РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

(БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 2 Логика предикатов

по дисциплине: Математическая логика и теория алгоритмов

Выполнил: ст. группы ПВ-212

Гринченко Алина Сергеевна

Проверили:

Рязанов Юрий Дмитриевич

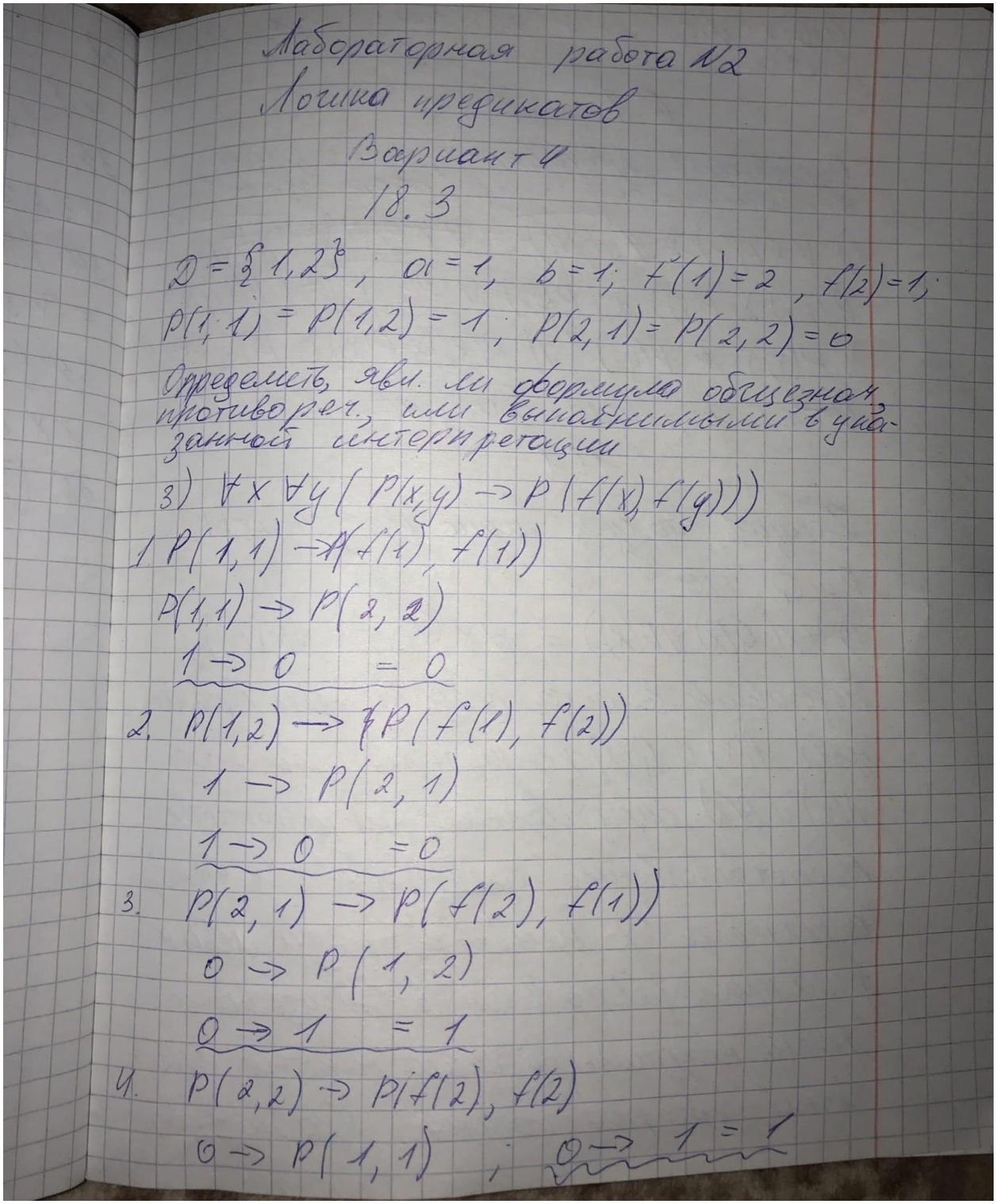
Осипов Олег Васильевич

Белгород 2022 г.

