Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №4**

*з предмету: «ТПКС»*

*по темі:*

«Автоматизація кодування графу переходів.»

Виконав: студент ФІОТ

групи ІО-92

Петрук В.О.

Київ 2012р.

**Мета:** Здобуття навичок з автоматизації процедури сумісного кодування графу

переходів

**Завдання**

1. Розробити алгоритм сумісного кодування графу переходів з попередньої роботи – будь-які вузли, що мають зв'язок повинні мати коди, які відрізняються лише у одному двійковому розряді. Блок-схему та опис розробленого алгоритму надати в протоколі роботи.

2. Реалізувати розроблений алгоритм. Кодування відобразити на графічному представленні графу переходів.

3. Модифікувати формат зберігання графу переходів таким чином, щоб він містив інформацію про коди вузлів. Реалізувати можливість збереження/відновлення закодованого графа переходів.

**Алгоритм сусіднього кодування**

1. Знайти усі вершини, зациклені на себе та доповнити граф додатковими вершинами.

2. Знайти усі з'єднання типу «трикутник» (наприклад, a -> b, b -> c, c -> a). Якщо хоч одна з вершин має еквівалентну, прибрати трикутник за допомогою цієї вершини, інакше створити еквівалент і використовувати його та перейти до пункту 1.

3. Відсортувати вершини за кількістю вихідних зв'язків. Зробити маркування для вершин з найбільшою кількістю вихідних зв’язків.

4. Маркувати першу вершину і всі пов'язані з нею

5. Перевірка наступної вершини: якщо вона маркована і має немарковані вершини, в які вона виходить, то маркувати їх. Якщо не маркована, але є марковані вершини, в які вона виходить, підібрати сусідній маркер для поточної вершини,а потім промаркувати всі сусідні не маровані вершини.

Граф автомата Мура з закодованими вершинами відображається в графічному інтерфесі користувача за допомогою бібліотеки jgraphx(рис. 1).

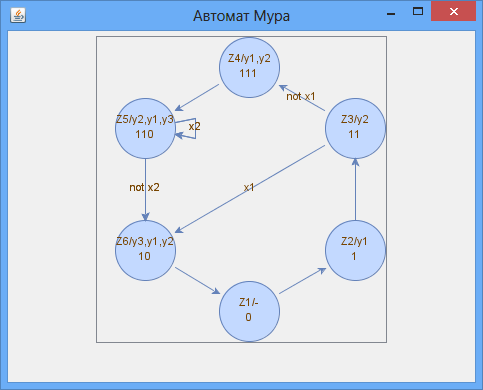


Рисунок 1 – Відображення графу автомата Мура

Для збереження та відновлення модифікованого автомату Мура у вигляді текстового файлу необхідно вибрати пункти меню Файл «Зберегти автомат Мура» та « Відкрити автомат Мура» відповідно . У діалоговому вікні необхідно ввести ім’я файлу.

**Код програми**

Представлені лише класи та методи, які безпосередньо були реалізовані або змінені для виконання даної лабораторної роботи.

package lab1.graph;

import java.util.ArrayList;

import java.util.HashMap;

public class MooreMachineActions {

private HashMap<Integer, Boolean> poolOfAvailableMarkers;

private HashMap<String, Integer> vertexMarking;

private String splitLine = ":";

public MooreMachineActions(ArrayList<String> moorGraphTable, int countOfVertexes, int countOfConditions) {

int sizeOfMarking = (int) (Math.ceil(Math.log(countOfVertexes) / Math.log(2.0)));

//якщо кілікість звязків пневищує поточну кільксть маркерів

ArrayList<String[]> lines = new ArrayList<String[]>();

if (moorGraphTable.size() != 2) {

for (int i = 2; i < moorGraphTable.size(); i++) {

lines.add(moorGraphTable.get(i).split(splitLine));

}

}

ArrayList<String> temp;

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

temp = new ArrayList<String>();

for (int j = 0; j < Math.pow(2, countOfConditions); j++) {

if (!temp.contains(lines.get(j)[countOfConditions + i])) {

temp.add(lines.get(j)[countOfConditions + i]);

}

}

if (temp.size() > sizeOfMarking)

sizeOfMarking = temp.size();

}

int[][] vertexesConnections = new int[countOfVertexes][countOfVertexes];

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

for (int j = 0; j < Math.pow(2, countOfConditions); j++) {

vertexesConnections[i][getVertexPos(lines.get(j)[countOfConditions + i], moorGraphTable.get(0).split(splitLine), countOfConditions, countOfVertexes)] = 1;

}

}

//вільні маркування

poolOfAvailableMarkers = new HashMap<Integer, Boolean>();

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

poolOfAvailableMarkers.put(i, true);

}

// колекція "Вершина - код"

vertexMarking = new HashMap<String, Integer>();

boolean isMarked = false;

while (!isMarked) {

int currentMarking = 0;

vertexMarking.put(moorGraphTable.get(0).split(splitLine)[countOfConditions], currentMarking);//перша вершина:0

isMarked = true;

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

for (int j = 0; j < countOfVertexes; j++) {

currentMarking = getNeighbourCode(currentMarking);

if (currentMarking != -1) {

vertexMarking.put(moorGraphTable.get(0).split(splitLine)[countOfConditions + i], getNeighbourCode(currentMarking));

} else {

isMarked = false;

break;

}

}

}

if (!isMarked){

sizeOfMarking++;

};

}

}

//пошук сусіднього маркування

public int getNeighbourCode(int connectedMarker) {

int temp;

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

if (poolOfAvailableMarkers.get(i)) {

temp = i ^ connectedMarker;

while (temp % 2 == 0)

temp /= 2;

if (temp == 1) {

poolOfAvailableMarkers.put(i, false);

return i;

}

}

}

return -1;

}

//визначає номер вершини за її позначенням

private int getVertexPos(String verName, String[] line1, int countOfConditions, int countOfVertexes) {

int count = 0;

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

if (line1[countOfConditions + i].equals(verName)) {

break;

}

count++;

}

return count;

}

//встановлення маркування для даного автомату

public void setVertexMarking(HashMap<String, Integer> vertexMarking){

this.vertexMarking = vertexMarking;

}

//повертає маркування даного автомату Мура

public HashMap<String, Integer> getVertexMarking(){

return vertexMarking;

}

}

package lab1.gui;

import com.mxgraph.layout.mxCircleLayout;

import com.mxgraph.layout.mxGraphLayout;

import com.mxgraph.swing.mxGraphComponent;

import com.mxgraph.util.mxConstants;

import com.mxgraph.view.mxGraph;

import com.mxgraph.view.mxStylesheet;

import lab1.graph.MooreMachine;

import lab1.graph.MooreMachineActions;

import javax.swing.\*;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Hashtable;

public class GraphPanel extends JPanel {

ArrayList<String> vertexNames;//найменування вершин

ArrayList<String> vertexNamesPlus;//найменування вершин(з сигналами)

private int counter = 0; //кількість вершин;

private MooreMachine mooreMachine;

private String splitLine = ":";

private String[] firstLine;

private ArrayList<Integer> fromVertex;

private ArrayList<Integer> toVertex;

private String[] conditions;

private mxGraphComponent graphComponent;

HashMap<String, Integer> vertexMarking;

// Конструктор панелі графу автомата Мура.

// mooreMachine - таблична форма автомату Мура

public GraphPanel(MooreMachine mooreMachine) {

this.mooreMachine = mooreMachine;

createGraph();

}

public ArrayList<String> getVertexNames(){

return vertexNames;

}

// Створення графу автомата МУра.

private void createGraph() {

// removeAll();

ArrayList<String> moorTable = mooreMachine.getMoorGraphTable();

//позначення вершин (+сигнали)

String[] secondLine = moorTable.get(1).split(splitLine);

int countOfConditions = 0;

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

if (!secondLine[i].equals("-")) {

countOfConditions++;

} else {

break;

}

}

firstLine = moorTable.get(0).split(splitLine);

vertexNames = new ArrayList<String>();

vertexNamesPlus = new ArrayList<String>();

int countOfVertexes = firstLine.length - countOfConditions;

// mooreMachine.printMoorMachine();

MooreMachineActions machineActions = mooreMachine.getMachineActions();

vertexMarking = machineActions.getVertexMarking();

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

vertexNames.add(firstLine[i + countOfConditions]);

vertexNamesPlus.add(firstLine[i + countOfConditions] + "/" + secondLine[i + countOfConditions]+

"\n"+Integer.toBinaryString(vertexMarking.get(firstLine[i + countOfConditions])));

}

//встановлюємо переходи

defineEdges(moorTable, countOfConditions, countOfVertexes);

//рисуємо граф

drawGraph(countOfVertexes);

}

//Створення вершин графу.

private Integer defineVertex(int i, int countOfConditions, String[] line) {

String zName = line[countOfConditions + i];

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

if (vertexNames.get(j).equals(zName)) {

return j;

}

}

return -1;

}

//Створення переходів графу.

private void defineEdges(ArrayList<String> moorTable, int countOfConditions, int countOfVertexes) {

ArrayList<String[]> lines = new ArrayList<String[]>();

if (moorTable.size() != 2) {

for (int i = 2; i < moorTable.size(); i++) {

lines.add(moorTable.get(i).split(splitLine));

}

}

fromVertex = new ArrayList<Integer>();

toVertex = new ArrayList<Integer>();

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

for (int j = 0; j < Math.pow(2, countOfConditions); j++) {

fromVertex.add(i);

toVertex.add(defineVertex(i, countOfConditions, lines.get(j)));

if (fromVertex.size() > 1) {

int fvs = fromVertex.size();

int tvs = toVertex.size();

if ((fromVertex.get(fvs - 2) == fromVertex.get(fvs - 1)) & (toVertex.get(tvs - 2) == toVertex.get(tvs - 1))) {

fromVertex.remove(fvs - 1);

toVertex.remove(tvs - 1);

}

}

}

}

int[][] condIsActive = new int[countOfConditions][countOfVertexes];//масив значимості умов переходах

