Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №6**

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

Виконав

студент групи ІО-73

Захожий Ігор

Номер залікової книжки: 7308

Київ-2010

**Тема роботи**

Автоматизація мінімізації булевих функцій.

**Мета роботи**

Здобуття навичок автоматизації процедури мінімізації булевих функцій методом Квайна-МакКласкі.

**Завдання**

1. Розробити процедуру мінімізації булевих функцій методом Квайна-МакКласкі.

2. Розробити засоби аналізу ефективності мінімізації по кількості елементів, кількості входів/виходів, довжині критичного шляху.

3. На основі розробленої процедури (п. 1) реалізувати модуль мінімізації булевих функцій переходів і функцій збудження тригерів з таблиці, побудованої в попередній роботі. Передбачити відображення немінімізованих/мінімізованих функцій та ефективності їх мінімізації (п. 2).

4. Реалізувати засоби збереження результатів мінімізації у файлі.

**Опис програми**

Для побудови аналітичної форми функцій переходів та функцій збудження тригерів необхідно натиснути кнопку «Build Functions» (Рис. 1). Результат для даної таблиці (Рис. 1) зображено на рисунку 2.

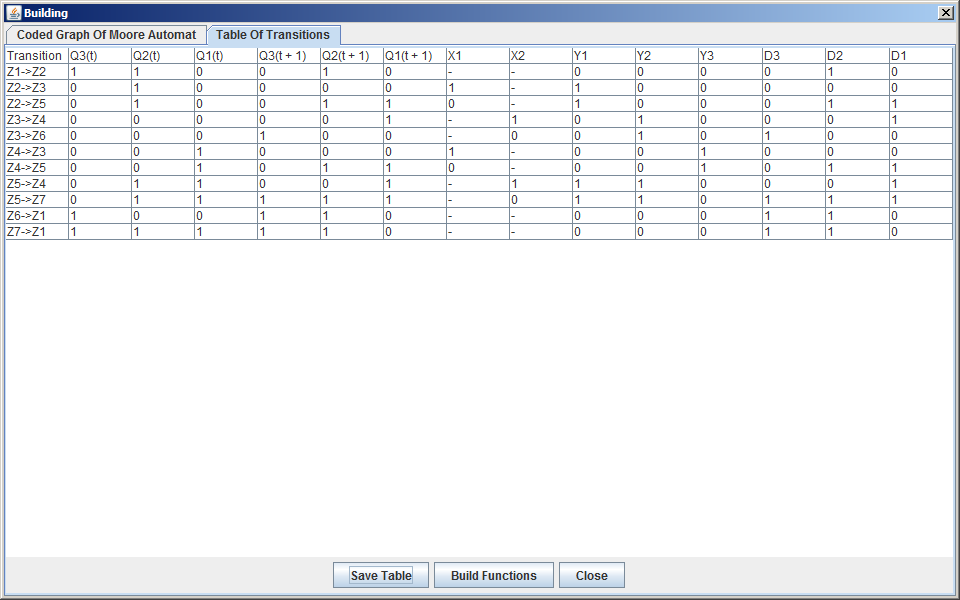


Рисунок 1

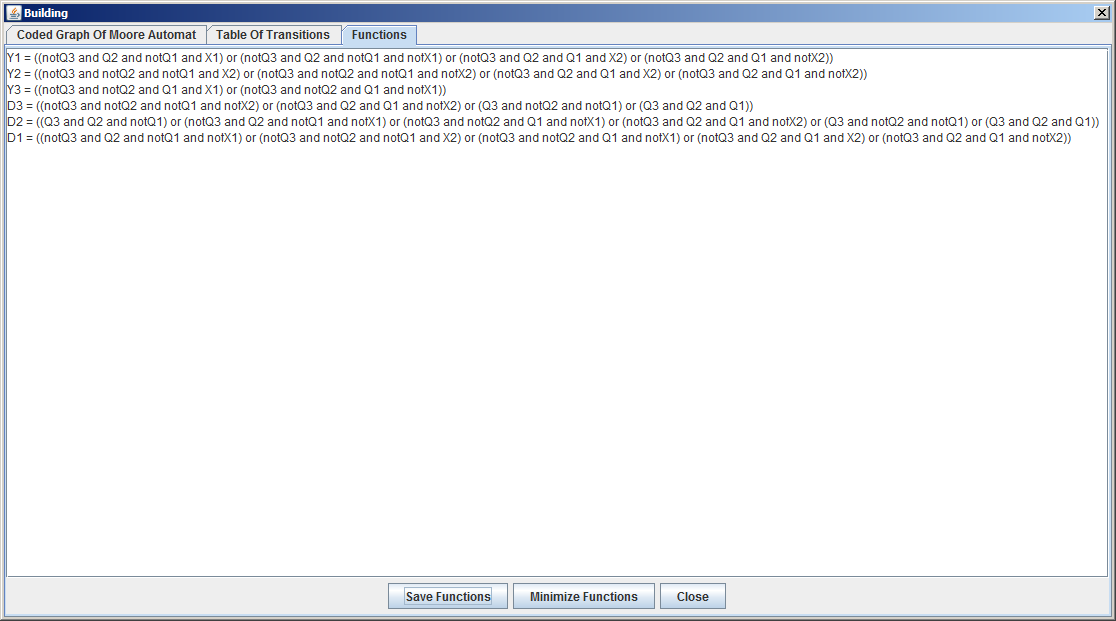


Рисунок 2

Для збереження аналітичного представлення не мінімізованих функцій необхідно натиснути кнопку «Save Functions» та в діалоговому вікні вибрати необхідний файл. Для мінімізації функцій необхідно натиснути кнопку «Minimize Functions». У результаті в новій вкладці будуть відображені мінімізовані функції (Рис. 3). Процедура мінімізації методом Квайна-МакКласкі реалізована в статичному методі minimizeFunctions() класу FunctionsWorker.

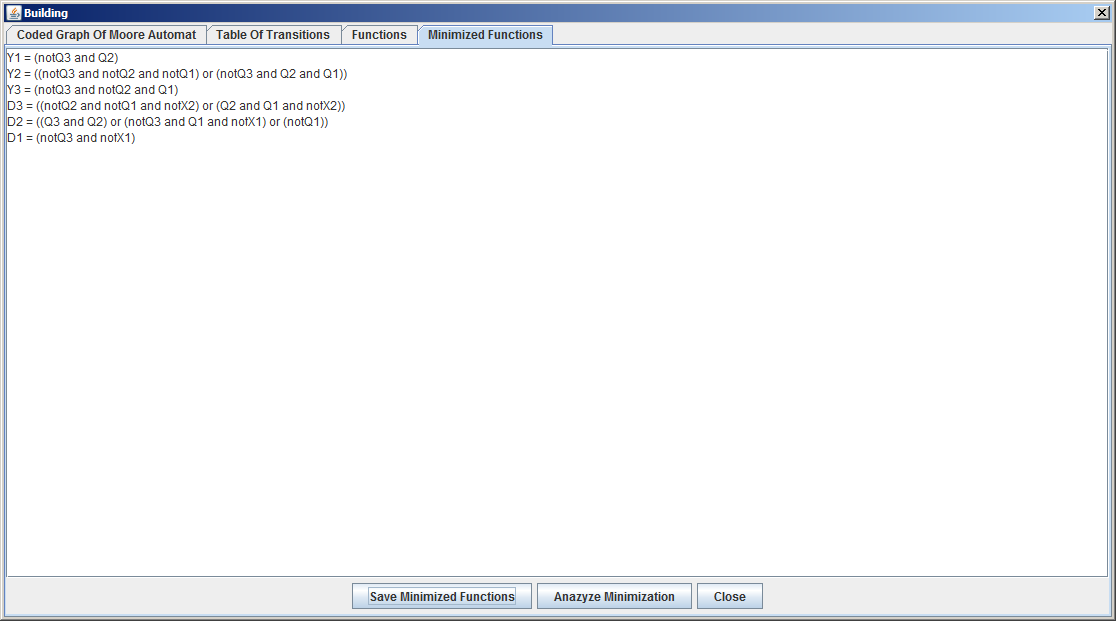


Рисунок 3

Для збереження мінімізованих функцій необхідно натиснути кнопку «Save Functions» та в діалоговому вікні вибрати необхідний файл. Результат показаний на рисунку 4.