Лабораторная работа № 2 Логика предикатов

Цель лабораторной работы: изучить логику предикатов

ТЕОРИТИЧЕСКАЯ ЧАСТЬ:



Ответ: формула является противоречивой.

6.1

Определить: противореч., общезначим., выполнимость след. формулы:

1) $\forall x P(x) \& \overline{\exists y P(y)} = \forall x P(x) \& \overline{\forall y P(y)}$

$= P(x_1) \& P(x_2) \& [\overline{P(y_1) \vee P(y_2)}] = [P(x_1) \& P(x_2) \& P(y_1)] \vee [P(x_1) \& P(x_2) \& \overline{P(y_2)}] = 1$

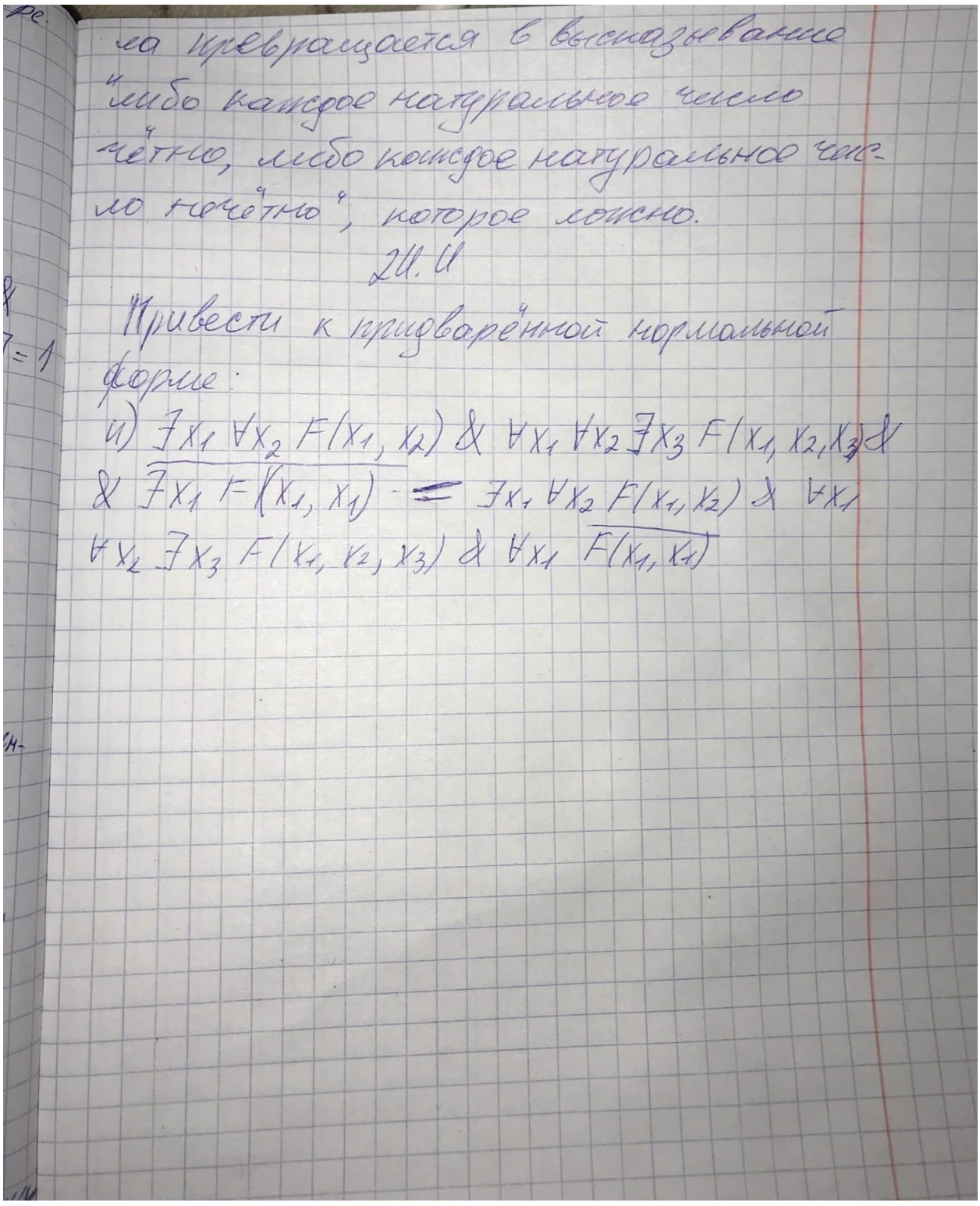
Ответ: формула - общезначимая

11.1

Доказать, что сущ. такие предикаты P и Q, что:

1) $\forall x (Q(x) \vee P(x)) \neq \forall x Q(x) \vee \forall x P(x)$

Подставим вместо предикатных переменных x $P(x)$ и $Q(x)$ конкретные предикаты $A(x)$ и $B(x)$, определённые на множестве N соответственно, где $A(x)$ есть "x - чётно", а $B(x)$ есть "x - нечётно". Тогда левая формула превратится в высказывание "каждое натуральное число нечётно, либо чётно", которое истинно. Правая форму-



ПРАКТИЧЕСКАЯ ЧАСТЬ:

Вариант 1. Разработать программу, способную считывать несколько формул-посылок логики высказываний и выводить на экран все формулы-следствия из этих посылок.

