Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Магнитогорский государственный технический университет

им. Г. И. Носова»

(ФГБОУ ВО «МГТУ им. Г.И. Носова»)

Кафедра вычислительной техники и программирования

**Лабораторная работа №2**

по дисциплине «Теория автоматов»

на тему: «Синтез автомата без памяти»

Исполнитель: Варламов М.Н. студент 2 курса, группа АВб-19-1

Руководитель: Ильина E.A., кандидат педагогических наук, доцент кафедры ВТиП

Магнитогорск, 2021

**ОПИСАНИЕ РАЗРАБОТАННЫХ КЛАССОВ И ФУНКЦИЙ**

Код программы состоит из двух основных классов. Класс Calculator сокращает введенное в него логическое выражение, а класс Painter отрисовывает по упрощенному выражению диаграмму. Рассмотрим эти классы подробнее.

Класс Calculator. Упрощение логического выражения происходит с помощью метода Куина-Макклауски. Данный метод рекурсивно обрабатывает входную последовательность логических выражений, записанных определенным образом. 1 обозначает, что логический оператор является “положительным”, а 0 – обозначает отрицание. Индекс обозначает определенный логический оператор. Каждый дизъюнкт записывается отдельной строкой и хранится в списке. После чего с данной последовательность производятся следующие операции:

1. Представим совершенную ДНФ булевой функции *f*(X1,…,Xn) списком двоичных векторов.

2. Упорядочим элементы списка по возрастанию весов векторов. В результате список разобьется на равновесные подмножества векторов.

3. Выполним операции склеивания над векторами соседних по весу подмножеств. Результаты склеиваний занесем в новый список, а векторы, участвующие в склеивании, отметим символом \*. В каждом новом списке векторы оказываются разбитыми на равновесные подмножества.

4. Если новый список пуст, переходим к п. 5 алгоритма. Иначе переходим к п. 3 и работаем с новым списком векторов.

5. Выпишем из всех полученных списков, в том числе из исходного упорядоченного списка булевых векторов, векторы, не отмеченный символом \*. Образуем из выписанных векторов список, представляющий сокращенную ДНФ булевой функции *f*(X1,…,Xn).

6. Переходим, если это требуется, к записи, сокращенной ДНФ в виде дизъюнкции конъюнкций.

На выходе мы получаем укороченную последовательность, с дополнительным обозначительным символом ‘-’, который означает отсутствие логического оператора в данной позиции. На рисунке 1 показан пример входной и выходной последовательности метода Куина-Макклауски.



Рисунок 1 – пример входной и выходной последовательности метода Куина-Макклауски

Класс Painter. Данный класс получает на вход упрощённую логическую последовательность и циклически начинает отрисовывать элементы. В каждой итерации цикла происходят следующие действия.

1. Отрисовывается логический блок “ИЛИ”
2. Запускается цикл для проверки каждой позиции в исходном дизъюнкте.
   1. Если на месте логического оператора стоит “-”, происходит пропуск отрисовки
   2. Если на месте логического оператора стоит “1”, происходит отрисовка входа логического оператора в блок дизъюнкта.
   3. Если на месте логического оператора стоит “0”, происходит отрисовка логического блока “НЕ”, с последующей отрисовкой входа в блок дизъюнкта.
3. Происходит увеличение переменных, отвечающих за координаты отрисовки последующих дизъюнктов.

В конце отрисовывается блок конъюнкта, в который входят все дизъюнкты, отрисованые раннее.

Остальные классы программы выполняют утилитарные задачи.

Класс Controller необходим для связывания между графической и логической частями программы. В частности, функция convertAndCalculate() преобразует введенные пользователем данные в тот вид, который понятен алгоритму, вызывает данный алгоритм и в результате возвращает объект класса DataHolder, который и содержит в себе преобразованные данные и результат работы алгоритма.

Класс Utils и его метод performSDNF() выполняет обратную функцию, он преобразует данные из вида, понятного для алгоритма в вид, понятный пользователю. Пример данного преобразования показан на рисунке 2.

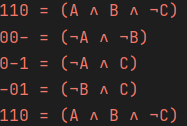


Рисунок 1 – пример преобразования данных методом performSDNF()

Класс Form является графическим интерфейсом программы. Он состоит из множества различных объектов, таких как: JTextField – для ввода и вывода данных, JLabel – для вывода текстовой информации, JPanel – для отрисовки диаграммы (класс Painter добавляется на данный объект, тем самым отрисовывая диаграмму).

Ну и класс Main – служит для настройки графического интерфейса и запуска программы.

**ОПИСАНИЕ ПРОГРАММЫ**

На рисунке 3 показан графический интерфейс разработанной программы.

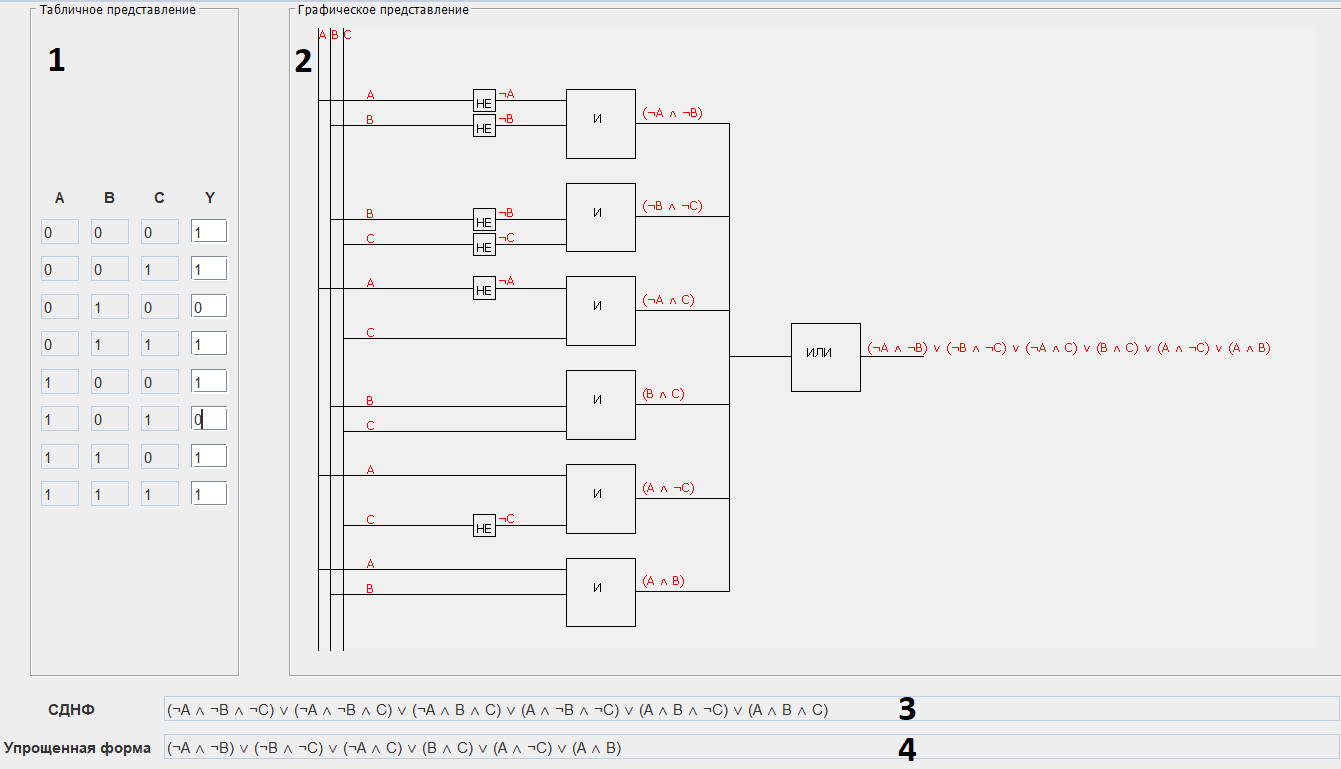


Рисунок 3 – графический интерфейс программы

На данном рисунке цифрой 1 отмечено табличное представление автомата без памяти. Пользователь должен ввести данные в столбец “Y” 0 либо 1.

Цифрой 2 отмечено графическое представление автомата без памяти. При изменении любой из строк столбца “Y” будет меняться диаграмма, на соответствующую сокращенной форме СДНФ, указанной цифрой 4.

Цифрой 3 обозначена совершенная дизъюнктивная нормальная форма, которая была составлена по таблице, указанной цифрой 1.

Интерактивность достигается при помощи отслеживания изменения состояния таблицы 1. Пользователю не нужно нажимать кнопок для просчёта и построения диаграммы, она будет строиться сама по введенным данным.

**ВЫВОД**

В результате проделанной работы был изучен метод Куина-Макклауски. Был разработан алгоритм для отрисовки логических диаграмм и из данных составляющих была разработана программа для составления автомата без памяти для совершенной дизъюнктивной нормальной формы.

Программа была написана на языке программирования JAVA с использованием графической библиотеки SWING. Листинг программы см. ниже.

**ЛИСТИНГ**

import javax.swing.\*;

import java.awt.\*;

public class Main {

public static void main(String[] args) {

Form gui = new Form();

gui.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

gui.setSize(new Dimension(620,500));

gui.pack();

gui.setVisible(true);

}

}

import javax.swing.\*;

import javax.swing.event.DocumentEvent;

import javax.swing.event.DocumentListener;

import javax.swing.text.BadLocationException;

import java.util.ArrayList;

import java.util.Arrays;

public class Form extends JFrame {

private JPanel panel;

private JTextField rTF0;

private JTextField yTF1;

private JTextField yTF2;

private JTextField yTF3;

private JTextField yTF4;

private JTextField yTF5;

private JTextField yTF6;

private JTextField yTF7;

private JTextField a0TextField;

private JTextField a0TextField10;

private JTextField a1TextField4;

private JTextField a1TextField5;

private JTextField a1TextField8;

private JTextField a0TextField8;

private JTextField a1TextField9;

private JTextField a1TextField3;

private JTextField SDNFTF;

private JTextField SDNFUTF;

private JTextField a0TextField2;

private JTextField a0TextField1;

private JTextField textField5;

private JTextField textField6;

private JTextField a1TextField7;

private JTextField a1TextField6;

private JTextField a1TextField2;

private JTextField textField3;

private JTextField textField4;

private JTextField textField7;

private JPanel paintPanel;

public Form() {

try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch (ClassNotFoundException | InstantiationException | IllegalAccessException | UnsupportedLookAndFeelException e) {

e.printStackTrace();

}

setContentPane(panel);

/\*SwingUtilities.invokeLater(() -> {

scrollPane.removeAll();

scrollPane.add(new Painter(null));

repaint();

revalidate();

});\*/

JTextField[] tfs = {rTF0, yTF1, yTF2, yTF3, yTF4, yTF5, yTF6, yTF7};

DocumentListener listener = new DocumentListener() {

@Override

public void insertUpdate(DocumentEvent e) {

if (e.getDocument().getLength() > 1) {

SwingUtilities.invokeLater(() -> {

try {

e.getDocument().remove(1, e.getLength());

} catch (BadLocationException ignored) {

}

});

}

try {

if (!e.getDocument().getText(0, 1).equals("0") && !e.getDocument().getText(0, 1).equals("1")) {

SwingUtilities.invokeLater(() -> {

try {

e.getDocument().remove(0, 1);

} catch (BadLocationException badLocationException) {

badLocationException.printStackTrace();

}

});

}

} catch (BadLocationException badLocationException) {

badLocationException.printStackTrace();

}

int[] m = new int[tfs.length];

try {

for (int i = 0; i < m.length; i++) {

String s = tfs[i].getText();

int a = Integer.parseInt(s);

if (a < 2)

m[i] = a;

}

drawData(Controller.convertAndCalculate(m));

} catch (NumberFormatException ignored) {

}

}

@Override

public void removeUpdate(DocumentEvent e) {}

@Override

public void changedUpdate(DocumentEvent e) {}

};

rTF0.getDocument().addDocumentListener(listener);

yTF1.getDocument().addDocumentListener(listener);

yTF2.getDocument().addDocumentListener(listener);

yTF3.getDocument().addDocumentListener(listener);

yTF4.getDocument().addDocumentListener(listener);

yTF5.getDocument().addDocumentListener(listener);

yTF6.getDocument().addDocumentListener(listener);

yTF7.getDocument().addDocumentListener(listener);

}

private void drawData(DataHolder dh) {

StringBuilder sbSDNF = new StringBuilder();

for (int i = 0; i < dh.getSDNF().size(); i++){

if (i != dh.getSDNF().size() - 1)

sbSDNF.append(Utils.performSDNF(dh.getSDNF().get(i))).append(" ∨ ");

else

sbSDNF.append(Utils.performSDNF(dh.getSDNF().get(i)));

}

StringBuilder sbSDNFU = new StringBuilder();

for (int i = 0; i < dh.getSDNFU().size(); i++){

if (i != dh.getSDNFU().size() - 1)

sbSDNFU.append(Utils.performSDNF(dh.getSDNFU().get(i))).append(" ∨ ");

else

sbSDNFU.append(Utils.performSDNF(dh.getSDNFU().get(i)));

}

SwingUtilities.invokeLater(() -> {

SDNFTF.setText(sbSDNF.toString());

SDNFUTF.setText(sbSDNFU.toString());

paintPanel.removeAll();

paintPanel.add(new Painter(dh.getSDNFU(),sbSDNFU.toString()));

panel.revalidate();

panel.repaint();

});

}

private void createUIComponents() {

ArrayList<String> bb = new ArrayList<>(Arrays.asList("123", "423", "234", "234"));

paintPanel = new JPanel();

Painter p = new Painter(bb, "");

paintPanel.add(p);

}

}

import javax.swing.\*;

import java.awt.\*;

import java.util.ArrayList;

public class Painter extends JPanel {

private static final int DIM\_WIDTH = 800;

private static final int DIM\_HEIGHT = 500;

private ArrayList<String> data;

String res;

private int x = 200, y = 50, w = 55, h = 55, wNOT = w / 3, hNOT = h / 3;

// x, y - начальные значения, h,w - размеры блоков И, wNOT,hNOT - размеры блоков НЕ

public Painter(ArrayList<String> data, String res) {

this.data = data;

this.res = res;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

y = 50;

g.setColor(Color.BLACK);

if (data != null) {

int y1 = y;

int[] xN = {1, 11, 21}; // A B C

g.drawLine(xN[0],1,1,800);

g.drawLine(xN[1],1,11,800);

g.drawLine(xN[2],1,21,800);

g.setColor(Color.RED);

g.drawString("A", xN[0] + 1, 10);

g.drawString("B", xN[1] + 1, 10);

g.drawString("C", xN[2] + 1, 10);

g.setColor(Color.BLACK);

for (int i = 0; i < data.size(); i++) {

String s = data.get(i);

g.drawRect(x, y1, w, h);

g.drawString("И", x + w/2 - 5, y1 + h/2);

g.setColor(Color.RED);

g.drawString(Utils.performSDNF(s), x + w + 5, y1 + w/2 - 5);

g.setColor(Color.BLACK);

g.drawLine(x + w,y1 + w/2, x + w + 75 ,y1 + w/2);

int yIn = y1;

for (int j = 0; j < 3; j++) {

if (s.charAt(j) == '-') {

yIn += 20;

continue;

}

if (s.charAt(j) == '1'){

g.setColor(Color.RED);

g.drawString(String.valueOf((char)(j + 65)), x - 160, yIn + 8);

g.setColor(Color.BLACK);

g.drawLine(xN[j],yIn + 9, x ,yIn + 9);

yIn += 20;

continue;

}

if (s.charAt(j) == '0'){

g.drawRect(x - 75, yIn, wNOT, hNOT);

g.drawString("НЕ", x - 72, yIn + 15);

g.setColor(Color.RED);

g.drawString("¬" + (char) (j + 65), x - 55, yIn + 7);

g.setColor(Color.BLACK);

g.drawLine(x - 57,yIn + 9, x ,yIn + 9);

g.setColor(Color.RED);

g.drawString(String.valueOf((char)(j + 65)), x - 160, yIn + 8);

g.setColor(Color.BLACK);

g.drawLine(xN[j],yIn + 9, x - 75 ,yIn + 9);

yIn += 20;

