Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Дискретная математика

Лабораторная работа № 4

Тема: «Определение полноты системы функций»

Выполнил студент группы

РИС-22-2б:

Прядеин И.А

Проверил: Ст. Преподаватель кафедры ИТАС:

Рустамханова Г.И.

г. Пермь – 2023

Оглавление

[Постановка задачи 3](#Постановка_Задачи)**[.](#Постановка_Задачи)**

[Алгоритм работы](#Алгоритм_работы) **[4.](#Алгоритм_работы)**

[Результат тестирования 7](#Результат_тестирования)**[.](#Результат_тестирования)**

[Исходный код 8](#Исходный_код)

# Постановка задачи

* Дана система булевых функций в виде векторов.
* Реализовать ввод системы.
* Реализовать вывод таблицы классов булевой функции.
* Определить является ли система полной.

# Алгоритм работы

Начало работы программы начинается с объявления массива векторов логических функций. Далее создается объект класса “FunctionalCompleteness”. Проверка исходных векторов происходит в методе “InputFunctions” класса “FunctionalCompleteness”, в этой же функции происходит начальное заполнение таблицы.

Проверка системы функций на полноту реализовано в методе “isFunctionalComplete” класса “FunctionalCompleteness”, изображенном на “Рис. 1”. Внутри данного метода вызываются методы проверящие каждый класс булевых функций.

Рис. 1 - Функция “isFunctionalComplete”

void FunctionalCompleteness::isFunctionalComplete() {

checkT0();

checkT1();

checkS();

checkM();

checkL();

PrintTable();

cout << '\n';

for (int j = 0; j < 5; j++) {

bool isComplete = true;

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

if (!\_table[i][j])

isComplete = false;

if (isComplete) {

cout << "The system is not functionally complete\n";

return;

}

}

cout << "The system is functionally complete\n";

}

После выполнения всех методов замкнутых классов булевых функций, выводиться конечная таблица. Проходя каждый столбец таблицы, определяется полнота системы булевых функций.

# Результат тестирования

В случае, если пользователь введёт неверную систему векторов появится сообщение об ошибке, так же, как изображено на “Рис. 2”.

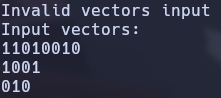


Рис. 2 - Неверный ввод системы векторов

Если введёная система векторов удовлетворяет условиям как на “Рис. 3”, программа успешно выводит таблицу замкнутых классов булевых функций и определяет полноту системы.

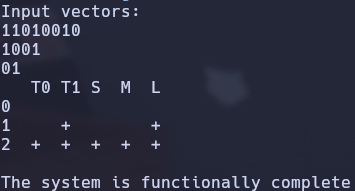


Рис. 3 - Успешное выполнение программы

# Исходный код

Файл “main.cpp”:

#include "functional\_completeness.h"

#include <iostream>

#include <vector>

using std::cout;

using std::vector;

int main() {

vector<string> functions = {

"1101 0010",

"1001",

"01"

};

FunctionalCompleteness com;

com.InputFunctions(functions);

cout << "Input vectors:\n";

com.PrintFunctions(functions);

com.isFunctionalComplete();

return 0;

}

Файл “functional\_completeness.cpp”:

#include "functional\_completeness.h"

void FunctionalCompleteness::PrintFunctions(const vector<string>& functions) {

for (string function : functions)

cout << function << '\n';

}

void FunctionalCompleteness::PrintTable() {

const int width = 3;

cout << left << setw(width) << "";

cout << left << setw(width) << "T0";

cout << left << setw(width) << "T1";

cout << left << setw(width) << "S";

cout << left << setw(width) << "M";

cout << left << setw(width) << "L";

cout << '\n';

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

cout << left << setw(width) << i;

for (int j = 0; j < \_table[i].size(); j++)

if (\_table[i][j])

cout << left << setw(width) << "+";

else

cout << left << setw(width) << "";

cout << '\n';

}

}

void FunctionalCompleteness::isFunctionalComplete() {

checkT0();

checkT1();

checkS();

checkM();

checkL();

