|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

по курсу «Конструирование компиляторов»

на тему «Распознавание цепочек регулярного языка»

Студент: Головнева Мария, ИУ7-22М

Преподаватель: Ступников А. А.

Москва

2021

**Цель работы:** приобретение практических навыков реализации важнейших элементов лексических анализаторов на примере распознавания цепочек регулярного языка.

**Задачи работы:**

1. Ознакомиться с основными понятиями и определениями, лежащими в основе построения лексических анализаторов.
2. Прояснить связь между регулярным множеством, регулярным выражением, праволинейным языком, конечно-автоматным языком и недетерминированным конечно-автоматным языком.
3. Разработать, тестировать и отладить программу распознавания цепочек регулярного или праволинейного языка в соответствии с предложенным вариантом грамматики.

**Вариант 5**

Напишите программу, которая в качестве входа принимает произвольное регулярное выражение, и выполняет следующие преобразования:

1. Преобразует регулярное выражение непосредственно в ДКА.
2. По ДКА строит эквивалентный ему КА, имеющий наименьшее возможное количество состояний. Для минимизации ДКА использовать алгоритм за O(n^2) с построением пар различимых состояний.
3. Моделирует минимальный КА для входной цепочки из терминалов исходной грамматики.

**Текст программы**

Class Main

|  |
| --- |
| import java.io.IOException;import java.util.\*;public class Main {public static void main(String[] args) throws IOException {Scanner in = new Scanner(System.in);System.out.print("Введите регулярное выражение: ");char[] regex = in.nextLine().toCharArray();TreeSet<Character> alphabet = new TreeSet<>();ArrayList<Character> arrayC = new ArrayList<>();for (char c : regex) {arrayC.add(c);if (!Regex.isOperator(c)) {alphabet.add(c);}}arrayC.add('#');Regex.addConcatenation(arrayC);arrayC = Regex.postfix(arrayC);System.out.println(arrayC);ExpressionTree et = new ExpressionTree();et.constructTree(arrayC);DFA dfa = new DFA(new ArrayList<Character>(alphabet), et.getRoot().firstpos);dfa.getDTran(et.getFollowpos());Graphiz.drawFA(dfa, "dfa.dot");MFA mfa = new MFA();mfa.minimize(dfa);Graphiz.drawFA(mfa, "mfa.dot");while (true){System.out.print("Введите входную цепочку:\n");char[] text = in.nextLine().toCharArray();if (mfa.match(text)){System.out.println("Входная цепочка допускается автоматом");}else {System.out.println("Входная цепочка не допускается автоматом");}}}} |

Класс с методами для преобразования регулярного выражения:

|  |
| --- |
| import java.util.ArrayList;  import java.util.Stack;  public class Regex {  public static boolean isOperator(char c) {  return c == '(' || c == ')' || c == '\*' || c == '.' || c == '|';  }  public static boolean isUnary(char c) {  return c == '\*';  }  public static int getPriority(char c){  if (c == '\*'){  return 1;  }  else if (c == '.'){  return 2;  }  else if (c == '|'){  return 3;  }  return 4;  }  public static void addConcatenation(ArrayList<Character> regex){  for (int i = 0; i < regex.size() - 1; i++){  if (!isOperator(regex.get(i)) && (!isOperator(regex.get(i+1)) || regex.get(i+1) == '(')){  regex.add(i+1, '.');  }  else if ((regex.get(i) == ')' || regex.get(i) == '\*')  && !isOperator(regex.get(i+1))){  regex.add(i+1, '.');  }  }  }  public static ArrayList<Character> postfix(ArrayList<Character> regex){  Stack<Character> s = new Stack<>();  ArrayList<Character> result = new ArrayList<>();  for (Character c : regex){  if (!isOperator(c)){  result.add(c);  }  else if (c == '('){  s.push(c);  }  else if (c == ')'){  while (s.peek() != '('){  result.add(s.pop());  }  s.pop();  }  else if (isOperator(c)){  while (!s.empty() && getPriority(c) >= getPriority(s.peek())){  result.add(s.pop());  }  s.push(c);  }  }  while (!s.empty()){  result.add(s.pop());  }  return result;  }  } |

Класс синтаксического дерева:

|  |
| --- |
| import java.util.ArrayList;  import java.util.Stack;  public class ExpressionTree {  public ArrayList<Followpos> getFollowpos() {  return followpos;  }  private ArrayList<Followpos> followpos;  public Node getRoot() {  return root;  }  private Node root;  ExpressionTree(){  followpos = new ArrayList<Followpos>();  }  public void constructTree(ArrayList<Character> regex){  Stack<Node> st = new Stack<Node>();  Node t, t1, t2;  int i = 0;  for (Character c: regex) {  if (!Regex.isOperator(c)) {  t = new Node(c);  t.nullable = false;  i++;  t.firstpos.add(i);  t.lastpos.add(i);  st.push(t);  followpos.add(new Followpos(c));  }  else if (Regex.isUnary(c)){  t = new Node(c);  t1 = st.pop();  t.left = t1;  t.nullable = nullable(t);  firstpos(t);  lastpos(t);  followpos(t);  st.push(t);  }  else {  t = new Node(c);  t1 = st.pop();  t2 = st.pop();  t.right = t1;  t.left = t2;  t.nullable = nullable(t);  firstpos(t);  lastpos(t);  followpos(t);  st.push(t);  }  }  root = st.peek();  st.pop();  }  boolean nullable(Node node){  if (node.value == '|'){  return node.left.nullable || node.right.nullable;  }  else if (node.value == '.'){  return node.left.nullable && node.right.nullable;  }  else  return true;  }  void firstpos(Node node){  if (node.value == '|'){  node.firstpos.addAll(node.left.firstpos);  node.firstpos.addAll(node.right.firstpos);  }  else if (node.value == '.'){  if (node.left.nullable){  node.firstpos.addAll(node.left.firstpos);  node.firstpos.addAll(node.right.firstpos);  }  else  node.firstpos.addAll(node.left.firstpos);  }  else  node.firstpos.addAll(node.left.firstpos);  }  void lastpos (Node node){  if (node.value == '|'){  node.lastpos.addAll(node.left.lastpos);  node.lastpos.addAll(node.right.lastpos);  }  else if (node.value == '.'){  if (node.right.nullable){  node.lastpos.addAll(node.left.lastpos);  node.lastpos.addAll(node.right.lastpos);  }  else  node.lastpos.addAll(node.right.lastpos);  }  else  node.lastpos.addAll(node.left.lastpos);  }  void followpos(Node node){  if (node.value == '.'){  for (int i: node.left.lastpos) {  followpos.get(i-1).followpos.addAll(node.right.firstpos);  }  }  else if (node.value == '\*'){  for (int i: node.lastpos) {  followpos.get(i-1).followpos.addAll(node.firstpos);  }  }  }  } |

Класс узел дерева

|  |
| --- |
| import java.util.TreeSet;  class Node {  char value;  Node left, right;  boolean nullable;  TreeSet<Integer> firstpos, lastpos;  Node(char item) {  value = item;  left = right = null;  firstpos = new TreeSet<Integer>();  lastpos = new TreeSet<Integer>();  }  } |

Класс Followpos

|  |
| --- |
| import java.util.TreeSet;  public class Followpos {  char label;  TreeSet<Integer> followpos;  Followpos(char label){  this.label = label;  followpos = new TreeSet<Integer>();  }  } |

Класс ДКА

|  |
| --- |
| import java.util.ArrayList;  import java.util.TreeSet;  class DFA {  ArrayList<TreeSet<Integer>> dStates;  ArrayList<Character> alphabet;  TreeSet<Integer> q0;  ArrayList<TreeSet<Integer>> f;  ArrayList<TreeSet<Integer>[]> dTran;  ArrayList<TreeSet<Integer>> getdStates() {  return dStates;  }  ArrayList<Character> getAlphabet() {  return alphabet;  }    TreeSet<Integer> getQ0() {  return q0;  }  ArrayList<TreeSet<Integer>> getF() {  return f;  }  ArrayList<TreeSet<Integer>[]> getdTran() {  return dTran;  }  DFA(ArrayList<Character> alphabet, TreeSet<Integer> q0){  dStates = new ArrayList<>();  dStates.add(q0);  this.alphabet = alphabet;  this.q0 = q0;  f = new ArrayList<>();  dTran = new ArrayList<>();  }  DFA(){  }  public void getDTran(ArrayList<Followpos> followpos){  for (int i = 0; i < dStates.size(); i++){  TreeSet[] s = new TreeSet[alphabet.size()];  for (int j = 0; j < s.length; j++) {  s[j] = new TreeSet<Integer>();  }  for (int j: dStates.get(i)) {  char a = followpos.get(j - 1).label;  if (a == '#'){  f.add(dStates.get(i));  }  else {  int k = alphabet.indexOf(a);  s[k].addAll(followpos.get(j - 1).followpos);  }  }  for (TreeSet ts: s) {  if (!dStates.contains(ts) && !ts.isEmpty())  dStates.add(ts);  }  dTran.add(s);  }  }  } |

Класс минимизированного ДКА

|  |
| --- |
| import java.util.\*;  public class MFA extends DFA{  MFA(){  super();  }  public void minimize(DFA dfa){  ArrayList[][] backEges = getBackEges(dfa.dStates, dfa.alphabet, dfa.dTran);  int[] reachable = new int[dfa.getdStates().size()];  reachable = bfs(dfa.dStates, dfa.dTran, dfa.q0, reachable);  boolean[][] marked = buildTable(dfa.dStates, dfa.f, backEges, dfa.alphabet);  int[] component = getEquivalenceClasses(reachable, marked);  initializeMFA(component, dfa);  }  public void initializeMFA(int[] component, DFA dfa){  this.dStates = new ArrayList<>();  this.alphabet = dfa.alphabet;  this.dTran = new ArrayList<>();  int startState = 0;  TreeSet<Integer> finalStates = new TreeSet<>();  for (int i = 0; i <= component[component.length - 1]; i++){  this.dStates.add(new TreeSet<>());  this.dTran.add(new TreeSet[this.alphabet.size()]);  }  for (int i = 0; i < component.length; i++){  this.dStates.get(component[i]).addAll(dfa.dStates.get(i));  if (dfa.dStates.get(i).equals(dfa.q0)){  startState = component[i];  }  if (dfa.f.contains(dfa.dStates.get(i))){  finalStates.add(component[i]);  }  for (int j = 0; j < this.alphabet.size(); j++){  if (this.dTran.get(component[i])[j] == null){  dTran.get(component[i])[j] = new TreeSet<>();  }  if (!dfa.dTran.get(i)[j].isEmpty()) {  int c = component[dfa.dStates.indexOf(dfa.dTran.get(i)[j])];  this.dTran.get(component[i])[j] = this.dStates.get(c);  }  }  }  this.q0 = this.dStates.get(startState);  this.f = new ArrayList<>();  for (int i: finalStates){  this.f.add(this.dStates.get(i));  }  }  public int[] getEquivalenceClasses(int[] reachable, boolean[][] marked){  int[] component = new int[reachable.length];  Arrays.fill(component, -1);  for (int i = 0; i < reachable.length; i++){  if (!marked[0][i]){  component[i] = 0;  }  }  int componentsCount = 0;  for (int i = 0; i < reachable.length; i++){  if (reachable[i] == 0){  continue;  }  if (component[i] == -1){  componentsCount++;  component[i] = componentsCount;  for (int j = i+1; j < reachable.length; j++){  if (!marked[i][j]){  component[j] = componentsCount;  }  }  }  }  return component;  }  public boolean[][] buildTable(ArrayList<TreeSet<Integer>> dStates, ArrayList<TreeSet<Integer>> f,  ArrayList<TreeSet>[][] backEges, ArrayList<Character> alphabet){  Queue queue = new LinkedList();  boolean[][] marked = new boolean[dStates.size()][dStates.size()];  for (int i = 0; i < dStates.size(); i++){  for (int j = 0; j < dStates.size(); j++){  if (!marked[i][j] && (f.contains(dStates.get(i)) != f.contains(dStates.get(j)))){  marked[i][j] = true;  marked[j][i] = true;  ArrayList pair = new ArrayList();  pair.add(dStates.get(i));  pair.add(dStates.get(j));  queue.add(pair);  }  }  }  while (!queue.isEmpty()){  ArrayList pair = (ArrayList) queue.poll();  for (int i = 0; i < alphabet.size(); i++){  int u = dStates.indexOf(pair.get(0));  int v = dStates.indexOf(pair.get(1));  if (backEges[u][i] != null && backEges[v][i] != null){  for (TreeSet r: backEges[u][i]){  for (TreeSet s: backEges[v][i]){  if (!marked[dStates.indexOf(r)][dStates.indexOf(s)]){  marked[dStates.indexOf(r)][dStates.indexOf(s)] = true;  marked[dStates.indexOf(s)][dStates.indexOf(r)] = true;  pair = new ArrayList();  pair.add(r);  pair.add(s);  queue.add(pair);  }  }  }  }  }  }  return marked;  }  public int[] bfs(ArrayList<TreeSet<Integer>> dStates, ArrayList<TreeSet<Integer>[]> dTran, TreeSet<Integer> q0,  int[] reachable){  Queue queue = new LinkedList();  queue.add(q0);  reachable[dStates.indexOf(q0)] = 1;  while (!queue.isEmpty()){  TreeSet<Integer> v = (TreeSet<Integer>) queue.poll();  TreeSet<Integer>[] neighbors = dTran.get(dStates.indexOf(v));  for (int i = 0; i < neighbors.length; i++){  if (!neighbors[i].isEmpty()){  if (reachable[dStates.indexOf(neighbors[i])] == 0){  queue.add(neighbors[i]);  reachable[dStates.indexOf(neighbors[i])] = 1;  }  }  }  }  return reachable;  }  public ArrayList[][] getBackEges(ArrayList<TreeSet<Integer>> dStates, ArrayList<Character> alphabet,  ArrayList<TreeSet<Integer>[]> dTran){  ArrayList[][] backEges = new ArrayList[dStates.size()][alphabet.size()];  int k = 0;  for (TreeSet<Integer>[] s: dTran){  for (int i = 0; i < s.length; i++){  if (!s[i].isEmpty()) {  int n = dStates.indexOf(s[i]);  if (backEges[n][i] == null) {  backEges[n][i] = new ArrayList();  }  backEges[n][i].add(dStates.get(k));  }  }  k++;  }  return backEges;  }  public boolean match(char[] text){  TreeSet<Integer> state = this.q0;  for (char c: text){  if (this.alphabet.indexOf(c) == -1){  return false;  }  state = this.dTran.get(this.dStates.indexOf(state))[this.alphabet.indexOf(c)];  }  return this.f.contains(state);  }  } |

Класс визуализации:

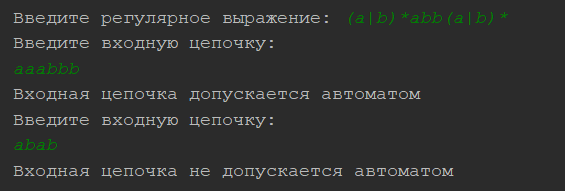
|  |
| --- |
| import java.io.FileWriter;  import java.io.IOException;  import java.util.TreeSet;  public class Graphiz {  public static void drawFA(DFA dfa, String filename) throws IOException {  FileWriter writer = new FileWriter(filename);  writer.write("digraph state\_machine {\n rankdir = LR;\n node [shape = doublecircle];");  String text = "";  for (TreeSet ts : dfa.getF()) {  text = text + " " + ts.toString().replaceAll("[ ,\\[\\]]+", "");  }  text = text + ";\n";  writer.write(text);  writer.write(" node [shape = oval];\n");  text = " start -> " + dfa.getQ0().toString().replaceAll("[ ,\\[\\]]+", "") + ";\n";  writer.write(text);  for (int i = 0; i < dfa.getdStates().size(); i++) {  for (int j = 0; j < dfa.getAlphabet().size(); j++) {  if (!dfa.getdTran().get(i)[j].isEmpty()) {  text = " ";  text = text + dfa.getdStates().get(i).toString().replaceAll("[ ,\\[\\]]+", "") + " -> "  + dfa.getdTran().get(i)[j].toString().replaceAll("[ ,\\[\\]]+", "") +  " [label = \"" + dfa.getAlphabet().get(j) + "\"];\n";  writer.write(text);  }  }  }  writer.write("}");  writer.flush();  }  } |

