Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №5

з курсу «Автоматизація проектування комп'ютерних систем»

Виконав

студент групи ІО-73

Захожий Ігор

Номер залікової книжки: 7308

Київ-2010

Тема роботи

Автоматизація синтезу таблиці переходів.

Мета роботи

Здобуття навичок з аналізу графових структур і автоматизації процедури побудови таблиці переходів.

Завдання

1. Представити номер залікової книжки в двійковому вигляді: $7308_{10} = 1110010001100_2$.
2. В залежності від молодшого розряду номера залікової книжки визначити тип тригера:

n_2	n_1	Тип тригера
0	0	D

3. Розробити модуль генерації таблиці переходів і функцій збудження тригерів на основі закодованого графу переходів.
4. Реалізувати засоби відображення таблиці (п. 3) та її збереження у файлі.

Опис програми

Для побудови таблиці переходів та функцій збудження тригерів мною був розроблений клас AutomatTableModel. Він приймає в конструкторі об'єкт класу CodedMooreAutomat, що містить всі дані про граф. З цих даних і будується таблиця. Для її відображення мною був використаний клас JTable.

Для побудови таблиці переходів необхідно після кодування графу (л. р. №4) натиснути кнопку «Build Table Of Transitions», після чого відкриється нова вкладка «Table Of Transitions», в якій і буде відображена побудована таблиця. Наприклад, для даного графу (рисунок 1) таблиця переходів зображена на рисунку 2.

