Lab 1: LFPC

Gherman Adrian FAF-203

Variant 11:

Variant 11.

$V_N=\{S, B, D\}$,
$V_T=\{a, b, c\}$,
$P=\{$

1. $S \rightarrow aB$
2. $S \rightarrow bB$
3. $B \rightarrow bD$
4. $D \rightarrow b$
5. $D \rightarrow aD$
6. $B \rightarrow cB$
7. $B \rightarrow aS$ $\}$

4. Determine the grammar type by the Chromsky classification

This specific grammer is of ***type 3*** - *regular grammar because*

The productions must be in the form X → a or X → aY

where X, Y ∈ N (Non terminal)

and a ∈ T (Terminal)

3. Convert regular grammar to Finite Automaton (FA)

Code in Java:

Edge class: here we create the structure of the edge, with a constructor which takes three parameters, the source, destination and weight.

```
public class Edge {
    private char src, dest, weight;

    public Edge(char src, char dest, char weight) {
        this.src = src;
        this.weight = weight;
        this.dest = dest;
    }

    public char getSrc(){
        return this.src;
    }

    public char getDest() {
        return this.dest;
    }

    public char getWeight() {
        return weight;
    }
}
```

Graph class: here we create the arrayList of edges and the arrayList with the edges.
We have two time of input:
NonTerminal -> Termina NonTerminal or NonTerminal -> Terminal
AddEdge checks if a NonTerminal node exists, if so, he adds to it his neigbours, otherwise the creates the new node.

```
import java.util.ArrayList;

public class Graph {
   private ArrayList<ArrayList<Edge>> adjList;
   private ArrayList<Character> vertices;

   public Graph(ArrayList<ArrayList<Edge>> adjList, ArrayList<Character>
vertices) {
       this.adjList = adjList;
       this.vertices = vertices;
   }

   public void addEdge(String userInput) {
       char[] chars = userInput.toCharArray();

       //When input is length 4 "S aB"
       if (chars.length == 4) {
           //Check if Node exists
           if (!vertices.contains(chars[0])) {
               vertices.add(chars[0]);
               adjList.add(new ArrayList<>());
               adjList.get(vertices.size() - 1).add(new Edge(chars[0],
chars[3], chars[2]));   //Create new Arraylist and add new node to start
           } else {//existing character
               adjList.get(getIndex(adjList, chars[0])).add(new
Edge(chars[0], chars[3], chars[2]));
           }
           //When input is length 3 "A a"
       } else if (chars.length == 3) {
           //Check if Node exists
           if (!vertices.contains(chars[0])) {
               vertices.add(chars[0]);
               adjList.add(new ArrayList<>());
               adjList.get(vertices.size() - 1).add(new Edge(chars[0], ' ',
chars[2])); //Create new ArrayList and add new node to start
           } else {//existing character
               adjList.get(getIndex(adjList, chars[0])).add(new
Edge(chars[0], ' ', chars[2]));
           }
       }
   }
```

Graph class *continuation:*

```
public void printGraph(){
       for (int i = 0; i < adjList.size(); i++) {
           System.out.print("\nAdjacency list of vertex: " +
adjList.get(i).get(0).getSrc());
           for (int j = 0; j < adjList.get(i).size(); j++) {
               if (adjList.get(i).get(j).getDest()==' '){
                   System.out.print(" -->  End Node ("+
adjList.get(i).get(j).getWeight() +") ");
               }else{
                   System.out.print(" --> " +
adjList.get(i).get(j).getDest() + "("+ adjList.get(i).get(j).getWeight() +")
");
               }
           }
           System.out.println();
       }
   }

   public static int getIndex(ArrayList<ArrayList<Edge>> adj, char start) {
       for (int i = 0; i < adj.size(); i++) {
           Edge e = adj.get(i).get(0);
           if (e.getSrc() == start)
               return i;
       }
       return -1;
   }
}
```