PrintTable();

cout << '\n';

for (int j = 0; j < 5; j++) {

bool isComplete = true;

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

if (!\_table[i][j])

isComplete = false;

if (isComplete) {

cout << "The system is not functionally complete\n";

return;

}

}

cout << "The system is functionally complete\n";

}

void FunctionalCompleteness::InputFunctions(vector<string>& functions) {

if (!isValidFunctions(functions)) {

cout << "Неверный ввод исходных векторов\n";

return;

}

for (string function : functions)

\_functions.push\_back(ConvertTo8(function));

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

\_table.push\_back(vector<bool>(5, false));

}

void FunctionalCompleteness::checkT0() {

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

if (\_functions[i][0] == '0')

\_table[i][0] = true;

else

\_table[i][0] = false;

}

void FunctionalCompleteness::checkT1() {

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

if (\_functions[i][7] == '1')

\_table[i][1] = true;

else

\_table[i][1] = false;

}

void FunctionalCompleteness::checkS() {

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

int left = 0;

int right = 7;

bool isS = true;

while (left < right)

if (\_functions[i][left++] == \_functions[i][right--]) {

isS = false;

break;

}

if (isS)

\_table[i][2] = true;

else

\_table[i][2] = false;

}

}

void FunctionalCompleteness::checkM() {

int patterns[6][4] = {

{ 0, 1, 3, 7 },

{ 0, 1, 5, 7 },

{ 0, 2, 3, 7 },

{ 0, 2, 6, 7 },

{ 0, 4, 5, 7 },

{ 0, 4, 6, 7 },

};

bool isNotM;

bool isChanged;

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

isNotM = false;

for (int j = 0; j < 6 && !isNotM; j++) {

isChanged = false;

for (int k = 0; k < 4; k++) {

if (\_functions[i][patterns[j][k]] == '1')

isChanged = true;

if (\_functions[i][patterns[j][k]] == '0' && isChanged) {

\_table[i][3] = false;

isNotM = true;

break;

}

}

}

if (!isNotM)

\_table[i][3] = true;

}

}

void FunctionalCompleteness::checkL() {

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

bool temp[4]{false};

temp[0] = (((((\_functions[i][0] - '0') + (\_functions[i][4] - '0')) % 2 +

(\_functions[i][2] - '0')) % 2) + (\_functions[i][6] - '0')) % 2; // Cxy

temp[1] = ((((\_functions[i][0] - '0') + (\_functions[i][2] - '0')) % 2 +

(\_functions[i][1] - '0')) + (\_functions[i][3] - '0')) % 2; // Cyz

temp[2] = ((((\_functions[i][0] - '0') + (\_functions[i][1] - '0')) % 2 +

(\_functions[i][4] - '0')) + (\_functions[i][5] - '0')) % 2; // Cxz

temp[3] = temp[0];

for (int j = 1; j < \_functions.size(); j++)

temp[3] = (temp[3] + temp[j]) % 2;

if (temp[0] == true || temp[1] == true || temp[2] == true || temp[3] == true)

\_table[i][4] = false;

else

\_table[i][4] = true;

}

}

string FunctionalCompleteness::ConvertTo8(const string& function) {

string result;

if (function.size() == 1)

for (int i = 0; i < 8; i++)

result += function;

if (function.size() == 2)

for (int i = 0; i < 2; i++) {

result += function[i];

result += function[i];

result += function[i];

result += function[i];

}

if (function.size() == 4)

for (int i = 0; i < 4; i++) {

result += function[i];

result += function[i];

}

if (function.size() == 8)

result = function;

return result;

}

bool FunctionalCompleteness::isValidFunctions(vector<string>& functions) {

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

if (!isValidFunction(functions[i]))

return false;

return true;

}

bool FunctionalCompleteness::isValidFunction(string& function) {

function.erase(remove\_if(function.begin(), function.end(), isspace), function.end());

if (function.size() == 1 ||

function.size() == 2 ||

function.size() == 4 ||

function.size() == 8) {

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

if (function[i] != '1' && function[i] != '0')

return false;

return true;

}

return false;

}