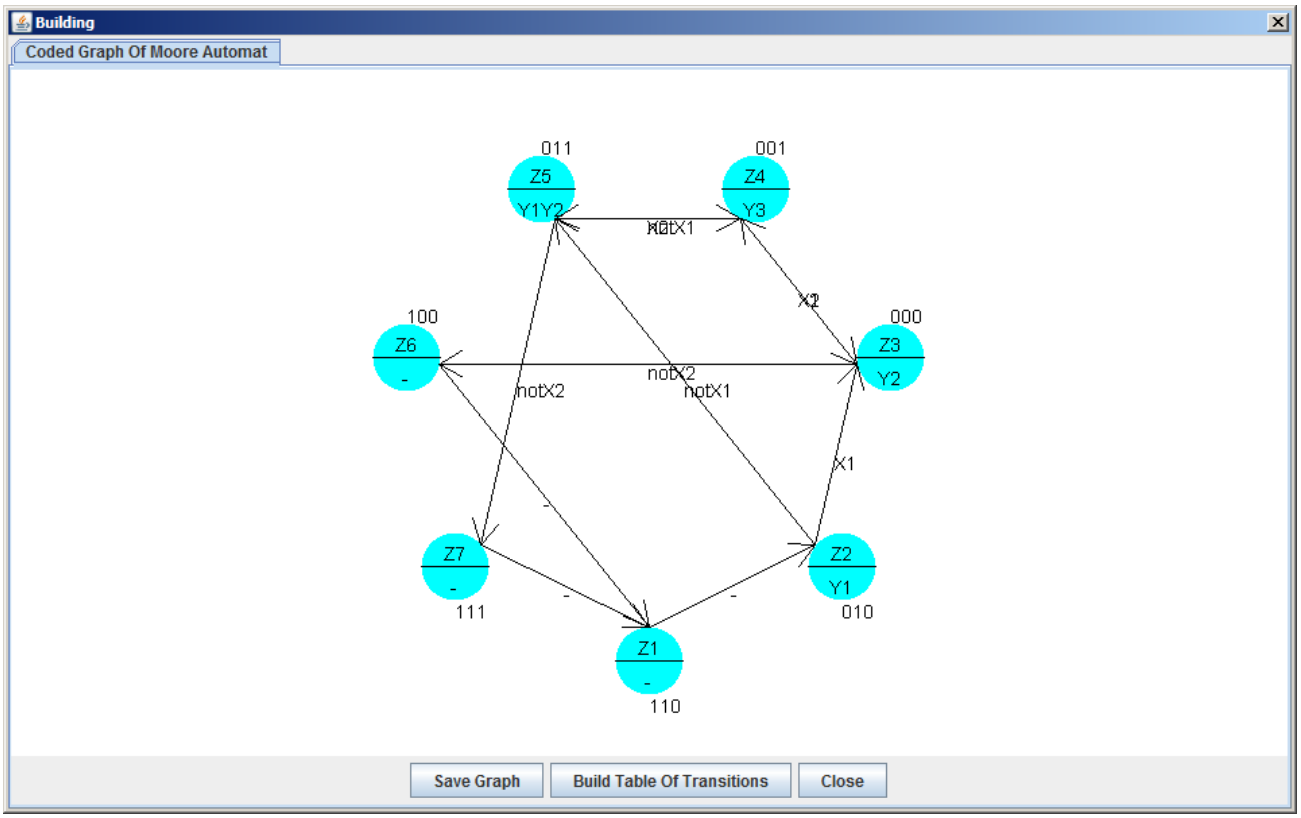
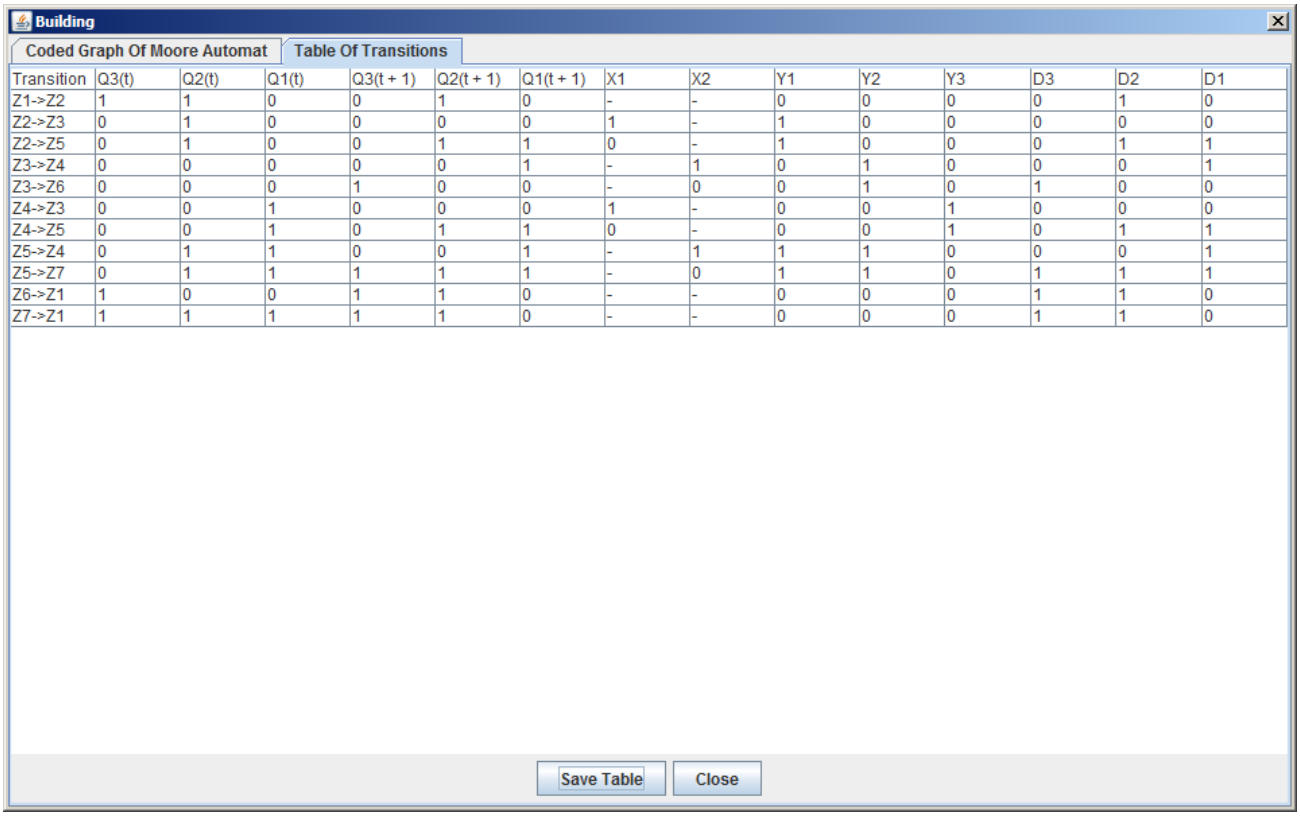


Рисунок 1



Transition	Q3(t)	Q2(t)	Q1(t)	Q3(t + 1)	Q2(t + 1)	Q1(t + 1)	X1	X2	Y1	Y2	Y3	D3	D2	D1
Z1->Z2	1	1	0	0	1	0	-	-	0	0	0	0	1	0
Z2->Z3	0	1	0	0	0	0	1	-	1	0	0	0	0	0
Z2->Z5	0	1	0	0	1	1	0	-	1	0	0	0	1	1
Z3->Z4	0	0	0	0	0	1	-	1	0	1	0	0	0	1
Z3->Z6	0	0	0	1	0	0	-	0	0	1	0	1	0	0
Z4->Z3	0	0	1	0	0	0	1	-	0	0	1	0	0	0
Z4->Z5	0	0	1	0	1	1	0	-	0	0	1	0	1	1
Z5->Z4	0	1	1	0	0	1	-	1	1	1	0	0	0	1
Z5->Z7	0	1	1	1	1	1	-	0	1	1	0	1	1	1
Z6->Z1	1	0	0	1	1	0	-	-	0	0	0	1	1	0
Z7->Z1	1	1	1	1	1	0	-	-	0	0	0	1	1	0