ValidationFunction:

```
public boolean isValid(String sequence) {
   char key = 'S';
   if (sequence.indexOf('b') != sequence.length() - 1) {
       return false;
   }

   for (Character c : sequence.toCharArray()) {

       for (Edge e : adjList.get(getIndex(adjList, key))) {

           if (e.getWeight() == c) {
               key = e.getDest();
               break;
           } else if (adjList.get(getIndex(adjList, key)).indexOf(e) ==
adjList.get(getIndex(adjList, key)).size() - 1) {
               return false;
           }
       }

       if (key == ' ' && sequence.indexOf(c) == sequence.length() - 1) {
           return true;
       } else if (key == ' ' && sequence.indexOf(c) != sequence.length() -
1) {
           return false;
       }
   }
   return true;
}
```

Main class:

```
import java.util.ArrayList;
import java.util.Scanner;


public class Main {
   public static void main(String[] args) {

       Scanner sc = new Scanner(System.in);
       System.out.println("Provide your input below. When finished type
!!!\"exit\"!!!");
       ArrayList<ArrayList<Edge>> adjList = new ArrayList<>();
       ArrayList<Character> vertices = new ArrayList<>();
       Graph FA = new Graph(adjList, vertices);

       while (true) {
           //Input
           String userInput = sc.nextLine();
           if (userInput.equals("exit") || userInput.equals("EXIT") ||
userInput.equals("Exit")) {
               break;
           } else {
               FA.addEdge(userInput);
           }
       }

       FA.printGraph();
   }
}
//Code done in collaboration with George Vragalev, as this is not the main
code, this was additional, cause java is popular for graphing
```

When we run the program: we give as input our product, with space in between, when we are done, we write exit in the *console.*

```
Provide your input below. When finished type !!!"exit"!!!
S aB
S bB
B bD
D b
D aD
B cB
B aS
exit
```

Output:

```
Adjacency list of vertex: S --> B(a)  --> B(b)

Adjacency list of vertex: B --> D(b)  --> B(c)  --> S(a)

Adjacency list of vertex: D -->  End Node (b)  --> D(a)

Process finished with exit code 0
```

Java is not very popular for graphing, so I had to use Python. The above code, was done more as a challenge, that is why is done with a classmate. Python has way too many libraries and it is less challenging that this Java code was.

Code in Python

Main:

```
import graphviz

f = graphviz.Digraph('finite_state_machine', filename='FinalAutomation.gv')
f.attr(rankdir='LR', size='8,5')
f.node('Start', shape='plaintext')
f.edge("Start", "q0")

print("Enter production rules, when ready type \"Exit\" ")
verticesMap = {}


while True:
   val = input()
   if val == "exit" or val == "Exit" or val == "EXIT":
       break
   else:
       if len(val) == 4:  # S aB
           if val[0] not in verticesMap.keys():
               verticesMap[val[0]] = "q" + str(len(verticesMap))

           if val[3] not in verticesMap.keys():
               verticesMap[val[3]] = "q" + str(len(verticesMap))

           f.attr('node', shape='circle')
           f.edge(verticesMap.get(val[0]), verticesMap.get(val[3]),
label=val[2])

       else:  # B b
           if val[0] not in verticesMap.keys():
               verticesMap[val[0]] = "q" + str(len(verticesMap))
           if val[2] not in verticesMap.keys():
               verticesMap[val[2]] = "q" + str(len(verticesMap))
           f.attr('node', shape='doublecircle')
           f.node(verticesMap.get(val[2]))
           f.edge(verticesMap.get(val[0]), verticesMap.get(val[2]),
label=val[2])

f.view()
```

Input:

```
Run:  main
C:\Users\gherm\AppData\Local\Programs\Python\Python39\python.exe U:/SecondYear_Sem2/LFPC/Labs/Lab1/Pyhton/main.py
Enter production rules, when ready type "Exit"
S aB
S bB
B bD
D b
D aD
B cB
B aS
exit

Process finished with exit code 0
```

Output: