Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №4

з курсу «Автоматизація проектування комп'ютерних систем»

Виконав

студент групи ІО-73

Захожий Ігор

Номер залікової книжки: 7308

Київ-2010

Тема роботи

Автоматизація кодування графу переходів.

Мета роботи

Здобуття навичок з автоматизації процедури сумісного кодування графу переходів.

Завдання

1. Розробити алгоритм сумісного кодування графу переходів з попередньої роботи – будь-які вузли, що мають зв’язок повинні мати коди, які відрізняються лише у одному двійковому розряді. Блок-схему та опис розробленого алгоритму надати в протоколі роботи.
2. Реалізувати розроблений алгоритм. Кодування відобразити на графічному представленні графу переходів.
3. Модифікувати формат зберігання графу переходів таким чином, щоб він містив інформацію про коди вузлів. Реалізувати можливість збереження/відновлення закодованого графа переходів.

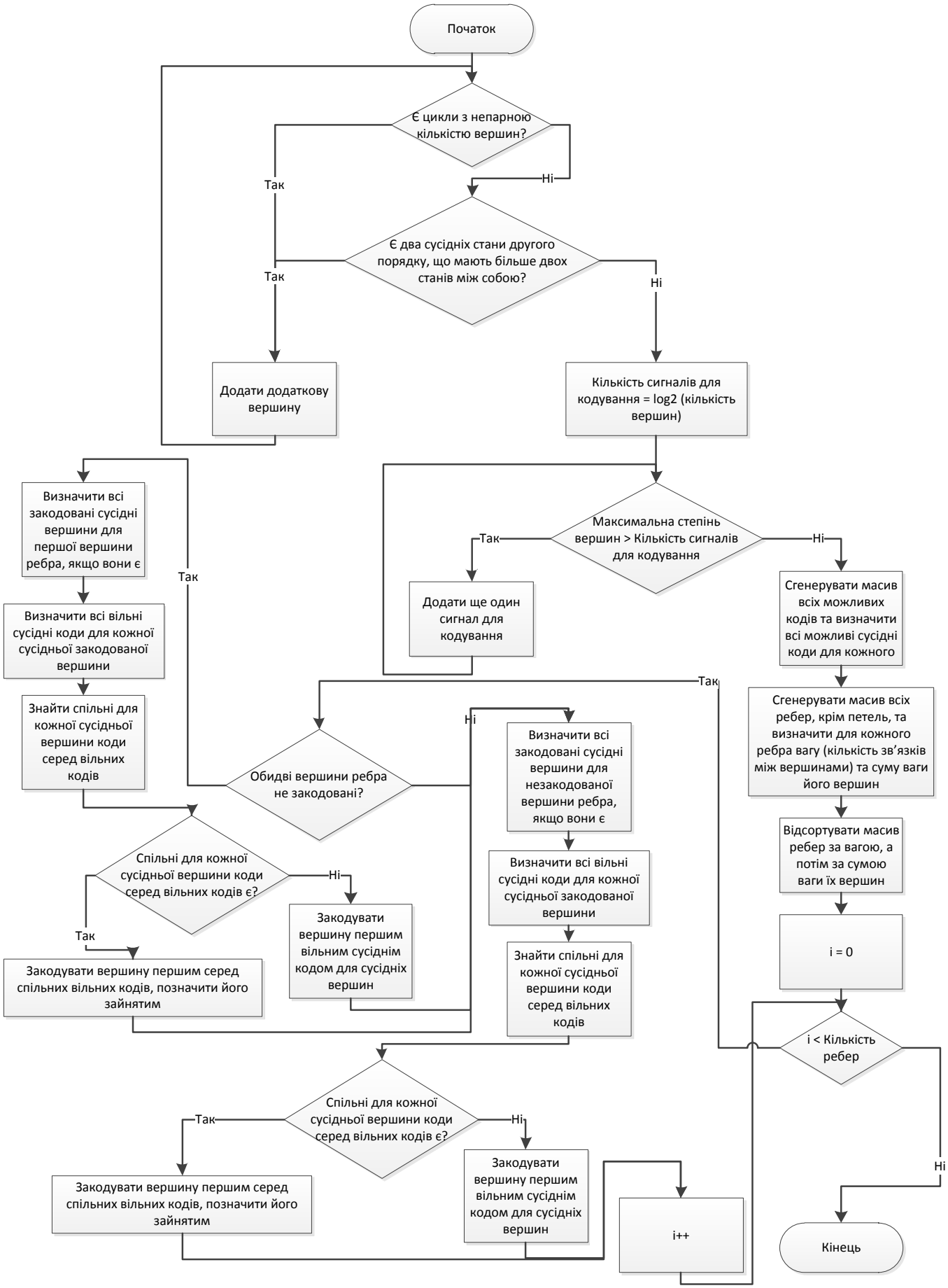
Опис алгоритму

Сумісне кодування можливе лише при виконанні таких умов:

1) У графі автомата не має бути циклів з непарною кількістю вершин;
2) Два сусідніх стани другого порядку не повинні мати більше двох станів, що лежать між ними.

Якщо граф не задовольняє дані умови, ми маємо додавати до нього додаткові вершини поки ці умови не будуть виконуватися. Після цього граф можна закодувати евристичним алгоритмом кодування. Його блок-схема зображена нижче.

Блок-схема алгоритму



Опис програми

Для виконання сумісного кодування графу переходів з попередньої лабораторної роботи необхідно натиснути кнопку «Code Graph» (рис. 1). Після чого в новій вкладці буде відображений закодований граф переходів. Для графу, зображеного на рисунку 1, закодований граф переходів показаний на рисунку 2.

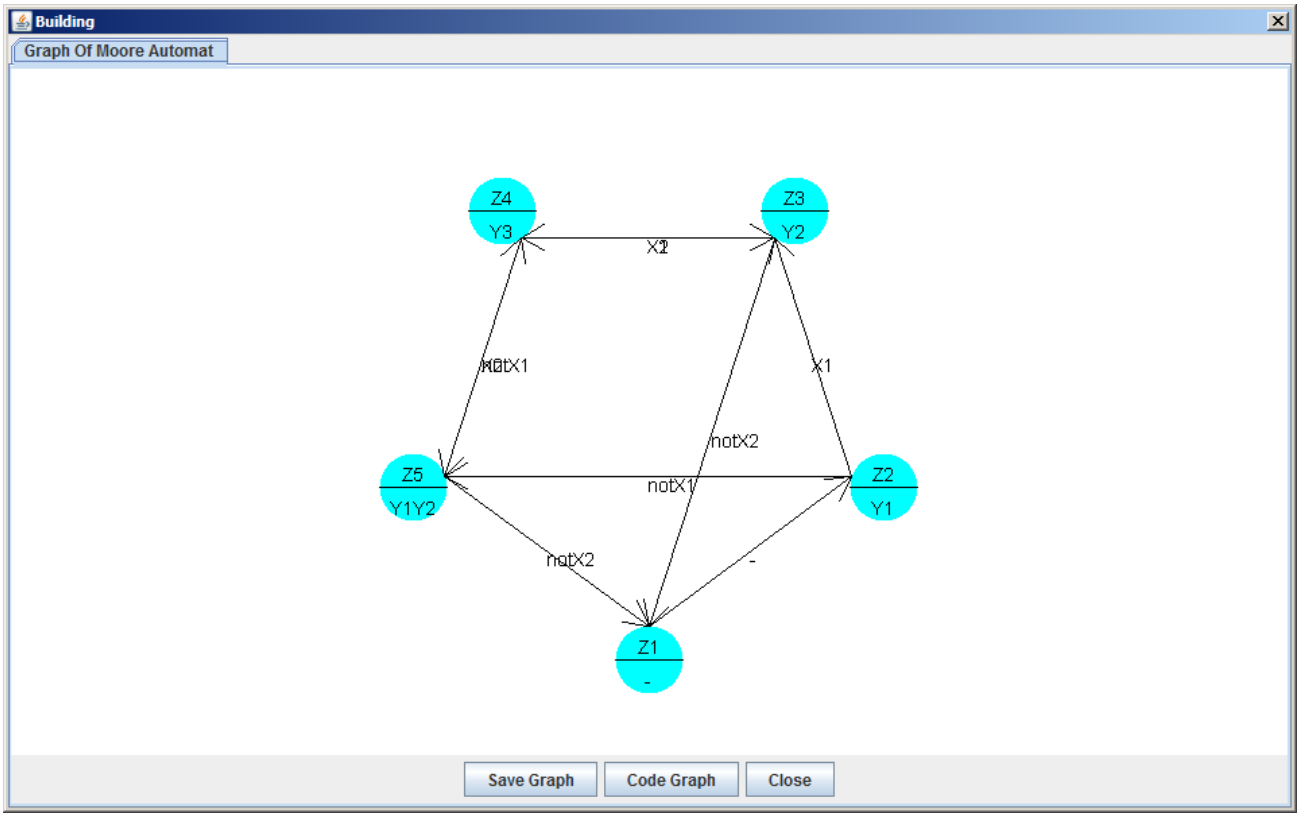


Рисунок 1

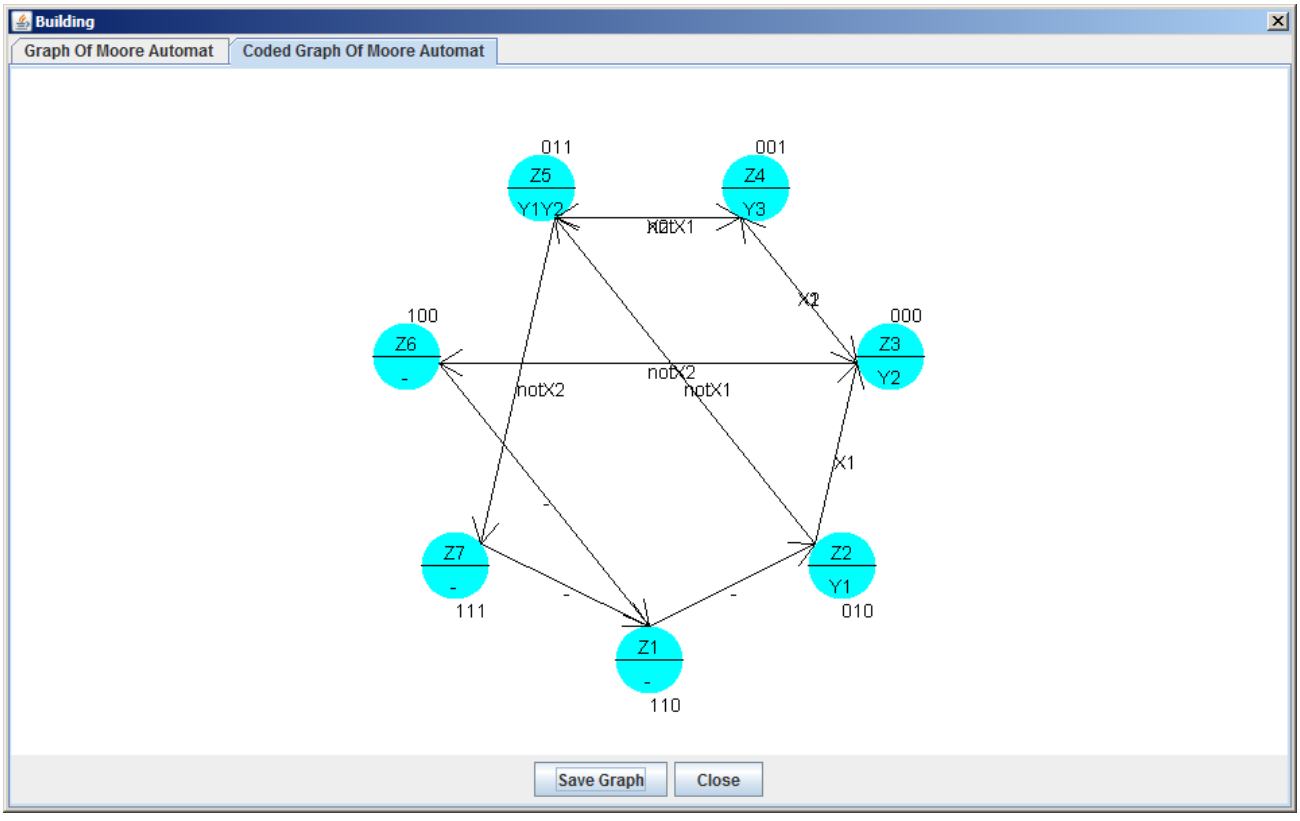


Рисунок 2

Для збереження закодованого графу переходів необхідно натиснути кнопку «Save Graph» у вкладці «Coded Graph Of Moore Automat» та ввести ім’я файлу в діалоговому вікні.

Лістинг програми

```
package automat.moore;

import java.io.*;
import java.util.LinkedList;
import java.util.ListIterator;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 25.10.2010
 * Time: 1:58:17
 * To change this template use File | Settings | File Templates.
 */
public class CodedMooreAutomat extends MooreAutomat {

    private String[] stateCodes;

    public CodedMooreAutomat(MooreAutomat automat) {
        String[] stateNames = automat.getStateNames();
        this.stateNames = new String[stateNames.length];
        for (int i = 0; i < stateNames.length; i++) {
            this.stateNames[i] = stateNames[i];
        }
        int[][] yNumbers = automat.getyNumbers();
        this.yNumbers = new int[yNumbers.length][];
        for (int i = 0; i < yNumbers.length; i++) {
            if (yNumbers[i] != null) {
                this.yNumbers[i] = new int[yNumbers[i].length];
                for (int j = 0; j < yNumbers[i].length; j++) {
                    this.yNumbers[i][j] = yNumbers[i][j];
                }
            }
            else {
                this.yNumbers[i] = null;
            }
        }
        int[][] connectionMatrix = automat.getConnectionMatrix();
        this.connectionMatrix = new int[connectionMatrix.length][];
        for (int i = 0; i < connectionMatrix.length; i++) {
            this.connectionMatrix[i] = new int[connectionMatrix[i].length];
            for (int j = 0; j < connectionMatrix[i].length; j++) {
                this.connectionMatrix[i][j] = connectionMatrix[i][j];
            }
        }
        int[][] xNumbers = automat.getxNumbers();
        this.xNumbers = new int[xNumbers.length][];
        for (int i = 0; i < xNumbers.length; i++) {
            if (xNumbers[i] != null) {
                this.xNumbers[i] = new int[xNumbers[i].length];
                for (int j = 0; j < xNumbers[i].length; j++) {
                    this.xNumbers[i][j] = xNumbers[i][j];
                }
            }
            else {
                this.xNumbers[i] = null;
            }
        }
        boolean[][] xValues = automat.getxValues();
        this.xValues = new boolean[xValues.length][];
        for (int i = 0; i < xValues.length; i++) {
            if (xValues[i] != null) {
                this.xValues[i] = new boolean[xValues[i].length];
                for (int j = 0; j < xValues[i].length; j++) {
                    this.xValues[i][j] = xValues[i][j];
                }
            }
            else {
                this.xValues[i] = null;
            }
        }
        codeAutomat();
    }

    private void codeAutomat() {
        while ((!checkForCycles()) || (!checkForSecondOrder())) {}
        double temp = Math.log(stateNames.length) / Math.log(2);
        int signalCount = (int) temp;
        if (temp > signalCount) {
            signalCount += 1;
        }
        while (getMaxStatePower() > signalCount) {
            signalCount++;
        }
        int combinationCount = (int) Math.pow(2, signalCount);
        boolean[] isBusy = new boolean[combinationCount];
        String[] combinations = new String[combinationCount];
        for (int i = 0; i < combinationCount; i++) {
            isBusy[i] = false;
            combinations[i] = intToBinary(i, signalCount);
        }
        int[] codes = new int[stateNames.length];
        for (int i = 0; i < codes.length; i++) {
            codes[i] = -1;
        }
        int[][] neighboursCombination = new int[combinations.length][];
        for (int i = 0; i < neighboursCombination.length; i++) {
            neighboursCombination[i] = new int[signalCount];
            int nCount = 0;
            for (int j = 0; j < combinations.length; j++) {
                if (i != j) {
                    int differentCount = 0;
                    for (int k = 0; k < signalCount; k++) {
                        if (combinations[i].charAt(k) != combinations[j].charAt(k)) {
                            differentCount++;
                        }
                    }
```

```
            if (differentCount == 1) {
                neighboursCombination[i][nCount++] = j;
            }
        }
    }
}
stateCodes = new String[stateNames.length];
int edgesCount = 0;
for (int i = 0; i < connectionMatrix.length; i++) {
    for (int j = 0; j < connectionMatrix[i].length; j++) {
        if ((connectionMatrix[i][j] >= 0) && (i != j)) {
            edgesCount++;
        }
    }
}
int[][] edges = new int[edgesCount][];
for (int i = 0; i < edges.length; i++) {
    edges[i] = new int[4];
}
edgesCount = 0;
for (int i = 0; i < connectionMatrix.length; i++) {
    for (int j = 0; j < connectionMatrix[i].length; j++) {
        if ((connectionMatrix[i][j] >= 0) && (i != j)) {
            boolean isAlready = false;
            for (int k = 0; k < edgesCount; k++) {
                if (((i == edges[k][0]) && (j == edges[k][1])) || ((i == edges[k][1]) && (j == edges[k][0]))) {
                    edges[k][2]++;
                    isAlready = true;
                }
            }
            if (!isAlready) {
                edges[edgesCount][0] = i;
                edges[edgesCount][1] = j;
                edges[edgesCount][2] = 1;
                edgesCount++;
            }
        }
    }
}
for (int i = 0; i < edgesCount; i++) {
    int max = i;
    for (int j = i + 1; j < edgesCount; j++) {
        if (edges[j][2] > edges[max][2]) {
            max = j;
        }
    }
    if (max != i) {
        int[] swapTemp = edges[i];
        edges[i] = edges[max];
        edges[max] = swapTemp;
    }
}
int[] stateWeights = new int[stateNames.length];
for (int i = 0; i < stateWeights.length; i++) {
    stateWeights[i] = 0;
    for (int j = 0; j < stateNames.length; j++) {
        if (i != j) {
            if ((connectionMatrix[i][j] >= 0) || (connectionMatrix[j][i] >= 0)) {
                stateWeights[i]++;
            }
        }
    }
}
for (int i = 0; i < edgesCount; i++) {
    edges[i][3] = stateWeights[edges[i][0]] + stateWeights[edges[i][1]];
}
int z = 0;
int previousEdgeWeight = -1;
while (z < edgesCount) {
    if (previousEdgeWeight == edges[z][2]) {
        int from = z - 1;
        while ((z < edgesCount) && (previousEdgeWeight == edges[z][2])) {
            z++;
        }
        for (int i = from; i < z; i++) {
            int max = i;
            for (int j = i + 1; j < z; j++) {
                if (edges[j][3] > edges[max][3]) {
                    max = j;
                }
            }
            int[] swapTemp = edges[max];
            edges[max] = edges[i];
            edges[i] = swapTemp;
        }

    }
    else {
        previousEdgeWeight = edges[z][2];
        z++;
    }
}
for (int i = 0; i < edgesCount; i++) {
    if ((codes[edges[i][0]] == -1) && (codes[edges[i][1]] == -1)) {
        boolean found = false;
        int j = 0;
        while (isBusy[j]) {
            j++;
        }
        codes[edges[i][0]] = j;
        isBusy[j] = true;
        stateCodes[edges[i][0]] = combinations[j];
        int k = 0;
        while (isBusy[neighboursCombination[j][k]]) {
            k++;
        }
        codes[edges[i][1]] = neighboursCombination[j][k];
        isBusy[neighboursCombination[j][k]] = true;
        stateCodes[edges[i][1]] = combinations[neighboursCombination[j][k]];
```

```
        }
        else {
            if (codes[edges[i][0]] == -1) {
                LinkedList<Integer> neighbourCodedStates = new LinkedList<Integer>();
                neighbourCodedStates.add(edges[i][1]);
                for (int j = 0; j < edgesCount; j++) {
                    if (i != j) {
                        if (edges[j][0] == edges[i][0]) {
                            if (codes[edges[j][1]] != -1) {
                                neighbourCodedStates.add(edges[j][1]);
                            }
                        } else {
                            if (edges[j][1] == edges[i][0]) {
                                if (codes[edges[j][0]] != -1) {
                                    neighbourCodedStates.add(edges[j][0]);
                                }
                            }
                        }
                    }
                }
                ListIterator<Integer> listIterator = neighbourCodedStates.listIterator();
                LinkedList<Integer> freeNeighboursCombinations = new LinkedList<Integer>();
                while (listIterator.hasNext()) {
                    int neighbourCodedState = listIterator.next();
                    for (int j = 0; j < neighboursCombination[codes[neighbourCodedState]].length; j++) {
                        if (!isBusy[neighboursCombination[codes[neighbourCodedState]][j]]) {
                            freeNeighboursCombinations.add(neighboursCombination[codes[neighbourCodedState]][j]);
                        }
                    }
                }
                boolean found = false;
                int j = 0;
                while ((!found) && (j < freeNeighboursCombinations.size())) {
                    int tempCount = 1;
                    for (int k = j + 1; k < freeNeighboursCombinations.size(); k++) {
                        if (freeNeighboursCombinations.get(j) == freeNeighboursCombinations.get(k)) {
                            tempCount++;
                        }
                    }
                    if (tempCount == neighbourCodedStates.size()) {
                        found = true;
                    }
                    j++;
                }
                codes[edges[i][0]] = freeNeighboursCombinations.get(--j);
                isBusy[freeNeighboursCombinations.get(j)] = true;
                stateCodes[edges[i][0]] = combinations[freeNeighboursCombinations.get(j)];
            }
            else {
                if (codes[edges[i][1]] == -1) {
                    LinkedList<Integer> neighbourCodedStates = new LinkedList<Integer>();
                    neighbourCodedStates.add(edges[i][0]);
                    for (int j = 0; j < edgesCount; j++) {
                        if (i != j) {
                            if (edges[j][0] == edges[i][1]) {
                                if (codes[edges[j][1]] != -1) {
                                    neighbourCodedStates.add(edges[j][1]);
                                }
                            } else {
                                if (edges[j][1] == edges[i][1]) {
                                    if (codes[edges[j][0]] != -1) {
                                        neighbourCodedStates.add(edges[j][0]);
                                    }
                                }
                            }
                        }
                    }
                    ListIterator<Integer> listIterator = neighbourCodedStates.listIterator();
                    LinkedList<Integer> freeNeighboursCombinations = new LinkedList<Integer>();
                    while (listIterator.hasNext()) {
                        int neighbourCodedState = listIterator.next();
                        for (int j = 0; j < neighboursCombination[codes[neighbourCodedState]].length; j++) {
                            if (!isBusy[neighboursCombination[codes[neighbourCodedState]][j]]) {
                                freeNeighboursCombinations.add(neighboursCombination[codes[neighbourCodedState]][j]);
                            }
                        }
                    }
                    boolean found = false;
                    int j = 0;
                    while ((!found) && (j < freeNeighboursCombinations.size())) {
                        int tempCount = 1;
                        for (int k = j + 1; k < freeNeighboursCombinations.size(); k++) {
                            if (freeNeighboursCombinations.get(j) == freeNeighboursCombinations.get(k)) {
                                tempCount++;
                            }
                        }
                        if (tempCount == neighbourCodedStates.size()) {
                            found = true;
                        }
                        j++;
                    }
                    codes[edges[i][1]] = freeNeighboursCombinations.get(--j);
                    isBusy[freeNeighboursCombinations.get(j)] = true;
                    stateCodes[edges[i][1]] = combinations[freeNeighboursCombinations.get(j)];
                }
            }
        }
    }
}

private boolean checkForCycles() {
    boolean result = true;
    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[] cycle = new int[2];
                    cycle[0] = i;
                    cycle[1] = j;
```

```
                    boolean flag = stepForCycles(cycle, j);
                    if (!flag) {
                        result = false;
                    }
                }
            }
        }
    }
    return result;
}

private boolean checkForSecondOrder() {
    boolean result = true;
    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) {
                    for (int k = 0; k < connectionMatrix[j].length; k++) {
                        if (connectionMatrix[j][k] > -1) {
                            if (j != k) {
                                if (connectionMatrix[i][k] > -1) {
                                    result = false;
                                    addAdditionalState(i, k);
                                }
                                else {
                                    if (connectionMatrix[k][i] > -1) {
                                        result = false;
                                        addAdditionalState(k, i);
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return result;
}

private boolean stepForCycles(int[] cycle, int current) {
    boolean result = true;
    for (int i = 0; i < connectionMatrix[current].length; i++) {
        if (connectionMatrix[current][i] > -1) {
            if (current != i) {
                boolean isAlready = false;
                int n = 0;
                for (int j = 0; j < cycle.length; j++) {
                    if (cycle[j] == i) {
                        isAlready = true;
                        n = j;
                    }
                }
                if (!isAlready) {
                    int[] newCycle = new int[cycle.length + 1];
                    for (int j = 0; j < cycle.length; j++) {
                        newCycle[j] = cycle[j];
                    }
                    newCycle[newCycle.length - 1] = i;
                    boolean flag = stepForCycles(newCycle, i);
                    if (!flag) {
                        result = false;
                    }
                }
                else {
                    if ((cycle.length - n) % 2 == 1) {
                        addAdditionalState(current, i);
                        return false;
                    }
                }
            }
        }
    }
    return result;
}

private int getMaxStatePower() {
    int result = -1;
    for (int i = 0; i < connectionMatrix.length; i++) {
        int power = 0;
        for (int j = 0; j < connectionMatrix[i].length; j++) {
            if ((i != j) && (connectionMatrix[i][j] > -1)) {
                power++;
            }
        }
        for (int j = 0; j < connectionMatrix.length; j++) {
            if ((i != j) && (connectionMatrix[j][i] > -1) && (connectionMatrix[i][j] < 0)) {
                power++;
            }
        }
        if (power > result) {
            result = power;
        }
    }
    return result;
}

private void addAdditionalState(int from, int to) {
    String[] newStateNames = new String[stateNames.length + 1];
    for (int i = 0; i < stateNames.length; i++) {
        newStateNames[i] = stateNames[i];
    }
    newStateNames[newStateNames.length - 1] = "Z" + String.valueOf(newStateNames.length);
    stateNames = newStateNames;
    int[][] newYNumbers = new int[yNumbers.length + 1][];
    for (int i = 0; i < yNumbers.length; i++) {
        newYNumbers[i] = yNumbers[i];
    }
    newYNumbers[newStateNames.length - 1] = null;
```

```
        yNumbers = newYNumbers;
        int[][] newXNumbers = new int[xNumbers.length + 1][];
        for (int i = 0; i < xNumbers.length; i++) {
            newXNumbers[i] = xNumbers[i];
        }
        newXNumbers[newXNumbers.length - 1] = null;
        xNumbers = newXNumbers;
        int newCondition = xNumbers.length - 1;
        boolean[][] newXValues = new boolean[xValues.length + 1][];
        for (int i = 0; i < xValues.length; i++) {
            newXValues[i] = xValues[i];
        }
        newXValues[newXValues.length - 1] = null;
        xValues = newXValues;
        int[][] newConnectionMatrix = new int[connectionMatrix.length + 1][];
        for (int i = 0; i < connectionMatrix.length; i++) {
            newConnectionMatrix[i] = new int[connectionMatrix.length + 1];
            for (int j = 0; j < connectionMatrix[i].length; j++) {
                newConnectionMatrix[i][j] = connectionMatrix[i][j];
            }
            newConnectionMatrix[i][newConnectionMatrix[i].length - 1] = -1;
        }
        newConnectionMatrix[newConnectionMatrix.length - 1] = new int[newConnectionMatrix.length];
        for (int i = 0; i < newConnectionMatrix[newConnectionMatrix.length - 1].length; i++) {
            newConnectionMatrix[newConnectionMatrix.length - 1][i] = -1;
        }
        connectionMatrix = newConnectionMatrix;
        int added = connectionMatrix.length - 1;
        connectionMatrix[from][added] = connectionMatrix[from][to];
        connectionMatrix[from][to] = -1;
        connectionMatrix[added][to] = newCondition;
    }

    private String intToBinary(int i, int n) {
        String s = "";
        int temp = i;
        int divider = (int) Math.pow(2, n - 1);
        while (divider > 1) {
            s += String.valueOf(temp / divider);
            temp = temp % divider;
            divider /= 2;
        }
        s += String.valueOf(temp);
        return s;
    }

    public String[] getStateCodes() {
        return stateCodes;
    }

    public static void writeToFile(File file, CodedMooreAutomat automat) throws IOException {
        ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream(file));
        output.writeObject(automat);
        output.close();
    }

    public static CodedMooreAutomat readFromFile(File file) throws IOException, ClassNotFoundException {
        ObjectInputStream input = new ObjectInputStream(new FileInputStream(file));
        CodedMooreAutomat automat = (CodedMooreAutomat) input.readObject();
        input.close();
        return automat;
    }

}
```

```
package automat.moore;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 25.10.2010
 * Time: 1:53:18
 * To change this template use File | Settings | File Templates.
 */
public class CodedGraphPanel extends GraphPanel {

    public CodedGraphPanel(GraphModel model) {
        super(model);
        this.model.setAutomat(new CodedMooreAutomat(this.model.getAutomat()));
    }

    @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 = stateCenterX;
            int stateSelfConnectorY;
            if (stateCenterY < getHeight() / 2) {
                stateSelfConnectorY = currentY;
            }
            else {
                stateSelfConnectorY = currentY + model.getStateDiametr();
            }
            stateSelfConnectors.add(new Point(stateSelfConnectorX, stateSelfConnectorY));
            drawState(g2, currentX, currentY, model.getStateDiametr(), automat.getStateNames()[i], automat.getyNumbers()[i]);
            angle += temp;
        }
        CodedMooreAutomat codedAutomat = (CodedMooreAutomat) automat;
        String[] stateCodes = codedAutomat.getStateCodes();
        g2.setColor(model.getTextColor());
        g2.setFont(model.getFont());
        FontRenderContext context = g2.getFontRenderContext();
        for (int i = 0; i < stateCodes.length; i++) {
            Rectangle2D bounds = g2.getFont().getStringBounds(stateCodes[i], context);
            if ((stateSelfConnectors.get(i).getX() <= (getWidth() / 2)) && (stateSelfConnectors.get(i).getY() < (getHeight() / 2))) {
                g2.drawString(stateCodes[i], (int) (stateSelfConnectors.get(i).getX() - 1.25 * bounds.getX()), (int)
stateSelfConnectors.get(i).getY());
            }
            else {
                if ((stateSelfConnectors.get(i).getX() <= (getWidth() / 2)) && (stateSelfConnectors.get(i).getY() >= (getHeight() / 2))) {
                    g2.drawString(stateCodes[i], (int) (stateSelfConnectors.get(i).getX() - 1.25 * bounds.getX()), (int)
(stateSelfConnectors.get(i).getY() - bounds.getY()));
                }
                else {
                    if ((stateSelfConnectors.get(i).getX() > (getWidth() / 2)) && (stateSelfConnectors.get(i).getY() >= (getHeight() / 2))) {
                        g2.drawString(stateCodes[i], (int) (stateSelfConnectors.get(i).getX() + 0.25 * bounds.getX()), (int)
(stateSelfConnectors.get(i).getY() - bounds.getY()));
                    }
                    else {
                        g2.drawString(stateCodes[i], (int) (stateSelfConnectors.get(i).getX() + 0.25 * bounds.getX()), (int)
stateSelfConnectors.get(i).getY());
                    }
                }
            }
        }
        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;
                        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();
                                }
                            }
                        }
                        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: 28.10.2010
 * Time: 2:56:28
 * To change this template use File | Settings | File Templates.
 */
public class CodedGraphFileFilter extends FileFilter {

    public static String CODED_GRAPH_EXTENSION = ".cgraph";

