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

# Formal Languages and Compiler Design

# Lab 2

Author:Vragalev George

Instructors: Cojuhari Irina

Academic group: FAF-203 2021

2022

Variant 23

$AF=(Q, \Sigma, \delta, q_0, F)$,

$Q = \{ q_0, q_1, q_2 \}$,

$\Sigma = \{ a, b \}$, $F = \{ q_2 \}$.

$\delta (q_0, a ) = q_0$ ,

$\delta (q_0, a ) = q_1$ ,

$\delta (q_1, b ) = q_2$ ,

$\delta (q_0, b ) = q_0$,

$\delta (q_2, b ) = q_2$,

$\delta (q_1, a ) = q_0$.

Convert NFA to DFA

Code:

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

```
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 + "  |  ");
   }

}
```

Graph class:

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

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();
       }
   }
}
```

NFAclass:

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

public class NFA {
   private Graph graph;
   private LinkedHashMap<String, ArrayList<Edge>> nfa;

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

   public LinkedHashMap<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);

               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;
    }
    //find unique weights in an array of edges
    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();
        }
    }
}
```

DFA class:

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

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


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

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

       //until we dont 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));
           }
           //if the state is not single we create a new space for it in dfa
           else {
               dfa.put(newState, new ArrayList<Edge>());
               //we concatenate the nodes at each weight using function
               concatenateNodes(newState);
           }
       }
   }
   //for a double+ state we have to combine the nodes at each weight
   public void concatenateNodes(String nodes) {
       //split the state into separate nodes to loop through
       String[] nodesList = usingSplitMethod(nodes);//q0 q1

       // for each weight we find an edge in the node array we are looping through the edge that has
a specific weight
       for (String weight : this.weights) {
           String resultNode = ""; //q0q1q0
           for (String node : nodesList) {
               if (!findEdgeWithWeight(node, weight).equals("")) {
                   resultNode += findEdgeWithWeight(node, weight); //append the result of found edge
of a specified weight to result
```

```
            }
        }
        //avoids adding empty nodes in dfa
        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 = ""; //q0q1q2
    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()) && !exists(edge.getDest())) {
                return edge.getDest();
            }
        }
    }
    return "empty";
}
```

```
   //function that checks to see if a possibly new node already exists in dfa just in different
order
   public boolean exists(String newNode){
       int check = 0;
       for (String node: dfa.keySet()){
           if (node.length() > 2){
               //create arrays of separate nodes for a complex node like //q0q1
               String[] nodesListNewNode = usingSplitMethod(newNode);//q0 q1
               String[] nodesListNode = usingSplitMethod(node);//q0 q1
               for (String s: nodesListNewNode) {
                   if ( !Arrays.asList(nodesListNode).contains(s)){
                       check = 0; break;
                   }else check++;
               }
               //if we have a check condition that was satisfied the number of elements times, then
the node exists and we should output true that it does
               if (check== nodesListNewNode.length && check == nodesListNode.length)
                   return true;
           }
       } //if the true condition isn't satisfied we return false
       return false;
   }
   //Prints dfa table and input for python code to generate the graph
   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("------------------------DFA state input------------------------");
       for (String s : dfa.keySet()) {
           for (Edge e : dfa.get(s)){
               System.out.println(e.getSrc() + " " + e.getWeight() + " " + e.getDest());
           }
       }
   }
}
```

Main class:

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

       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("----------------Finite automaton state----------------");
       FA.printGraph();

       System.out.println("------------------------NFA state------------------------");
       NFA nfa = new NFA(FA, adjListNFA);
       nfa.graphToNFA();
       nfa.printNFA();

       System.out.println("------------------------DFA state------------------------");
       DFA dfa = new DFA(nfa, adjListDFA);
       dfa.nfaToDfa();
       dfa.printDFA();
   }
}
```

For the input:

```
q0 a q0
q0 a q1
q1 b q2
q0 b q0
q2 b q2
q1 a q0
exit
```

The program will produce the following output:

```
----------------Finite automaton state----------------
q0 : q0 (a) q0  |  q0 (a) q1  |  q0 (b) q0  |
q1 : q1 (b) q2  |  q1 (a) q0  |
q2 : q2 (b) q2  |
------------------------NFA state-----------------------
->q0 : q0 (a) q0q1  |  q0 (b) q0  |
q1 : q1 (b) q2  |  q1 (a) q0  |
*q2 : q2 (b) q2  |
------------------------DFA state-----------------------
->q0 : q0 (a) q0q1  |  q0 (b) q0  |
q0q1 : q0q1 (a) q0q1  |  q0q1 (b) q0q2  |
*q0q2 : q0q2 (a) q0q1  |  q0q2 (b) q0q2  |
------------------------DFA state input-----------------------
q0 a q0q1
q0 b q0
q0q1 a q0q1
q0q1 b q0q2
q0q2 a q0q1
q0q2 b q0q2
```

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()
   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()
"""
Sample input
q0 a q0q1
q0 b q0
q0q1 a q0q1
q0q1 b q0q2
q0q2 a q0q1
q0q2 b q0q2
exit
"""
```

INPUT:

```
C:\Users\vraga\PycharmProjects\PythonScripts\venv\Scripts\python.exe C:/Users/vraga/PycharmProjects/PythonScripts/main.py
Enter rules, when ready type "Exit"
```

```
Enter rules, when ready type "Exit"
q0 a q0q1
q0 b q0
q0q1 a q0q1
q0q1 b q0q2
q0q2 a q0q1
q0q2 b q0q2
exit

Process finished with exit code 0
```

OUTPUT: