package main.CourseWork;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Point;

import java.awt.Rectangle;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.awt.event.MouseMotionListener;

import java.util.ArrayList;

import javax.swing.JComponent;

**public class Component extends JComponent implements MouseMotionListener,**

MouseListener {

protected DrawPanel drawPanel;

protected int numOfInputs;

protected boolean not;

protected boolean or;

protected final int w = 75;

protected int h;

protected final int d = 3;

protected final int dx = w / 4;

protected final int pinH = 20;

protected ArrayList<Link> output = new ArrayList();

protected Link[] input;

protected Point p;

protected static Component component;

protected int index = -2;

public Component(DrawPanel drawPanel, int numOfInputs, boolean not, boolean or) {

this.drawPanel = drawPanel;

this.numOfInputs = numOfInputs;

this.not = not;

this.or = or;

input = new Link[numOfInputs];

addMouseMotionListener(this);

addMouseListener(this);

h = (numOfInputs + 1) \* pinH;

setBounds(0, h, w, h);

}

@Override

public void paintComponent(Graphics g) {

g.setColor(Color.WHITE);

g.fillRect(dx, 0, w - dx \* 2, h - d);

g.setColor(Color.BLACK);

g.drawRect(dx, 0, w - dx \* 2, h - d);

String text = "";

if (not) {

text = "N";

}

if (or) {

text += "OR";

} else {

text += "AND";

}

g.drawString(text, dx + d, pinH);

if (index == -1) {

g.setColor(Color.red);

}

g.drawLine(w - dx, pinH, w, pinH);

if (index == -1) {

g.setColor(Color.black);

}

if (not) {

g.fillOval(w - dx - d \* 2, pinH - d \* 2, d \* 4, d \* 4);

}

int th = pinH;

for (int i = 0; i < numOfInputs; i++) {

if (index == i) {

g.setColor(Color.red);

}

g.drawLine(0, th, dx, th);

if (index == -1) {

g.setColor(Color.black);

}

th += pinH;

}

}

public int getX(Link l) {

for (Link link : output) {

if (link == l) {

return getX() + w;

}

}

return getX();

}

public int getY(Link l) {

for (Link link : output) {

if (l == link) {

return getY() + pinH;

}

}

for (int i = 0; i < numOfInputs; i++) {

if (input[i] == l) {

return getY() + pinH + i \* pinH;

}

}

return -1;

}

protected void removeLink(Link l) {

if (!output.remove(l)) {

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

if (input[i] == l) {

input[i] = null;

break;

}

}

}

}

@Override

public void mouseDragged(MouseEvent e) {

setLocation(getX() - (p.x - e.getLocationOnScreen().x), getY() - (p.y - e.getLocationOnScreen().y));

p = e.getLocationOnScreen();

for (Link l : output) {

l.repaint();

}

for (Link l : input) {

if (l != null) {

l.repaint();

}

}

}

@Override

public void mouseMoved(MouseEvent e) {

}

private int cont(Point p) {

if (new Rectangle(w - dx, pinH / 2, dx, pinH).contains(p)) {

return -1;

}

int th = pinH / 2;

for (int i = 0; i < numOfInputs; i++) {

if (new Rectangle(0, th, dx, pinH).contains(p)) {

return i;

}

th += pinH;

}

if (component == this) {

component = null;

}

return -2;

}

protected void setLink(Link l) {

if (index == -1) {

output.add(l);

} else if (index >= 0) {

input[index] = l;

}

index = -2;

repaint();

}

@Override

public void mouseClicked(MouseEvent e) {

if (e.getButton() == 3) {

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

output.get(i).removeLink();

}

for (int i = 0; i < numOfInputs; i++) {

if (input[i] != null) {

input[i].removeLink();

}

}

drawPanel.remove(this);

drawPanel.repaint();

return;

}

index = cont(e.getPoint());

if (index != -2) {

if (component == null) {

if (index == -1) {

component = this;

} else if (index >= 0) {

if (input[index] == null) {

component = this;

}

}

} else if (component != this) {

Link l;

if (index >= 0) {

l = new Link(drawPanel, component, this);

} else {

l = new Link(drawPanel, this, component);

}

component.setLink(l);

setLink(l);

drawPanel.add(l);

drawPanel.repaint();

component = null;

}

}

repaint();

}

@Override

public void mousePressed(MouseEvent e) {

p = e.getLocationOnScreen();

}

@Override

public void mouseReleased(MouseEvent e) {

}

@Override

public void mouseEntered(MouseEvent e) {

}

@Override

public void mouseExited(MouseEvent e) {

}

public boolean isConnected() {

if (output.isEmpty()) {

return false;

}

for (int i = 0; i < numOfInputs; i++) {

if (input[i] == null) {

return false;

}

}

return true;

}

protected boolean value;

public boolean isValue() {

if (or) {

value = false;

} else {

value = true;

}

for (Link l : input) {

if (l.getFrom().getX() < getX()) {

if (or) {

if (not) {

value = value | !l.isValue();

}else{

value = value | l.isValue();

}

} else {

if (not) {

value = value & !l.isValue();

}else{

value = value & l.isValue();

}

}

}

}

return value;

}

public void setValue(boolean value) {

this.value = value;

}

}

package main.CourseWork;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Polygon;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.awt.event.MouseMotionListener;

import java.util.PriorityQueue;

import java.util.Queue;

import javax.swing.JComponent;

**public class Link extends JComponent implements** MouseMotionListener,

MouseListener {

private static final Queue<Integer> ids = new PriorityQueue<>();

private Component from, to;

private DrawPanel drawPanel;

private boolean test;

private boolean value;

private final int index;

private static int pollID() {

if (ids.isEmpty()) {

ids.offer(0);

}

int result = ids.poll();

if (ids.isEmpty()) {

ids.add(result + 1);

}

return result;

}

public Link(DrawPanel drawPanel, Component from, Component to) {

setLocation(0, 0);

setSize(drawPanel.getSize());

addMouseListener(this);

addMouseMotionListener(this);

this.from = from;

this.to = to;

this.drawPanel = drawPanel;

index = pollID();

}

@Override

public void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g;

if (!selected) {

g2.setColor(Color.BLACK);

} else {

g2.setColor(Color.RED);

}

g2.drawLine(getX1Line(), getY1Line(), getX2Line(), getY2Line());

g2.drawString("L " + index, (getX1Line() + getX2Line()) / 2, (getY1Line() + getY2Line()) / 2);

}

@Override

public boolean contains(int x, int y) {

if (getX1Line() - DELTA < x && x < getX1Line() + DELTA && getY1Line() - DELTA < y && y < getY1Line() + DELTA) {

return true;

} else if (getX2Line() - DELTA < x && x < getX2Line() + DELTA && getY2Line() - DELTA < y && y < getY2Line() + DELTA) {

return true;

} else {

int minX = 0;

int maxX = 0;

int minY = 0;

int maxY = 0;

if (getX1Line() > getX2Line()) {

minX = getX2Line();

maxX = getX1Line();

} else {

minX = getX1Line();

maxX = getX2Line();

}

if (getY1Line() > getY2Line()) {

minY = getY2Line();

maxY = getY1Line();

} else {

minY = getY1Line();

maxY = getY2Line();

}

if (getX1Line() < getX2Line()) {

if (getY1Line() < getY2Line()) {

int[] xp = {minX - DELTA, minX - DELTA, minX + DELTA, maxX + DELTA, maxX + DELTA, maxX - DELTA};

int[] yp = {minY + DELTA, minY - DELTA, minY - DELTA, maxY - DELTA, maxY + DELTA, maxY + DELTA};

if (new Polygon(xp, yp, xp.length).contains(x, y)) {

return true;

}

} else {

int[] xp = {minX + DELTA, minX - DELTA, minX - DELTA, maxX - DELTA, maxX + DELTA, maxX + DELTA};

int[] yp = {maxY + DELTA, maxY + DELTA, maxY - DELTA, minY - DELTA, minY - DELTA, minY + DELTA};

if (new Polygon(xp, yp, xp.length).contains(x, y)) {

return true;

}

}

} else {

if (getY1Line() < getY2Line()) {

int[] xp = {minX + DELTA, minX - DELTA, minX - DELTA, maxX - DELTA, maxX + DELTA, maxX + DELTA};

int[] yp = {maxY + DELTA, maxY + DELTA, maxY - DELTA, minY - DELTA, minY - DELTA, minY + DELTA};

if (new Polygon(xp, yp, xp.length).contains(x, y)) {

return true;

}

} else {

int[] xp = {minX - DELTA, minX - DELTA, minX + DELTA, maxX + DELTA, maxX + DELTA, maxX - DELTA};

int[] yp = {minY + DELTA, minY - DELTA, minY - DELTA, maxY - DELTA, maxY + DELTA, maxY + DELTA};

if (new Polygon(xp, yp, xp.length).contains(x, y)) {

return true;

}

}

}

return false;

}

}

@Override

public void mouseClicked(MouseEvent me) {

if (me.getButton() == 3) {

removeLink();

}

}

@Override

public void mousePressed(MouseEvent me) {

}

@Override

public void mouseReleased(MouseEvent me) {

}

@Override

public void mouseEntered(MouseEvent me) {

selected = true;

repaint();

}

@Override

public void mouseExited(MouseEvent me) {

selected = false;

repaint();

}

@Override

public void mouseDragged(MouseEvent me) {

}

@Override

public void mouseMoved(MouseEvent me) {

}

private final int DELTA = 10;

private boolean selected;

private int getY1Line() {

return from.getY(this);

}

private int getY2Line() {

return to.getY(this);

}

private int getX1Line() {

return from.getX(this);

}

private int getX2Line() {

return to.getX(this);

}

public void removeLink() {

from.removeLink(this);

to.removeLink(this);

ids.offer(index);

drawPanel.remove(this);

drawPanel.repaint();

}

public Component getFrom() {

return from;

}

public Component getTo() {

return to;

}

public int getIndex() {

return index;

}

public void setValue(boolean value) {

test = true;

this.value = value;

}

public boolean isValue() {

if (test) {

return value;

} else {

return getFrom().isValue();

}

}

public void reset(){

test = false;

}

@Override

public String toString() {

return "L" + index;

}

}

package main.CourseWork;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Point;

import java.awt.Rectangle;

import java.awt.event.MouseEvent;

**public class InputPin extends Component {**

public InputPin(DrawPanel drawPanel, String caption) {

super(drawPanel, 0, true, true);

this.caption = caption;

addMouseMotionListener(this);

addMouseListener(this);

h = 2 \* pinH;

setBounds(0, h, w, h);

}

@Override

public void paintComponent(Graphics g) {

g.setColor(Color.WHITE);

g.fillRect(0, 0, w - dx, h - d);

g.setColor(Color.BLACK);

g.drawRect(0, 0, w - dx, h - d);

g.drawString(caption, d, pinH);

if (index == -1) {

g.setColor(Color.red);

}

g.drawLine(w - dx, pinH, w, pinH);

if (index == -1) {

g.setColor(Color.black);

}

}

private String caption;

private int cont(Point p) {

if (new Rectangle(w - dx, pinH / 2, dx, pinH).contains(p)) {

return -1;

}

if (component == this) {

component = null;

}

return -2;

}

@Override

protected void setLink(Link l) {

if (index == -1) {

output.add(l);

}

index = -2;

repaint();

}

@Override

public void mouseClicked(MouseEvent e) {

if (e.getButton() == 3) {

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

output.get(i).removeLink();

}

for (int i = 0; i < numOfInputs; i++) {

if (input[i] != null) {

input[i].removeLink();

}

}

drawPanel.remove(this);

drawPanel.repaint();

return;

}

index = cont(e.getPoint());

if (index != -2) {

if (component == null) {

if (index == -1) {

component = this;

}

} else if (component != this) {

Link l = new Link(drawPanel, this, component);

component.setLink(l);

setLink(l);

drawPanel.add(l);

drawPanel.repaint();

component = null;

}

}

repaint();

}

@Override

public boolean isConnected() {

if (output.isEmpty()) {

return false;

}

return true;

}

@Override

public boolean isValue() {

return value;

}

@Override

public void setValue(boolean value) {

this.value = value;

}

public String getCaption() {

return caption;

}

}

package main.CourseWork;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Point;

import java.awt.Rectangle;

import java.awt.event.MouseEvent;

**public class OutputPin extends Component {**

private String caption;

public OutputPin(DrawPanel drawPanel, String caption) {

super(drawPanel, 1, true, true);

this.drawPanel = drawPanel;

this.caption = caption;

addMouseMotionListener(this);

addMouseListener(this);

h = 2 \* pinH;

setBounds(0, h, w, h);

}

@Override

public void paintComponent(Graphics g) {

g.setColor(Color.WHITE);

g.fillRect(dx, 0, w - dx - d, h - d);

g.setColor(Color.BLACK);

g.drawRect(dx, 0, w - dx - d, h - d);

g.drawString(caption, dx + d, pinH);

if (index == 0) {

g.setColor(Color.red);

}

g.drawLine(0, pinH, dx, pinH);

if (index == 0) {

g.setColor(Color.black);

}

}

private int cont(Point p) {

if (new Rectangle(0, pinH / 2, dx, pinH).contains(p)) {

return 0;

}

if (component == this) {

component = null;

}

return -2;

}

@Override

protected void setLink(Link l) {

if (index == 0) {

input[index] = l;

}

index = -2;

repaint();

}

@Override

public void mouseClicked(MouseEvent e) {

if (e.getButton() == 3) {

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

output.get(i).removeLink();

}

for (int i = 0; i < numOfInputs; i++) {

if (input[i] != null) {

input[i].removeLink();

}

}

drawPanel.remove(this);

drawPanel.repaint();

return;

}

index = cont(e.getPoint());

if (index != -2) {

if (component == null) {

if (index == 0) {

if (input[index] == null) {

component = this;

}

}

} else if (component != this) {

Link l = new Link(drawPanel, component, this);

component.setLink(l);

setLink(l);

drawPanel.add(l);

drawPanel.repaint();

component = null;

}

}

repaint();

}

@Override

public boolean isConnected() {

for (int i = 0; i < numOfInputs; i++) {

if (input[i] == null) {

return false;

}

}

return true;

}

@Override

public boolean isValue() {

return input[0].isValue();

}

@Override

public void setValue(boolean value) {

this.value = value;

}

public String getCaption() {

return caption;

}

}

package main.CourseWork;

import javax.swing.event.TableModelListener;

import javax.swing.table.TableModel;

**public class TruthTableModel<T> implements TableModel** {

public TruthTableModel(T[][] matrix, String[] names) {

this.matrix = matrix;

this.names = names;

}

@Override

public int getRowCount() {

return matrix.length;

}

@Override

public int getColumnCount() {

return names.length;

}

@Override

public String getColumnName(int columnIndex) {

return names[columnIndex];

}

@Override

public Class<?> getColumnClass(int columnIndex) {

if (matrix.length > 0) {

if (matrix[0].length > 0) {

return matrix[0][0].getClass();

}

}

return String.class;

}

@Override

public boolean isCellEditable(int rowIndex, int columnIndex) {

return false;

}

@Override

public Object getValueAt(int rowIndex, int columnIndex) {

return matrix[rowIndex][columnIndex];

}

@Override

public void setValueAt(Object aValue, int rowIndex, int columnIndex) {

}

@Override

public void addTableModelListener(TableModelListener l) {

}

@Override

public void removeTableModelListener(TableModelListener l) {

}

private T[][] matrix;

private String[] names;

}

package main.CourseWork;

import java.util.ArrayList;

import java.util.Random;

import javax.swing.JOptionPane;

**public class MainFrame extends javax.swing.JFrame {**

public MainFrame() {

}

private void addComponentButtonActionPerformed(java.awt.event.ActionEvent evt) {

int t = Math.abs(new Integer(numOfInputsSpinner.getValue().toString()));

if (t == 0) {

t = 1;

}

drawPanel.add(new Component(drawPanel, t, notCheckBox.isSelected(), orRadioButton.isSelected()));

drawPanel.repaint();

}

private void addInputButtonActionPerformed(java.awt.event.ActionEvent evt) {

if (!captionTextField.getText().isEmpty()) {

drawPanel.add(new InputPin(drawPanel, captionTextField.getText()));

drawPanel.repaint();

}

}

private void addOutputButtonActionPerformed(java.awt.event.ActionEvent evt) {

if (!captionTextField.getText().isEmpty()) {

drawPanel.add(new OutputPin(drawPanel, captionTextField.getText()));

drawPanel.repaint();

}

}

private void autoMenuItemActionPerformed(java.awt.event.ActionEvent evt) {

ArrayList<Component> components = new ArrayList<Component>();

ArrayList<InputPin> in = new ArrayList<InputPin>();

ArrayList<OutputPin> out = new ArrayList<OutputPin>();

ArrayList<Link> links = new ArrayList<Link>();

for (java.awt.Component c : drawPanel.getComponents()) {

if (c instanceof InputPin) {

in.add((InputPin) c);

if (!in.get(in.size() - 1).isConnected()) {

JOptionPane.showMessageDialog(this, "Error not connected", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

} else if (c instanceof OutputPin) {

out.add((OutputPin) c);

if (!out.get(out.size() - 1).isConnected()) {

JOptionPane.showMessageDialog(this, "Error not connected", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

} else if (c instanceof Component) {

components.add((Component) c);

if (!components.get(components.size() - 1).isConnected()) {

JOptionPane.showMessageDialog(this, "Error not connected", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

} else if (c instanceof Link) {

links.add((Link) c);

}

}

Integer[][] matrix = new Integer[(int) Math.pow(2, in.size())][in.size() + out.size()];

String[] names = new String[in.size() + out.size()];

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

names[i] = in.get(i).getCaption();

}

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

names[i + in.size()] = out.get(i).getCaption();

}

for (int i = 0; i < Math.pow(2, in.size()); i++) {

String bin = Integer.toBinaryString(i);

boolean[] values = new boolean[in.size()];

int d = values.length - bin.length();

for (int j = bin.length() - 1; j >= 0; j--) {

if (bin.charAt(j) == '1') {

values[d + j] = true;

}

}

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

in.get(j).setValue(values[j]);

if (values[j]) {

matrix[i][j] = 1;

} else {

matrix[i][j] = 0;

}

}

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

if (out.get(j).isValue()) {

matrix[i][j + in.size()] = 1;

} else {

matrix[i][j + in.size()] = 0;

}

}

}

truthTable.setModel(new TruthTableModel(matrix, names));

Random r = new Random();

int[][] faults = new int[links.size()][links.size()];

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

for (int j = 0; j < Math.abs(r.nextInt()) % (links.size() / 2) + 1; j++) {

while (true) {

int index = Math.abs(r.nextInt()) % links.size();

if (faults[i][index] == 0) {

if (Math.random() > 0.5) {

faults[i][index] = 1;

} else {

faults[i][index] = -1;

}

break;

}

}

}

}

String[] faultsNames = {"N", "Faults"};

String[][] faultsMatrix = new String[links.size()][2];

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

faultsMatrix[i][0] = i + "";

String value = "";

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

if (faults[i][j] > 0) {

value += links.get(j).toString()+ "/" + 1 +" ";

} else if (faults[i][j] < 0) {

value += links.get(j).toString()+ "/" + 0 +" ";

}

}

faultsMatrix[i][1] = value;

}

faultTable.setModel(new TruthTableModel(faultsMatrix, faultsNames));

Integer[][] discrimination = new Integer[matrix.length][faults.length];

String[] discriminationNames = new String[faults.length];

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

discriminationNames[i] = "" + i;

}

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

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

boolean t = false;

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

if (faults[j][k] > 0) {

links.get(k).setValue(true);

} else if (faults[j][k] < 0) {

links.get(k).setValue(false);

}

}

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

if (matrix[i][k] > 0) {

in.get(k).setValue(true);

} else {

in.get(k).setValue(false);

}

}

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

if (out.get(k).isValue()) {

if (matrix[i][k + in.size()] == 0) {

t = true;

}

} else {

if (matrix[i][k + in.size()] == 1) {

t = true;

}

}

}

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

links.get(k).reset();

}

if (t) {

discrimination[i][j] = 1;

} else {

discrimination[i][j] = 0;

}

}

}

disctiminationTable.setModel(new TruthTableModel(discrimination, discriminationNames));

ArrayList<Integer> f = new ArrayList();

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

f.add(i);

}

String result = "";

ArrayList<Integer> m = new ArrayList();

while (f.size() > 0) {

int max = 0;

int maxIndex = 0;

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

int count = 0;

boolean need = false;

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

if (discrimination[i][j] > 0) {

count++;

if (f.indexOf(j) != -1) {

need = true;

}

}

}

if (count > max && need) {

max = count;

maxIndex = i;

}

}

if (m.indexOf(maxIndex) != -1 || max == 0) {

break;

}

m.add(maxIndex);

result += maxIndex + " ";

for (Integer i = 0; i < discrimination[maxIndex].length; i++) {

if (discrimination[maxIndex][i] > 0) {

f.remove(i);

}

}

}

JOptionPane.showMessageDialog(this, result, "Result", JOptionPane.INFORMATION\_MESSAGE);

}

private void manualMenuItemActionPerformed(java.awt.event.ActionEvent evt) {

ArrayList<Component> components = new ArrayList<Component>();

ArrayList<InputPin> in = new ArrayList<InputPin>();

ArrayList<OutputPin> out = new ArrayList<OutputPin>();

ArrayList<Link> links = new ArrayList<Link>();

for (java.awt.Component c : drawPanel.getComponents()) {

if (c.getClass() == InputPin.class) {

in.add((InputPin) c);

if (!in.get(in.size() - 1).isConnected()) {

JOptionPane.showMessageDialog(this, "Error not connected", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

} else if (c.getClass() == OutputPin.class) {

out.add((OutputPin) c);

if (!out.get(out.size() - 1).isConnected()) {

JOptionPane.showMessageDialog(this, "Error not connected", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

} else if (c.getClass() == Component.class) {

components.add((Component) c);

if (!components.get(components.size() - 1).isConnected()) {

JOptionPane.showMessageDialog(this, "Error not connected", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

} else if (c.getClass() == Link.class) {

links.add((Link) c);

}

}

String[] fNames = new String[links.size()];

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

fNames[i] = links.get(i).toString();

}

MDialog md = new MDialog(this, true, fNames, links.size());

md.setVisible(true);

int[][] faults = md.getMatrix();

if (faults == null) {

return;

}

Integer[][] matrix = new Integer[(int) Math.pow(2, in.size())][in.size() + out.size()];

String[] names = new String[in.size() + out.size()];

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

names[i] = in.get(i).getCaption();

}

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

names[i + in.size()] = out.get(i).getCaption();

}

for (int i = 0; i < Math.pow(2, in.size()); i++) {

String bin = Integer.toBinaryString(i);

boolean[] values = new boolean[in.size()];

int d = values.length - bin.length();

for (int j = bin.length() - 1; j >= 0; j--) {

if (bin.charAt(j) == '1') {

values[d + j] = true;

}

}

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

in.get(j).setValue(values[j]);

if (values[j]) {

matrix[i][j] = 1;

} else {

matrix[i][j] = 0;

}

}

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

if (out.get(j).isValue()) {

matrix[i][j + in.size()] = 1;

} else {

matrix[i][j + in.size()] = 0;

