Міністерство освіти та науки України

Національний технічний університет України «КПІ»

Факультет інформатики та обчислювальної техніки

**Автоматизація проектування комп'ютерних систем**

Лабораторна робота №7

Автоматизація генерації аналітичних форм булевих функцій з табличної форми

Виконав

ст. гр. ІО-71

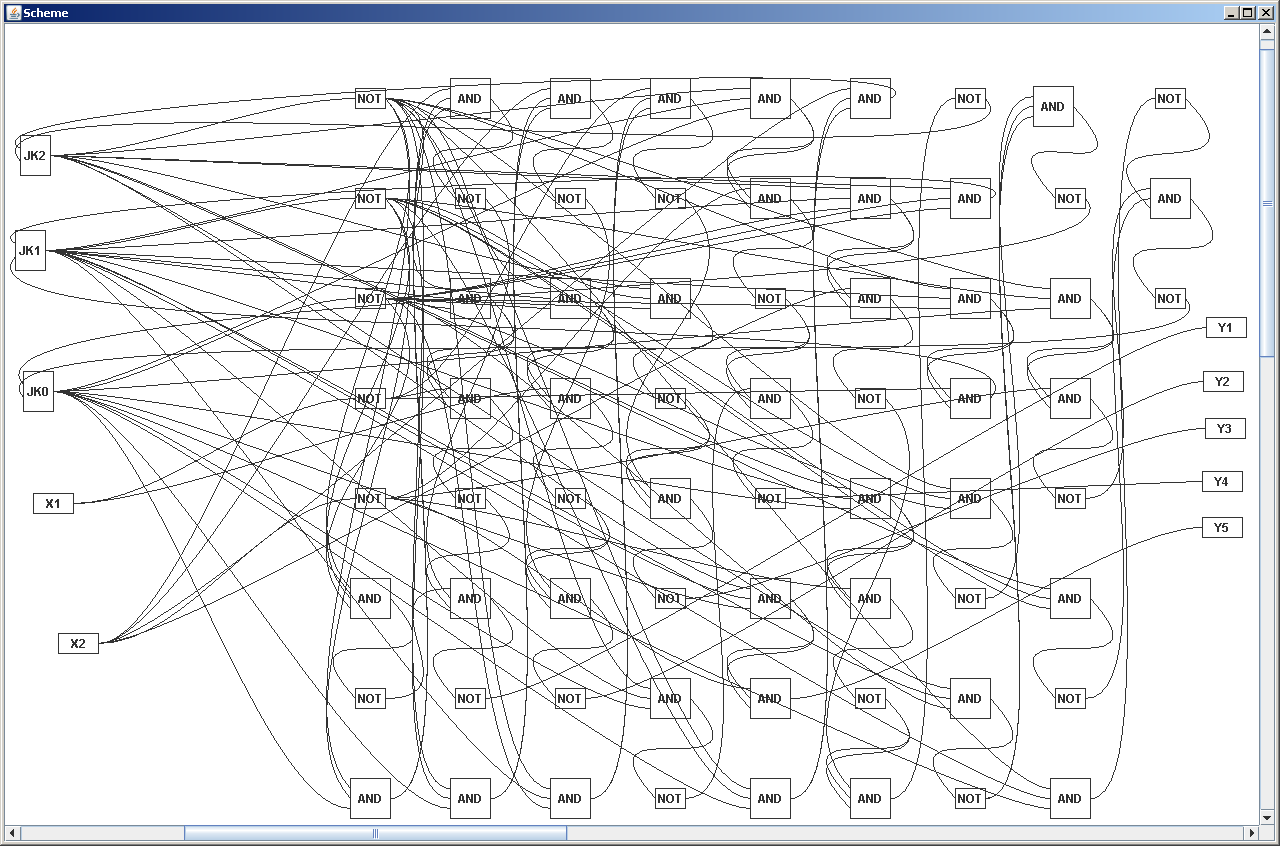
Ювженко Д.

**Київ 2011**

Мета: здобуття навичок автоматизації перетворення представлення булевих функцій з табличної до аналітичної форми для заданого елементного базису.

Елементний базис: NOT,3AND

Скріншот програми:



Лістинг програми:

package Scheme;

import java.util.ArrayList;

import java.util.Arrays;

import Scheme.Elements.Connector;

import Scheme.Elements.Element;

import Scheme.Elements.JKTrigger;

import Scheme.Elements.NOT;

import Scheme.Elements.OutputVariable;

import Scheme.Elements.ThreeAND;

import Scheme.Elements.InputVariable;

public class Generator {

private ArrayList<Element> SchemeElements;

private ArrayList<Connector> InputConnectors;

private ArrayList<Connector> OutputConnectors;

public ArrayList<Element> createScheme(String[] InputSignals, String[] ColumnNames, String[][] TableData) {

System.out.println(Arrays.toString(InputSignals));

SchemeElements = new ArrayList<Element>();

InputConnectors = new ArrayList<Connector>();

OutputConnectors = new ArrayList<Connector>();

createInputConnectors(InputSignals);

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

if (ColumnNames[i].contains("m")) {

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

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

if (TableData[j][i]!=null)

ConstituentsList.add(TableData[j][i]);

String[] ConstituentsArray = new String[ConstituentsList.size()];

ConstituentsArray = ConstituentsList.toArray(ConstituentsArray);

convertFunctionToScheme(ColumnNames[i].substring(1), InputSignals, ConstituentsArray);

}

}

return SchemeElements;

}

private void createInputConnectors(String[] InputSignals) {

InputVariable CurrentVariable;

NOT CurrentNOT;

Connector CurrentPositiveConnector;

Connector CurrentNegativeConnector;

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

if (InputSignals[i].contains("Q")) {

CurrentPositiveConnector = createTrigger(InputSignals[i]);

} else {

CurrentVariable = new InputVariable();

CurrentPositiveConnector = new Connector(InputSignals[i], CurrentVariable);

CurrentVariable.setOutput(CurrentPositiveConnector);

SchemeElements.add(CurrentVariable);

}

CurrentNOT = new NOT(CurrentPositiveConnector);

CurrentNegativeConnector = new Connector("!"+InputSignals[i], CurrentNOT);

SchemeElements.add(CurrentNOT);

InputConnectors.add(CurrentPositiveConnector);

InputConnectors.add(CurrentNegativeConnector);

}

}

private Connector createTrigger(String OutputSignalName) {

JKTrigger Trigger;

Connector Q;

Trigger = new JKTrigger();

Q = new Connector(OutputSignalName, Trigger);

Trigger.setQ(Q);

SchemeElements.add(Trigger);

InputConnectors.add(Q);

return Q;

}

private Connector getInputConnectorByName(String Name) {

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

if (InputConnectors.get(i).getName().equals(Name)) {

return InputConnectors.get(i);

}

}

return null;

}

private JKTrigger getTriggerByQ(Connector Q) {

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

if (SchemeElements.get(i).getClass().equals(JKTrigger.class) && ((JKTrigger) SchemeElements.get(i)).getQ().equals(Q)) {

return (JKTrigger) SchemeElements.get(i);

}

}

return null;

}

@SuppressWarnings("unchecked")

private ArrayList<Connector>[] parseConstituents(String[] InputSignals, String[] Constituents) {

ArrayList<Connector>[] Result = new ArrayList[Constituents.length];

ArrayList<Connector> ParsedConstituent;

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

ParsedConstituent = new ArrayList<Connector>();

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

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

ParsedConstituent.add(getInputConnectorByName("!"+InputSignals[j]));

} else if (Constituents[i].charAt(j) == '1') {

ParsedConstituent.add(getInputConnectorByName(InputSignals[j]));

}

}

Result[i] = ParsedConstituent;

}

return Result;

}

private void convertFunctionToScheme(String FunctionName, String[] InputSignals, String[] Constituents) {

ArrayList<Connector>[] ParsedConstituents = this.parseConstituents(InputSignals, Constituents);

ArrayList<Connector> FinalConstituent = new ArrayList<Connector>();

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

FinalConstituent.add(convertConstituentToScheme(null, ParsedConstituents[i], ParsedConstituents.length));

}

Connector FunctionOutput = convertConstituentToScheme(FunctionName, FinalConstituent, ParsedConstituents.length);

FunctionOutput.setName(FunctionName);

OutputConnectors.add(FunctionOutput);

if (FunctionName.contains("J") || FunctionName.contains("K")) {

JKTrigger CurrentTrigger = getTriggerByQ(getInputConnectorByName("Q"+FunctionName.substring(1)));

if (FunctionName.contains("J")) {

CurrentTrigger.setJ(FunctionOutput);

} else {

CurrentTrigger.setK(FunctionOutput);

}

} else {

OutputVariable FunctionOut = new OutputVariable();

FunctionOut.setInput(FunctionOutput);

SchemeElements.add(FunctionOut);

}

}

private Connector convertConstituentToScheme(String OutputName, ArrayList<Connector> Constituent, int ConstituentsQuantityInFunction) {

ArrayList<Connector> CurrentElementInputs = new ArrayList<Connector>();

Connector CurrentOutput;

Element CurrentElement;

while (Constituent.size() > 1) {

CurrentElementInputs.clear();

while (CurrentElementInputs.size()<3) {

if (Constituent.size()>0) {

CurrentElementInputs.add(Constituent.remove(0));

} else {

CurrentElementInputs.add(CurrentElementInputs.get(CurrentElementInputs.size()-1));

}

}

CurrentElement = new ThreeAND(CurrentElementInputs.get(0), CurrentElementInputs.get(1), CurrentElementInputs.get(2));

CurrentOutput = new Connector(null, CurrentElement);

SchemeElements.add(CurrentElement);

Constituent.add(CurrentOutput);

}

if (ConstituentsQuantityInFunction>1) {

CurrentElement = new NOT(Constituent.remove(0));

CurrentOutput = new Connector(OutputName, CurrentElement);

SchemeElements.add(CurrentElement);

Constituent.add(CurrentOutput);

}

return Constituent.remove(0);

}

}

**package** Scheme;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.Font;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** java.awt.event.MouseEvent;

**import** java.awt.event.MouseListener;

**import** java.awt.event.MouseMotionListener;

**import** java.awt.geom.Point2D;

**import** java.awt.geom.Rectangle2D;

**import** java.awt.image.BufferedImage;

**import** java.util.ArrayList;

**import** javax.swing.JPanel;

**import** lab1\_8.MainWindow;

**import** Scheme.Elements.Element;

**import** Scheme.Elements.InputVariable;

**import** Scheme.Elements.JKTrigger;

**import** Scheme.Elements.OutputVariable;

**public** **class** SchemePanel **extends** JPanel **implements** MouseListener, MouseMotionListener {

**private** **static** **final** **long** *serialVersionUID* = -4763106806489158220L;

**private** MainWindow Owner;

**private** ArrayList<Element> SchemeElements;

**private** ArrayList<Element> ActiveElements = **new** ArrayList<Element>();

**private** Point2D PreviousDraggingPoint;

**private** Point2D BeginSelectionPoint;

**private** Rectangle2D SelectionRectangle;

**private** **boolean** isSelecting;

**private** **boolean** isDragging;

**public** SchemePanel(MainWindow Owner) {

**this**.Owner = Owner;

}

**public** **void** setElements(ArrayList<Element> SchemeElements) {

**this**.SchemeElements = SchemeElements;

**this**.setBackground(Color.*WHITE*);

**this**.addMouseListener(**this**);

**this**.addMouseMotionListener(**this**);

initElementsPosition();

}

**private** **void** initElementsPosition() {

**int** GridStep = 100;

**int** i = 0;

**for** (Element e : SchemeElements) {

**if** (e.getClass().equals(InputVariable.**class**)

|| e.getClass().equals(JKTrigger.**class**)) {

e.setLocation(5\*GridStep, GridStep+GridStep\*(i++));

}

}

i = 0;

**for** (Element e : SchemeElements) {

**if** (!e.getClass().equals(InputVariable.**class**)

&& !e.getClass().equals(JKTrigger.**class**)

&& !e.getClass().equals(OutputVariable.**class**)) {

e.setLocation(9\*GridStep+GridStep\*(i/8), GridStep+GridStep\*((i++) % 8));

}

}

**int** j = 0;

**for** (Element e : SchemeElements) {

**if** (e.getClass().equals(OutputVariable.**class**)) {

e.setLocation((12+i/8)\*GridStep, GridStep+GridStep\*(j++));

}

}

}

@Override

**public** **void** paint(Graphics g) {

Graphics2D g2 = (Graphics2D) g;

**super**.paint(g2);

g2.setFont(**new** Font(g2.getFont().getName(), Font.*BOLD*, g2.getFont().getSize()));

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

**if** (ActiveElements.contains(SchemeElements.get(i))) {

g2.setColor(Color.*BLUE*);

SchemeElements.get(i).paint(g2);

g2.setColor(Color.*BLACK*);

}

**else** {

SchemeElements.get(i).paint(g2);

}

}

**if** (SelectionRectangle!=**null**) {

g2.draw(SelectionRectangle);

}

**this**.setPreferredSize(**new** Dimension(4000, 2000));

**this**.setSize(**this**.getPreferredSize());

}

**public** **void** mouseClicked(MouseEvent e) {

**if** (e.getButton() == MouseEvent.*BUTTON1* && e.getClickCount() == 1) {

ActiveElements.clear();

**for** (Element E : SchemeElements) {

**if** (E.isContainsPoint(e.getPoint())) {

ActiveElements.add(E);

**break**;

}

}

**this**.repaint();

**return**;

}

**if** (e.getButton() == MouseEvent.*BUTTON1* && e.getClickCount() == 2) {

BufferedImage Image = **new** BufferedImage(

(**int**) **this**.getPreferredSize().getWidth(),

(**int**) **this**.getPreferredSize().getHeight(),

BufferedImage.*TYPE\_INT\_RGB*);

Graphics2D g2 = Image.createGraphics();

**this**.paint(g2);

Owner.saveImage(Image);

}

}

**public** **void** mouseEntered(MouseEvent e) {}

**public** **void** mouseExited(MouseEvent e) {}

**public** **void** mousePressed(MouseEvent e) {

**if** (e.getButton() == MouseEvent.*BUTTON1*) {

Element PressedElment = **null**;

**for** (Element E : SchemeElements) {

**if** (E.isContainsPoint(e.getPoint())) {

PressedElment = E;

**break**;

}

}

**if** (PressedElment == **null**) {

ActiveElements.clear();

isSelecting = **true**;

} **else** {

**if** (!ActiveElements.contains(PressedElment)) {

ActiveElements.clear();

ActiveElements.add(PressedElment);

**this**.repaint();

}

PreviousDraggingPoint = e.getPoint();

isDragging = **true**;

}

}

}

**public** **void** mouseReleased(MouseEvent e) {

PreviousDraggingPoint = **null**;

BeginSelectionPoint = **null**;

SelectionRectangle = **null**;

isSelecting = **false**;

isDragging = **false**;

**this**.repaint();

}

**public** **void** mouseDragged(MouseEvent e) {

**if** (isSelecting) {

**if** (BeginSelectionPoint==**null**) {

BeginSelectionPoint = e.getPoint();

} **else** {

SelectionRectangle = **new** Rectangle2D.Double(

Math.*min*(BeginSelectionPoint.getX(), e.getX()),

Math.*min*(BeginSelectionPoint.getY(), e.getY()),

Math.*abs*(e.getX()-BeginSelectionPoint.getX()),

Math.*abs*(e.getY()-BeginSelectionPoint.getY()));

ActiveElements.clear();

**for** (Element E : SchemeElements) {

**if** (SelectionRectangle.contains(E.getShapeFrame())) {

ActiveElements.add(E);

}

}

**this**.repaint();

**return**;

}

}

**if** (isDragging) {

**for** (Element E : ActiveElements) {

E.changeLocation(

e.getX()-PreviousDraggingPoint.getX(),

e.getY()-PreviousDraggingPoint.getY());

}

PreviousDraggingPoint.setLocation(

e.getX(),

e.getY());

**this**.repaint();

}

}

**public** **void** mouseMoved(MouseEvent e) {}

}

Висновок: під час виконання даної лабораторної роботи були здобуті навички автоматизації перетворення представлення булевих функцій з табличної до аналітичної форми для заданого елементного базису NOT,3AND. Також були реалізовані засоби збереження результатів у графічний файл.