Код программы:

```
#include <iostream>
#include <string>
#include <set>
#include <map>
#include <stack>
```

```cpp
#include <queue>
#include <cassert>
#include <cstdlib>
#include <windows.h>

using namespace std;

// Объявление типов.
// Токен (лексема):
typedef char Token;

// Стек токенов:
typedef stack<Token> Stack;

// Последовательность токенов:
typedef queue<Token> Queue;

// Множество различных токенов:
typedef set<Token> Set;

// Таблица значений переменных:
typedef map<Token, Token> Map;

// Пара переменная—значение:
typedef pair<Token, Token> VarVal;

// Строка символов:
typedef string String;

// Является ли токен числом?
inline bool isNumber(Token t) {
    return t == '0' || t == '1';
}

// Является ли токен переменной?
inline bool isVariable(Token t) {
    return (t >= 'A' && t <= 'Z') || (t >= 'a' && t <= 'z');
}

// Является ли токен операцией?
inline bool isOperation(Token t) {
    return (t == '|' || t == '&' || t == '-' || t == '>' || t == '~');
}

// Является ли токен открывающей скобкой?
inline bool isOpeningPar(Token t) {
    return t == '(';
}

// Является ли токен закрывающей скобкой?
inline bool isClosingPar(Token t) {
    return t == ')';
}

// Вернуть величину приоритета операции
// (чем больше число, тем выше приоритет)
inline int priority(Token op) {
    assert(isOperation(op));
    int res = 0;
    switch (op) {
        case '-':
            // Отрицание — наивысший приоритет
            res = 5;
            break;
```

```
        case '&':
            // Конъюнкция
            res = 4;
            break;
        case '|':
            // Дизъюнкция
            res = 3;
            break;
        case '>':
            // Импликация
            res = 2;
            break;
        case '~':
            // Эквивалентность — наинизший приоритет
            res = 1;
            break;
    }
    return res;
}

// Преобразовать последовательность токенов,
// представляющих выражение в инфиксной записи,
// в последовательность токенов, представляющих
// выражение в обратной польской записи
// (алгоритм Дейкстры «Сортировочная станция»)
Queue infixToPostfix(Queue input) {
    // Выходная последовательность (очередь вывода):
    Queue output;
    // Рабочий стек:
    Stack s;
    // Текущий входной токен:
    Token t;
    // Пока есть токены во входной последовательности:
    while (!input.empty()) {
        // Получить токен из начала входной последовательности
        t = input.front();
        input.pop();
        // Если токен — число или переменная, то:
        if (isNumber(t) || isVariable(t)) {
            // Добавить его в очередь вывода
            output.push(t);
            // Если токен — операция op1, то:
        } else if (isOperation(t)) {
            // Пока на вершине стека присутствует токен-операция op2
            // и у op1 приоритет меньше либо равен приоритету op2, то:
            while (!s.empty() && isOperation(s.top())
                   && priority(t) <= priority(s.top())
                    ) {
                // переложить op2 из стека в выходную очередь
                output.push(s.top());
                s.pop();
            }
            // Положить op1 в стек
            s.push(t);
            // Если токен — открывающая скобка, то:
        } else if (isOpeningPar(t)) {
            // Положить его в стек
            s.push(t);
            // Если токен — закрывающая скобка, то:
        } else if (isClosingPar(t)) {
            // Пока токен на вершине стека не является открывающей скобкой:
            while (!s.empty() && !isOpeningPar(s.top())) {
                // Перекладывать токены-операции из стека
                // в выходную очередь
```

```
                assert(isOperation(s.top()));
                output.push(s.top());
                s.pop();
            }
            // Если стек закончился до того,
            // как был встречен токен-«открывающая скобка», то:
            if (s.empty()) {
                // В выражении пропущена открывающая скобка
                throw String("Пропущена открывающая скобка!");
            } else {
                // Иначе выкинуть открывающую скобку из стека
                // (но не добавлять в очередь вывода)
                s.pop();
            }
        } else {
            // В остальных случаях входная последовательность