// 0 - активний

// 1 - перехід не залежить від умови

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

for (int j = 0; j < countOfConditions; j++) {

String toV = lines.get(0)[countOfConditions + i];

for (int k = 0; k < Math.pow(2, countOfConditions); k++) {

if (lines.get(k)[j].equals("1")) {

if (toV.equals(lines.get(k)[countOfConditions + i])) {

condIsActive[j][i] = 1;

break;

}

}

}

}

}

conditions = new String[fromVertex.size()];

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

conditions[i] = "";

for (int j = 0; j < countOfConditions; j++) {

if (condIsActive[j][fromVertex.get(i)] == 0) {

// умова чи антиумова

int row = -1;

for (int k = 0; k < Math.pow(2, countOfConditions); k++) {

if (lines.get(k)[countOfConditions + fromVertex.get(i)].equals(firstLine[countOfConditions + toVertex.get(i)])) {

if (lines.get(k)[j].equals("0")) {

if (!conditions[i].equals(firstLine[j])) {

conditions[i] += firstLine[j];

}

} else {

if (!conditions[i].equals("not " + firstLine[j])) {

conditions[i] += "not " + firstLine[j];

}

}

}

}

}

}

}

}

//рисування графу на панелі

public void drawGraph(int counter) {

mxGraph graph = new mxGraph();

Object parent = graph.getDefaultParent();

graph.getModel().beginUpdate();

try {

mxStylesheet stylesheet = graph.getStylesheet();

Hashtable<String, Object> vertexStyle = new Hashtable<String, Object>();

vertexStyle.put(mxConstants.STYLE\_SHAPE, mxConstants.SHAPE\_ELLIPSE);

vertexStyle.put(mxConstants.STYLE\_OPACITY, 100);

stylesheet.putCellStyle("ROUNDED", vertexStyle);

Hashtable<String, Object> arrowStyle = new Hashtable<String, Object>();

arrowStyle.put(mxConstants.STYLE\_EDGE, mxConstants.EDGESTYLE\_TOPTOBOTTOM);

arrowStyle.put(mxConstants.STYLE\_OPACITY, 100);

arrowStyle.put(mxConstants.STYLE\_FONTCOLOR, "#774400");

stylesheet.putCellStyle("ARROW", arrowStyle);

ArrayList<Object> vertexes = new ArrayList<Object>();

ArrayList<Object> edges = new ArrayList<Object>();

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

Object vertex = graph.insertVertex(parent, null, vertexNamesPlus.get(i), 60, 60, 60, 60, "ROUNDED");

vertexes.add(vertex);

}

//рисуємо переходи

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

if (toVertex.get(i) != -1) {

Object edge = graph.insertEdge(parent, null, conditions[i], vertexes.get(fromVertex.get(i)),

vertexes.get(toVertex.get(i)), "ARROW");

edges.add(edge);

}

}

} finally {

graph.getModel().endUpdate();

}

graph.setCellsEditable(false);

graph.setCellsResizable(false);

graph.setCellsBendable(false);

graph.setConnectableEdges(false);

graphComponent = new mxGraphComponent(graph);

add(graphComponent);

mxGraphLayout graphLayout = new mxCircleLayout(graph);

graphLayout.execute(parent);

repaint();

revalidate();

}

}

package lab1.file;

import lab1.graph.MooreMachine;

import lab1.gui.MainPanel;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

public class SaveMoorListener implements ActionListener {

private MainPanel mainPanel;

public SaveMoorListener(MainPanel mainPanel) {

this.mainPanel = mainPanel;

}

public void actionPerformed(ActionEvent e) {

JFileChooser chooser = new JFileChooser();

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

chooser.setSelectedFile(new File(".moor"));

int result = chooser.showSaveDialog(null);

if (result != JFileChooser.CANCEL\_OPTION) {

String fileName = chooser.getSelectedFile().getPath();

try {

writeToTXTFile(fileName);

} catch (IOException e1) {

e1.printStackTrace();

}

}

}

private void writeToTXTFile(String fileName) throws IOException {

BufferedWriter newMoorFile = new BufferedWriter( new FileWriter(fileName));

MooreMachine mooreMachine = new MooreMachine();

ArrayList<String> moorTable = mooreMachine.createGraph(mainPanel.getThisLsa());

String[] addMoor = new String[moorTable.size()];

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

addMoor[i] = moorTable.get(i)+":";

}

HashMap<String, Integer> vertexMarking = mooreMachine.getMachineActions().getVertexMarking();

for (String s: addMoor){

newMoorFile.write(s+vertexMarking.put(s.substring(0,2))"\n");

}

newMoorFile.close();

}

}

**Висновок**

Розроблено та реалізовано алгоритми сумісного кодування графу переходів автомата Мура. Граф із закодованими вершинами відображено в графічному інтерфейсі користувача.

Розроблено процедури збереження та відновлення модифікованого автомата Мура в табличному представленні в текстовий файл. Реалізована можливість преглянути збережені алгоритми у вигляді графу в графічному інтерефесі користувача.