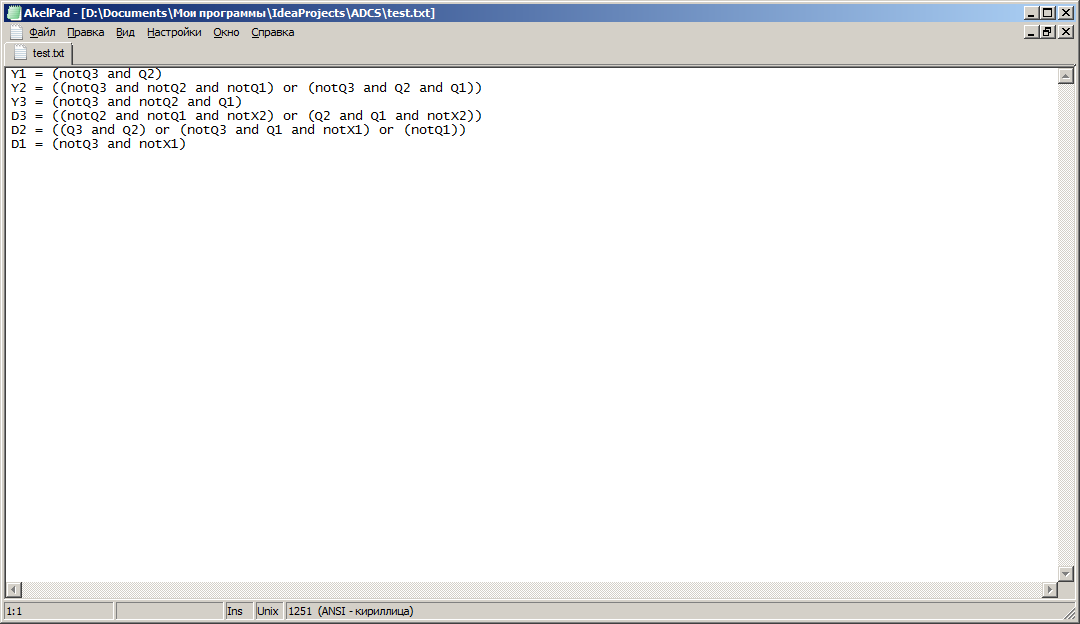


Рисунок 4

Для перегляду параметрів ефективності мінімізації функцій необхідно натиснути кнопку «Analyze Minimization» (Рис. 5).

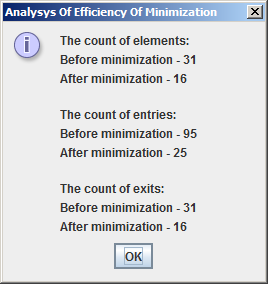


Рисунок 5

**Лістинг програми**

package automat.functions;

import java.util.ArrayList;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 16.11.2010

\* Time: 19:20:26

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

\*/

public class Function {

private String name;

private ArrayList<Implicant> implicants;

private int boolFunction;

public Function(String name, ArrayList<Implicant> implicants, int boolFunction) {

this.name = name;

this.implicants = implicants;

this.boolFunction = boolFunction;

}

public String getName() {

return name;

}

public ArrayList<Implicant> getImplicants() {

return implicants;

}

public int getBoolFunction() {

return boolFunction;

}

public String toString() {

StringBuilder builder = new StringBuilder();

builder.append(name);

builder.append(" = ");

String boolFunctionString;

if (boolFunction <= 2) {

boolFunctionString = BoolFunction.getBoolFunctionString(boolFunction);

} else {

builder.append(BoolFunction.getBoolFunctionString(4));

boolFunctionString = BoolFunction.getBoolFunctionString(boolFunction - 2);

}

if (implicants.size() > 1) {

builder.append("(");

}

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

builder.append(implicants.get(i).toString());

builder.append(" ");

builder.append(boolFunctionString);

builder.append(" ");

}

builder.append(implicants.get(implicants.size() - 1).toString());

if (implicants.size() > 1) {

builder.append(")");

}

return builder.toString();

}

}

package automat.functions;

import java.util.ArrayList;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 16.11.2010

\* Time: 19:29:13

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

\*/

class Implicant implements Cloneable {

private ArrayList<String> names;

private ArrayList<Boolean> values;

private int boolFunction;

public Implicant(ArrayList<String> names, ArrayList<Boolean> values, int boolFunction) {

this.names = names;

this.values = values;

this.boolFunction = boolFunction;

}

public ArrayList<String> getNames() {

return names;

}

public ArrayList<Boolean> getValues() {

return values;

}

public int getBoolFunction() {

return boolFunction;

}

public void setNames(ArrayList<String> names) {

this.names = names;

}

public void setValues(ArrayList<Boolean> values) {

this.values = values;

}

public String toString() {

StringBuilder builder = new StringBuilder();

String boolFunctionString;

if (boolFunction <= 2) {

boolFunctionString = BoolFunction.getBoolFunctionString(boolFunction);

} else {

builder.append(BoolFunction.getBoolFunctionString(4));

boolFunctionString = BoolFunction.getBoolFunctionString(boolFunction - 2);

}

builder.append("(");

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

if (!values.get(i)) {

builder.append(BoolFunction.getBoolFunctionString(4));

}

builder.append(names.get(i));

builder.append(" ");

builder.append(boolFunctionString);

builder.append(" ");

}

if (!values.get(names.size() - 1)) {

builder.append(BoolFunction.getBoolFunctionString(4));

}

builder.append(names.get(names.size() - 1));

builder.append(")");

return builder.toString();

}

public Implicant clone() {

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

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

cloneNames.add(new String(names.get(i)));

}

ArrayList<Boolean> cloneValues = new ArrayList<Boolean>();

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

if (values.get(i) != null) {

cloneValues.add(new Boolean(values.get(i)));

}

else {

cloneValues.add(null);

}

}

return new Implicant(cloneNames, cloneValues, boolFunction);

}

}

package automat.functions;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 16.11.2010

\* Time: 19:34:41

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

\*/

class BoolFunction {

private static final String AND = "and";

private static final String OR = "or";

private static final String NAND = "nand";

private static final String NOR = "nor";

private static final String NOT = "not";

public static String getBoolFunctionString(int f) {

String result = "";

switch (f) {

case 0: {

result = AND;

break;

}

case 1: {

result = OR;

break;

}

case 2: {

result = NAND;

break;

}

case 3: {

result = NOR;

break;

}

case 4: {

result = NOT;

break;

}

}

return result;

}

}

package automat.functions;

import automat.moore.AutomatTableModel;

import java.util.ArrayList;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 17.11.2010

\* Time: 1:11:19

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

\*/

public class FunctionsWorker {

public static ArrayList<Function> getFunctions(AutomatTableModel tableModel) {

ArrayList<Function> functions = new ArrayList<Function>();

String[][] table = tableModel.getTable();

int index1 = tableModel.getyStartIndex();

for (int i = 0; i < tableModel.getyCount(); i++) {

ArrayList<Implicant> implicants = new ArrayList<Implicant>();

for (int j = 1; j < table.length; j++) {

if (table[j][index1].compareTo("1") == 0) {

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

ArrayList<Boolean> values = new ArrayList<Boolean>();

int index2 = tableModel.getqStartIndex();

for (int k = 0; k < tableModel.getqCount(); k++) {

names.add(table[0][index2].substring(0, table[0][index2].length() - 3));

if (table[j][index2].compareTo("0") == 0) {

values.add(false);

}

else {

values.add(true);

}

index2++;

}

index2 = tableModel.getxStartIndex();

for (int k = 0; k < tableModel.getxCount(); k++) {

if (table[j][index2].compareTo("-") != 0) {

names.add(table[0][index2]);

if (table[j][index2].compareTo("0") == 0) {

values.add(false);

} else {

values.add(true);

}

}

index2++;

}

implicants.add(new Implicant(names, values, 0));

}

}

functions.add(new Function(table[0][index1], implicants, 1));

index1++;

}

index1 = tableModel.getdStartIndex();

for (int i = 0; i < tableModel.getdCount(); i++) {

ArrayList<Implicant> implicants = new ArrayList<Implicant>();

for (int j = 1; j < table.length; j++) {

if (table[j][index1].compareTo("1") == 0) {

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

ArrayList<Boolean> values = new ArrayList<Boolean>();

int index2 = tableModel.getqStartIndex();

for (int k = 0; k < tableModel.getqCount(); k++) {

names.add(table[0][index2].substring(0, table[0][index2].length() - 3));

if (table[j][index2].compareTo("0") == 0) {

values.add(false);

}

else {

values.add(true);

}

index2++;

}

index2 = tableModel.getxStartIndex();

for (int k = 0; k < tableModel.getxCount(); k++) {

if (table[j][index2].compareTo("-") != 0) {

names.add(table[0][index2]);

if (table[j][index2].compareTo("0") == 0) {

values.add(false);

} else {

values.add(true);

}

}

index2++;

}

implicants.add(new Implicant(names, values, 0));

}

}

functions.add(new Function(table[0][index1], implicants, 1));

index1++;

}

return functions;

}

public static ArrayList<Function> prepareFunctionsToMinimization(ArrayList<Function> functions) {

ArrayList<Function> newFunctions = new ArrayList<Function>();

for (Function f : functions) {

ArrayList<Implicant> implicants = f.getImplicants();

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

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

ArrayList<String> names = implicants.get(i).getNames();

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

boolean contains = false;

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

if (allNames.get(k).compareTo(names.get(j)) == 0) {

contains = true;

}

}

if (!contains) {

allNames.add(new String(names.get(j)));

}

}

}

ArrayList<Implicant> newImplicants = new ArrayList<Implicant>();

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

ArrayList<String> names = implicants.get(i).getNames();

ArrayList<Boolean> values = implicants.get(i).getValues();

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

ArrayList<Boolean> newValues = new ArrayList<Boolean>();

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

newNames.add(new String(allNames.get(j)));

newValues.add(null);

}

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

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

if (names.get(j).compareTo(newNames.get(k)) == 0) {

newValues.set(k, new Boolean(values.get(j)));

}

}

}

newImplicants.add(new Implicant(newNames, newValues, implicants.get(i).getBoolFunction()));

}

newFunctions.add(new Function(new String(f.getName()), newImplicants, f.getBoolFunction()));

}

return newFunctions;

}

public static ArrayList<Function> minimizeFunctions(ArrayList<Function> functions) {

ArrayList<Function> minimizedFunctions = new ArrayList<Function>();

for (Function f : functions) {

ArrayList<ArrayList<Implicant>> implicants = new ArrayList<ArrayList<Implicant>>();

ArrayList<ArrayList<Boolean>> isCovered = new ArrayList<ArrayList<Boolean>>();

ArrayList<Implicant> originalImplicats = f.getImplicants();

ArrayList<Implicant> startImplicants = new ArrayList<Implicant>();

ArrayList<Boolean> startIsCovered = new ArrayList<Boolean>();

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

startImplicants.add(originalImplicats.get(i).clone());

startIsCovered.add(false);

}

implicants.add(startImplicants);

isCovered.add(startIsCovered);

boolean flag = false;

while (!flag) {

ArrayList<Implicant> coveringImplicants = implicants.get(implicants.size() - 1);

ArrayList<Boolean> coveringIsCover = isCovered.get(isCovered.size() - 1);

ArrayList<Implicant> coverImplicants = new ArrayList<Implicant>();

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

if (!coveringIsCover.get(i)) {

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

int difference = 0;

int differenceIndex = -1;

ArrayList<String> names1 = coveringImplicants.get(i).getNames();

ArrayList<Boolean> values1 = coveringImplicants.get(i).getValues();

ArrayList<String> names2 = coveringImplicants.get(j).getNames();

ArrayList<Boolean> values2 = coveringImplicants.get(j).getValues();

for (int k = 0; k < names1.size(); k++) {

if (names1.get(k).compareTo(names2.get(k)) != 0) {

difference++;

differenceIndex = k;

}

else {

if ((values1.get(k) != null) && (values2.get(k) != null) &&

(values1.get(k).compareTo(values2.get(k)) != 0)) {

difference++;

differenceIndex = k;

}

}

}

if (difference < 2) {

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

ArrayList<Boolean> newValues = new ArrayList<Boolean>();

for (int k = 0; k < names1.size(); k++) {

if (k != differenceIndex) {

newNames.add(new String(names1.get(k)));

if (values1.get(k) != null) {

newValues.add(new Boolean(values1.get(k)));

}

else {

newValues.add(null);

}

}

}

coverImplicants.add(new Implicant(newNames, newValues,

coveringImplicants.get(i).getBoolFunction()));

coveringIsCover.set(i, true);

coveringIsCover.set(j, true);

}

}

}

}

implicants.add(coverImplicants);

isCovered.add(new ArrayList<Boolean>());

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

isCovered.get(isCovered.size() - 1).add(false);

}

flag = true;

for (int i = 0; i < isCovered.get(isCovered.size() - 2).size(); i++) {

if (isCovered.get(isCovered.size() - 2).get(i)) {

flag = false;

}

}

}

ArrayList<Implicant> minimizedImplicants = new ArrayList<Implicant>();

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

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

if (!isCovered.get(i).get(j)) {

ArrayList<String> names = implicants.get(i).get(j).getNames();

ArrayList<Boolean> values = implicants.get(i).get(j).getValues();

boolean hasNull = true;

while (hasNull) {

hasNull = false;

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

if (values.get(k) == null) {

hasNull = true;

names.remove(k);

values.remove(k);

}

}

}

implicants.get(i).get(j).setNames(names);

implicants.get(i).get(j).setValues(values);

minimizedImplicants.add(implicants.get(i).get(j));

}

}

}

minimizedFunctions.add(new Function(new String(f.getName()), minimizedImplicants, f.getBoolFunction()));

}

return minimizedFunctions;

}

public static MinimizationEfficiencyObject analyzeMinimization(ArrayList<Function> functions,

ArrayList<Function> minimizedFunctions) {

int elementCount1 = 0;

int elementCount2 = 0;

int entryCount1 = 0;

int entryCount2 = 0;

int exitCount1 = 0;

int exitCount2 = 0;

for (Function f : functions) {

elementCount1++;

exitCount1++;

for (Implicant i : f.getImplicants()) {

elementCount1++;

entryCount1 += i.getNames().size();

exitCount1++;

}

}

for (Function f : minimizedFunctions) {

elementCount2++;

exitCount2++;

for (Implicant i : f.getImplicants()) {

elementCount2++;

entryCount2 += i.getNames().size();

exitCount2++;

}

}

return new MinimizationEfficiencyObject(elementCount1, elementCount2, entryCount1, entryCount2, exitCount1,

exitCount2);

}

}

package automat.functions;

/\*\*

\* Created by IntelliJ IDEA.

\* User: Zak

\* Date: 18.11.2010

\* Time: 3:44:35

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

\*/

public class MinimizationEfficiencyObject {

private int elementCount1;

private int elementCount2;

private int entryCount1;

private int entryCount2;

private int exitCount1;

private int exitCount2;

private int wayLength1;

private int wayLength2;

public MinimizationEfficiencyObject(int elementCount1, int elementCount2, int entryCount1, int entryCount2,

int exitCount1, int exitCount2) {

this.elementCount1 = elementCount1;

this.elementCount2 = elementCount2;

this.entryCount1 = entryCount1;

this.entryCount2 = entryCount2;

this.exitCount1 = exitCount1;

this.exitCount2 = exitCount2;

}

public String toString() {

StringBuilder builder = new StringBuilder();

builder.append("The count of elements:\n");

builder.append("Before minimization - ");

builder.append(String.valueOf(elementCount1));

builder.append("\nAfter minimization - ");

builder.append(String.valueOf(elementCount2));

builder.append("\n\nThe count of entries:\n");

builder.append("Before minimization - ");

builder.append(String.valueOf(entryCount1));

builder.append("\nAfter minimization - ");

builder.append(String.valueOf(entryCount2));

builder.append("\n\nThe count of exits:\n");

builder.append("Before minimization - ");

builder.append(String.valueOf(exitCount1));

builder.append("\nAfter minimization - ");

builder.append(String.valueOf(exitCount2));

return builder.toString();

}

}

package face;

import automat.functions.Function;

import automat.functions.FunctionsWorker;

import automat.moore.\*;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.ArrayList;

/\*\*

\* 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;

private String functionsString;

private ArrayList<Function> functions;

private JButton buildFunctionsButton;

private ArrayList<Function> minimizedFunctions;

private String minimizedFunctionsString;

private JButton minimizeFunctionsButton;

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");

buildFunctionsButton = new JButton(new BuildFunctionsAction(frame));

buildFunctionsButton.setText("Build Functions");

JButton closeButton = new JButton(new AbstractAction() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

closeButton.setText("Close");

tableButtonsPanel.add(saveTableButton);

tableButtonsPanel.add(buildFunctionsButton);

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);

}

}

private class SaveFunctionsAction extends AbstractAction {

private BuildFrame frame;

public SaveFunctionsAction(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 {

PrintWriter output = new PrintWriter(new FileWriter(chooser.getSelectedFile()));

output.print(functionsString);

output.close();

} catch (IOException e1) {

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

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

}

private class BuildFunctionsAction extends AbstractAction {

private BuildFrame frame;

public BuildFunctionsAction(BuildFrame frame) {

this.frame = frame;

}

public void actionPerformed(ActionEvent e) {

JPanel functionsPanel = new JPanel();

functionsPanel.setLayout(new BorderLayout());

JTextArea functionsArea = new JTextArea();

functionsArea.setEditable(false);

functions = FunctionsWorker.getFunctions(tableModel);

StringBuilder builder = new StringBuilder();

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

builder.append(functions.get(i).toString());

builder.append("\n");

}

functionsString = builder.toString();

functionsArea.setText(functionsString);

JPanel functionsButtonPanel = new JPanel();

JButton saveFunctionsButton = new JButton(new SaveFunctionsAction(frame));

saveFunctionsButton.setText("Save Functions");

minimizeFunctionsButton = new JButton(new MinimizeFunctionsAction(frame));

minimizeFunctionsButton.setText("Minimize Functions");

JButton closeButton = new JButton(new AbstractAction() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

closeButton.setText("Close");

functionsButtonPanel.add(saveFunctionsButton);

functionsButtonPanel.add(minimizeFunctionsButton);

functionsButtonPanel.add(closeButton);

functionsPanel.add(functionsButtonPanel, BorderLayout.SOUTH);

functionsPanel.add(new JScrollPane(functionsArea));

tabbedPane.addTab("Functions", functionsPanel);

tabbedPane.setSelectedIndex(tabbedPane.getTabCount() - 1);

buildFunctionsButton.setEnabled(false);

}

}

private class SaveMinimizedFunctionsAction extends AbstractAction {

private BuildFrame frame;

public SaveMinimizedFunctionsAction(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 {

PrintWriter output = new PrintWriter(new FileWriter(chooser.getSelectedFile()));

output.print(minimizedFunctionsString);

output.close();

} catch (IOException e1) {

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

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

}

private class MinimizeFunctionsAction extends AbstractAction {

private BuildFrame frame;

public MinimizeFunctionsAction(BuildFrame frame) {

this.frame = frame;

}

public void actionPerformed(ActionEvent e) {

JPanel minimizedFunctionsPanel = new JPanel();

minimizedFunctionsPanel.setLayout(new BorderLayout());

JTextArea minimizedFunctionsArea = new JTextArea();

minimizedFunctionsArea.setEditable(false);

minimizedFunctions = FunctionsWorker.minimizeFunctions(FunctionsWorker.prepareFunctionsToMinimization(functions));

StringBuilder builder = new StringBuilder();

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

builder.append(minimizedFunctions.get(i).toString());

builder.append("\n");

}

minimizedFunctionsString = builder.toString();

minimizedFunctionsArea.setText(minimizedFunctionsString);

JPanel minimizedFunctionsButtonPanel = new JPanel();

JButton saveMinimizedFunctionsButton = new JButton(new SaveMinimizedFunctionsAction(frame));

saveMinimizedFunctionsButton.setText("Save Minimized Functions");

JButton analyzeButton = new JButton(new AbstractAction() {

public void actionPerformed(ActionEvent e) {

JOptionPane.showMessageDialog(frame, FunctionsWorker.analyzeMinimization(functions, minimizedFunctions).toString(),

"Analysys Of Efficiency Of Minimization", JOptionPane.INFORMATION\_MESSAGE);

}

});

analyzeButton.setText("Anazyze Minimization");

JButton closeButton = new JButton(new AbstractAction() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

closeButton.setText("Close");

minimizedFunctionsButtonPanel.add(saveMinimizedFunctionsButton);

minimizedFunctionsButtonPanel.add(analyzeButton);

minimizedFunctionsButtonPanel.add(closeButton);

minimizedFunctionsPanel.add(minimizedFunctionsButtonPanel, BorderLayout.SOUTH);

minimizedFunctionsPanel.add(new JScrollPane(minimizedFunctionsArea));

tabbedPane.addTab("Minimized Functions", minimizedFunctionsPanel);

tabbedPane.setSelectedIndex(tabbedPane.getTabCount() - 1);

minimizeFunctionsButton.setEnabled(false);

}

}

}

**Висновки**

У результаті виконання даної лабораторної роботи я здобув навички автоматизації процедури мінімізації булевих функцій методом Квайна-МакКласкі. Мною були реалізовані процедури побудови функцій переходів та функцій збудження тригерів, мінімізації цих функцій, аналізу ефективності їх мінімізації по різним параметрам, а також була реалізована можливість збереження результатів у текстовому файлі. Всі процедури були реалізовані на мові програмування Java.