Министерство образования и науки Российской Федерации Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌ высшего‌ ‌образования‌

**«Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №3**

Дисциплина: «Дискретная математика и математическая логика»

Тема: Минимизация булевых выражений

Выполнила работу

студентка группы ИВТ-22-2б

Ищенко Д. О.

Проверила

Старший преподаватель кафедры ИТАС

Рустамханова Г. И.

Пермь, 2023

**Постановка задачи**

Необходимо создать программу, которая будет реализовывать минимизацию булевых (логических) выражений с четырьмя переменными. Программа должна иметь следующие возможности:

1. Считывание вектора из 16 логических значений («1» или «0»).
2. Вывод таблицы истинности.
3. Построение совершенной дизъюнктивной нормальной формы (СДНФ).
4. Неполное склеивание (по одному одной переменной) полученной СДНФ.
5. Построение импликантной матрицы в результате склеек.

**Анализ задачи**

Для реализации реализован класс Boolean4 (рис. 1), содержащий поля:

1. int[] function - хранит вектор булевого выражения
2. int[,] truthTable - хранит таблицу истинности (для каждого булевого выражения она одинакова)
3. List<string> minterms - хранит элементарные дизъюнкции после слейки
4. List<string> sdnf - хранит полный набор элементарных дизъюнкций

В классе также реализованы следующие методы:

1. string Variable(int j) - по индексу получает строковое значение переменной. «0» соответствует «x», «1» соответствует «y», «2» соответствует «z», «3» соответствует «w».
2. void TruthTable() - заполнение таблицы истинности двоичными числами от 0 до 15 в порядке возрастания.
3. void PrintTruthTable() - печать таблицы истинности и вектора булевого выражения в консоль.
4. void makeSDNF() - заполнение списка *sdnf*  элементарными дизъюнкциями.
5. void SDNF() - вывод СДНФ в консоль.
6. MatchCollection GetVars(string st) - получение списка переменных из элементарной дизъюнкции любой длины.
7. bool Splice(string first, string second, out string minterm) - склейка двух элементарных дизъюнкций.
8. void Splicing() - процесс склейки с выводом шагов в консоль.
9. void ImplicantMatrix() - вывод импликантной матрицы в консоль.

В классе реализовано одно свойство Function, которое при внесении нового вектора классу заново строит СДНФ и таблицу истинности.

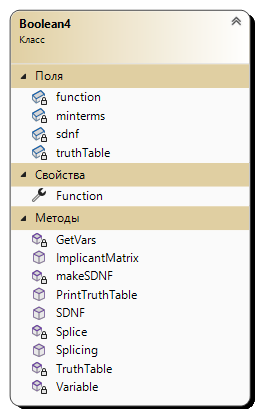


Рисунок 1 - UML-диаграмма класса Boolean4

Таблица истинности заполняется числами от 0 до 15 в двоичной системе счисления с помощью парсера целочисленного типа и конвертера строкового типа.

Совершенная дизъюнктивная нормальная форма (СДНФ) создаётся с помощью функции makeSDNF(). В функции используется функция Variable(), которая ставит в соответствие индексу строковое значение переменной: «0» соответствует «x», «1» соответствует «y», «2» соответствует «z», «3» соответствует «w». В for-loop происходит проход по таблице истинности. Если значение функции для данного набора переменных равно «1», то двоичное число конвертируется в строковый тип с помощью Variable(). Если значение переменной равно «1», то записывается только значение функции Variable(), если значение равно «0», то перед значением функции Variable() записывается символ «-», обозначающий отрицание. Значения элементарных дизъюнкций записываются в список sdnf.

Неполное склеивание (склейка) происходит в функции Splicing(), которая использует функции Splice() и GetVars(). Функция GetVars() разделяет элементарную дизъюнкцию на переменные (с отрицанием или без). В Splice() происходит склеивание двух любых дизъюнкций. Если количество переменных в дизъюнкциях не равно, то функция возвращает значение false. Иначе, дизъюнкции разделяются на переменные, и они сравниваются друг с другом. Если переменные равны, то они записываются в переменную minterm. Далее, если в minterm количество переменных на одну меньше, чем в изначальной дизъюнкции, то возвращается true, иначе false. Переменная minterm со служебным словом out.

Функция Splicing() производит полную склейку по СДНФ до тех пор, пока она возможна. Если функция Splice() даёт истинное значение, то в в переменную minterms записывается результат. Далее, когда все возможные склейки были произведены, склеивание запускается заново. Когда склеивание не происходит, цикл do-while заканчивается, и результат склейки редактируется: остаются только уникальные элементарные дизъюнкции, результат выводится на экран.

Импликантная матрица выводится на экран. Заполнение происходит одновременно. Функция ImplicantMatrix() использует функцию GetVars(). Если в дизъюнкции СДНФ содержатся все переменные, которые есть в дизъюнкции, то в таблице выводится «+».

**Тесты**

*Тест 1*

Введите вектор:

1111111111111111

-> Таблица истинности

x | y | z | w | f

------------------

0 0 0 0 | 1

0 0 0 1 | 1

0 0 1 0 | 1

0 0 1 1 | 1

0 1 0 0 | 1

0 1 0 1 | 1

0 1 1 0 | 1

0 1 1 1 | 1

1 0 0 0 | 1

1 0 0 1 | 1

1 0 1 0 | 1

1 0 1 1 | 1

1 1 0 0 | 1

1 1 0 1 | 1

1 1 1 0 | 1

1 1 1 1 | 1

-> СДНФ

-x-y-z-w + -x-y-zw + -x-yz-w + -x-yzw + -xy-z-w + -xy-zw + -xyz-w + -xyzw + x-y-z-w + x-y-zw + x-yz-w + x-yzw + xy-z-w + xy-zw + xyz-w + xyzw

-> Склейка

1-2: -x-y-z

1-3: -x-y-w

1-5: -x-z-w

1-9: -y-z-w

2-4: -x-yw

2-6: -x-zw

2-10: -y-zw

3-4: -x-yz

3-7: -xz-w

3-11: -yz-w

4-8: -xzw

4-12: -yzw

5-6: -xy-z

5-7: -xy-w

5-13: y-z-w

6-8: -xyw

6-14: y-zw

7-8: -xyz

7-15: yz-w

8-16: yzw

9-10: x-y-z

9-11: x-y-w

9-13: x-z-w

10-12: x-yw

10-14: x-zw

11-12: x-yz

11-15: xz-w

12-16: xzw

13-14: xy-z

13-15: xy-w

14-16: xyw

15-16: xyz

Результат: -x-y-z + -x-y-w + -x-z-w + -y-z-w + -x-yw + -x-zw + -y-zw + -x-yz + -xz-w + -yz-w + -xzw + -yzw + -xy-z + -xy-w + y-z-w + -xyw + y-zw + -xyz + yz-w + yzw + x-y-z + x-y-w + x-z-w + x-yw + x-zw + x-yz + xz-w + xzw + xy-z + xy-w + xyw + xyz

-> Склейка

1-8: -x-y

1-13: -x-z

1-21: -y-z

2-5: -x-y

2-14: -x-w

2-22: -y-w

3-6: -x-z

3-9: -x-w

3-23: -z-w

4-7: -y-z

4-10: -y-w

4-15: -z-w

5-16: -xw

5-24: -yw

6-11: -xw

6-25: -zw

7-12: -yw

7-17: -zw

8-18: -xz

8-26: -yz

9-11: -xz

9-27: z-w

10-12: -yz

10-19: z-w

11-28: zw

12-20: zw

13-18: -xy

13-29: y-z

14-16: -xy

14-30: y-w

15-17: y-z

15-19: y-w

16-31: yw

17-20: yw

18-32: yz

19-20: yz

21-26: x-y

21-29: x-z

22-24: x-y

22-30: x-w

23-25: x-z

23-27: x-w

24-31: xw

25-28: xw

26-32: xz

27-28: xz

29-32: xy

30-31: xy

Результат: -x-y + -x-z + -y-z + -x-w + -y-w + -z-w + -xw + -yw + -zw + -xz + -yz + z-w + zw + -xy + y-z + y-w + yw + yz + x-y + x-z + x-w + xw + xz + xy

-> Склейка

1-14: -x

1-19: -y

2-10: -x

2-20: -z

3-11: -y

3-15: -z

4-7: -x

4-21: -w

5-8: -y

5-16: -w

6-9: -z

6-12: -w

7-22: w

8-17: w

9-13: w

10-23: z

11-18: z

12-13: z

14-24: y

15-18: y

16-17: y

19-24: x

20-23: x

21-22: x

Результат: -x + -y + -z + -w + w + z + y + x

-> Склейка завершена

Результат: -x + -y + -z + -w + w + z + y + x

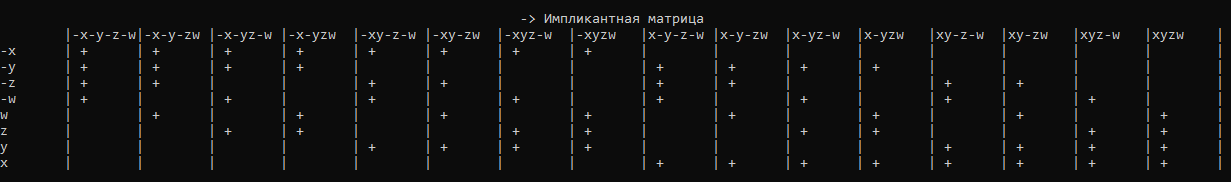


Рисунок 2 - Импликантная матрица для теста 1

Тест 2

Введите вектор:

0000000000000000

-> Таблица истинности

x | y | z | w | f

------------------

0 0 0 0 | 0

0 0 0 1 | 0

0 0 1 0 | 0

0 0 1 1 | 0

0 1 0 0 | 0

0 1 0 1 | 0

0 1 1 0 | 0

0 1 1 1 | 0

1 0 0 0 | 0

1 0 0 1 | 0

1 0 1 0 | 0

1 0 1 1 | 0

1 1 0 0 | 0

1 1 0 1 | 0

1 1 1 0 | 0

1 1 1 1 | 0

Выражение является противоречием: построение СДНФ, склейки и импликантной матрицы невозможно

Тест 3

Введите вектор:

1100010101101000

-> Таблица истинности

x | y | z | w | f

------------------

0 0 0 0 | 1

0 0 0 1 | 1

0 0 1 0 | 0

0 0 1 1 | 0

0 1 0 0 | 0

0 1 0 1 | 1

0 1 1 0 | 0

0 1 1 1 | 1

1 0 0 0 | 0

1 0 0 1 | 1

1 0 1 0 | 1

1 0 1 1 | 0

1 1 0 0 | 1

1 1 0 1 | 0

1 1 1 0 | 0

1 1 1 1 | 0

-> СДНФ

-x-y-z-w + -x-y-zw + -xy-zw + -xyzw + x-y-zw + x-yz-w + xy-z-w

-> Склейка

1-2: -x-y-z

2-3: -x-zw

2-5: -y-zw

3-4: -xyw

Результат: -x-y-z + -x-zw + -y-zw + -xyw + x-yz-w + xy-z-w

-> Склейка завершена

Результат: -x-y-z + -x-zw + -y-zw + -xyw + x-yz-w + xy-z-w

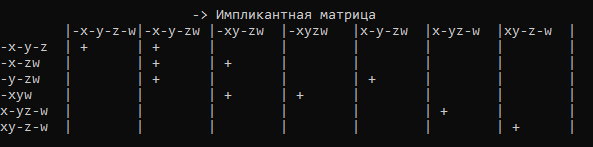


Рисунок 3 - Импликантная матрица для теста 3

Тест 4

Введите вектор:

