted by PDA." << endl;

    } else {

        cout << "String rejected by PDA." << endl;

    }

}



private:

    vector<string> stack;



    string stackToString(const vector<string>& stack) {

        string result;

        for (const string& symbol : stack) {

            result += symbol;

        }
```

```
        return result;

    }

};



int main() {

    FileHandler fh;

    PDA pda;
    string automataFilePath;

    cout << "Enter the automata file path: ";

    cin >> automataFilePath; // Context Free Language

    vector<string> lines;

    try {

        lines = fh.readFile(automataFilePath);
} catch (const invalid_argument& e) {

        cerr << e.what() << endl;

        return 1;

    }

    cout << "Reading Automata File" << endl;

    sleep(2);

    cout << "Automata File Successfully Read" << endl;

    sleep(2);

    string inputString;

    cout << "Enter input String: ";

    cin >> inputString;
```

```
inputString.erase(inputString.find_last_not_of(" \n\r\t") + 1);

cout << "Loading Details from Automata File: " << endl;

sleep(3);

map<string, vector<string>> parsedLines = fh.parseFile(lines);

cout << "States: ";

for (const string& state : parsedLines["states"]) {

    cout << state << " ";
}

cout << endl;

cout << "Input Symbols: ";

for (const string& symbol : parsedLines["input_symbols"]) {

    cout << symbol << " ";

}

cout << endl;

cout << "Stack Symbols: ";

for (const string& symbol : parsedLines["stack_symbols"]) {

    cout << symbol << " ";

}

cout << endl;

cout << "Initial State: " << parsedLines["initial_state"][0] << endl;

cout << "Initial Stack Symbol: " << parsedLines["initial_stack"][0] << endl;

cout << "Final States: ";

for (const string& state : parsedLines["final_states"]) {
```

```
        cout << state << " ";
    }
    cout << endl;
    cout << "Productions List:" << endl;
    for (const string& production : parsedLines["productions"]) {
        cout << "\t" << production << endl;
    }
    sleep(2);
    cout << "Details loaded" << endl;
    cout << "Computing the Transition Table:" << endl;
    pda.compute(inputString, parsedLines);
    return 0;
}
```

# CFL Text File Code:

- For anbn:

```
Q P F
a b
Z a
Q
Z
F
Q a Z Q aZ
Q a a Q aa
```

Q b a P e

P b Z P aZ

P e Z F Z

P b a P e

Q e Z F Z

P a Z R aZ

- For 0n1n:

  Q P F

  0 1

  Z 0

  Q

  Z

  F

  Q 0 Z Q 0Z

  Q 0 0 Q 00

  Q 1 0 P e

  P 1 Z P 0Z

  P e Z F Z

  P 1 0 P e

  Q e Z F Z

  P 0 Z R 0Z

# Output:

When string is accepted:

```
Enter the automata file path: anbn.txt
Reading Automata File
Automata File Successfully Read
Enter input String: aaaabbbb
Loading Details from Automata File:
States: Q P F
Input Symbols: a b
Stack Symbols: Z a
Initial State: Q
Initial Stack Symbol: Z
Final States: F
Productions List:
        Q a Z Q aZ
        Q a a Q aa
        Q b a P e
        P b Z P aZ
        P e Z F Z
        P b a P e
        Q e Z F Z
        P a Z R aZ
Details loaded
Computing the Transition Table:
State   Input   Stack   Move
Q       _       Z       (Z, Z)
Q       a       Z       (a, Za)
Q       a       a       (a, Zaa)
Q       a       a       (a, Zaaa)
Q       a       a       (a, Zaaaa)
P       b       a       (a, Zaaa)
P       b       a       (a, Zaa)
P       b       a       (a, Za)
P       b       a       (Z, Z)
F       e       Z       (Z, Z)
String accepted by PDA.
```

When string is rejected:

```
Enter the automata file path: anbn.txt
Reading Automata File
Automata File Successfully Read
Enter input String: aaabbbb
Loading Details from Automata File:
States: Q P F
Input Symbols: a b
Stack Symbols: Z a
Initial State: Q
Initial Stack Symbol: Z
Final States: F
Productions List:
        Q a Z Q aZ
        Q a a Q aa
        Q b a P e
        P b Z P aZ
        P e Z F Z
        P b a P e
        Q e Z F Z
        P a Z R aZ
Details loaded
Computing the Transition Table:
State   Input   Stack   Move
Q       _       Z       (Z, Z)
Q       a       Z       (a, Za)
Q       a       a       (a, Zaa)
Q       a       a       (a, Zaaa)
P       b       a       (a, Zaa)
P       b       a       (a, Za)
P       b       a       (Z, Z)
P       b       Z       (b, Zb)
P       e       b       (b, Zb)
String rejected by PDA.
```

# Conclusion:

The Development of Pushdown Automaton (PDA) Simulator project aims to create a versatile and user-friendly tool for simulating the behavior of PDAs. By providing interactive visualization and step-by-step execution capabilities, the simulator facilitates learning, research, and development in the field of automata theory and computational linguistics. With its robust features and intuitive interface, the simulator promises to be a valuable asset for students, educators, researchers, and practitioners alike.