**Набор тестов**

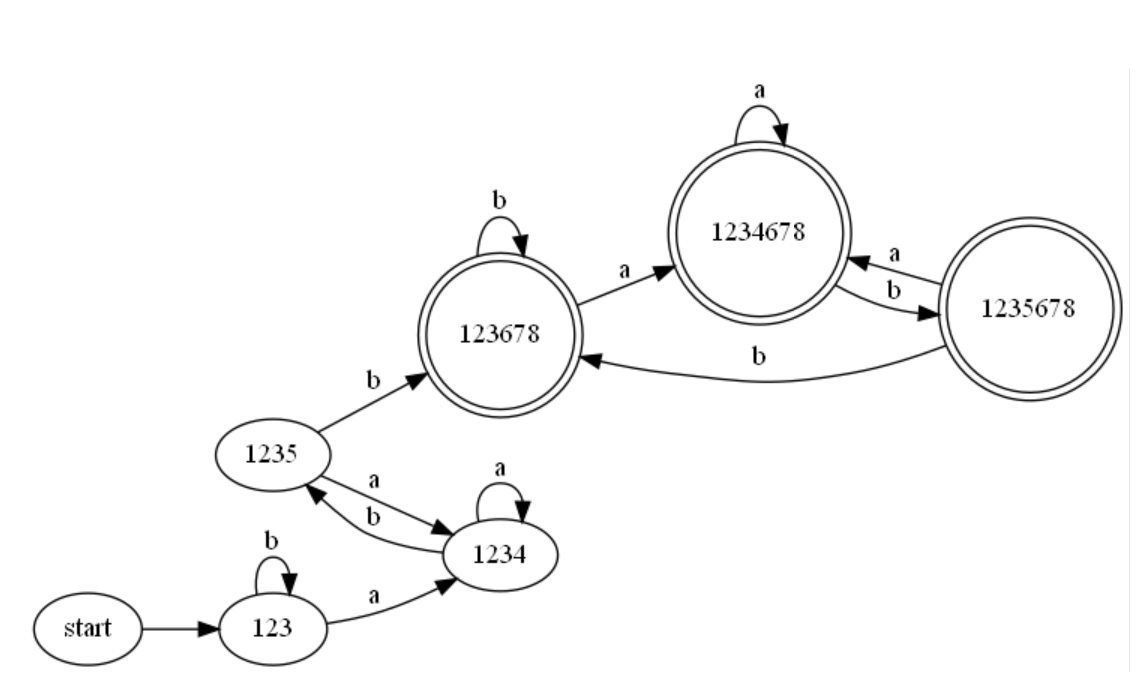
|  |  |  |
| --- | --- | --- |
| Регулярное выражение | Входная цепочка | Результат |
| (a|b)\*abb | abab  abb  ababaaabb  aaabbb | false  true  true  false |
| a\* | a  aaa  пустая строка  пробел  b | true true  true  false  false |
| a|b | a  b  aaaa  abab  пустая строка  пробел  c | true  true  false  false  false  false  false |

**Пример работы программы**

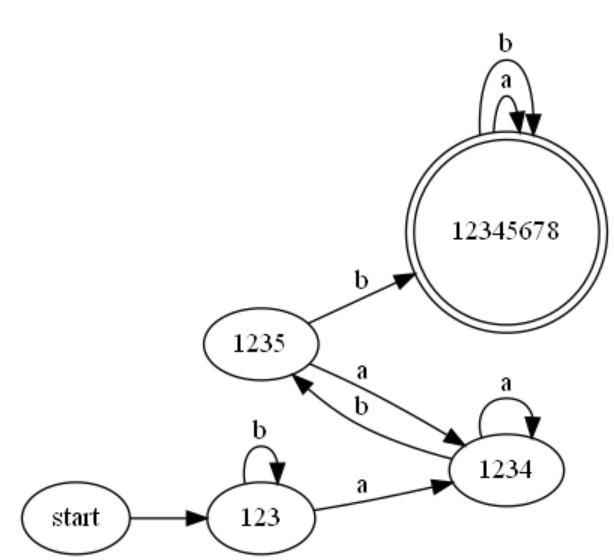
Консоль:

****

Построенный ДКА:

****

Минимизированный ДКА:



**Вывод**

Распознавание цепочек регулярного языка является центральной задачей лексического анализа, который образует первый этап процесса компиляции. В данной лабораторной работе мы:

1. Ознакомились с основными понятиями и определениями, лежащими в основе построения лексических анализаторов.
2. Прояснили связь между регулярным множеством, регулярным выражением, праволинейным языком, конечно-автоматным языком и недетерминированным конечно-автоматным языком.
3. Разработали, протестировали и отладили программу распознавания цепочек регулярного языка в соответствии с предложенным вариантом грамматики.

**Список литературы**

1. Converting Regular Expressions to Postfix Notation with the Shunting-Yard Algorithm [Электронный ресурс] — Режим доступа. — URL: <https://gregorycernera.medium.com/converting-regular-expressions-to-postfix-notation-with-the-shunting-yard-algorithm-63d22ea1cf88>
2. Expression Tree [Электронный ресурс] — Режим доступа. — URL: <https://www.geeksforgeeks.org/expression-tree/>
3. Минимизация ДКА, алгоритм за O(n^2) с построением пар различимых состояний [Электронный ресурс] — Режим доступа. — URL: <http://neerc.ifmo.ru/wiki/index.php?title=Минимизация_ДКА,_алгоритм_за_O(n%5E2)_с_построением_пар_различимых_состояний>
4. АХО А.В, ЛАМ М.С., СЕТИ Р., УЛЬМАН Дж.Д. Компиляторы: принципы, технологии и инструменты. – М.: Вильямс, 2008.