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

Formal Languages and Compiler Design

Laboratory work nr.2
Finite Automata

Authorship:

Gherman Adrian

Coordinator:

Vdovicenco Alexandru

Chisinau, 2022

Contents

1 Automation

AF=(Q, Σ, δ, q_0, F),

Q = { q_0, q_1, q_2 , q_3 },

Σ = { a, b, c }, F = { q_3 }.

δ (q_0, a) = q_1 ,

δ (q_1, b) = q_2,

δ (q_2, c) = q_0 ,

δ (q_1, a) = q_3,

δ (q_0, b) = q_2,

δ (q_2, c) = q_3.

1.1 NFA relation table

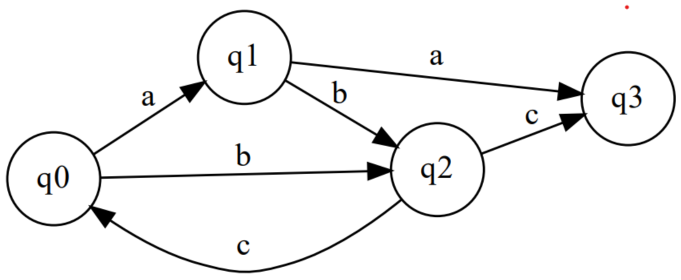
NFA			
	a	**b**	**c**
q0	q1	q3	-
q1	q3	q2	-
q2	-	-	q0q3
q3	-	-	-

1.2 DFA relation table

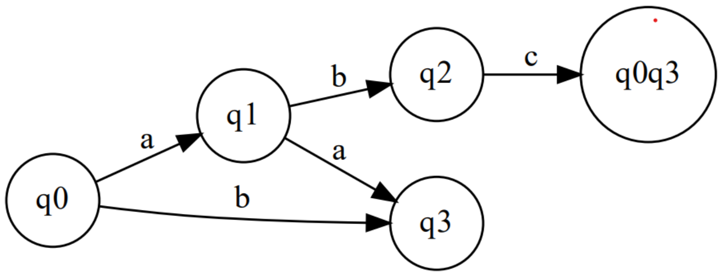
DFA			
	a	**b**	**c**
q0	q1	q3	-
q1	q3	q2	-
q2	-	-	q0q3
q3	-	-	-

2 Graphing

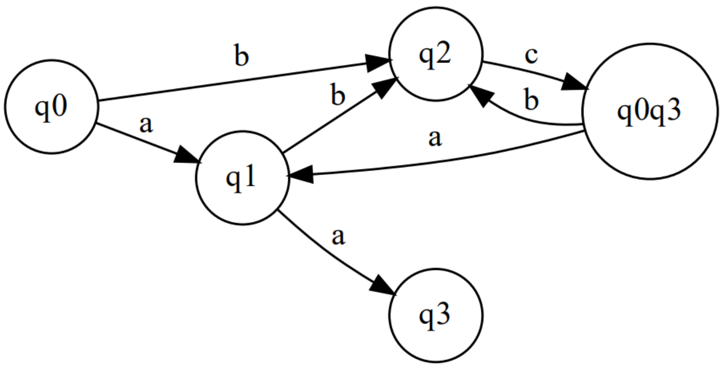
2.1 Finite automation FA



2.2 Non deterministic finite automaton NFA



2.3 Deterministic finite automaton DFA



3 Implementation

3.1 Edge class

```java
public class Edge {
    private String src, dest, weight;


    public String getSrc() {
        return src;
    }

    public String getDest() {
        return dest;
    }

    public String getWeight() {
        return weight;
    }

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

    public void printEdge(){
        System.out.print(src + " (" + weight + ") " + dest + "  |  ");
    }

}
```

3.2 Graph class

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedHashMap;

//Grammar to FA
public class Graph {
    private LinkedHashMap<String, ArrayList<Edge>> adjList;
    private ArrayList<String> vertices;

    public HashMap<String, ArrayList<Edge>> getAdjList() {
        return adjList;
    }


