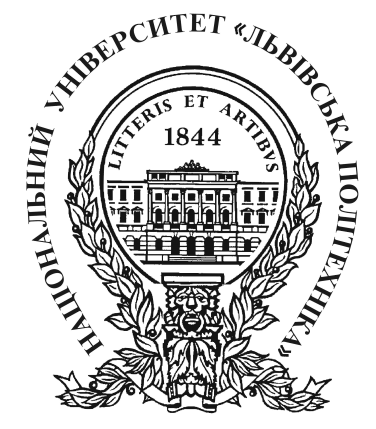
МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”



Лабораторна робота №7

з курсу «Інженерія даних та знань»

на тему: **«Розробка інтерпретатора логіки висловлень»**

Виконав:

ст. гр. ITIT-12

Галах А.Б. Прийняла:

Рибчак З.Л.

Львів 2017

1. **Мета роботи**

Мета лабораторної роботи полягає у вивченні логіки та застосування її для розв’язування складних логічних задач.

1. **Індивідуальне завдання**

**Варіант 3**

1. Розробити додаток для обробки виразів логіки висловлень та інтерфейс для введення виразів з урахуванням операторів логічних операцій.

Вихідні дані. Нехай p та q - висловлення: p - сьогодні холодно; q – падає сніг. Задати висловлення а) - ж), використовуючи p , q та логічні операції (логічні зв’язки) а) сьогодні холодно, але сніг не падає; б) сьогодні падає сніг або холодно (або одне і друге); в) сьогодні холодно або падає сніг, але немає снігу, якщо холодно;

2. Побудувати істиннісні таблиці для кожного із складних висловлень:

а)

б)

в)

3. Показати, що логічні зв’язки , , , , , де F - стале значення false, мають ту саму істиннісну таблицю, що і імплікація

4. Виконати побітові операції OR, AND та XOR для таких пар бітових рядків: а) 0001110001, 1001001000;

5. Обчислити значення висловлення f у нечіткій логіці, якщо p, q та r мають задані значення

1. **Програмна реалізація**

Користувач задає атоми і їх значення, після чого за допомогою елементів інтерфейсу вводить буле вий вираз. Програма перетворює заданий вираз у постфіксний запис, після чого його можна обчислити на основі заданих значень атомів та отримати таблицю істинності даного виразу.

функції для отримання постфіксного запису з інфіксного:

private List<Symbol> GetPostfix()

{

\_postfixList = new List<Symbol>();

Stack<Symbol> operatorStack = new Stack<Symbol>();

foreach (var s in \_symbolList)

{

ApplyRules(s, operatorStack);

}

while (operatorStack.Count > 0)

{

\_postfixList.Add(operatorStack.Pop());

}

return \_postfixList;

}

private void ApplyRules(Symbol s, Stack<Symbol> operatorStack)

{

if (s.IsAtom())

{

\_postfixList.Add(s);

}

else

{

if (s.OperationType == OperationType.LBracket)

{

operatorStack.Push(s);

}

else if (s.OperationType == OperationType.RBracket)

{

Symbol current;

do

{

current = operatorStack.Pop();

if (current.OperationType != OperationType.LBracket)

{

\_postfixList.Add(current);

}

}

while (current.OperationType != OperationType.LBracket);

}

else if (operatorStack.Count == 0 || operatorStack.Peek().OperationType == OperationType.LBracket)

{

operatorStack.Push(s);

}

else if (operatorStack.Peek().OperationType == s.OperationType)

{

\_postfixList.Add(operatorStack.Pop());

ApplyRules(s, operatorStack);

}

else if (s.OperationType == OperationType.Not)

{

operatorStack.Push(s);

}

else if (operatorStack.Peek().OperationType != OperationType.Not &&

(s.OperationType == OperationType.And))

{

operatorStack.Push(s);

}

else if (operatorStack.Peek().OperationType != OperationType.Not &&

operatorStack.Peek().OperationType != OperationType.And &&

(s.OperationType == OperationType.Or))

{

operatorStack.Push(s);

}

else if (operatorStack.Peek().OperationType != OperationType.Not &&

operatorStack.Peek().OperationType != OperationType.And &&

operatorStack.Peek().OperationType != OperationType.Or &&

(s.OperationType == OperationType.Imp))

{

operatorStack.Push(s);

}

else

{

\_postfixList.Add(operatorStack.Pop());

ApplyRules(s, operatorStack);

}

}

}

функція для знаходження результату булевого виразу:

public bool GetResult()

{

\_postfixList = GetPostfix();

string str = GetStringFromSymbolList(\_postfixList);

Stack<bool> calculationStack = new Stack<bool>();

foreach (var s in \_postfixList)

{

if (s.IsAtom())

{

calculationStack.Push(s.Atom.Value);

}

else if (calculationStack.Count > 0)

{

if (s.OperationType == OperationType.Not)

{

bool left = !calculationStack.Pop();

calculationStack.Push(left);

}

else if (s.OperationType == OperationType.And)

{

bool right = calculationStack.Pop();

bool left = calculationStack.Pop();

calculationStack.Push(left && right);

}

else if (s.OperationType == OperationType.Or)

{

bool right = calculationStack.Pop();

bool left = calculationStack.Pop();

calculationStack.Push(left || right);

}

else if (s.OperationType == OperationType.Equals)

{

bool right = calculationStack.Pop();

bool left = calculationStack.Pop();

calculationStack.Push(left == right);

}

else if (s.OperationType == OperationType.Imp)

{

bool right = calculationStack.Pop();

bool left = !calculationStack.Pop();

calculationStack.Push(left || right);

}

}

}

bool result = calculationStack.Pop();

return result;

}

функція для отримання таблиці істинності:

public bool[,] GetTruthTable()

{

List<Atom> atoms = GetAtoms();

int columnCount = atoms.Count + 1;

int rowCount = (int)Math.Pow(2, atoms.Count);

bool[,] table = new bool[rowCount, columnCount];

int divisor = 1;

for (int j = 0; j < columnCount - 1; j++)

{

divisor \*= 2;

int counter = 1;

bool cellValue = true;

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

{

if (counter > rowCount / divisor)

{

counter = 1;

cellValue = !cellValue;

}

table[i, j] = cellValue;

counter++;

}

}

bool[] savedValues = new bool[atoms.Count];

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

savedValues[i] = atoms[i].Value;

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

{

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

{

atoms[j].Value = table[i,j];

}

table[i, columnCount - 1] = GetResult();

}

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

atoms[i].Value = savedValues[i];

return table;

}

Також програма може працювати з бітовими рядками введеними користувачем.

команда для знаходження побітового «і»:

public class BitwiseAndCommand : ICommand

{

public BitwiseAndCommand(MainWindowViewModel viewModel)

{

this.ViewModel = viewModel;

}

public bool CanExecute(object parameter)

{

return true;

}

public void Execute(object parameter)

{

if (ViewModel.BitRow1 == "")

{

ViewModel.BitRow1 = "0";

}

if (ViewModel.BitRow2 == "")

{

ViewModel.BitRow2 = "0";

}

int num1 = Convert.ToInt32(ViewModel.BitRow1, 2);

int num2 = Convert.ToInt32(ViewModel.BitRow2, 2);

ViewModel.BitwiseResult = Convert.ToString(num1 & num2, 2);

}

public MainWindowViewModel ViewModel { get; set; }

public event EventHandler CanExecuteChanged;

}

команда для знаходження побітового «або»:

public class BitwiseOrCommand : ICommand

{

public BitwiseOrCommand(MainWindowViewModel viewModel)

{

this.ViewModel = viewModel;

}

public bool CanExecute(object parameter)

{

return true;

}

public void Execute(object parameter)

{

if (ViewModel.BitRow1 == "")

{

ViewModel.BitRow1 = "0";

}

if (ViewModel.BitRow2 == "")

{

ViewModel.BitRow2 = "0";

}

int num1 = Convert.ToInt32(ViewModel.BitRow1, 2);

int num2 = Convert.ToInt32(ViewModel.BitRow2, 2);

ViewModel.BitwiseResult = Convert.ToString(num1 | num2, 2);

}

public MainWindowViewModel ViewModel { get; set; }

public event EventHandler CanExecuteChanged;

}

команда для знаходження побітового «виключного або»:

public class BitwiseXorCommand : ICommand

{

public BitwiseXorCommand(MainWindowViewModel viewModel)

{

this.ViewModel = viewModel;

}

public bool CanExecute(object parameter)

{

return true;

}

public void Execute(object parameter)

{

if (ViewModel.BitRow1 == "")

{

ViewModel.BitRow1 = "0";

}

if (ViewModel.BitRow2 == "")

{

ViewModel.BitRow2 = "0";

}

int num1 = Convert.ToInt32(ViewModel.BitRow1, 2);

int num2 = Convert.ToInt32(ViewModel.BitRow2, 2);

ViewModel.BitwiseResult = Convert.ToString(num1 ^ num2, 2);

}

public MainWindowViewModel ViewModel { get; set; }

public event EventHandler CanExecuteChanged;

}

1. **Результат виконання програми**

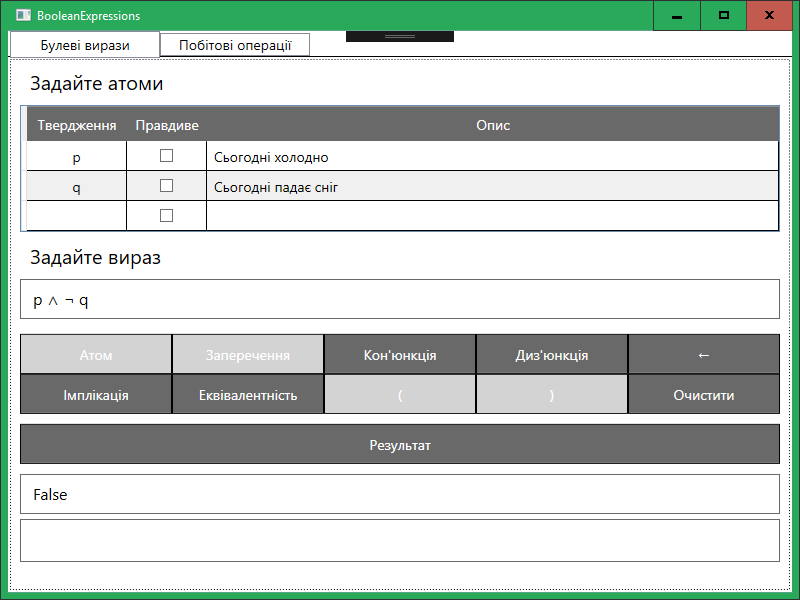
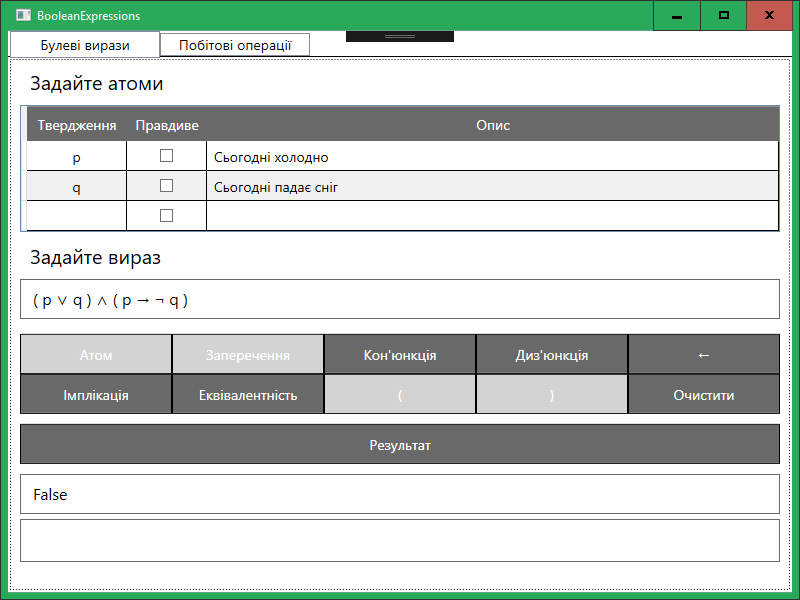
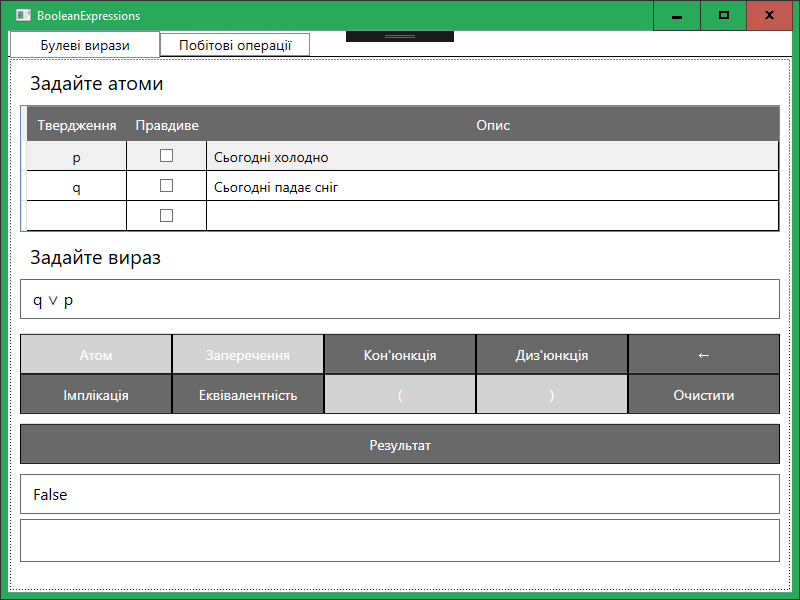
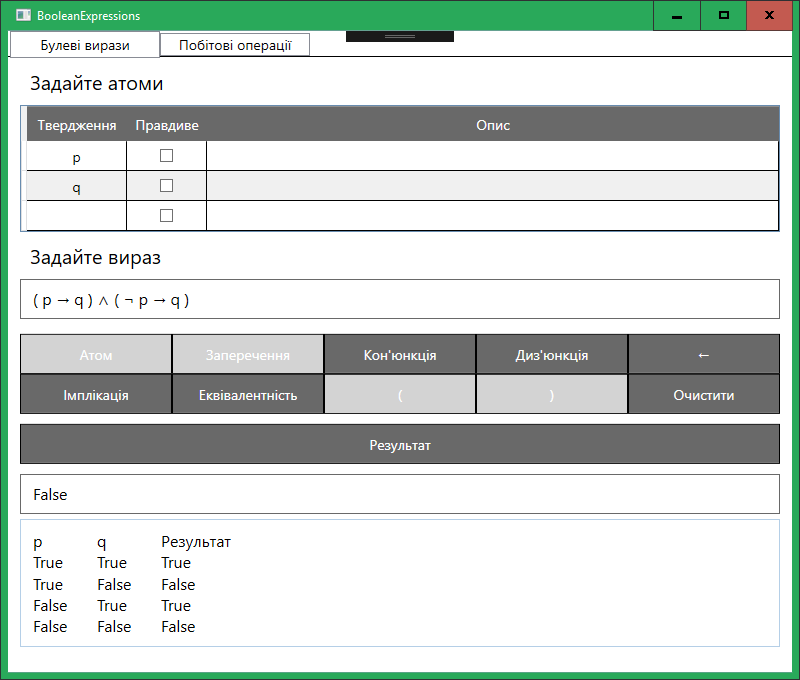
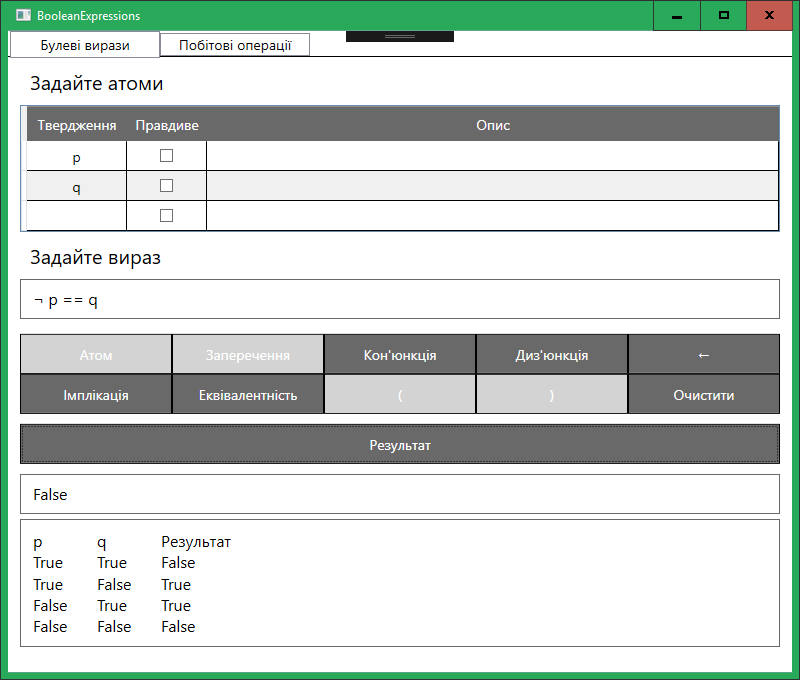


Рис.1-3 Задання виразів відповідно до завдання 1



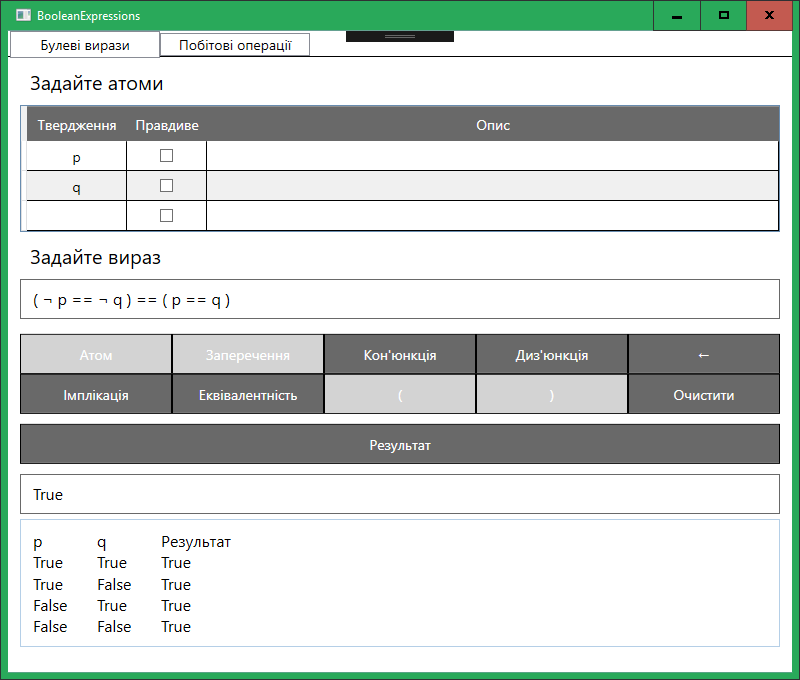


Рис.4-6 Таблиці істинності для заданих виразів у завданні 2

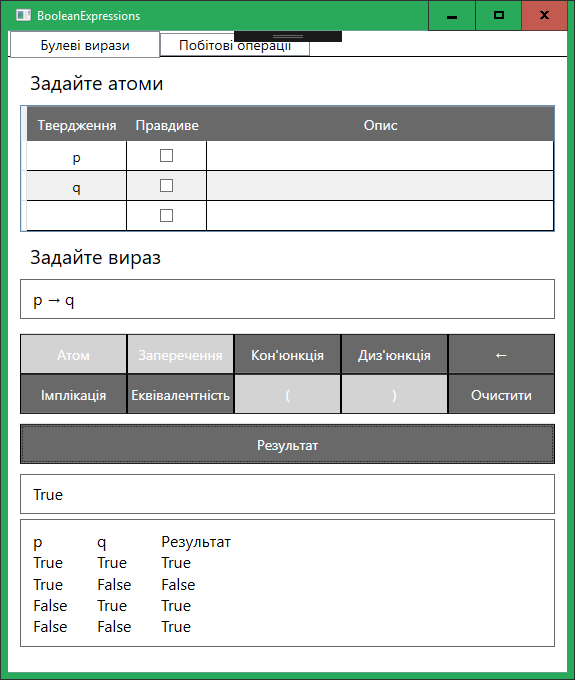
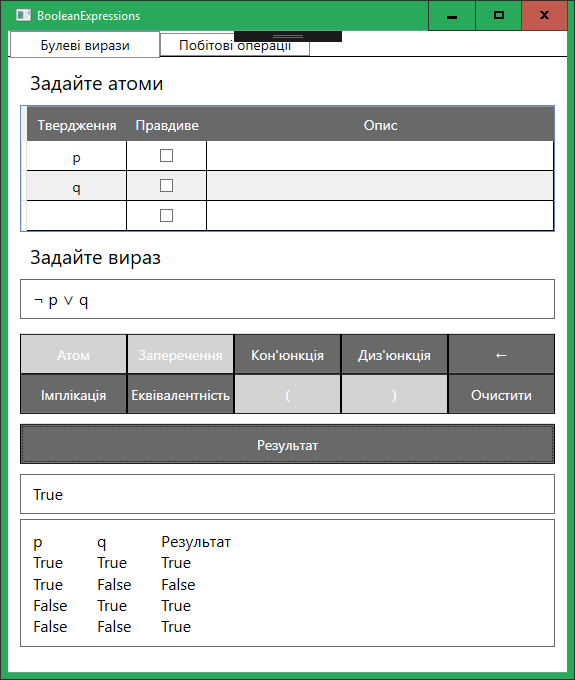
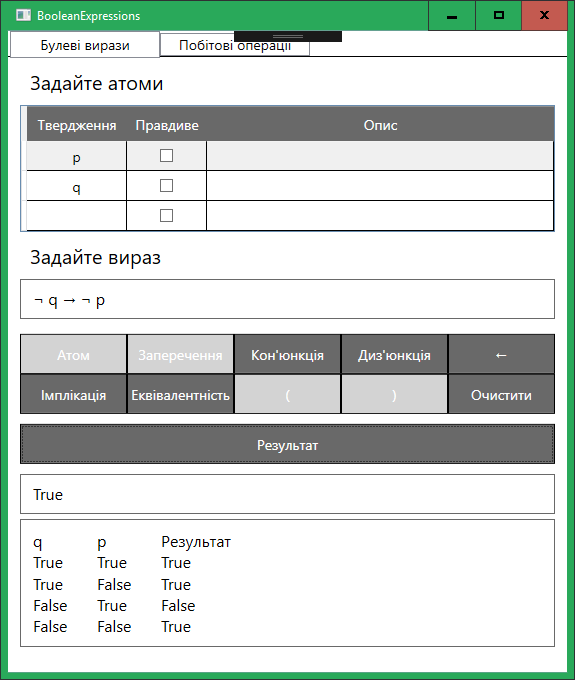
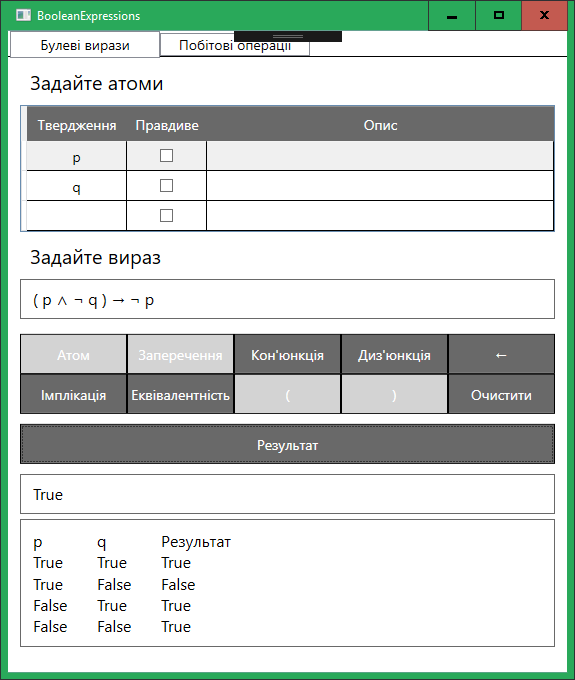
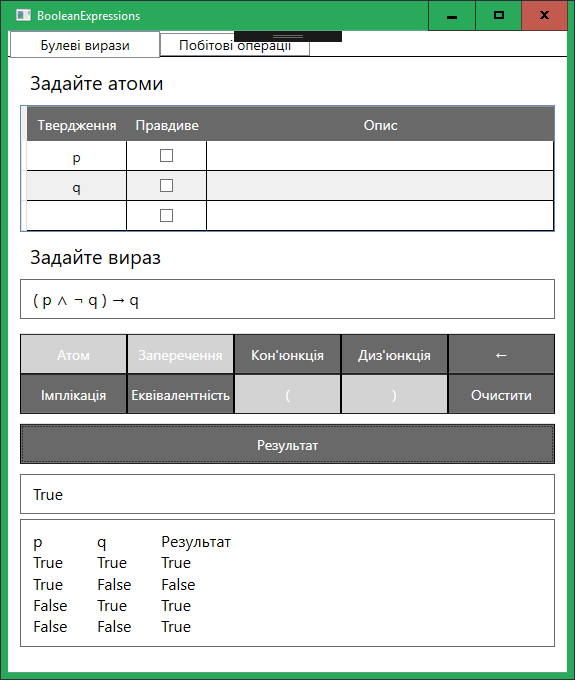
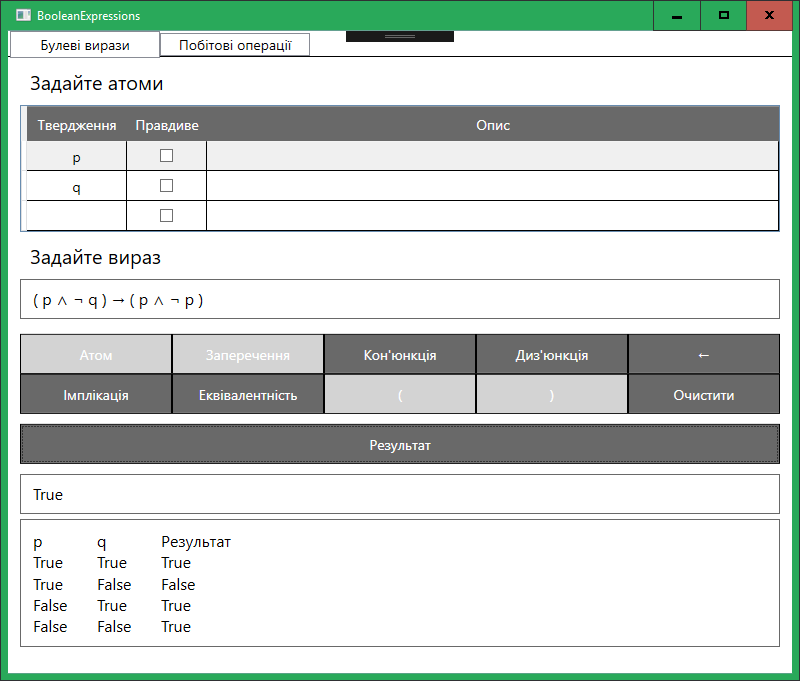