            // содержит токен неизвестного типа
            String msg("Неизвестный символ \'");
            msg += t + String("\'!");
            throw msg;
        }
    }

    // Токенов на входе больше нет, но ещё могут остаться токены в стеке.
    // Пока стек не пустой:
    while (!s.empty()) {
        // Если токен на вершине стека — открывающая скобка, то:
        if (isOpeningPar(s.top())) {
            // В выражении присутствует незакрытая скобка
            throw String("Незакрытая скобка!");
        } else {
            // Иначе переложить токен-операцию из стека в выходную очередь
            assert(isOperation(s.top()));
            output.push(s.top());
            s.pop();
        }
    }
    // Конец алгоритма.
    // Выдать полученную последовательность
    return output;
}

// Напечатать последовательность токенов
void printSequence(Queue q) {
    while (!q.empty()) {
        cout << q.front();
        q.pop();
    }
    cout << endl;
}

// Является ли символ пробельным?
inline bool isSpace(char c) {
    return c <= ' ';
}

// Если символ — маленькая буква, преобразовать её в большую,
// иначе просто вернуть этот же символ
inline char toUpperCase(char c) {
    if (c >= 'a' && c <= 'z') {
        return c - 'a' + 'A';
    } else {
        return c;
    }
```

```
}

// Преобразовать строку с выражением в последовательность токенов
// (лексический анализатор)
Queue stringToSequence(const String &s) {
    Queue res;
    for (char i: s) {
        if (!isSpace(i)) {
            res.push(toUpperCase(i));
        }
    }
    return res;
}

// Напечатать сообщение об ошибке
inline void printErrorMessage(const String &err) {
    cerr << "*** ОШИБКА! " << err << endl;
}

// Ввести выражение с клавиатуры
inline String inputExpr() {
    String expr;
    cout << "Формула логики высказываний: ";
    getline(cin, expr);
    return expr;
}

// Выделить из последовательности токенов переменные
unsigned getVariables(Queue s, Set &res) {
    //Set res;
    unsigned c = 0;
    while (!s.empty()) {
        if (isVariable(s.front()) && res.count(s.front()) == 0) {
            res.insert(s.front());
            c++;
        }
        s.pop();
    }
    return c;
}

// Получить значения переменных с клавиатуры
Map inputVarValues(const Set &var) {
    Token val;
    Map res;
    for (char i: var) {
        do {
            cout << i << " = ";
            cin >> val;
            if (!isNumber(val)) {
                cerr << "Введите 0 или 1!" << endl;
            }
        } while (!isNumber(val));
        res.insert(VarVal(i, val));
    }
    return res;
}

// Заменить переменные их значениями
Queue substValues(Queue expr, Map &varVal) {
    Queue res;
    while (!expr.empty()) {
        if (isVariable(expr.front())) {
            res.push(varVal[expr.front()]);
```

```
        } else {
            res.push(expr.front());
        }
        expr.pop();
    }
    return res;
}

// Является ли операция бинарной
inline bool isBinOp(Token t) {
    return t == '&' || t == '|' || t == '>' || t == '~';
}

// Является ли операция унарной
inline bool isUnarOp(Token t) {
    return t == '-';
}

// Получить bool-значение токена-числа (true или false)
inline bool logicVal(Token x) {
    assert(isNumber(x));
    return x == '1';
}

// Преобразовать bool-значение в токен-число
inline Token boolToToken(bool x) {
    if (x) {
        return '1';
    } else {
        return '0';
    }
}

// Вычислить результат бинарной операции
inline Token evalBinOp(Token a, Token op, Token b) {
    assert(isNumber(a) && isBinOp(op) && isNumber(b));
    bool res;
    // Получить bool-значения операндов
    bool left = logicVal(a);
    bool right = logicVal(b);
    switch (op) {
        case '&':
            // Конъюнкция
            res = left && right;
            break;
        case '|':
            // Дизъюнкция
            res = left || right;
            break;
        case '>':
            // Импликация
            res = !left || right;
            break;
        case '~':
            // Эквивалентность
            res = (!left || right) && (!right || left);
            break;
    }
    return boolToToken(res);
}

// Вычислить результат унарной операции
inline Token evalUnarOp(Token op, Token a) {
    assert(isUnarOp(op) && isNumber(a));
```