1100000000001111

-> Таблица истинности

x | y | z | w | f

------------------

0 0 0 0 | 1

0 0 0 1 | 1

0 0 1 0 | 0

0 0 1 1 | 0

0 1 0 0 | 0

0 1 0 1 | 0

0 1 1 0 | 0

0 1 1 1 | 0

1 0 0 0 | 0

1 0 0 1 | 0

1 0 1 0 | 0

1 0 1 1 | 0

1 1 0 0 | 1

1 1 0 1 | 1

1 1 1 0 | 1

1 1 1 1 | 1

-> СДНФ

-x-y-z-w + -x-y-zw + xy-z-w + xy-zw + xyz-w + xyzw

-> Склейка

1-2: -x-y-z

3-4: xy-z

3-5: xy-w

4-6: xyw

5-6: xyz

Результат: -x-y-z + xy-z + xy-w + xyw + xyz

-> Склейка

2-5: xy

3-4: xy

Результат: xy + -x-y-z

-> Склейка завершена

Результат: xy + -x-y-z

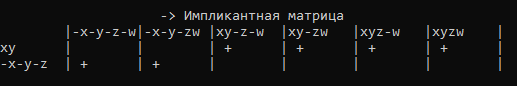


Рисунок 4 - Импликантная матрица для теста 4

**Код программы**

*Boolean4.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

namespace lab3

{

internal class Boolean4

{

private int[] function;

private int[,] truthTable = new int[16, 4];

private List<string> minterms = new List<string> ();

private List<string> sdnf = new List<string>();

public int[] Function

{

get => function;

set

{

function = value;

TruthTable();

makeSDNF();

}

}

private string Variable(int j)

{

if (j == 0)

return "x";

if (j == 1)

return "y";

if (j == 2)

return "z";

else

return "w";

}

private void TruthTable()

{

for (int i = 0; i < 16; ++i)

{

string res = Convert.ToString(i, 2);

for (int j = 0; j < (4 - res.Length); ++j)

{

truthTable[i, j] = 0;

}

int counter = 0;

for (int j = (4 - res.Length); j < 4; j++)

{

truthTable[i, j] = int.Parse(res[counter].ToString());

++counter;

}

}

}

public void PrintTruthTable()

{

Console.WriteLine("\n-> Таблица истинности");

Console.WriteLine(" x | y | z | w | f");

Console.WriteLine("------------------");

for (int i = 0; i < 16; ++i)

{

string result = " " + truthTable[i, 0] + " "

+ truthTable[i, 1] + " " + truthTable[i, 2]

+ " " + truthTable[i, 3] + " | "

+ Function[i];

Console.WriteLine(result);

}

Console.WriteLine();

}

private void makeSDNF()

{

for (int i = 0; i < 16; ++i)

{

if (Function[i] == 1)

{

string temp = "";

for (int j = 0; j < 4; j++)

{

string variable = Variable(j);

if (truthTable[i, j] == 0)

{

temp = String.Concat(temp, "-"+variable);

}

else

{

temp = String.Concat(temp, variable);

}

}

minterms.Add(temp);

sdnf.Add(temp);

}

}

}

public void SDNF()

{

if (sdnf.Count == 0)

{

Console.WriteLine("Выражение является противоречием: построение СДНФ, склейки и импликантной матрицы невозможно");

return;

}

Console.WriteLine("-> СДНФ");

for (int i = 0; i < sdnf.Count - 1; ++i)

{

Console.Write(sdnf[i] + " + ");

}

Console.Write(sdnf[sdnf.Count-1] + "\n");

}

private MatchCollection GetVars(string st)

{

Regex regex = new Regex("-\*[xywz]");

return regex.Matches(st);

}

private bool Splice(string first, string second, out string minterm)

{

int vars = 0;

minterm = "";

MatchCollection firstMatches = GetVars(first);

MatchCollection secondMatches = GetVars(second);

if (firstMatches.Count != secondMatches.Count)

return false;

for (int i = 0; i < firstMatches.Count; ++i)

{

first = firstMatches[i].Value;

second = secondMatches[i].Value;

if (first.IndexOf(second[second.Length-1]) != -1){

if (first == second)

{

vars++;

minterm += firstMatches[i].Value;

}

}

else

{

return false;

}

}

return ((vars == firstMatches.Count-1)&&(firstMatches.Count!=1)) ? true : false;

}

public void Splicing()

{

if (sdnf.Count == 0)

return;

List<string> res;

bool[] used;

bool needContinue;

string result = "";

string minterm = "";

do

{

result = "";

Console.Write("\n-> Склейка");

res = new List<string>();

used = new bool[minterms.Count];

needContinue = false;

for (int i = 0; i < minterms.Count; ++i)

{

for (int j = i + 1; j < minterms.Count; ++j)

{

string first = minterms[i];

string second = minterms[j];

if (Splice(first, second, out minterm))

{

used[i] = true;

used[j] = true;

needContinue = true;

res.Add(minterm);

Console.Write($"\n{i+1}-{j+1}: {minterm}");

}

}

}

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

{

if (!used[i])

{

res.Add(minterms[i]);

}

}

minterms = res;

string[] temp = minterms.ToArray();

var tempRes = temp.Distinct();

minterms = tempRes.ToList();

if (!needContinue)

{

Console.Write(" завершена");

}

Console.WriteLine();

for (int i = 0; i < minterms.Count; ++i)

{

result = result + minterms[i] + " + ";

}

result = result.Substring(0, result.Length - 2);

Console.WriteLine($"Результат: {result}");

} while (needContinue);

}

public void ImplicantMatrix()

{

if (minterms.Count == 0)

return;

bool isInMinterm;

Console.Write("\n");

string implicate = "-> Импликантная матрица";

int tab = ((9 \* (sdnf.Count + 1)) - implicate.Length)/2;

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

{

implicate = String.Concat(" ", implicate);

}

Console.WriteLine(implicate);

Console.Write("{0, -8}|", "");

for (int i = 0; i < sdnf.Count; ++i)

{

Console.Write("{0, -8}|", sdnf[i]);

}

Console.Write("\n");

for (int j = 0; j < minterms.Count; ++j)

{

Console.Write("{0, -8}|", minterms[j]);

for (int i = 0; i < sdnf.Count; ++i)

{

isInMinterm = true;

MatchCollection vars = GetVars(minterms[j]);

foreach (Match m in vars)

{

int index = sdnf[i].IndexOf(m.Value);

if (index == -1)

{

isInMinterm = false;

}

else if (index != 0)

{

if (sdnf[i][index-1] == '-')

{

isInMinterm = false;

}

}

}

if (isInMinterm)

{

Console.Write("{0, -8}|", " +");

}

else

{

Console.Write("{0, -8}|", "");

}

}

Console.Write("\n");

}

}

}

}

*Program.cs*

using System;

namespace lab3

{

internal class Program

{

static void Main(string[] args)

{

Boolean4 work;

while (true)

{

Console.Clear();

work = new Boolean4();

Console.WriteLine("Введите вектор: ");

string input = Console.ReadLine();

int[] vec = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

for (int i = 0; i < 16; i++)

{

vec[i] = int.Parse(input[i].ToString());

}

work.Function = vec;

work.PrintTruthTable();

work.SDNF();

work.Splicing();

work.ImplicantMatrix();

Console.Write("\nНажмите, чтобы повторить...");

Console.ReadKey(true);

}

}

}

}