    private static String CODED_GRAPH_DESCRIPTION = "Coded Graph File";

    public boolean accept(File pathname) {
        return (pathname.getName().toLowerCase().endsWith(CODED_GRAPH_EXTENSION) || pathname.isDirectory());
    }

    public String getDescription() {
        return CODED_GRAPH_DESCRIPTION;
    }

}
```

```
package face;

import automat.moore.*;

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;
    private CodedGraphPanel codedGraphPanel;
    private JButton codeGraphButton;

    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");
        codeGraphButton = new JButton(new CodeGraphAction(this));
        codeGraphButton.setText("Code Graph");
        JButton closeButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                setVisible(false);
            }
        });
        closeButton.setText("Close");
```

```
        mooreGraphButtonsPanel.add(saveGraphButton);
        mooreGraphButtonsPanel.add(codeGraphButton);
        mooreGraphButtonsPanel.add(closeButton);
        mooreGraphPanel.add(mooreGraphButtonsPanel, BorderLayout.SOUTH);
        mooreGraphPanel.add(graphPanel);
        tabbedPane.addTab("Graph Of Moore Automat", mooreGraphPanel);
    }

    public BuildFrame(MainFrame frame, Rectangle bounds, CodedMooreAutomat automat) {
        super(frame);
        mainFrame = frame;
        setBounds(bounds);
        setMinimumSize(bounds.getSize());
        setResizable(true);
        setModal(true);
        setTitle("Building");
        tabbedPane = new JTabbedPane();
        add(tabbedPane);
        JPanel mooreCodedGraphPanel = new JPanel();
        mooreCodedGraphPanel.setLayout(new BorderLayout());
        codedGraphPanel = new CodedGraphPanel(new GraphModel(automat));
        JPanel mooreGraphButtonsPanel = new JPanel();
        JButton saveCodedGraphButton = new JButton(new SaveCodedGraphAction(this));
        saveCodedGraphButton.setText("Save Graph");
        JButton closeButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                setVisible(false);
            }
        });
        closeButton.setText("Close");
        mooreGraphButtonsPanel.add(saveCodedGraphButton);
        mooreGraphButtonsPanel.add(closeButton);
        mooreCodedGraphPanel.add(mooreGraphButtonsPanel, BorderLayout.SOUTH);
        mooreCodedGraphPanel.add(codedGraphPanel);
        tabbedPane.addTab("Coded Graph Of Moore Automat", mooreCodedGraphPanel);
    }

    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);
                }
            }
        }

    }

    private class SaveCodedGraphAction extends AbstractAction {

        private BuildFrame frame;

        public SaveCodedGraphAction(BuildFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            JFileChooser chooser = mainFrame.getChooser();
            chooser.resetChoosableFileFilters();
            chooser.addChoosableFileFilter(new CodedGraphFileFilter());
            int result = chooser.showSaveDialog(frame);
            if (result == JFileChooser.APPROVE_OPTION) {
                if (!chooser.getSelectedFile().getName().endsWith(CodedGraphFileFilter.CODED_GRAPH_EXTENSION)) {
                    chooser.setSelectedFile(new File(chooser.getSelectedFile().getAbsolutePath() + CodedGraphFileFilter.CODED_GRAPH_EXTENSION));
                }
                try {
                    CodedMooreAutomat.writeToFile(chooser.getSelectedFile(), (CodedMooreAutomat) codedGraphPanel.getModel().getAutomat());
                } catch (IOException e1) {
                    JOptionPane.showMessageDialog(frame, "Error! Can't create file.",
                            "Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        }

    }

    private class CodeGraphAction extends AbstractAction {

        private BuildFrame frame;

        public CodeGraphAction(BuildFrame frame) {
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent e) {
            JPanel mooreCodedGraphPanel = new JPanel();
            mooreCodedGraphPanel.setLayout(new BorderLayout());
            codedGraphPanel = new CodedGraphPanel(new GraphModel(graphPanel.getModel().getAutomat()));
            JPanel mooreGraphButtonsPanel = new JPanel();
            JButton saveCodedGraphButton = new JButton(new SaveCodedGraphAction(frame));
            saveCodedGraphButton.setText("Save Graph");
            JButton closeButton = new JButton(new AbstractAction() {
```

```
                public void actionPerformed(ActionEvent e) {
                    setVisible(false);
                }
            });
            closeButton.setText("Close");
            mooreGraphButtonsPanel.add(saveCodedGraphButton);
            mooreGraphButtonsPanel.add(closeButton);
            mooreCodedGraphPanel.add(mooreGraphButtonsPanel, BorderLayout.SOUTH);
            mooreCodedGraphPanel.add(codedGraphPanel);
            tabbedPane.addTab("Coded Graph Of Moore Automat", mooreCodedGraphPanel);
            tabbedPane.setSelectedIndex(1);
            codeGraphButton.setEnabled(false);
        }

    }

}
```

Висновки

При виконанні даної лабораторної роботи я здобув навички з автоматизації сумісного кодування графу переходів автомату Мура. Сумісне кодування графу я реалізував за допомогою евристичного алгоритму кодування. Також мною була реалізована можливість зберігання/відновлення закодованого графу переходів за допомогою механізму серіалізації об’єктів мови програмування Java.