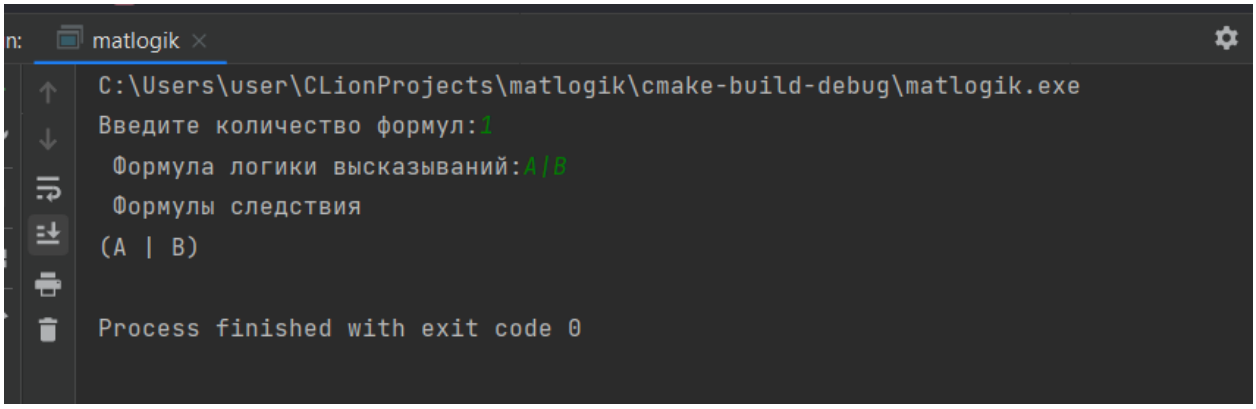
```
    bool res = logicVal(a);
    switch (op) {
        case '-':
            // Отрицание
            res = !res;
            break;
    }
    return boolToToken(res);
}

// Вычислить значение операции, модифицируя стек.
// Результат помещается в стек
void evalOpUsingStack(Token op, Stack &s) {
    assert(isOperation(op));
    // Если операция бинарная, то:
    if (isBinOp(op)) {
        // В стеке должны быть два операнда
        if (s.size() >= 2) {
            // Если это так, то извлекаем правый операнд-число
            Token b = s.top();
            if (!isNumber(b)) {
                throw String("Неверное выражение!");
            }
            s.pop();
            // Затем извлекаем левый операнд-число
            Token a = s.top();
            if (!isNumber(a)) {
                throw String("Неверное выражение!");
            }
            s.pop();
            // Помещаем в стек результат операции
            s.push(evalBinOp(a, op, b));
        } else {
            throw String("Неверное выражение!");
        }
        // Иначе операция унарная
    } else if (isUnarOp(op) && !s.empty()) {
        // Извлекаем операнд
        Token a = s.top();
        if (!isNumber(a)) {
            throw String("Неверное выражение!");
        }
        s.pop();
        // Помещаем в стек результат операции
        s.push(evalUnarOp(op, a));
    } else {
        throw String("Неверное выражение!");
    }
}

// Вычислить значение выражения, записанного в обратной польской записи
Token evaluate(Queue expr) {
    // Рабочий стек
    Stack s;
    // Текущий токен
    Token t;
    // Пока входная последовательность содержит токены:
    while (!expr.empty()) {
        // Считать очередной токен
        t = expr.front();
        assert(isNumber(t) || isOperation(t));
        expr.pop();
        // Если это число, то:
        if (isNumber(t)) {
```

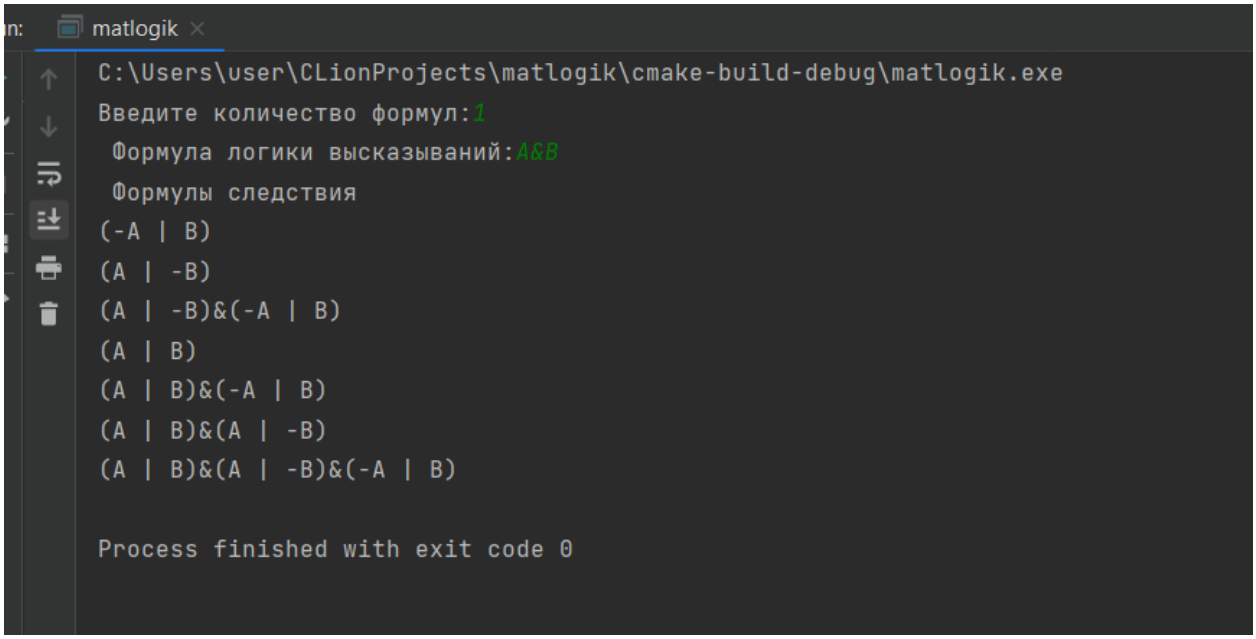
```
            // Поместить его в стек
            s.push(t);
            // Если это операция, то:
        } else if (isOperation(t)) {
            // Вычислить её, модифицируя стек
            // (результат также помещается в стек)
            evalOpUsingStack(t, s);
        }
    }
    // Результат — единственный элемент в стеке
    if (s.size() == 1) {
        // Вернуть результат
        return s.top();
    } else {
        throw String("Неверное выражение!");
    }
}

// Вывести результат вычисления на экран
void printResult(Token r) {
    assert(isNumber(r));
    cout << "Значение выражения: " << r << endl;
}

//построение матрицы СКНФ формул
int SKNF(Queue expr, int **matr, Set vars, unsigned countVars) {
    unsigned mask;
    unsigned lim = 1 << countVars;
    unsigned c = 0;
    for (size_t i = 0; i < lim; i++) {
        Map varVals;
        mask = lim;
        auto k = vars.begin();
        for (size_t j = 0; j < countVars; j++) {
            mask >>= 1;
            bool t = i & mask;
            Token T = boolToToken(t);
            varVals.insert(VarVal(*k, T));
            k++;
        }
        Queue sExpr = substValues(expr, varVals);
        Token res = evaluate(sExpr);
        if (res == '0') {
            mask = lim;
            for (size_t j = 0; j < countVars; j++) {
                mask >>= 1;
                bool t = i & mask;
                if (t) {
                    (*matr)[j] = -1;
                } else
                    (*matr)[j] = 0;
            }
            matr++;
            c++;
        }
    }
    return c;
}

void writeExp(int **a, int nums, unsigned n, Set vars) {
    set<int>::iterator it;
    for (int i = 0, it = *vars.begin(); i < n; i++, it++) {
        if (a[nums][i] == -1)
            cout << '-';
```

