Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

Дискретная математика

Семестр 3

Лабораторная работа 4

Выполнил  
студент группы РИС-22-2б  
Баяндин К.С.

Проверила  
старший преподаватель кафедры ИТАС  
Рустамханова Г.И.

Пермь, 2023

Постановка задачи

Создать консольное приложение, которое определяет полноту системы булевых функций, заданных вектором

Для этого разработать:

1. Ввод системы функций векторным способом
2. Вывод таблицы с векторами и их классами (T0, T1, S, M, L)
3. Определение полноты и вывод результатов

Алгоритмы работы программы

Для взаимодействия пользователя с программой, был разработан массив из возможных команд, вывод этих команд на экран, а также возможность выбора одной из них. Листинг представлен ниже:

static void Main()

{

string[] menu = {

"1. Ввод системы функций",

"2. Расчет полноты системы функций",

"3. Выход из программы",

};

string[] functions = Array.Empty<string>();

int command;

for (int i = 0; i < menu.Length; i++)

Console.WriteLine(menu[i]);

do

{

command = InputInt("Введите номер команды: ");

switch (command)

{

case 1:

functions = InputSystem();

break;

case 2:

CompletenessSystem(functions);

break;

case 3:

Console.WriteLine("Спасибо что пользуетесь нашей программой!");

break;

default:

Console.WriteLine("Неверная команда");

break;

}

} while (command != 3);

}

Как можно заметить, программа позволяет ввести систему функций (векторов) и расчитать полноту системы функций.

Ввод системы функций осуществляется с помощью функции *InputSustem()* (код представлен ниже). Ввод размера системы проверяется с помощью функции *InputInt()*, ввод векторов проверяется с помощью функции *InputVector()*.

static string[] InputSystem()

{

var size = InputInt("Введите размер системы: ");

string[] functions = new string[size];

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

{

functions[i] = InputVector($"Введите {i + 1} ф-ию: ");

}

return functions;

}

Вычисление полноты системы осуществляется с помощью функции *CompletenessSystem()* (код представлен ниже). Проверяется существование системы (размер не равен 0), создается матрица в функции *CreateFunctionsSystem(functions)* со значениями *true* и *false* принадлежности функции к данному классу функции:

* Сохраняющий 0 (*IsT0()*)
* Сохраняющий 1 (*IsT1()*)
* Самодвойственный (*IsSelfDualityFunction()*)
* Монононный (*IsManatonicFunction()*)
* Линейный (*IsLinearityFunction()*)

Далее эта матрица выводится и передается в функцию *IsCompletenessSystem()*, в которой и производится определение полноты системы функций, результат выводится на основе данных вычислений. Листинг всех функций можно посмотреть в конце отчета.

static void CompletenessSystem(string[] functions)

{

if (functions.Length == 0)

{

Console.WriteLine("Задайте систему функций.");

return;

}

var functionSystem = CreateFunctionsSystem(functions);

DisplayClassFunction(functions, functionSystem);

bool isCompletenessSystem = IsCompletenessSystem(functionSystem);

if (isCompletenessSystem)

Console.WriteLine("Система функций полная");

else

Console.WriteLine("Система функций неполная");

}

**Результаты работы программы:**

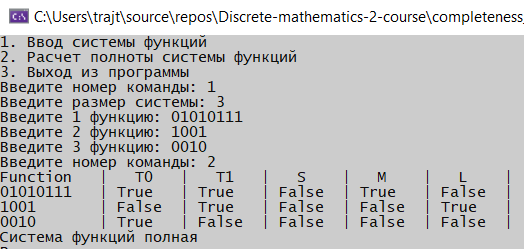


Рисунок 1. Система функций полная

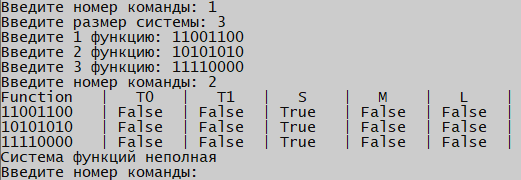


Рисунок 2. Система функций неполная

Листинг

using System;

using System.Reflection.Metadata.Ecma335;

using System.Numerics;

using System.ComponentModel.DataAnnotations;

namespace completeness\_system

{

class Program

{

static void Main()

{

string[] menu = {

"1. Ввод системы функций",

"2. Расчет полноты системы функций",

"3. Выход из программы",

};

string[] functions = Array.Empty<string>();

int command;

for (int i = 0; i < menu.Length; i++)

Console.WriteLine(menu[i]);

do

{

command = InputInt("Введите номер команды: ");

switch (command)

{

case 1:

functions = InputSystem();

break;

case 2:

CompletenessSystem(functions);

break;

case 3:

Console.WriteLine("Спасибо что пользуетесь нашей программой!");

break;

default:

Console.WriteLine("Неверная команда");

break;

}

} while (command != 3);

}

//проверка на ввод целого числа

static int InputInt(string message)

{

int number;

Console.Write(message);

if (!int.TryParse(Console.ReadLine(), out number) || (number < 0))

{

do

{

Console.WriteLine("Ошибка!");

if (number < 0)

Console.WriteLine("Число не может быть отрицательным");

else

Console.WriteLine("Введите число типа int");

Console.Write(message);

} while (!int.TryParse(Console.ReadLine(), out number) || (number < 0));

}

return number;

}

static bool ValidVector(string? inputVector)

{

return !(inputVector is null || inputVector == "");

}

//проверка на ввод вектора

static string InputVector(string message)

{

string? inputVector;

Console.Write(message);

bool isCorrect = false;

do

{

inputVector = Console.ReadLine();

if (!ValidVector(inputVector))

Console.WriteLine("Ошибка ввода вектора булевой функции. ");

else

isCorrect = true;

} while (!isCorrect);

return inputVector;

}

static string[] InputSystem()

{

var size = InputInt("Введите размер системы: ");

string[] functions = new string[size];

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

{

functions[i] = InputVector($"Введите {i + 1} функцию: ");

}

return functions;

}

//{ "1001", "0100", "11" };

static void CompletenessSystem(string[] functions)

{

if (functions.Length == 0)

{

Console.WriteLine("Задайте систему функций.");

return;

}

var functionSystem = CreateFunctionsSystem(functions);

DisplayClassFunction(functions, functionSystem);

bool isCompletenessSystem = IsCompletenessSystem(functionSystem);

if (isCompletenessSystem)

Console.WriteLine("Система функций полная");

else

Console.WriteLine("Система функций неполная");

}

static void DisplayClassFunction(string[] function, bool[,] functionSystem)

{

Console.WriteLine("Function | T0 | T1 | S | M | L |");

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

{

Console.Write($"{function[i],-10} | ");

for (int j = 0; j < functionSystem.GetLength(1); j++)

{

Console.Write($"{functionSystem[i, j],-6} | ");

}

Console.WriteLine();

}

}

static bool IsCompletenessSystem(bool[,] functionsSystem)

{

for (int col = 0; col < functionsSystem.GetLength(1); col++)

{

bool hasFalse = false;

for (int row = 0; row < functionsSystem.GetLength(0); row++)

{

if (!functionsSystem[row, col])

{

hasFalse = true;

break;

}

}

if (!hasFalse)

{

return false;

}

}

return true;

}

static bool IsT0(string function) => function[0] == '0';

static bool IsT1(string function) => function[function.Length - 1] == '1';

static bool IsSelfDualityFunction(string function)

{

bool isSelfDual = true;

int length = function.Length;

for (int i = 0; i < length / 2; i++)

{

if (function[i] == function[length - 1 - i])

{

isSelfDual = false;

break;

}

}

return isSelfDual;

}

static bool Manatonic(string function, string[][] tree)

{

bool isM = true;

int a = 1, b = 0, c = 0;

int count = 0;

while (a < tree.Length && isM)

{

while (b < tree[a - 1].Length && isM)

{

while (c < tree[a].Length && isM)

{

for (int i = 0; i < tree[a][c].Length; i++)

{

if (tree[a - 1][b][i] != tree[a][c][i])

{

count++;

}

}

if (count == 1)

{

char One = function[Convert.ToInt32(tree[a - 1][b], 2)];

char Two = function[Convert.ToInt32(tree[a][c], 2)];

if (One > Two)

{

isM = false;

}

}

c++;

count = 0;

}

b++;

c = 0;

}

a++;

b = 0;

}

return isM;

}

static bool IsManatonicFunction(string function, ref string[][] tree)

{

bool isM = true;

int n = 1;

int N = 2;

while (N != function.Length)

{

n++;

N \*= 2;

}

string[] tableTrue = new string[function.Length];

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

{

string str = "";

string strTable = "";

str = Convert.ToString(i, 2);

for (int j = 0; j < n - str.Length; j++)

{

strTable += '0';

}

for (int j = 0; j < str.Length; j++)

{

strTable += str[j];

}

tableTrue[i] = strTable;

}

int[] counter = new int[n + 1];

foreach (string strTable in tableTrue)

{

int count = 0;

for (int i = 0; i < strTable.Length; i++)

{

if (strTable[i] == '1')

{

count++;

}

}

counter[count]++;

}

tree = new string[n + 1][];

for (int i = 0; i < counter.Length; i++)

{

tree[i] = new string[counter[i]];

}

foreach (string strTable in tableTrue)

{

int count = 0;

for (int i = 0; i < strTable.Length; i++)

{

if (strTable[i] == '1')

{

count++;

}

}

tree[count][counter[count] - 1] = strTable;

counter[count]--;

}

isM = Manatonic(function, tree);

return isM;

}

static bool IsLinearityFunction(string function, string[][] tree)

{

string str;

bool isL = true;

char[][] polinomJigal = new char[tree.Length][];

for (int i = 0; i < tree.Length; i++)

{

polinomJigal[i] = new char[tree[i].Length];

}

polinomJigal[0][0] = function[0];

int a = 1, b = 0, c = 0, d = 0;

int count = 0;

while (a < polinomJigal.Length)

{

str = "" + polinomJigal[0][0];

while (b < polinomJigal[a].Length)

{

string element = tree[a][b];

c = a - 1;

while (c > 0)

{

while (d < polinomJigal[c].Length)

{

string chek = tree[c][d];

for (int i = 0; i < chek.Length; i++)

{

if (chek[i] == '1' && element[i] == '0')

{

count++;

}

}

if (count == 0)

{

str += polinomJigal[c][d];

}

count = 0;

d++;

}

d = 0;

c--;

}

for (int i = 0; i < str.Length; i++)

{

if (str[i] == '1')

{

count++;

}

}

if (count % 2 == 0)

{

str = "" + '0';

}

else

{

str = "" + '1';

}

if ('0' == function[Convert.ToInt32(tree[a][b], 2)])

{

polinomJigal[a][b] = str[0];

}

else

{

if (str[0] == '1')

{

polinomJigal[a][b] = '0';

}

else

{

polinomJigal[a][b] = '1';

}

}

b++;

}

a++;

b = 0;

}

a = 1;

while (a < polinomJigal.Length && isL)

{

while (b < polinomJigal[a].Length && isL)

{

if (a > 1)

{

if (polinomJigal[a][b] == '1')

{

isL = false;

}

}

b++;

}

a++;

b = 0;

}

return isL;

}

static bool[] CreateFunctionClass(string function)

{

var functionClass = new bool[5];

if (function.Length != 1)

{

functionClass[0] = IsT0(function);

functionClass[1] = IsT1(function);

functionClass[2] = IsSelfDualityFunction(function);

string[][] tree = new string[0][];

functionClass[3] = IsManatonicFunction(function, ref tree);

functionClass[4] = IsLinearityFunction(function, tree);

return functionClass;

}

else

{

if (function == "0")

functionClass = new bool[] { true, false, false, true, true };

else

functionClass = new bool[] { false, true, false, true, true };

}

return functionClass;

}

static bool[,] CreateFunctionsSystem(string[] functions)

{

var functionSystem = new bool[functions.Length, 5];

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

{

var functionClass = CreateFunctionClass(functions[i]);

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

{

functionSystem[i, j] = functionClass[j];

}

}

return functionSystem;

}

}

}