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

    public void addEdge(String userInput) {
        String[] arrOfStr = userInput.split(" ");
        String src = arrOfStr[0];
        String weight = arrOfStr[1];
        String dest = arrOfStr[2];

        if (!vertices.contains(src)) {//q0 a q2
            vertices.add(src);
            Edge e = new Edge(src, dest, weight);
            adjList.put(src, new ArrayList<Edge>());
            adjList.get(src).add(e);
        }
        else {
            Edge e = new Edge(src, dest, weight);
            adjList.get(src).add(e);
        }
        if(!vertices.contains(dest)){
            vertices.add(dest);
            adjList.put(dest, new ArrayList<Edge>());
        }
    }

    public void printGraph(){
        for (String s : adjList.keySet()) {
            System.out.print(s+" : ");
            for(Edge e: adjList.get(s))
                e.printEdge();
            System.out.println();
        }
    }
}
```

3.3 NFA class

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedHashMap;

//Grath to NFA
public class NFA {
    Graph graph;
    LinkedHashMap<String, ArrayList<Edge>> nfa;

    public NFA(Graph graph, LinkedHashMap<String, ArrayList<Edge>> nfa) {
        this.graph = graph;
        this.nfa = nfa;
    }

    public HashMap<String, ArrayList<Edge>> getNfa() {
        return nfa;
    }

    public void graphToNFA() {
        //for each state we loop through its array
        for (String src : graph.getAdjList().keySet()) {
            if (graph.getAdjList().get(src).isEmpty()) {
                nfa.put(src, new ArrayList<Edge>());
            }
            //we store the unique weights that the state has in an array
            ArrayList<String> weights = uniqueWeights(graph.getAdjList().get(src));
            //for each weight in the list we find the edges that have the same weight
            for (String weight : weights) {
                //We create an array of edges that have the same weight q0q1, q1q2
                //for each weight of the list we call weightArray function that returns the array of
     edges that have the same weight
                ArrayList<Edge> edgesSameWeight = weightArray(graph.getAdjList().get(src), weight);

                //Concatenate nodes with same weight
                String newState = "";
                for (Edge e : edgesSameWeight) {
                    newState += e.getDest();
                }
                Edge newStateEdge = new Edge(src, newState, weight);
                if (nfa.containsKey(src)) {
                    //appending to existing src
                    nfa.get(src).add(newStateEdge);
                } else {
                    //creating new src and appending to it the new edge
                    nfa.put(src, new ArrayList<Edge>());
                    nfa.get(src).add(newStateEdge);
                }
            }
        }
    }
    //Returns an array of edges that have the same weight that is specified
    public ArrayList<Edge> weightArray(ArrayList<Edge> list, String weight) {
        ArrayList<Edge> outputEdge = new ArrayList<>();
        for (Edge edge : list) {
            if (edge.getWeight().equals(weight)) {
                outputEdge.add(edge);
            }
        }
        return outputEdge;
    }

    public ArrayList<String> uniqueWeights(ArrayList<Edge> list) {
        ArrayList<String> outputWeights = new ArrayList<>();
        for (Edge edge : list) {
            if (!outputWeights.contains(edge.getWeight())) {
                outputWeights.add(edge.getWeight());
            }
        }
        return outputWeights;
    }

    public ArrayList<String> uniqueWeightsVoid() {
        ArrayList<String> weights = new ArrayList<>();

        for (String s : nfa.keySet()) {
            for (Edge e : nfa.get(s)) {
                if (!weights.contains(e.getWeight())) {
                    weights.add(e.getWeight());
                }
            }
        }
        return weights;
    }

    public void printNFA() {
        String endState = "";
        int nOfElements=nfa.keySet().size()-1;
        int count = 0;
        for (String key: nfa.keySet()){
            if (count==nOfElements){
                endState = key;
            }
            count++;
        }

        for (String s : nfa.keySet()) {
            if (s.contains(endState) && !endState.equals("")){
                System.out.print("*" + s + " : ");
            }
            else if (s.equals("q0")){
                System.out.print("->" + s + " : ");
            }
            else{
                System.out.print(s + " : ");
            }
            for (Edge e : nfa.get(s))
                e.printEdge();
            System.out.println();
        }
    }


}
```

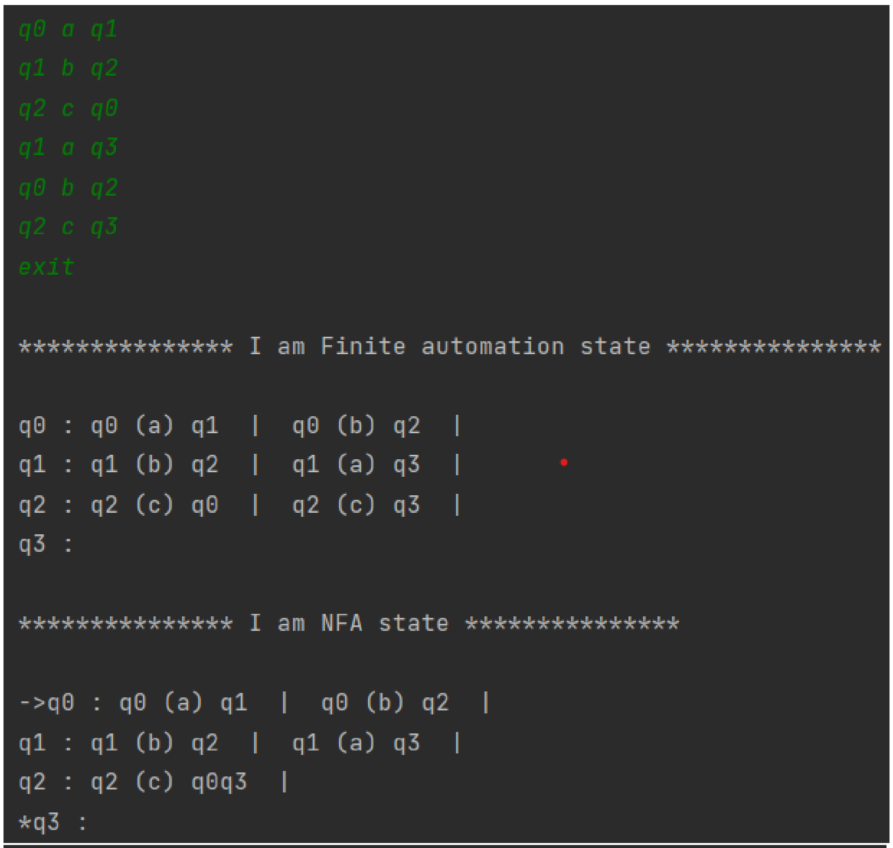
3.4 DFA class

```java
import java.util.ArrayList;
import java.util.LinkedHashMap;

public class DFA {
    NFA nfa;
    LinkedHashMap<String, ArrayList<Edge>> dfa;


    public DFA(NFA nfa, LinkedHashMap<String, ArrayList<Edge>> dfa) {
        this.nfa = nfa;
        this.dfa = dfa;
    }

    public void nfaToDfa() {
        dfa.put("q0", nfa.getNfa().get("q0")); //q0 : (a) q1 , (b) q2

        //until we don't find a new empty state
        while (!findNewState().equals("empty")) {
            String newState = findNewState();
            //if the state is single
            if (newState.length() == 2) {
                //we put that new state in dfa!
                dfa.put(newState, nfa.getNfa().get(newState));
            } else { //if the state is not single
                dfa.put(newState, new ArrayList<Edge>());
                concatenateNodes(newState);
            }
        }

    }


    public void concatenateNodes(String nodes) {
        String[] nodesList = usingSplitMethod(nodes);//q0 q1
        ArrayList<String> weights = nfa.uniqueWeightsVoid();
        for (String weight : weights) {
            String resultNode = ""; //q0q1q0
            for (String node : nodesList) {
                if (!findEdgeWithWeight(node, weight).equals("")) {
                    resultNode += findEdgeWithWeight(node, weight);
                }
            }
            if (!resultNode.equals("")) {
                resultNode = removeDuplicates(resultNode);
                Edge newNode = new Edge(nodes, resultNode, weight);
                dfa.get(nodes).add(newNode);
            }
        }
    }

    //Removing duplicates from string
    public String[] usingSplitMethod(String text) {
        return text.split("(?<=\\G.{" + 2 + "})");
    }

    public String removeDuplicates(String s) {
        String[] variables = usingSplitMethod(s); //q0 q0 q1 q2
        String result = "";
        for (String node : variables) {
            if (!result.contains(node)) {
                result += node;
            }
        }
        return result;
    }
    //Searches through a specific array list of a node and find an edge that has a specific weight
    public String findEdgeWithWeight(String node, String weight) {
        for (Edge e : nfa.getNfa().get(node)) {
            if (e.getWeight().equals(weight)) {
                return e.getDest();
            }
        }
        return "";
    }

    //loops through whole dfa and finds states that haven't been added to dfa yet
    public String findNewState() {
        for (String s : dfa.keySet()) {
            for (Edge edge : dfa.get(s)) {
                if (!dfa.containsKey(edge.getDest())) {
                    return edge.getDest();
                }
            }
        }
        return "empty";
    }


    public void printDFA() {
        String endState = "";
        int nOfElements = nfa.getNfa().keySet().size() - 1;
        int count = 0;
        for (String key : nfa.getNfa().keySet()) {
            if (count == nOfElements) {
                endState = key;
            }
            count++;
        }

        for (String s : dfa.keySet()) {
            if (s.contains(endState) && !endState.equals("")) {
                System.out.print("*" + s + " : ");
            } else if (s.equals("q0")) {
                System.out.print("->" + s + " : ");
            } else {
                System.out.print(s + " : ");
            }

            for (Edge e : dfa.get(s))
                e.printEdge();
            System.out.println();
        }

        System.out.println("*************** PythonInput ***************");
        for (String s : dfa.keySet()) {
            for (Edge e : dfa.get(s)) {
                System.out.println(e.getSrc() + " " + e.getWeight() + " " + e.getDest());
            }
        }
    }
}
```

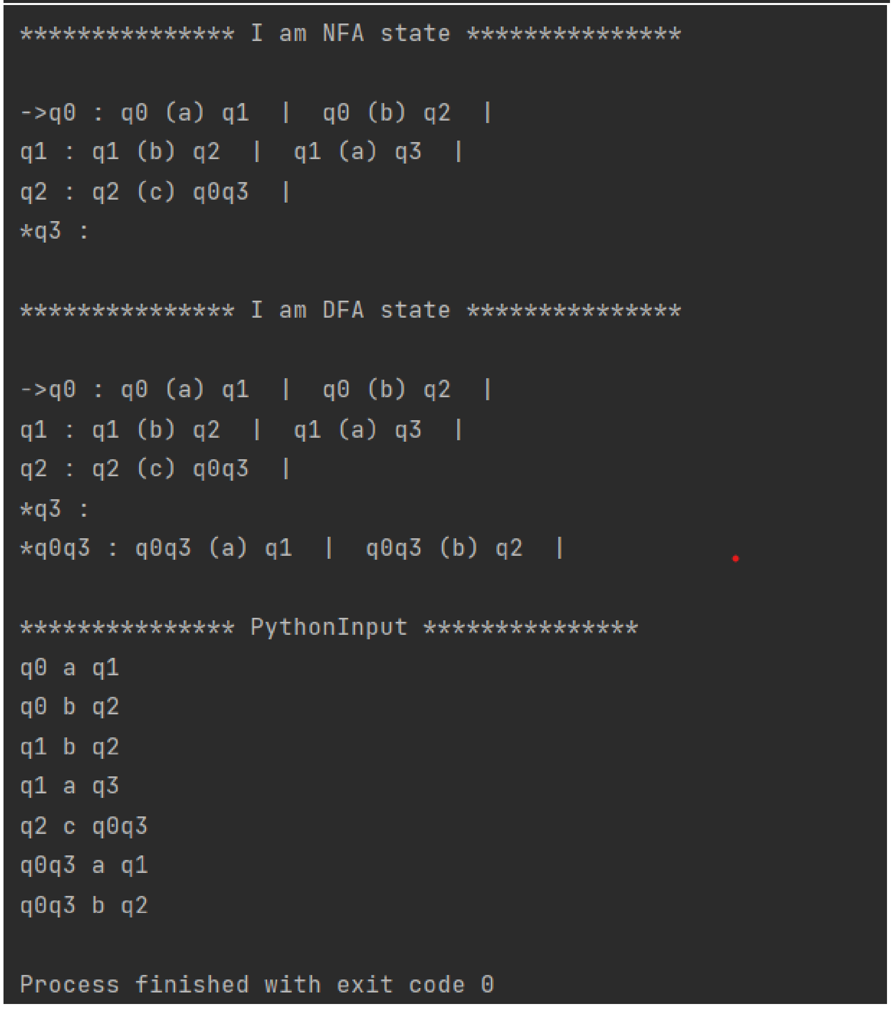
3.5 Main class

```java
import java.util.ArrayList;
import java.util.LinkedHashMap;
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\"!!!");

        LinkedHashMap<String, ArrayList<Edge>> adjList = new LinkedHashMap<>();
        LinkedHashMap<String, ArrayList<Edge>> adjListNFA = new LinkedHashMap<>();
        LinkedHashMap<String, ArrayList<Edge>> adjListDFA = new LinkedHashMap<>();
        ArrayList<String> vertices = new ArrayList<>();
        Graph FA = new Graph(adjList, vertices);

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

        System.out.println("\n" + "*************** I am Finite automation state ***************" + "
    \n");
        FA.printGraph();

        System.out.println("\n" + "*************** I am NFA state ***************" + "\n");
        NFA nfa = new NFA(FA, adjListNFA);
        nfa.graphToNFA();
        nfa.printNFA();

        System.out.println("\n" + "*************** I am DFA state ***************" + "\n");
        DFA dfa = new DFA(nfa, adjListDFA);
        dfa.nfaToDfa();
        dfa.printDFA();

    }
}
```

3.6 Output



```
q0 a q1
q1 b q2
q2 c q0
q1 a q3
q0 b q2
q2 c q3
exit

*************** I am Finite automation state ***************

q0 : q0 (a) q1  |  q0 (b) q2  |
q1 : q1 (b) q2  |  q1 (a) q3  |
q2 : q2 (c) q0  |  q2 (c) q3  |
q3 :

*************** I am NFA state ***************

->q0 : q0 (a) q1  |  q0 (b) q2  |
q1 : q1 (b) q2  |  q1 (a) q3  |
q2 : q2 (c) q0q3  |
*q3 :
```



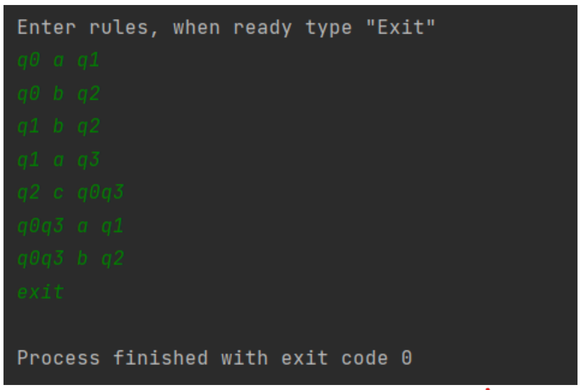
```
*************** I am NFA state ***************

->q0 : q0 (a) q1  |  q0 (b) q2  |
q1 : q1 (b) q2  |  q1 (a) q3  |
q2 : q2 (c) q0q3  |
*q3 :

*************** I am DFA state ***************

->q0 : q0 (a) q1  |  q0 (b) q2  |
q1 : q1 (b) q2  |  q1 (a) q3  |
q2 : q2 (c) q0q3  |
*q3 :
*q0q3 : q0q3 (a) q1  |  q0q3 (b) q2  |

*************** PythonInput ***************
q0 a q1
q0 b q2
q1 b q2
q1 a q3
q2 c q0q3
q0q3 a q1
q0q3 b q2

Process finished with exit code 0
```

4 Implementation graphics

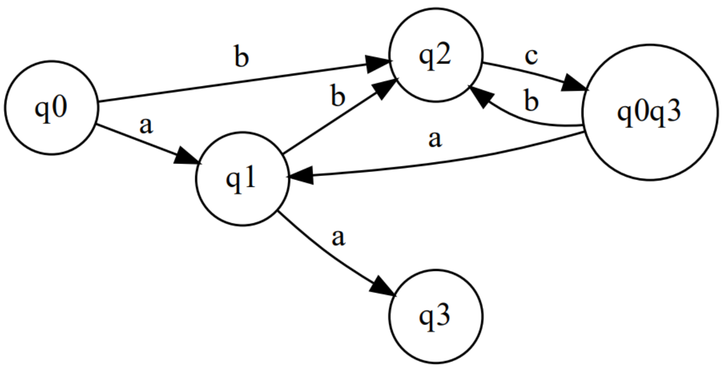
4.1 Code

```
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()
    array = val.split(" ")
    if val == "exit" or val == "Exit":
        break
    else:
        if len(array) == 3:  # S aB

            f.attr('node', shape='circle')
            f.edge(array[0], array[2], label=array[1])


f.view()
```

4.2 Output



```
Enter rules, when ready type "Exit"
q0 a q1
q0 b q2
q1 b q2
q1 a q3
q2 c q0q3
q0q3 a q1
q0q3 b q2
exit

Process finished with exit code 0
```


q0
q1
q2
q3
q0q3
a
b
b
a
c
b
a