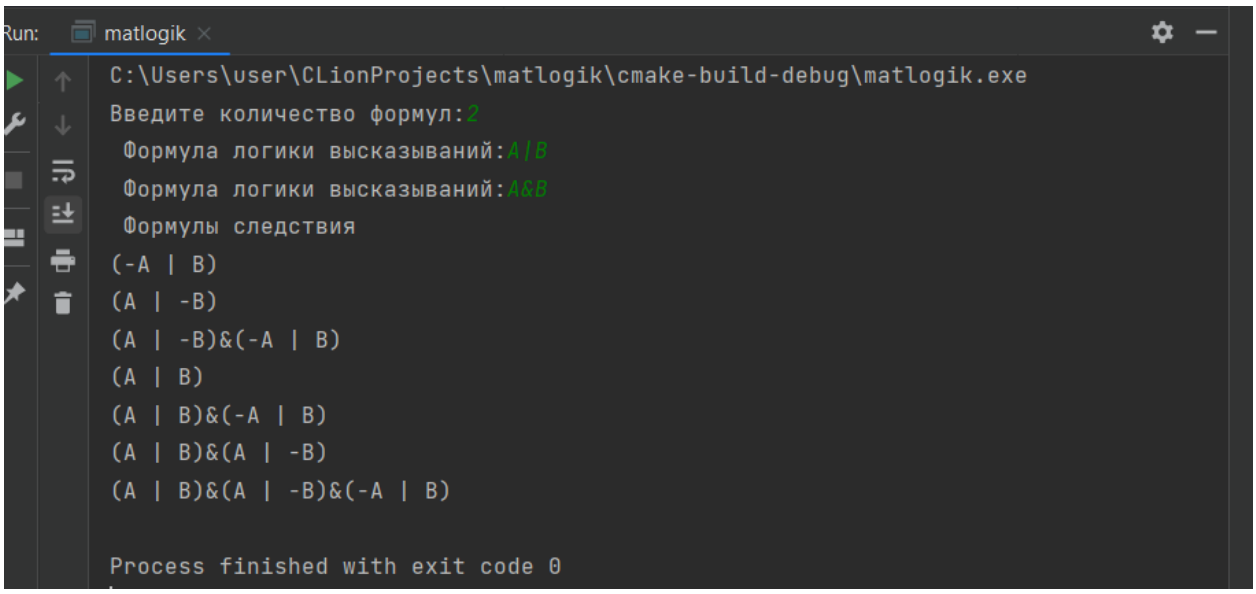
```
        cout << (Token) (it) << " | ";
    }
    cout << "\b\b\b";
}

//вывод следствия формулы
void writeConseq(int *D, int **matr, Set vars, unsigned countVars, unsigned
countString) {
    for (int i = 0; i < countString; i++) {
        if (D[i] != 0) {
            cout << "(";
            writeExp(matr, i, countVars, vars);
            cout << ")&";
        }
    }
    cout << "\b \n";
}

//построение двоичных векторов
static void subsets_inner(int i, unsigned n, Set vars, int **matr, unsigned
m) {
    static int D[100];
    for (int x = 0; x <= 1; x++) {
        D[i] = x;
        if (i == n - 1)
            writeConseq(D, matr, vars, m, n);
        else
            subsets_inner(i + 1, n, vars, matr, m);
    }
}

void subsets(unsigned n, unsigned m, int **matr, Set vars) {
    subsets_inner(0, n, vars, matr, m);
}

// Главная программа
int main() {
    setlocale(LC_ALL, "Russian");
    try {
        cout << "Введите количество формул: ";
        int n;
        String expres;
        cin >> n;
        getchar();
        //ввод формул
        for (int i = 0; i < n - 1; i++) {
            String exp = inputExpr();
            expres += '(' + exp + ")&";
        }
        String exp = inputExpr();
        expres += '(' + exp + ')';
        // Преобразовать выражение в последовательность токенов
        Queue input = stringToSequence(expres);
        // Преобразовать последовательность токенов в ОПЗ
        Queue output = infixToPostfix(input);
        Set vars;
        unsigned countVars = getVariables(output, vars);
        unsigned t = 1 << countVars;
        //создание матрицы
        int **matr = new int *[t];
        for (int i = 0; i < t; i++) {
            matr[i] = new int[countVars];
        }
        //построение матрицы СКНФ
```