}

}

}

truthTable.setModel(new TruthTableModel(matrix, names));

String[] faultsNames = {"N", "Faults"};

String[][] faultsMatrix = new String[links.size()][2];

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

faultsMatrix[i][0] = i + "";

String value = "";

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

if (faults[i][j] > 0) {

value += links.get(j).toString()+ "/" + 1+" ";

} else if (faults[i][j] < 0) {

value += links.get(j).toString()+ "/" + 0+" ";

}

}

faultsMatrix[i][1] = value;

}

faultTable.setModel(new TruthTableModel(faultsMatrix, faultsNames));

Integer[][] discrimination = new Integer[matrix.length][faults.length];

String[] discriminationNames = new String[faults.length];

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

discriminationNames[i] = "" + i;

}

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

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

boolean t = false;

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

if (faults[j][k] > 0) {

links.get(k).setValue(true);

} else if (faults[j][k] < 0) {

links.get(k).setValue(false);

}

}

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

if (matrix[i][k] > 0) {

in.get(k).setValue(true);

} else {

in.get(k).setValue(false);

}

}

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

if (out.get(k).isValue()) {

if (matrix[i][k + in.size()] == 0) {

t = true;

}

} else {

if (matrix[i][k + in.size()] == 1) {

t = true;

}

}

}

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

links.get(k).reset();

}

if (t) {

discrimination[i][j] = 1;

} else {

discrimination[i][j] = 0;

}

}

}

disctiminationTable.setModel(new TruthTableModel(discrimination, discriminationNames));

ArrayList<Integer> f = new ArrayList();

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

f.add(i);

}

String result = "";

ArrayList<Integer> m = new ArrayList();

while (f.size() > 0) {

int max = 0;

int maxIndex = 0;

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

int count = 0;

boolean need = false;

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

if (discrimination[i][j] > 0) {

count++;

if (f.indexOf(j) != -1) {

need = true;

}

}

}

if (count > max && need) {

max = count;

maxIndex = i;

}

}

if (m.indexOf(maxIndex) != -1 || max == 0) {

break;

}

m.add(maxIndex);

result += maxIndex + " ";

for (Integer i = 0; i < discrimination[maxIndex].length; i++) {

if (discrimination[maxIndex][i] > 0) {

f.remove(i);

}

}

}

JOptionPane.showMessageDialog(this, result, "Result", JOptionPane.INFORMATION\_MESSAGE);

}

public static void main(String args[]) {

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new MainFrame().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JButton addComponentButton;

private javax.swing.JButton addInputButton;

private javax.swing.JButton addOutputButton;

private javax.swing.JRadioButton andRadioButton;

private javax.swing.JMenuItem autoMenuItem;

private javax.swing.JLabel captionLabel;

private javax.swing.JTextField captionTextField;

private javax.swing.ButtonGroup componentTypeButtonGroup;

private javax.swing.JScrollPane discriminationTableScrollPane;

private javax.swing.JTable disctiminationTable;

private main.CourseWork.DrawPanel drawPanel;

private javax.swing.JScrollPane drawScrollPane;

private javax.swing.JTable faultTable;

private javax.swing.JScrollPane faultTableScrollPane;

private javax.swing.JMenu generateTablesMenu;

private javax.swing.JMenuBar jMenuBar1;

private javax.swing.JToolBar.Separator jSeparator3;

private javax.swing.JToolBar.Separator jSeparator4;

private javax.swing.JToolBar.Separator jSeparator5;

private javax.swing.JTabbedPane jTabbedPane;

private javax.swing.JMenuItem manualMenuItem;

private javax.swing.JCheckBox notCheckBox;

private javax.swing.JLabel numOfInputsLabel;

private javax.swing.JSpinner numOfInputsSpinner;

private javax.swing.JRadioButton orRadioButton;

private javax.swing.JMenu testsMenu;

private javax.swing.JToolBar toolBar;

private javax.swing.JTable truthTable;

private javax.swing.JScrollPane truthTableScrollPane;

}