Рисунок 2

Для збереження таблиці у текстовому файлі необхідно натиснути на кнопку «Save Table» і вибрати необхідний файл у діалоговому вікні. Вміст файлу для таблиці з рисунку 2 показаний на рисунку 3.

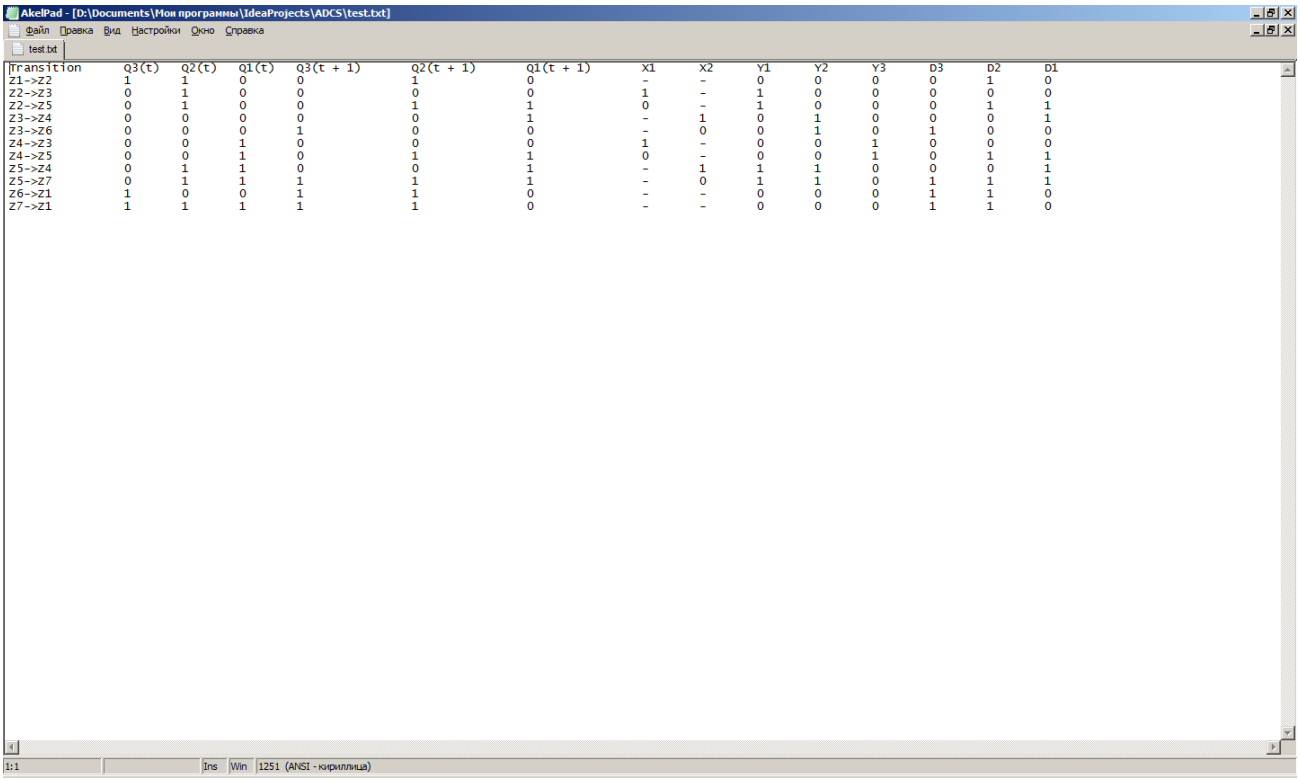


Рисунок 3

Лістинг програми

```
package automat.moore;

import javax.swing.table.AbstractTableModel;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
```

```
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 03.11.2010
 * Time: 5:26:16
 * To change this template use File | Settings | File Templates.
 */
public class AutomatTableModel extends AbstractTableModel {

    private String[][] table;

    public AutomatTableModel(CodedMooreAutomat automat) {
        String[] stateNames = automat.getStateNames();
        String[] stateCodes = automat.getStateCodes();
        int[][] connectionMatrix = automat.getConnectionMatrix();
        int[][] yNumbers = automat.getyNumbers();
        int[][] xNumbers = automat.getxNumbers();
        boolean[][] xValues = automat.getxValues();
        int colCount = 1 + 3 * stateCodes[0].length();
        ArrayList<Integer> x = new ArrayList<Integer>();
        for (int i = 0; i < xNumbers.length; i++) {
            if (xNumbers[i] != null) {
                for (int j = 0; j < xNumbers[i].length; j++) {
                    if (!x.contains(xNumbers[i][j])) {
                        x.add(xNumbers[i][j]);
                    }
                }
            }
        }
        Collections.sort(x);
        ArrayList<Integer> y = new ArrayList<Integer>();
        for (int i = 0; i < yNumbers.length; i++) {
            if (yNumbers[i] != null) {
                for (int j = 0; j < yNumbers[i].length; j++) {
                    if (!y.contains(yNumbers[i][j])) {
                        y.add(yNumbers[i][j]);
                    }
                }
            }
        }
        Collections.sort(y);
        colCount += x.size() + y.size();
        table = new String[xNumbers.length + 1][];
        for (int i = 0; i < table.length; i++) {
            table[i] = new String[colCount];
        }
        table[0][0] = "Transition";
        int column = 1;
        for (int i = 0; i < stateCodes[0].length(); i++) {
            table[0][column] = "Q" + String.valueOf(stateCodes[0].length() - i) + "(t)";
            column++;
        }
        for (int i = 0; i < stateCodes[0].length(); i++) {
            table[0][column] = "Q" + String.valueOf(stateCodes[0].length() - i) + "(t + 1)";
            column++;
        }
        for (int i = 0; i < x.size(); i++) {
            table[0][column] = "X" + String.valueOf(x.get(i));
            column++;
        }
        for (int i = 0; i < y.size(); i++) {
            table[0][column] = "Y" + String.valueOf(y.get(i));
            column++;
        }
        for (int i = 0; i < stateCodes[0].length(); i++) {
            table[0][column] = "D" + String.valueOf(stateCodes[0].length() - i);
            column++;
        }
        int row = 1;
        for (int i = 0; i < connectionMatrix.length; i++) {
            for (int j = 0; j < connectionMatrix[i].length; j++) {
                if (connectionMatrix[i][j] > -1) {
                    table[row][0] = stateNames[i] + "->" + stateNames[j];
                    column = 1;
                    for (int k = 0; k < stateCodes[i].length(); k++) {
                        table[row][column] = stateCodes[i].substring(k, k + 1);
                        column++;
                    }
```

```
                for (int k = 0; k < stateCodes[j].length(); k++) {
                    table[row][column] = stateCodes[j].substring(k, k + 1);
                    column++;
                }
                for (int k = 0; k < x.size(); k++) {
                    if (xNumbers[connectionMatrix[i][j]] != null) {
                        int found = -1;
                        for (int z = 0; z < xNumbers[connectionMatrix[i][j]].length; z++) {
                            if (x.get(k) == xNumbers[connectionMatrix[i][j]][z]) {
                                found = z;
                            }
                        }
                        if (found > -1) {
                            if (xValues[connectionMatrix[i][j]][found]) {
                                table[row][column] = "1";
                            }
                            else {
                                table[row][column] = "0";
                            }
                        }
                        else {
                            table[row][column] = "-";
                        }
                    }
                    else {
                        table[row][column] = "-";
                    }
                    column++;
                }
                for (int k = 0; k < y.size(); k++) {
                    if (yNumbers[i] != null) {
                        int found = -1;
                        for (int z = 0; z < yNumbers[i].length; z++) {
                            if (y.get(k) == yNumbers[i][z]) {
                                found = z;
                            }
                        }
                        if (found > -1) {
                            table[row][column] = "1";
                        }
                        else {
                            table[row][column] = "0";
                        }
                    }
                    else {
                        table[row][column] = "0";
                    }
                    column++;
                }
                for (int k = 0; k < stateCodes[j].length(); k++) {
                    table[row][column] = stateCodes[j].substring(k, k + 1);
                    column++;
                }
                row++;
            }
        }
    }
}

public void writeToFile(File file) throws IOException {
    PrintWriter output = new PrintWriter(new FileWriter(file));
    int[] maxColumnLength = new int[table[0].length];
    for (int i = 0; i < table.length; i++) {
        for (int j = 0; j < table[i].length; j++) {
            if (table[i][j].length() > maxColumnLength[j]) {
                maxColumnLength[j] = table[i][j].length();
            }
        }
    }
    for (int i = 0; i < table.length; i++) {
        for (int j = 0; j < table[i].length; j++) {
            int disparity = maxColumnLength[j] - table[i][j].length();
            output.print(table[i][j]);
            for (int k = 0; k < disparity; k++) {
                output.print(" ");
            }
            output.print("\t");
        }
        output.println();
    }
    output.close();
}
```

```
    public int getRowCount() {
        return table.length;
    }

    public int getColumnCount() {
        return table[0].length;
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        return table[rowIndex][columnIndex];
    }

}
```

```
package automat.moore;

import javax.swing.filechooser.FileFilter;
import java.io.File;

/**
 * Created by IntelliJ IDEA.
 * User: Zak
 * Date: 03.11.2010
 * Time: 15:29:26
 * To change this template use File | Settings | File Templates.
 */
public class TextFileFilter extends FileFilter {

    public static String TEXT_FILE_EXTENSION = ".txt";

    private static String TEXT_FILE_DESCRIPTION = "Text File";

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

    public String getDescription() {
        return TEXT_FILE_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;
    private AutomatTableModel tableModel;
    private JButton buildTableButton;

    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");
        buildTableButton = new JButton(new BuildTableAction(this));
        buildTableButton.setText("Build Table Of Transitions");
        JButton closeButton = new JButton(new AbstractAction() {
            public void actionPerformed(ActionEvent e) {
                setVisible(false);
            }
        });
        closeButton.setText("Close");
        mooreGraphButtonsPanel.add(saveCodedGraphButton);
        mooreGraphButtonsPanel.add(buildTableButton);
        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");
            buildTableButton = new JButton(new BuildTableAction(frame));
            buildTableButton.setText("Build Table Of Transitions");
            JButton closeButton = new JButton(new AbstractAction() {
                public void actionPerformed(ActionEvent e) {
                    setVisible(false);
                }
            });
            closeButton.setText("Close");
            mooreGraphButtonsPanel.add(saveCodedGraphButton);
            mooreGraphButtonsPanel.add(buildTableButton);
            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);
        }

    }

    private class SaveTableAction extends AbstractAction {

        private BuildFrame frame;

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

        public void actionPerformed(ActionEvent e) {
            JFileChooser chooser = mainFrame.getChooser();
            chooser.resetChoosableFileFilters();
            chooser.addChoosableFileFilter(new TextFileFilter());
            int result = chooser.showSaveDialog(frame);
            if (result == JFileChooser.APPROVE_OPTION) {
                if (!chooser.getSelectedFile().getName().endsWith(TextFileFilter.TEXT_FILE_EXTENSION)) {
```

```
                    chooser.setSelectedFile(new File(chooser.getSelectedFile().getAbsolutePath() +
TextFileFilter.TEXT_FILE_EXTENSION));
                }
                try {
                    tableModel.writeToFile(chooser.getSelectedFile());
                } catch (IOException e1) {
                    JOptionPane.showMessageDialog(frame, "Error! Can't create file.",
                            "Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        }

    }

    private class BuildTableAction extends AbstractAction {

        private BuildFrame frame;

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

        public void actionPerformed(ActionEvent e) {
            JPanel tablePanel = new JPanel();
            tablePanel.setLayout(new BorderLayout());
            tableModel = new AutomatTableModel((CodedMooreAutomat) codedGraphPanel.getModel().getAutomat());
            JTable table = new JTable(tableModel);
            JPanel tableButtonsPanel = new JPanel();
            JButton saveTableButton = new JButton(new SaveTableAction(frame));
            saveTableButton.setText("Save Table");
            JButton closeButton = new JButton(new AbstractAction() {
                public void actionPerformed(ActionEvent e) {
                    setVisible(false);
                }
            });
            closeButton.setText("Close");
            tableButtonsPanel.add(saveTableButton);
            tableButtonsPanel.add(closeButton);
            tablePanel.add(tableButtonsPanel, BorderLayout.SOUTH);
            tablePanel.add(table);
            tabbedPane.addTab("Table Of Transitions", tablePanel);
            tabbedPane.setSelectedIndex(tabbedPane.getTabCount() - 1);
            buildTableButton.setEnabled(false);
        }

    }

}
```

Висновки

При виконанні даної лабораторної роботи я здобув навички з аналізу графових структур і автоматизації процедури побудови таблиці переходів. Мною був розроблений модуль для генерації таблиці переходів. А також я реалізував засоби для візуального відображення сгенерованої таблиці переходів автомату. Для цього мною був використаний клас JTable мови програмування Java.