Міністерство освіти та науки України

Національний технічний університет України «КПІ»

Факультет інформатики та обчислювальної техніки

**Автоматизація проектування комп'ютерних систем**

**Лабораторна робота №4**

**Автоматизація розмітки блок-схем алгоритмів**

**Виконав**

**ст. гр. ІО-71**

**Ювженко Д.**

**Київ 2010**

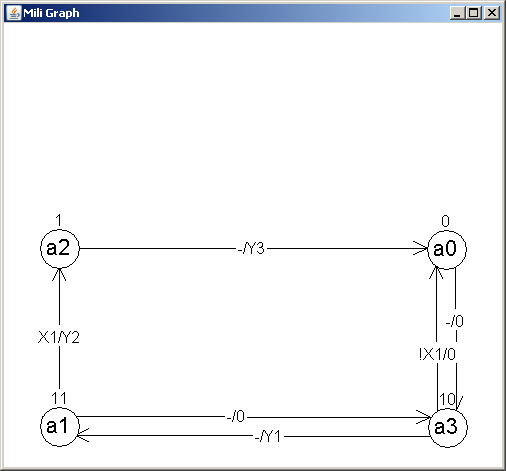
Варіант 7122 = 1101111010010

Мета: здобуття навичок з автоматизації процедури розмітки алгоритмів за методом Мілі.

Завдання:

Метод розмітки: Мілі.

Тип формату: текстовий.



Лістинг програми:

package lab;

import java.awt.Dimension;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.Insets;

import java.awt.Rectangle;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.util.ArrayList;

import javax.swing.JButton;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

public class MainWindow extends JFrame {

private static final long serialVersionUID = 1L;

private JPanel ButtonPanel = new JPanel(new GridBagLayout());

private JButton bAddBegin = new JButton("Begin");

private JButton bAddEnd = new JButton("End");

private JButton bAddArithmetical = new JButton("Arithmetical");

private JButton bAddLogical = new JButton("Logical");

private JButton bAddOpenBracket = new JButton("(");

private JButton bAddCloseBracket = new JButton(")");

private JButton bAddInArrow = new JButton("In Arrow");

private JButton bAddOutArrow = new JButton("Out Arrow");

private JButton bOpenLSA = new JButton("Open LSA");

private JButton bSaveLSA = new JButton("Save LSA");

private JButton bDrawMG = new JButton("Draw Mili Graph");

private JButton bMarkMG = new JButton("Mark Mili Graph");

private JButton bSaveMG = new JButton("Save Mili Graph");

private JButton bOpenMG = new JButton("Open Mili Graph");

private LSADrawPanel lsaPanel = new LSADrawPanel(true);

private JScrollPane lsaPanelScroll = new JScrollPane(lsaPanel);

private JFrame fMiliGraph = new JFrame("Mili Graph");

private MiliGraphDrawPanel mgPanel = new MiliGraphDrawPanel(true);

private JFrame fStructureTable = new JFrame("Structure Table");

private JTable tStructureTable = new JTable();

private JScrollPane spStructureTable = new JScrollPane(tStructureTable);

private JFrame fMinimizedFunctions = new JFrame("Minimized Functions");

private JTable tMinimizedFunctions = new JTable();

private JScrollPane spMinimizedFunctions = new JScrollPane(tMinimizedFunctions);

private JFileChooser FileChoosingDialog = new JFileChooser();

private ObjectOutputStream outObject;

private ObjectInputStream inObject;

private BufferedWriter outText;

private BufferedReader inText;

private String InputSignalsNames = "";

public MainWindow() {

this.setLayout(new GridBagLayout());

this.setTitle("LSA Editor");

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setSize(850, 241);

this.setMinimumSize(new Dimension(850, 241));

this.setMaximizedBounds(new Rectangle(0, 0, (int) this.getToolkit().getScreenSize().getWidth(), (int) this.getMinimumSize().getHeight()-27));

this.setLocation((int)(this.getToolkit().getScreenSize().getWidth()-this.getWidth())/2, (int)(this.getToolkit().getScreenSize().getHeight()-this.getHeight())/2);

setLookOfButtonPanel();

setLookOflsaPanel();

setAllListeners();

setLookOfMiliGraphWindow();

setLookOfStructureTableWindow();

setLookOfMinimizedFunctionsWindow();

FileChoosingDialog.setCurrentDirectory(new File("."));

FileChoosingDialog.setAcceptAllFileFilterUsed(false);

this.setVisible(true);

}

private void setLookOfStructureTableWindow() {

fStructureTable.setDefaultCloseOperation(JFrame.HIDE\_ON\_CLOSE);

fStructureTable.add(spStructureTable);

fStructureTable.setSize(700, 300);

}

private void setLookOfMiliGraphWindow() {

fMiliGraph.setDefaultCloseOperation(JFrame.HIDE\_ON\_CLOSE);

fMiliGraph.add(mgPanel);

fMiliGraph.setSize(500, 500);

}

private void setLookOfButtonPanel() {

ButtonPanel.add(bAddArithmetical, new GridBagConstraints(0, 0, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bAddLogical, new GridBagConstraints(0, 1, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bAddInArrow, new GridBagConstraints(0, 2, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bAddOutArrow, new GridBagConstraints(0, 3, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bAddBegin, new GridBagConstraints(1, 0, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bAddEnd, new GridBagConstraints(2, 0, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bAddOpenBracket, new GridBagConstraints(1, 1, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bAddCloseBracket, new GridBagConstraints(2, 1, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bOpenLSA, new GridBagConstraints(1, 2, 2, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bSaveLSA, new GridBagConstraints(1, 3, 2, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bMarkMG, new GridBagConstraints(3, 0, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bDrawMG, new GridBagConstraints(3, 1, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bOpenMG, new GridBagConstraints(3, 2, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

ButtonPanel.add(bSaveMG, new GridBagConstraints(3, 3, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(1, 1, 1, 1), 0 , 0));

this.add(ButtonPanel, new GridBagConstraints(0, 0, 1, 1, 100, 0, GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 0 , 0));

}

private void setLookOflsaPanel() {

lsaPanel.setParentWindow(this);

lsaPanelScroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

this.add(lsaPanelScroll, new GridBagConstraints(0, 1, 1, 1, 100, 100, GridBagConstraints.CENTER, GridBagConstraints.BOTH, new Insets(1, 1, 1, 1), 0 , 0));

}

@SuppressWarnings("unchecked")

private String[] minimizeFunction(ArrayList<String> Constituents, ArrayList<String> CoveringTableConstituents) {

ArrayList<String> NewCube = (ArrayList<String>) Constituents.clone();

ArrayList<String> CurrentCube;

ArrayList<Integer> CurrentAbsorptionCube = new ArrayList<Integer>();

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

ArrayList<ArrayList<Integer>> AbsorptionCubes = new ArrayList<ArrayList<Integer>>();

boolean Splicing = false;

do {

Splicing = false;

CurrentCube = (ArrayList<String>) NewCube.clone();

NewCube.clear();

CurrentAbsorptionCube.clear();

for (int i = 0; i<CurrentCube.size()-1; i++)

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

boolean OneBitDifferece = false;

int OneBitDiffereceIndex = 0;

for (int k = 0; k<CurrentCube.get(i).length(); k++)

if (CurrentCube.get(i).charAt(k)!=CurrentCube.get(j).charAt(k))

if ((CurrentCube.get(i).charAt(k)!='-') && (CurrentCube.get(j).charAt(k)!='-') && (!OneBitDifferece)) {

OneBitDifferece = true;

OneBitDiffereceIndex = k;

}

else {

OneBitDifferece = false;

break;

}

if (OneBitDifferece) {

if ((!NewCube.contains(CurrentCube.get(i).substring(0, OneBitDiffereceIndex)+"-"+CurrentCube.get(i).substring(OneBitDiffereceIndex+1, CurrentCube.get(i).length()))))

NewCube.add(CurrentCube.get(i).substring(0, OneBitDiffereceIndex)+"-"+CurrentCube.get(i).substring(OneBitDiffereceIndex+1, CurrentCube.get(i).length()));

if (!CurrentAbsorptionCube.contains(i))

CurrentAbsorptionCube.add(i);

if (!CurrentAbsorptionCube.contains(j))

CurrentAbsorptionCube.add(j);

}

}

Cubes.add((ArrayList<String>) CurrentCube.clone());

AbsorptionCubes.add((ArrayList<Integer>) CurrentAbsorptionCube.clone());

if (!NewCube.isEmpty())

Splicing = true;

} while (Splicing);

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

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

for (int j = 0; j<Cubes.get(i).size(); j++)

if (!AbsorptionCubes.get(i).contains(j))

TDNF.add(Cubes.get(i).get(j));

Integer[][] CoveringTable = new Integer[TDNF.size()][CoveringTableConstituents.size()];

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

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

boolean Cover = true;

for (int k = 0; k<TDNF.get(i).length(); k++)

if ((TDNF.get(i).charAt(k)!=CoveringTableConstituents.get(j).charAt(k)) && (TDNF.get(i).charAt(k)!='-')) {

Cover = false;

break;

}

if (Cover)

CoveringTable[i][j] = 1;

else

CoveringTable[i][j] = 0;

}

Boolean[] CoveredConstituents = new Boolean[CoveringTableConstituents.size()];

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

CoveredConstituents[i] = false;

Boolean AllConstituentsAreCovered = false;

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

while (!AllConstituentsAreCovered) {

int MinConstituentOnesQuantity = Integer.MAX\_VALUE;

ArrayList<Integer> MinConstituentOnesNumbers = new ArrayList<Integer>();

for (int j = 0; j<CoveringTable[0].length; j++)

if (!CoveredConstituents[j]) {

int CurrentConstituentOnesQuantity = 0;

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

if (CoveringTable[i][j]==1)

CurrentConstituentOnesQuantity++;

if (CurrentConstituentOnesQuantity<MinConstituentOnesQuantity) {

MinConstituentOnesQuantity = CurrentConstituentOnesQuantity;

MinConstituentOnesNumbers.clear();

}

if (CurrentConstituentOnesQuantity==MinConstituentOnesQuantity)

MinConstituentOnesNumbers.add(j);

}

int MaxUncoveredConstituentsQuantity = 0;

int MaxUncoveredConstituentsNumber = 0;

for (int j = 0; j<MinConstituentOnesNumbers.size(); j++) {

ArrayList<Integer> CurrentConstituentCoveringImplicantsNumbers = new ArrayList<Integer>();

for (int k = 0; k<CoveringTable.length; k++)

if (CoveringTable[k][MinConstituentOnesNumbers.get(j)] == 1)

CurrentConstituentCoveringImplicantsNumbers.add(k);

ArrayList<Integer> CurrentConstituentCoveringConstituentsNumbers = new ArrayList<Integer>();

for (int k = 0; k<CurrentConstituentCoveringImplicantsNumbers.size(); k++)

for (int l = 0; l<CoveringTable[CurrentConstituentCoveringImplicantsNumbers.get(k)].length; l++)

if ((CoveringTable[CurrentConstituentCoveringImplicantsNumbers.get(k)][l]==1) && (!CoveredConstituents[l]) && (!CurrentConstituentCoveringConstituentsNumbers.contains(l)))

CurrentConstituentCoveringConstituentsNumbers.add(l);

if (CurrentConstituentCoveringConstituentsNumbers.size()>MaxUncoveredConstituentsQuantity) {

MaxUncoveredConstituentsQuantity = CurrentConstituentCoveringConstituentsNumbers.size();

MaxUncoveredConstituentsNumber = MinConstituentOnesNumbers.get(j);

}

}

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

if (CoveringTable[i][MaxUncoveredConstituentsNumber]==1) {

MDNF.add(TDNF.get(i));

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

if (CoveringTable[i][j]==1)

CoveredConstituents[j] = true;

}

AllConstituentsAreCovered = true;

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

if (!CoveredConstituents[i]) {

AllConstituentsAreCovered = false;

break;

}

}

String[] ArrayMDNF = new String[MDNF.size()];

ArrayMDNF = MDNF.toArray(ArrayMDNF);

return ArrayMDNF;

}

int SignalsQuantityForOR3 = Constituents.length;

while (SignalsQuantityForOR3>3) {

if (SignalsQuantityForOR3%3==0)

SignalsQuantityForOR3 = SignalsQuantityForOR3/3;

else

SignalsQuantityForOR3 = SignalsQuantityForOR3/3+1;

SignalsQuantity += SignalsQuantityForOR3;

}

if (!negativeSignalsNames.isEmpty() || SignalsQuantity!=0)

Result += "signal ";

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

if ((i==(negativeSignalsNames.size()-1)) && (SignalsQuantity==0))

Result += "S"+i+" : bit;\n";

else

Result += "not"+negativeSignalsNames.get(i)+", ";

for (int i = 1; i<=SignalsQuantity; i++)

if (i!=SignalsQuantity)

Result += "S"+i+", ";

else

Result += "S"+i+" : bit;\n";

Result += "begin\n";

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

Result += "\tnot"+negativeSignalsNames.get(i)+" <= not "+negativeSignalsNames.get(i)+";\n";

int LabelCounter = 1;

int SignalCounter = 1;

ArrayList<String> SignalsNamesForOR3 = new ArrayList<String>();

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

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

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

if (Constituents[i].charAt(j)!='-')

if (Constituents[i].charAt(j)=='0')

CurrentSignalsNames.add("not"+InputSignals[j]);

else

CurrentSignalsNames.add(InputSignals[j]);

while (CurrentSignalsNames.size()>3) {

int AND3Quantity = 0;

if (CurrentSignalsNames.size()%3==0)

AND3Quantity = CurrentSignalsNames.size()/3;

else

AND3Quantity = CurrentSignalsNames.size()/3+1;

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

for (int k = 0; k<AND3Quantity; k++) {

Result += "\tL"+(LabelCounter++)+": entity AND3(BEH) port map (";

for (int l = 3\*k; l<3\*k+3; l++)

if (l>=CurrentSignalsNames.size())

Result += CurrentSignalsNames.get(CurrentSignalsNames.size()-1)+", ";

else

Result += CurrentSignalsNames.get(l)+", ";

Result += "S"+SignalCounter+");\n";

TempSignalsNames.add("S"+(SignalCounter++));

}

CurrentSignalsNames = TempSignalsNames;

}

Result += "\tL"+(LabelCounter++)+": entity AND3(BEH) port map (";

for (int l = 0; l<3; l++)

if (l>=CurrentSignalsNames.size())

Result += CurrentSignalsNames.get(CurrentSignalsNames.size()-1)+", ";

else

Result += CurrentSignalsNames.get(l)+", ";

Result += "S"+SignalCounter+");\n";

SignalsNamesForOR3.add("S"+(SignalCounter++));

}

while (SignalsNamesForOR3.size()>3) {

int OR3Quantity = 0;

if (SignalsNamesForOR3.size()%3==0)

OR3Quantity = SignalsNamesForOR3.size()/3;

else

OR3Quantity = SignalsNamesForOR3.size()/3+1;

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

for (int k = 0; k<OR3Quantity; k++) {

Result += "\tL"+(LabelCounter++)+": entity OR3(BEH) port map (";

for (int l = 3\*k; l<3\*k+3; l++)

if (l>=SignalsNamesForOR3.size())

Result += SignalsNamesForOR3.get(SignalsNamesForOR3.size()-1)+", ";

else

Result += SignalsNamesForOR3.get(l)+", ";

Result += "S"+SignalCounter+");\n";

TempSignalsNames.add("S"+(SignalCounter++));

}

SignalsNamesForOR3 = TempSignalsNames;

}

Result += "\tL"+(LabelCounter++)+": entity OR3(BEH) port map (";

for (int l = 0; l<3; l++)

if (l>=SignalsNamesForOR3.size())

Result += SignalsNamesForOR3.get(SignalsNamesForOR3.size()-1)+", ";

else

Result += SignalsNamesForOR3.get(l)+", ";

Result += FunctionName+");\n";

Result += "end BEH;\n\n";

return Result;

}

ResultList.add(ISNames.substring(0, ISNames.length()));

String[] ResultArray = new String[ResultList.size()];

ResultArray = ResultList.toArray(ResultArray);

return ResultArray;

}

bAddOutArrow.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

ArrowNodePropertiesDialog currentDialog;

currentDialog = new ArrowNodePropertiesDialog(MainWindow.this, 1);

if (currentDialog.isDisplayable()) {

lsaPanel.addDrawObject(new OutArrowNode(currentDialog.getNodeNumber()));

currentDialog.dispose();

}

}

});

bSaveLSA.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (lsaPanel.isCorrectInput()) {

FileChoosingDialog.setFileFilter(new LSAF\_Filter());

if (JFileChooser.APPROVE\_OPTION==FileChoosingDialog.showSaveDialog(MainWindow.this)) {

try {

if (FileChoosingDialog.getSelectedFile().getName().endsWith(".lsaf"))

outObject = new ObjectOutputStream(new FileOutputStream(FileChoosingDialog.getSelectedFile().getPath()));

else

outObject = new ObjectOutputStream(new FileOutputStream(FileChoosingDialog.getSelectedFile().getPath()+".lsaf"));

outObject.writeObject(lsaPanel.getDrawObjects());

outObject.writeObject(lsaPanel.getCoherenceMatrix());

outObject.close();

} catch (FileNotFoundException e1) {

e1.printStackTrace();

JOptionPane.showMessageDialog(MainWindow.this, "Can't Save Algorithm.");

} catch (IOException e1) {

e1.printStackTrace();

JOptionPane.showMessageDialog(MainWindow.this, "Can't Save Algorithm.");

}

}

}

else

JOptionPane.showMessageDialog(MainWindow.this, "Can't Save Algorithm.");

}

});

bOpenLSA.addActionListener(new ActionListener() {

@SuppressWarnings("unchecked")

public void actionPerformed(ActionEvent e) {

FileChoosingDialog.setFileFilter(new LSAF\_Filter());

if (JFileChooser.APPROVE\_OPTION==FileChoosingDialog.showOpenDialog(MainWindow.this)) {

try {

if (FileChoosingDialog.getSelectedFile().getName().endsWith(".lsaf"))

inObject = new ObjectInputStream(new FileInputStream(FileChoosingDialog.getSelectedFile().getPath()));

else

inObject = new ObjectInputStream(new FileInputStream(FileChoosingDialog.getSelectedFile().getPath()+".lsaf"));

lsaPanel.addObjectsFromFile((ArrayList<AbstractNode>)inObject.readObject(), (int[][])inObject.readObject());

inObject.close();

} catch (FileNotFoundException e1) {

JOptionPane.showMessageDialog(MainWindow.this, "File Not Found.");

} catch (IOException e1) {

JOptionPane.showMessageDialog(MainWindow.this, "Can't Open Algorithm.");

} catch (ClassNotFoundException e1) {

JOptionPane.showMessageDialog(MainWindow.this, "Wrong File Format.");

}

}

}

});

bMarkMG.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (!mgPanel.getDrawObjects().isEmpty()) {

mgPanel.setGrayCode();

if (!fMiliGraph.isVisible())

fMiliGraph.setVisible(true);

}

else

JOptionPane.showMessageDialog(MainWindow.this, "Can't Mark Mili Graph.");

}

});

bDrawMG.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (lsaPanel.isCorrectInput()) {

mgPanel.addDrawObjects(lsaPanel.getMiliDrawObjects());

if (!fMiliGraph.isVisible())

fMiliGraph.setVisible(true);

}

else

JOptionPane.showMessageDialog(MainWindow.this, "Can't Draw Mili's Graph.");

}

});

bSaveMG.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (!mgPanel.getDrawObjects().isEmpty()) {

FileChoosingDialog.setFileFilter(new MGF\_Filter());

if (JFileChooser.APPROVE\_OPTION==FileChoosingDialog.showSaveDialog(MainWindow.this)) {

try {

if (FileChoosingDialog.getSelectedFile().getName().endsWith(".mgf"))

outText = new BufferedWriter(new FileWriter(FileChoosingDialog.getSelectedFile().getPath()));

else

outText = new BufferedWriter(new FileWriter(FileChoosingDialog.getSelectedFile().getPath()+".mgf"));

outText.write(mgPanel.getGraphTextFormat());

outText.close();

} catch (FileNotFoundException e1) {

e1.printStackTrace();

JOptionPane.showMessageDialog(MainWindow.this, "Can't Save Mili Graph.");

} catch (IOException e1) {

e1.printStackTrace();

JOptionPane.showMessageDialog(MainWindow.this, "Can't Save Mili Graph.");

}

}

}

else

JOptionPane.showMessageDialog(MainWindow.this, "Can't Save Mili Graph.");

}

});

bOpenMG.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

FileChoosingDialog.setFileFilter(new MGF\_Filter());

if (JFileChooser.APPROVE\_OPTION==FileChoosingDialog.showOpenDialog(MainWindow.this)) {

try {

if (FileChoosingDialog.getSelectedFile().getName().endsWith(".mgf"))

inText = new BufferedReader(new FileReader(FileChoosingDialog.getSelectedFile().getPath()));

else

inText = new BufferedReader(new FileReader(FileChoosingDialog.getSelectedFile().getPath()+".mgf"));

mgPanel.addDrawObjectsFromString(inText);

if (!fMiliGraph.isVisible())

fMiliGraph.setVisible(true);

} catch (FileNotFoundException e1) {

JOptionPane.showMessageDialog(MainWindow.this, "File Not Found.");

}

}

}

});

lsaPanel.addKeyListener(new KeyListener() {

public void keyPressed(KeyEvent e) {

if (e.getKeyCode()==KeyEvent.VK\_DELETE)

lsaPanel.removeSelectedNode();

}

public void keyReleased(KeyEvent e) {}

public void keyTyped(KeyEvent e) {}

});

}

}

Висновок: під час виконання даної лабораторної роботи були отримані навички з автоматизації процедури розмітки алгоритмів за методом Мілі та графічного відображення графу переходів. Також були реалізовані функції збереження/відновлення графу переходів.