Technical University of Moldova
Software Engineering and Automatics department
Study program in Software Engineering

Formal Languages and Compiler Design

Lab 1

Author:Vragalev George

Instructors: Cojuhari Irina

Academic group: FAF-203 2021

2022

Variant 23.

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

P={

1. $S \rightarrow aB$
2. $B \rightarrow aC$
3. $C \rightarrow bB$
4. $C \rightarrow c$
5. $C \rightarrow aS$
6. $B \rightarrow bB$ }

3. Convert regular grammar to Finite Automaton (FA).

4. Determine the grammar type by the Chromsky classification.

Grammar type

This grammar is of **type 3- regular grammar**, since the productions rules are of the form:

$X \rightarrow a$ or $X \rightarrow aY$, where $X,Y \in V_N$ and $a \in V_T$

Code:

Edge class is defined to store the connection between nodes with the details about source, destination and weight of edge

```
package LABFA;

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:

```
package LABFA;
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]));
           }
       }
   }

   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) { //S
        for (int i = 0; i < adj.size(); i++) {
            Edge e = adj.get(i).get(0);
            if (e.getSrc() == start)
                return i;
        }
        return -1;
    }

    public boolean isValid(String sequence){
        char key = 'S';
        //if sequence does not end with the ending character c then it is wrong
        if (sequence.indexOf('c') != sequence.length()-1 ){
            return false;
        }
        //loop through the check string - sequence
        for (Character c: sequence.toCharArray()) {
            //for each adjacency list, whose starting character is key
            for (Edge e: adjList.get(getIndex(adjList, key))) {
                //if edge weight is that of the character c we set the key to the destination of that
edge and go to the next character
                if (e.getWeight()==c){
                    key = e.getDest();
                    break;
                }
                //if we don't find the character having looped through the whole sub array then its
false
                else if(adjList.get(getIndex(adjList, key)).indexOf(e) ==
adjList.get(getIndex(adjList, key)).size()-1){
                    return false;
                }
            }
            //if the next destination node is empty, and we have reached the end of string TRUE
            if (key == ' ' && sequence.indexOf(c)==sequence.length()-1){
                return true;
            }
            //if the next destination node is empty, but we have NOT reached the end of string FALSE
            else if (key == ' ' && sequence.indexOf(c)!=sequence.length()-1){
                return false;
            }
        }
        return true;
    }
}
```

Main class:

```
package LABFA;
import java.util.ArrayList;
import java.util.Scanner;
public class LabFA {
   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();
       if (FA.isValid("abc"))  System.out.println("Correct"); //wrong
       else System.out.println("Wrong");

       if (FA.isValid("aaaabac")) System.out.println("Correct"); //correct
       else System.out.println("Wrong");

       if (FA.isValid("aaaabbbabac"))System.out.println("Correct"); //correct
       elseSystem.out.println("Wrong");
   }
}
/*
SAMPLE INPUT
S aB
B aC
C bB
C c
C aS
B bB
exit
Corresponding Output:
Adjacency list of vertex: S --> B(a)

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

Adjacency list of vertex: C --> B(b)  -->  End Node (c)  --> S(a)
*/
```

For the input:

```
S aB
B aC
C bB
C c
C aS
B bB
exit
```

The program will produce the following output:

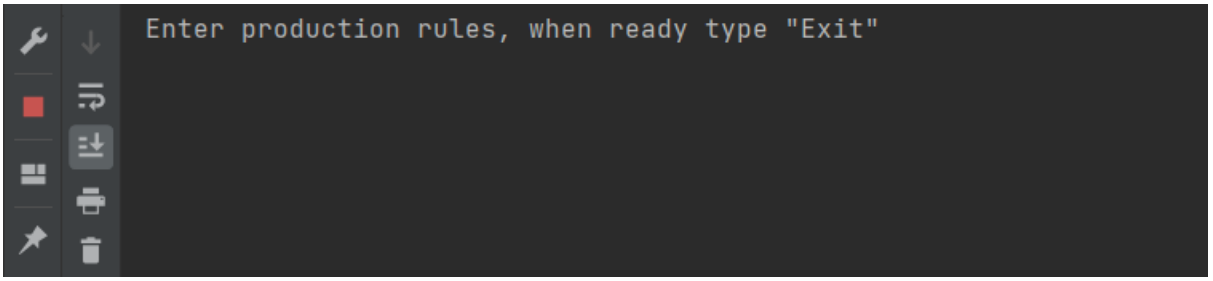
```
Adjacency list of vertex: S --> B(a)

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

Adjacency list of vertex: C --> B(b)  -->  End Node (c)  --> S(a)
```

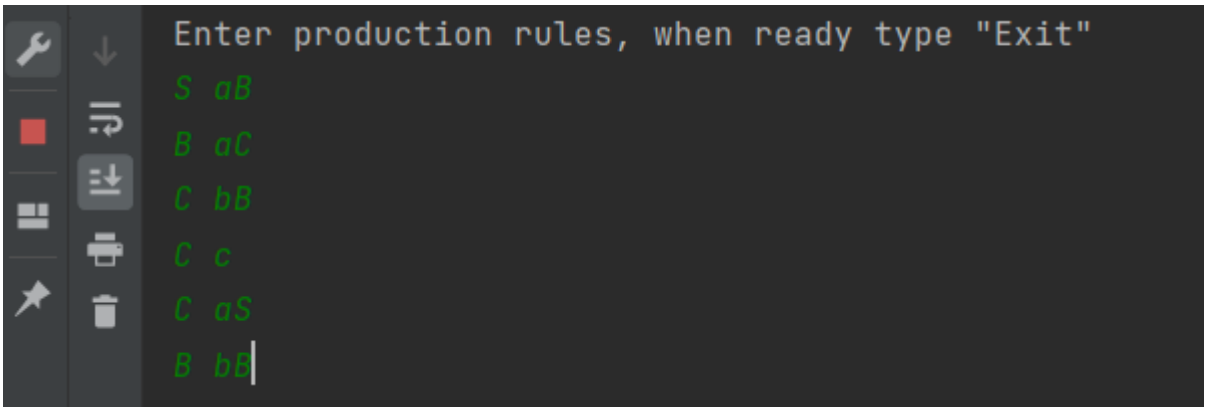
Python code that visualises the graph and creates a file with the graph image

```
import graphviz

f = graphviz.Digraph('finite_state_machine', filename='Lab1GraphViz.gv')
f.attr(rankdir='LR', size='8,5')


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

while True:
   val = input()
   if 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:





```
Enter production rules, when ready type "Exit"
S aB
B aC
C bB
C c
C aS
B bB
exit

Process finished with exit code 0
```

OUTPUT: