Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №3**

*з курсу «Автоматизація проектування комп’ютерних систем»*

Виконав

студент групи ІО-73

Захожий Ігор

Номер залікової книжки: 7308

Київ-2010

**Тема роботи**

Автоматизація розмітки блок-схем алгоритмів.

**Мета роботи**

Здобуття навичок з автоматизації процедури розмітки алгоритмів за методами Мілі та Мура.

**Завдання**

1. Представити номер залікової книжки в двійковому вигляді:

730810 = 11100100011002

2. В залежності значення розрядів номера залікової книжки розмітити алгоритм, перевірений на наявність помилок у попередній роботі згідно методу:

|  |  |
| --- | --- |
| **n4 n1** | **Метод розмітки** |
| 1 | Мура |

3. Згідно отриманої розмітки, графічно відобразити граф переходів для алгоритму.

4. Розробити формат файлу для зберігання графу переходів. Реалізувати функцію збереження/відновлення графу переходів. Тип формату в залежності від розряду номера залікової книжки:

|  |  |
| --- | --- |
| **n2** | **Тип формату** |
| 0 | Бінарний |

**Опис програми**

Для того, щоб розмітити методом Мура блок-схему алгоритму, побудовану редактором блок-схем (лабораторна робота №1), та перевірену на наявність безкінечних циклів (лабораторна робота №2), необхідно вибрати пункт головного меню Execute -> Mark Algorithm For Moore Automat. Для алгоритму, що зображений на рисунку 1, розмічена таким чином блок-схема показана на рисунку 2.

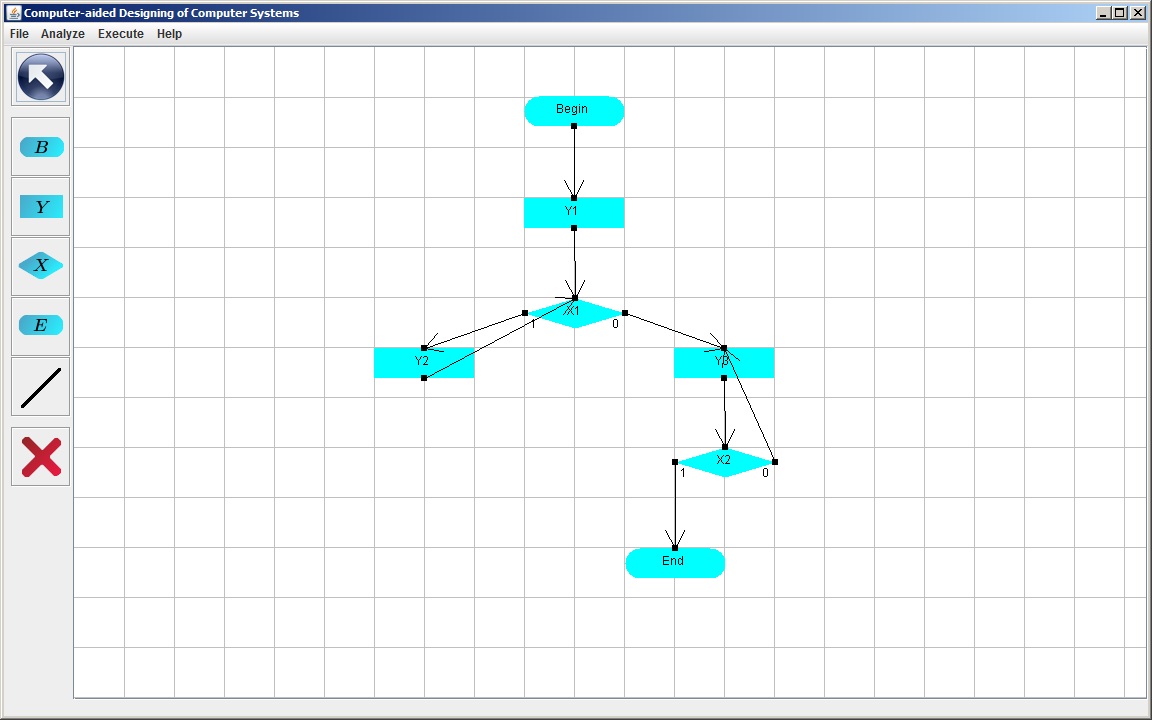


Рисунок 1

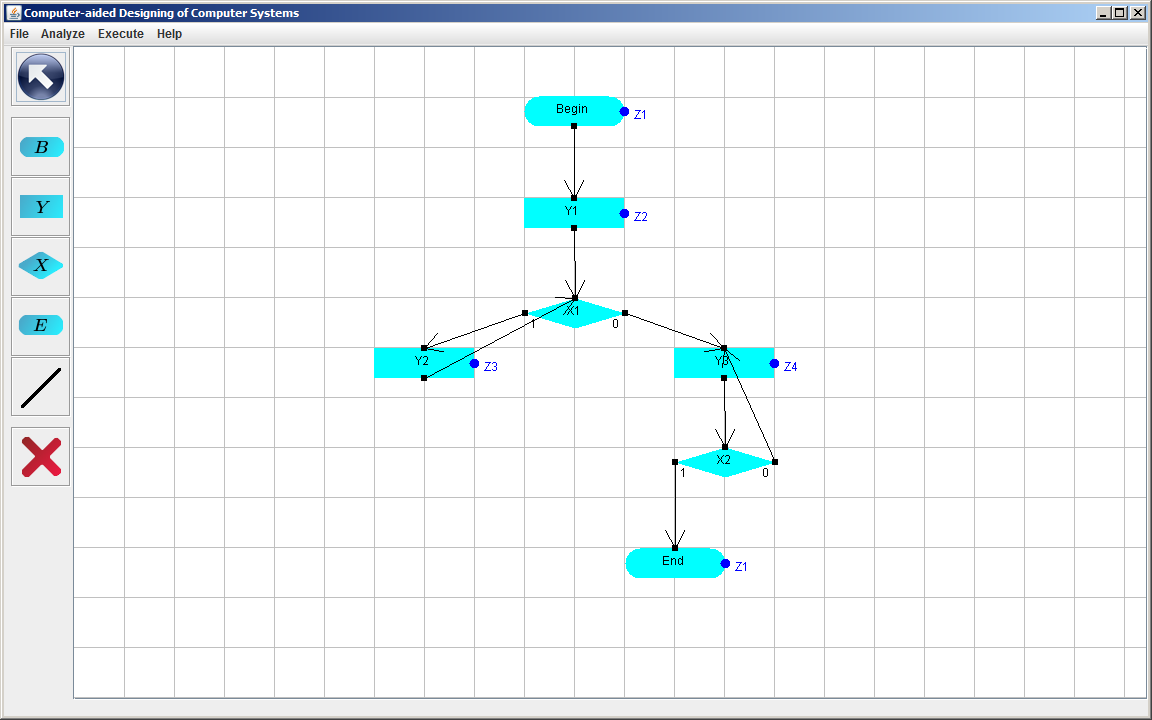


Рисунок 2

Після цього можна побудувати граф переходів для автомату Мура. Для цього необхідно вибрати пункт головного меню Execute -> Build Graph of Moore Automat… . Результат даної дії для даного алгоритму (рисунок 2) зображений на рисунку 3. Дана операція була реалізована за допомогою видалення з матриці зв’язків логічних вершин та запису в матрицю умов переходів, що визначені за допомогою рекурсивного алгоритму пошуку наступної кінцевої або операторної вершини від кожної операторної та початкової вершин алгоритму.

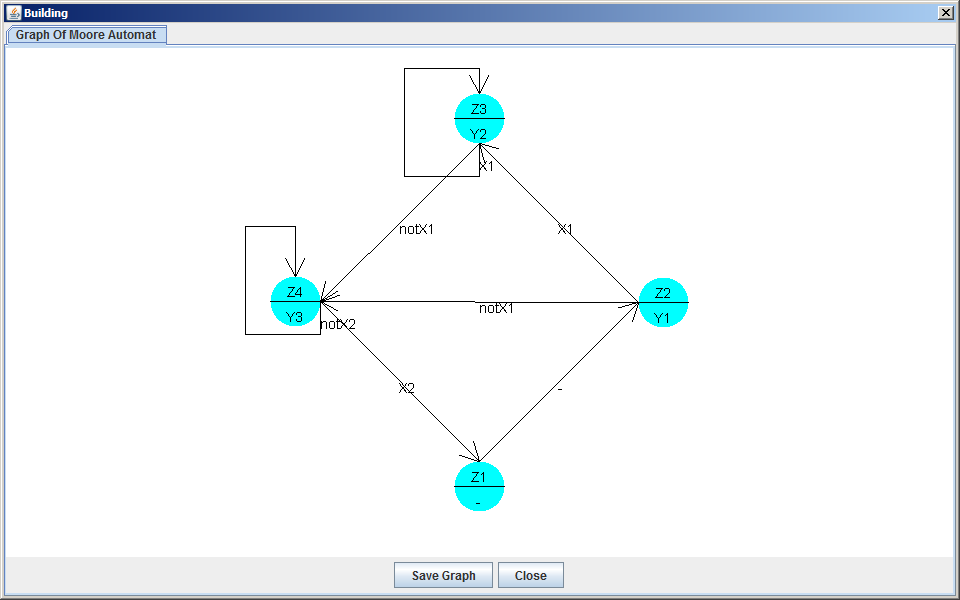


Рисунок 3

Для збереження отриманого графу переходів необхідно натиснути кнопку Save Graph та ввести ім’я файлу.

Для відновлення збереженого графу переходів необхідно скористатися пунктом головного меню File -> Open Graph Of Moore Automat. Граф зберігається у файл бінарного формату. Це реалізовано за допомогою серіалізації об’єкту класу MooreAutomat, що містить всі дані про граф переходів.

**Лістинг програми**

package automat.moore;

import gsa.GSAModel;

import java.io.\*;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 16.10.2010

\* Time: 14:16:11

\* To change this template use File | Settings | File Templates.

\*/

public class MooreAutomat implements Serializable {

private String[] stateNames;

private int[][] yNumbers;

private int[][] connectionMatrix;

private int[][] xNumbers;

private boolean[][] xValues;

public MooreAutomat(String[] stateNames, int[][] yNumbers, int[][] connectionMatrix, int[][] xNumbers, boolean[][] xValues) {

this.stateNames = stateNames;

this.yNumbers = yNumbers;

this.connectionMatrix = connectionMatrix;

this.xNumbers = xNumbers;

this.xValues = xValues;

}

public MooreAutomat(GSAModel model) {

int[] nodesType = model.getNodesType();

int[][] nodesConnectionMatrix = model.getConnectionMatrix();

int[][] signalMatrix = model.getSignalMatrix();

int count = 0;

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

if (nodesType[i] == 2) {

count++;

}

}

stateNames = new String[nodesType.length - count - 1];

connectionMatrix = new int[nodesType.length - count - 1][];

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

connectionMatrix[i] = new int[nodesType.length - count - 1];

for (int j = 0; j < connectionMatrix[i].length; j++) {

connectionMatrix[i][j] = -1;

}

}

yNumbers = new int[nodesType.length - count - 1][];

int z1 = -1;

int z = 2;

int j = 0;

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

if (((nodesType[i] == 0) || (nodesType[i] == 3)) && (z1 == -1)) {

z1 = j;

stateNames[j] = "Z1";

yNumbers[j] = null;

j++;

}

else {

if (nodesType[i] == 1) {

stateNames[j] = "Z" + String.valueOf(z++);

yNumbers[j] = new int[signalMatrix[i].length];

for (int k = 0; k < signalMatrix[i].length; k++) {

yNumbers[j][k] = signalMatrix[i][k];

}

j++;

}

}

}

xNumbers = null;

xValues = null;

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

if ((nodesType[i] == 0) || (nodesType[i] == 1)) {

for (j = 0; j < nodesConnectionMatrix[i].length; j++) {

if (nodesType[j] == 1) {

if (nodesConnectionMatrix[i][j] > 0) {

int[][] newXNumbers;

if (xNumbers != null) {

newXNumbers = new int[xNumbers.length + 1][];

for (int x = 0; x < xNumbers.length; x++) {

newXNumbers[x] = xNumbers[x];

}

}

else {

newXNumbers = new int[1][];

}

newXNumbers[newXNumbers.length - 1] = null;

xNumbers = newXNumbers;

boolean[][] newXValues;

if (xValues != null) {

newXValues = new boolean[xValues.length + 1][];

for (int x = 0; x < xValues.length; x++) {

newXValues[x] = xValues[x];

}

} else {

newXValues = new boolean[1][];

}

newXValues[newXValues.length - 1] = null;

xValues = newXValues;

connectionMatrix[getStateNumber(nodesType, i)][getStateNumber(nodesType, j)] = xNumbers.length - 1;

}

}

else {

if (nodesType[j] == 3) {

if (nodesConnectionMatrix[i][j] > 0) {

int[][] newXNumbers;

if (xNumbers != null) {

newXNumbers = new int[xNumbers.length + 1][];

for (int x = 0; x < xNumbers.length; x++) {

newXNumbers[x] = xNumbers[x];

}

} else {

newXNumbers = new int[1][];

}

newXNumbers[newXNumbers.length - 1] = null;

xNumbers = newXNumbers;

boolean[][] newXValues;

if (xValues != null) {

newXValues = new boolean[xValues.length + 1][];

for (int x = 0; x < xValues.length; x++) {

newXValues[x] = xValues[x];

}

} else {

newXValues = new boolean[1][];

}

newXValues[newXValues.length - 1] = null;

xValues = newXValues;

connectionMatrix[getStateNumber(nodesType, i)][getStateNumber(nodesType, j)] = xNumbers.length - 1;

}

}

else {

if (nodesType[j] == 2) {

if (nodesConnectionMatrix[i][j] > 0) {

int[] newXNumbers = null;

boolean[] newXValues = null;

stepToOperatorNode(getStateNumber(nodesType, i), j, newXNumbers, newXValues, nodesType, nodesConnectionMatrix, signalMatrix);

}

}

}

}

}

}

}

}

public static void writeToFile(File file, MooreAutomat automat) throws IOException {

ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream(file));

output.writeObject(automat);

output.close();

}

public static MooreAutomat readFromFile(File file) throws IOException, ClassNotFoundException {

ObjectInputStream input = new ObjectInputStream(new FileInputStream(file));

MooreAutomat automat = (MooreAutomat) input.readObject();

input.close();

return automat;

}

private void stepToOperatorNode(int start, int from, int[] stepXNumbers, boolean[] stepXValues, int[] nodesType,

int[][] nodesConnectionMatrix, int[][] signalMatrix) {

for (int i = 0; i < nodesConnectionMatrix[from].length; i++) {

if (nodesConnectionMatrix[from][i] == 1) {

int[] stepNewXNumbers;

if (stepXNumbers != null) {

stepNewXNumbers = new int[stepXNumbers.length + 1];

for (int j = 0; j < stepXNumbers.length; j++) {

stepNewXNumbers[j] = stepXNumbers[j];

}

} else {

stepNewXNumbers = new int[1];

}

stepNewXNumbers[stepNewXNumbers.length - 1] = signalMatrix[from][0];

boolean[] stepNewXValues;

if (stepXValues != null) {

stepNewXValues = new boolean[stepXValues.length + 1];

for (int j = 0; j < stepXValues.length; j++) {

stepNewXValues[j] = stepXValues[j];

}

} else {

stepNewXValues = new boolean[1];

}

stepNewXValues[stepNewXValues.length - 1] = true;

if (nodesType[i] == 2) {

stepToOperatorNode(start, i, stepNewXNumbers, stepNewXValues, nodesType, nodesConnectionMatrix, signalMatrix);

} else {

int[][] newXNumbers;

if (xNumbers != null) {

newXNumbers = new int[xNumbers.length + 1][];

for (int j = 0; j < xNumbers.length; j++) {

newXNumbers[j] = xNumbers[j];

}

} else {

newXNumbers = new int[1][];

}

newXNumbers[newXNumbers.length - 1] = stepNewXNumbers;

xNumbers = newXNumbers;

boolean[][] newXValues;

if (xValues != null) {

newXValues = new boolean[xValues.length + 1][];

for (int j = 0; j < xValues.length; j++) {

newXValues[j] = xValues[j];

}

} else {

newXValues = new boolean[1][];

}

newXValues[newXValues.length - 1] = stepNewXValues;

xValues = newXValues;

connectionMatrix[start][getStateNumber(nodesType, i)] = newXNumbers.length - 1;

}

} else {

if (nodesConnectionMatrix[from][i] == 2) {

int[] stepNewXNumbers;

if (stepXNumbers != null) {

stepNewXNumbers = new int[stepXNumbers.length + 1];

for (int j = 0; j < stepXNumbers.length; j++) {

stepNewXNumbers[j] = stepXNumbers[j];

}

} else {

stepNewXNumbers = new int[1];

}

stepNewXNumbers[stepNewXNumbers.length - 1] = signalMatrix[from][0];

boolean[] stepNewXValues;

if (stepXValues != null) {

stepNewXValues = new boolean[stepXValues.length + 1];

for (int j = 0; j < stepXValues.length; j++) {

stepNewXValues[j] = stepXValues[j];

}

} else {

stepNewXValues = new boolean[1];

}

stepNewXValues[stepNewXValues.length - 1] = false;

if (nodesType[i] == 2) {

stepToOperatorNode(start, i, stepNewXNumbers, stepNewXValues, nodesType, nodesConnectionMatrix, signalMatrix);

} else {

int[][] newXNumbers = new int[xNumbers.length + 1][];

for (int j = 0; j < xNumbers.length; j++) {

newXNumbers[j] = xNumbers[j];

}

newXNumbers[newXNumbers.length - 1] = stepNewXNumbers;

xNumbers = newXNumbers;

boolean[][] newXValues = new boolean[xValues.length + 1][];

for (int j = 0; j < xValues.length; j++) {

newXValues[j] = xValues[j];

}

newXValues[newXValues.length - 1] = stepNewXValues;

xValues = newXValues;

connectionMatrix[start][getStateNumber(nodesType, i)] = newXNumbers.length - 1;

}

}

}

}

}

private int getStateNumber(int[] nodesType, int n) {

int z1 = -1;

int j = 0;

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

if (nodesType[i] == 1) {

j++;

}

else {

if (((nodesType[i] == 0) || (nodesType[i] == 3)) && (z1 == -1)) {

z1 = j;

j++;

}

}

}

if ((nodesType[n] == 0) || (nodesType[n] == 3)) {

if (z1 == -1) {

return n;

}

else {

return z1;

}

}

else {

return j;

}

}

public String[] getStateNames() {

return stateNames;

}

public int[][] getyNumbers() {

return yNumbers;

}

public int[][] getConnectionMatrix() {

return connectionMatrix;

}

public int[][] getxNumbers() {

return xNumbers;

}

public boolean[][] getxValues() {

return xValues;

}

}

package automat.moore;

import java.awt.\*;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 16.10.2010

\* Time: 14:15:35

\* To change this template use File | Settings | File Templates.

\*/

public class GraphModel {

private static final Color DEFAULT\_BACKGROUND\_COLOR = Color.WHITE;

private static final Color DEFAULT\_STATE\_COLOR = Color.CYAN;

private static final Color DEFAULT\_LINES\_COLOR = Color.BLACK;

private static final Color DEFAULT\_TEXT\_COLOR = Color.BLACK;

private static final Font DEFAULT\_FONT = new Font("Sans Serif", Font.PLAIN, 14);

private static final int DEFAULT\_STATE\_RADIUS = 50;

private static final int DEFAULT\_ARROW\_WIDTH = 20;

private static final double DEFAULT\_ARROW\_ANGLE = 0.5;

private static final int DEFAULT\_DISTANCE = 70;

private int stateDiametr;

private int arrowWidth;

private double arrowAngle;

private int distance;

private Font font;

private MooreAutomat automat;

private Color stateColor;

private Color backgroundColor;

private Color linesColor;

private Color textColor = DEFAULT\_TEXT\_COLOR;

public GraphModel(MooreAutomat automat) {

this.automat = automat;

stateColor = DEFAULT\_STATE\_COLOR;

backgroundColor = DEFAULT\_BACKGROUND\_COLOR;

linesColor = DEFAULT\_LINES\_COLOR;

stateDiametr = DEFAULT\_STATE\_RADIUS;

textColor = DEFAULT\_TEXT\_COLOR;

font = DEFAULT\_FONT;

arrowWidth = DEFAULT\_ARROW\_WIDTH;

arrowAngle = DEFAULT\_ARROW\_ANGLE;

distance = DEFAULT\_DISTANCE;

}

public MooreAutomat getAutomat() {

return automat;

}

public void setAutomat(MooreAutomat automat) {

this.automat = automat;

}

public Font getFont() {

return font;

}

public void setFont(Font font) {

this.font = font;

}

public Color getStateColor() {

return stateColor;

}

public void setStateColor(Color stateColor) {

this.stateColor = stateColor;

}

public Color getBackgroundColor() {

return backgroundColor;

}

public void setBackgroundColor(Color backgroundColor) {

this.backgroundColor = backgroundColor;

}

public Color getLinesColor() {

return linesColor;

}

public void setLinesColor(Color linesColor) {

this.linesColor = linesColor;

}

public int getStateDiametr() {

return stateDiametr;

}

public void setStateDiametr(int stateDiametr) {

this.stateDiametr = stateDiametr;

}

public Color getTextColor() {

return textColor;

}

public void setTextColor(Color textColor) {

this.textColor = textColor;

}

public int getArrowWidth() {

return arrowWidth;

}

public void setArrowWidth(int arrowWidth) {

this.arrowWidth = arrowWidth;

}

public double getArrowAngle() {

return arrowAngle;

}

public void setArrowAngle(double arrowAngle) {

this.arrowAngle = arrowAngle;

}

public int getDistance() {

return distance;

}

public void setDistance(int distance) {

this.distance = distance;

}

}

package automat.moore;

import javax.swing.\*;

import java.awt.\*;

import java.awt.font.FontRenderContext;

import java.awt.geom.Rectangle2D;

import java.util.ArrayList;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 16.10.2010

\* Time: 14:15:17

\* To change this template use File | Settings | File Templates.

\*/

public class GraphPanel extends JPanel {

private GraphModel model;

public GraphPanel(GraphModel model) {

super();

this.model = model;

setBackground(this.model.getBackgroundColor());

}

public GraphModel getModel() {

return model;

}

public void setModel(GraphModel model) {

this.model = model;

}

private void drawState(Graphics2D g2, int x, int y, int radius, String name, int[] yNumbers) {

g2.setColor(model.getStateColor());

g2.fillOval(x, y, radius, radius);

g2.setColor(model.getLinesColor());

g2.drawLine(x, y + (radius / 2), x + radius, y + (radius / 2));

g2.setColor(model.getTextColor());

g2.setFont(model.getFont());

FontRenderContext context = g2.getFontRenderContext();

Rectangle2D bounds = g2.getFont().getStringBounds(name, context);

g2.drawString(name, (int) (x + (radius - bounds.getWidth()) / 2) ,

(int) (y + (radius / 2 + bounds.getHeight()) / 2));

StringBuilder builder = new StringBuilder();

if (yNumbers != null) {

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

builder.append("Y");

builder.append(String.valueOf(yNumbers[i]));

}

} else {

builder.append("-");

}

bounds = g2.getFont().getStringBounds(builder.toString(), context);

g2.drawString(builder.toString(), (int) (x + (radius - bounds.getWidth()) / 2),

(int) (y + radius / 2 + (radius / 2 + bounds.getHeight()) / 2));

}

private void drawArrowLine(Graphics2D g2, int[] x, int[] y, int[] xNumbers, boolean[] xValues) {

g2.setColor(model.getLinesColor());

g2.drawPolyline(x, y, x.length);

double temp = Math.atan2(x[x.length - 2] - x[x.length - 1], y[y.length - 2] - y[y.length - 1]);

g2.drawLine(x[x.length - 1], y[y.length - 1],

(int) (x[x.length - 1] + model.getArrowWidth() \* Math.sin(temp + model.getArrowAngle())),

(int) (y[y.length - 1] + model.getArrowWidth() \* Math.cos(temp + model.getArrowAngle())));

g2.drawLine(x[x.length - 1], y[y.length - 1],

(int) (x[x.length - 1] + model.getArrowWidth() \* Math.sin(temp - model.getArrowAngle())),

(int) (y[y.length - 1] + model.getArrowWidth() \* Math.cos(temp - model.getArrowAngle())));

g2.setColor(model.getTextColor());;

g2.setFont(model.getFont());

StringBuilder builder = new StringBuilder();

if (xNumbers != null) {

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

if (!xValues[i]) {

builder.append("not");

}

builder.append("X");

builder.append(String.valueOf(xNumbers[i]));

}

} else {

builder.append("-");

}

FontRenderContext context = g2.getFontRenderContext();

Rectangle2D bounds = g2.getFont().getStringBounds(builder.toString(), context);

int textX = (int) (x[0] + (x[1] - x[0] - bounds.getX()) / 2);

int textY = (int) (y[0] + (y[1] - y[0] - bounds.getY()) / 2 + 5);

g2.drawString(builder.toString(), textX, textY);

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2 = (Graphics2D) g;

int centerX = (getWidth() - model.getStateDiametr()) / 2;

int centerY = (getHeight() - model.getStateDiametr()) / 2;

int currentX;

int currentY;

MooreAutomat automat = model.getAutomat();

double temp = 2 \* Math.PI / automat.getStateNames().length;

double angle = 0;

int radius;

if (getWidth() > getHeight()) {

radius = (getHeight() - 2 \* model.getDistance()) / 2;

}

else {

radius = (getWidth() - 2 \* model.getDistance()) / 2;

}

ArrayList<Point> stateConnectors = new ArrayList<Point>();

ArrayList<Point> stateSelfConnectors = new ArrayList<Point>();

for (int i = 0; i < model.getAutomat().getStateNames().length; i++) {

currentX = (int) (centerX + radius \* Math.sin(angle));

currentY = (int) (centerY + radius \* Math.cos(angle));

int stateCenterX = currentX + model.getStateDiametr() / 2;

int stateCenterY = currentY + model.getStateDiametr() / 2;

int stateConnectorX = (int) (stateCenterX + model.getStateDiametr() / 2 \* Math.sin(angle + Math.PI));

int stateConnectorY = (int) (stateCenterY + model.getStateDiametr() / 2 \* Math.cos(angle + Math.PI));

stateConnectors.add(new Point(stateConnectorX, stateConnectorY));

//int stateSelfConnectorX = (int) (stateCenterX + model.getStateDiametr() / 2 \* Math.sin(angle));

//int stateSelfConnectorY = (int) (stateCenterY + model.getStateDiametr() / 2 \* Math.cos(angle));

// Temporary!!!

// |

// v

int stateSelfConnectorX = stateCenterX;

int stateSelfConnectorY;

if (stateCenterY < getHeight() / 2) {

stateSelfConnectorY = currentY;

}

else {

stateSelfConnectorY = currentY + model.getStateDiametr();

}

// ^

// |

// Temporary!!!

stateSelfConnectors.add(new Point(stateSelfConnectorX, stateSelfConnectorY));

drawState(g2, currentX, currentY, model.getStateDiametr(), automat.getStateNames()[i], automat.getyNumbers()[i]);

angle += temp;

}

int[][] connectionMatrix = automat.getConnectionMatrix();

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

for (int j = 0; j < connectionMatrix[i].length; j++) {

if (connectionMatrix[i][j] > -1) {

if (i != j) {

int[] lineX = new int[2];

int[] lineY = new int[2];

lineX[0] = (int) stateConnectors.get(i).getX();

lineX[1] = (int) stateConnectors.get(j).getX();

lineY[0] = (int) stateConnectors.get(i).getY();

lineY[1] = (int) stateConnectors.get(j).getY();

drawArrowLine(g2, lineX, lineY, automat.getxNumbers()[connectionMatrix[i][j]],

automat.getxValues()[connectionMatrix[i][j]]);

}

else {

int[] lineX;

int[] lineY;

// Temporary!!!

// |

// v

if ((stateConnectors.get(i).getX() <= (getWidth() / 2)) && (stateConnectors.get(i).getY() < (getHeight() / 2))) {

lineX = new int[6];

lineY = new int[6];

lineX[0] = (int) stateConnectors.get(i).getX();

lineY[0] = (int) stateConnectors.get(i).getY();

lineX[1] = (int) stateConnectors.get(i).getX();

lineY[1] = (int) stateConnectors.get(i).getY() + model.getStateDiametr() \* 2 / 3;

lineX[2] = (int) stateConnectors.get(i).getX() - 3 \* model.getStateDiametr() / 2;

lineY[2] = lineY[1];

lineX[3] = lineX[2];

lineY[3] = (int) stateConnectors.get(i).getY() - 3 \* model.getStateDiametr() / 2;

lineX[4] = (int) stateSelfConnectors.get(i).getX();

lineY[4] = lineY[3];

lineX[5] = (int) stateSelfConnectors.get(i).getX();

lineY[5] = (int) stateSelfConnectors.get(i).getY();

}

else {

if ((stateConnectors.get(i).getX() <= (getWidth() / 2)) && (stateConnectors.get(i).getY() >= (getHeight() / 2))) {

lineX = new int[6];

lineY = new int[6];

lineX[0] = (int) stateConnectors.get(i).getX();

lineY[0] = (int) stateConnectors.get(i).getY();

lineX[1] = (int) stateConnectors.get(i).getX();

lineY[1] = (int) stateConnectors.get(i).getY() - model.getStateDiametr() \* 2 / 3;

lineX[2] = (int) stateConnectors.get(i).getX() - 3 \* model.getStateDiametr() / 2;

lineY[2] = lineY[1];

lineX[3] = lineX[2];

lineY[3] = (int) stateConnectors.get(i).getY() + 3 \* model.getStateDiametr() / 2;

lineX[4] = (int) stateSelfConnectors.get(i).getX();

lineY[4] = lineY[3];

lineX[5] = (int) stateSelfConnectors.get(i).getX();

lineY[5] = (int) stateSelfConnectors.get(i).getY();

}

else {

if ((stateConnectors.get(i).getX() > (getWidth() / 2)) && (stateConnectors.get(i).getY() >= (getHeight() / 2))) {

lineX = new int[6];

lineY = new int[6];

lineX[0] = (int) stateConnectors.get(i).getX();

lineY[0] = (int) stateConnectors.get(i).getY();

lineX[1] = (int) stateConnectors.get(i).getX();

lineY[1] = (int) stateConnectors.get(i).getY() - model.getStateDiametr() \* 2 / 3;

lineX[2] = (int) stateConnectors.get(i).getX() + 3 \* model.getStateDiametr() / 2;

lineY[2] = lineY[1];

lineX[3] = lineX[2];

lineY[3] = (int) stateConnectors.get(i).getY() + 3 \* model.getStateDiametr() / 2;

lineX[4] = (int) stateSelfConnectors.get(i).getX();

lineY[4] = lineY[3];

lineX[5] = (int) stateSelfConnectors.get(i).getX();

lineY[5] = (int) stateSelfConnectors.get(i).getY();