Рис.7-12 Істинні таблиці для виразів на рис.7-11 співпадають з таблицею істинності на рис.12

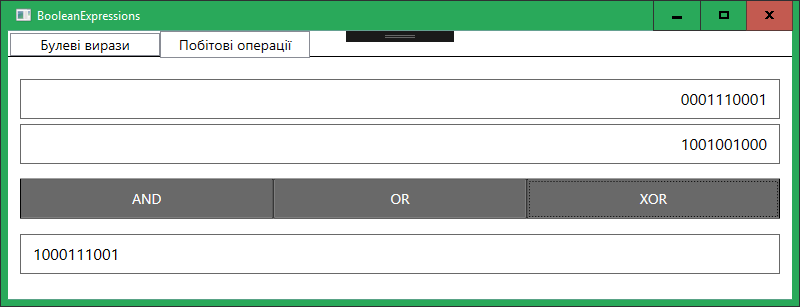
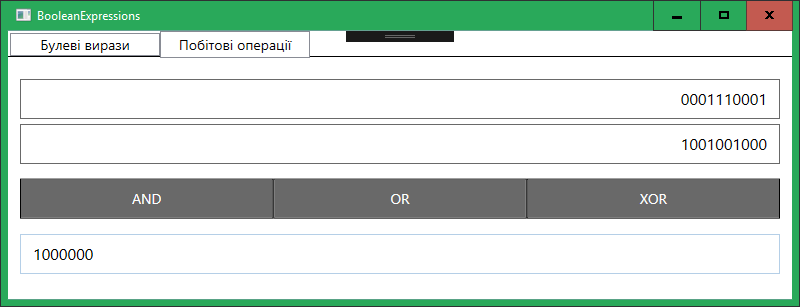
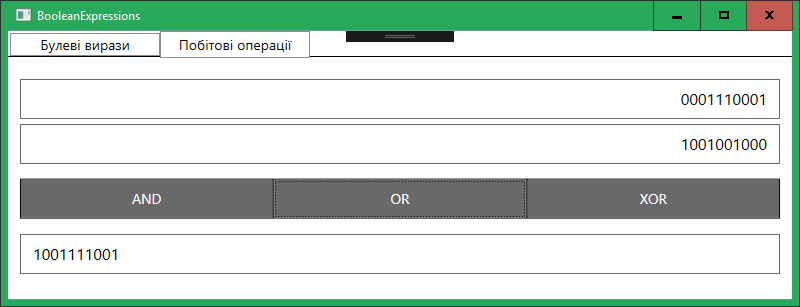


Рис.13-15 Виконання програмою побітових операцій «і», «або» та «виключне або» для завдання 4

1. **Висновок**

У цій лабораторній роботі я вивчив булеву логіку та застосування її для розв’язування складних логічних задач. Написав програму для роботи з булевими виразами та бітовими рядками.