}

}

y1 += 75;

}

g.drawLine(x + w + 75,y + w/2,x + w + 75,y1 - w + 6);

int temY = ((y1 - w + 6) + (y + w/2)) / 2;

g.drawLine(x + w + 75, temY, x + w + 125, temY);

g.drawRect(x + w + 125, temY - w/2, w, h);

g.drawString("ИЛИ",x + w + 125 + w/2 - 15, temY);

g.setColor(Color.RED);

g.drawString(res, x + w + 130 + w, temY - 4);

g.setColor(Color.BLACK);

g.drawLine(x + w + 125 + w, temY, x + w + 175 + w, temY);

}

}

@Override

public Dimension getPreferredSize() {

return new Dimension(DIM\_WIDTH, DIM\_HEIGHT);

}

}

import java.util.ArrayList;

import java.util.Collections;

public class Controller {

public static DataHolder convertAndCalculate(int[] data) {

ArrayList<String> list = new ArrayList<>();

for (int i = 0; i < data.length; i++) {

StringBuilder sb = new StringBuilder();

sb.append(Integer.toBinaryString(i + 8));

sb.delete(0, 1);

if (data[i] == 1) {

list.add(sb.toString());

}

}

System.err.println(list);

for (int a : data){

if (a != 1) {

System.err.println(Calculator.calculate(list));

return new DataHolder(list, Calculator.calculate(list));

}

}

return new DataHolder(list, new ArrayList<>(Collections.singletonList("111")));

}

}

import java.util.ArrayList;

import java.util.Arrays;

public class Calculator {

public static ArrayList<String> calculate(ArrayList<String> data) {

String[] newData = Arrays.copyOf(data.toArray(), data.size(), String[].class);

int size = newData.length;

ArrayList<String> im = new ArrayList<>();

String[] mark = repeatelem(0, size);

for (int i = 0; i < size; i++) {

for (int j = i + 1; j < size; j++) {

String c = combine(newData[i], newData[j]);

if (!c.equals("")) {

im.add(c);

mark[i] = "1";

mark[j] = "1";

}

}

}

String[] mark2 = repeatelem(0, im.size());

for (int i = 0; i < im.size(); i++) {

for (int j = i + 1; j < im.size(); j++) {

if ((i != j) && (mark2[j].equals("0")) && (im.get(i).equals(im.get(j)))) {

mark2[j] = "1";

}

}

}

ArrayList<String> im2 = new ArrayList<>();

for (int i = 0; i < im.size(); i++) {

if (mark2[i].equals("0")) {

im2.add(im.get(i));

}

}

int m = 0;

ArrayList<String> IM = new ArrayList<>();

for (int i = 0; i < size; i++) {

if (mark[i].equals("0")) {

IM.add(newData[i]);

m++;

}

}

if ((m != size) && (size != 1)) {

IM.addAll(calculate(im2));

}

return IM;

}

private static String[] addOneAndRecurse(ArrayList<String> st, int remaining, int elem) {

st.add(String.valueOf(elem));

if (remaining > 1)

addOneAndRecurse(st, remaining - 1, elem);

return st.toArray(new String[0]);

}

private static String[] repeatelem(int elem, int count) {

ArrayList<String> st = new ArrayList<>();

return addOneAndRecurse(st, count, elem);

}

private static String combine(String m, String n) {

int count = 0;

StringBuilder sb = new StringBuilder();

for (int i = 0; i < m.length(); i++) {

if (m.charAt(i) == n.charAt(i)) {

sb.append(m.charAt(i));

} else if (m.charAt(i) != n.charAt(i)) {

sb.append('-');

count++;

}

}

if (count > 1) {

return "";

}

return sb.toString();

}

}

import java.util.ArrayList;

public class DataHolder {

private final ArrayList<String> SDNF, SDNFU;

public DataHolder(ArrayList<String> SDNF, ArrayList<String> SDNFU){

this.SDNF = SDNF;

this.SDNFU = SDNFU;

}

public ArrayList<String> getSDNF() {

return SDNF;

}

public ArrayList<String> getSDNFU() {

return SDNFU;

}

}