}

else {

lineX = new int[6];

lineY = new int[6];

lineX[0] = (int) stateConnectors.get(i).getX();

lineY[0] = (int) stateConnectors.get(i).getY();

lineX[1] = (int) stateConnectors.get(i).getX();

lineY[1] = (int) stateConnectors.get(i).getY() + model.getStateDiametr() \* 2 / 3;

lineX[2] = (int) stateConnectors.get(i).getX() + 3 \* model.getStateDiametr() / 2;

lineY[2] = lineY[1];

lineX[3] = lineX[2];

lineY[3] = (int) stateConnectors.get(i).getY() - 3 \* model.getStateDiametr() / 2;

lineX[4] = (int) stateSelfConnectors.get(i).getX();

lineY[4] = lineY[3];

lineX[5] = (int) stateSelfConnectors.get(i).getX();

lineY[5] = (int) stateSelfConnectors.get(i).getY();

}

}

}

// ^

// |

// Temporary!!!

drawArrowLine(g2, lineX, lineY, automat.getxNumbers()[connectionMatrix[i][j]],

automat.getxValues()[connectionMatrix[i][j]]);

}

}

}

}

}

}

package automat.moore;

import javax.swing.filechooser.FileFilter;

import java.io.File;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 20.10.2010

\* Time: 2:13:35

\* To change this template use File | Settings | File Templates.

\*/

public class GraphFileFilter extends FileFilter {

public static String GRAPH\_EXTENSION = ".graph";

private static String GRAPH\_DESCRIPTION = "Graph File";

public boolean accept(File pathname) {

return (pathname.getName().toLowerCase().endsWith(GRAPH\_EXTENSION) || pathname.isDirectory());

}

public String getDescription() {

return GRAPH\_DESCRIPTION;

}

}

package face;

import automat.moore.GraphFileFilter;

import automat.moore.GraphModel;

import automat.moore.GraphPanel;

import automat.moore.MooreAutomat;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.io.File;

import java.io.IOException;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 20.10.2010

\* Time: 1:17:35

\* To change this template use File | Settings | File Templates.

\*/

class BuildFrame extends JDialog {

private MainFrame mainFrame;

private JTabbedPane tabbedPane;

private GraphPanel graphPanel;

public BuildFrame(MainFrame frame, Rectangle bounds, MooreAutomat automat) {

super(frame);

mainFrame = frame;

setBounds(bounds);

setMinimumSize(bounds.getSize());

setResizable(true);

setModal(true);

setTitle("Building");

tabbedPane = new JTabbedPane();

add(tabbedPane);

JPanel mooreGraphPanel = new JPanel();

mooreGraphPanel.setLayout(new BorderLayout());

graphPanel = new GraphPanel(new GraphModel(automat));

JPanel mooreGraphButtonsPanel = new JPanel();

JButton saveGraphButton = new JButton(new SaveGraphAction(this));

saveGraphButton.setText("Save Graph");

JButton closeButton = new JButton(new AbstractAction() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

closeButton.setText("Close");

mooreGraphButtonsPanel.add(saveGraphButton);

mooreGraphButtonsPanel.add(closeButton);

mooreGraphPanel.add(mooreGraphButtonsPanel, BorderLayout.SOUTH);

mooreGraphPanel.add(graphPanel);

tabbedPane.addTab("Graph Of Moore Automat", mooreGraphPanel);

}

private class SaveGraphAction extends AbstractAction {

private BuildFrame frame;

public SaveGraphAction(BuildFrame frame) {

this.frame = frame;

}

public void actionPerformed(ActionEvent e) {

JFileChooser chooser = mainFrame.getChooser();

chooser.resetChoosableFileFilters();

chooser.addChoosableFileFilter(new GraphFileFilter());

int result = chooser.showSaveDialog(frame);

if (result == JFileChooser.APPROVE\_OPTION) {

if (!chooser.getSelectedFile().getName().endsWith(GraphFileFilter.GRAPH\_EXTENSION)) {

chooser.setSelectedFile(new File(chooser.getSelectedFile().getAbsolutePath() + GraphFileFilter.GRAPH\_EXTENSION));

}

try {

MooreAutomat.writeToFile(chooser.getSelectedFile(), graphPanel.getModel().getAutomat());

} catch (IOException e1) {

JOptionPane.showMessageDialog(frame, "Error! Can't create file.",

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

}

}

**Висновки**

В результаті виконання даної лабораторної роботи я здобув навички з автоматизації процедури розмітки алгоритму методом Мура. Я реалізував побудову графу за допомогою видалення логічних вершин з матриці зв’язків та запису в неї умов переходів, які я визначив за допомогою рекурсивного алгоритму пошуку наступної операторної або кінцевої вершини від початкової та кожної операторної вершини. Також я здобув навички з візуалізації графів та роботи з бінарними файлами. Програма була написана на мові Java. Для візуалізації графу переходів був використаний пакет javax.swing.\*.