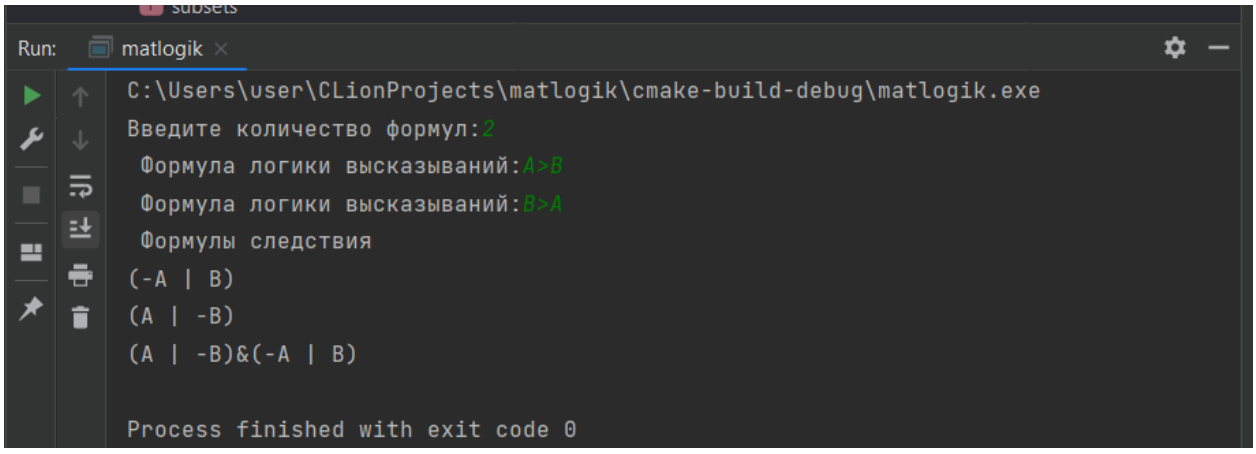
```
        unsigned k = SKNF(output, matr, vars, countVars);
        cout << "Формулы следствия:";
        subsets(k, countVars, matr, vars);
        //удаление матрицы
        for (int i = 0; i < t; i++) {
            delete[] matr[i];
        }
        delete[] matr;
    }
    catch (const String &err) {
        // Если возникла ошибка, вывести сообщение
        printErrorMessage(err);
        // И выйти из программы с неудачным кодом завершения
        exit(1);
    }
    // Конец программы
    return 0;
}
```

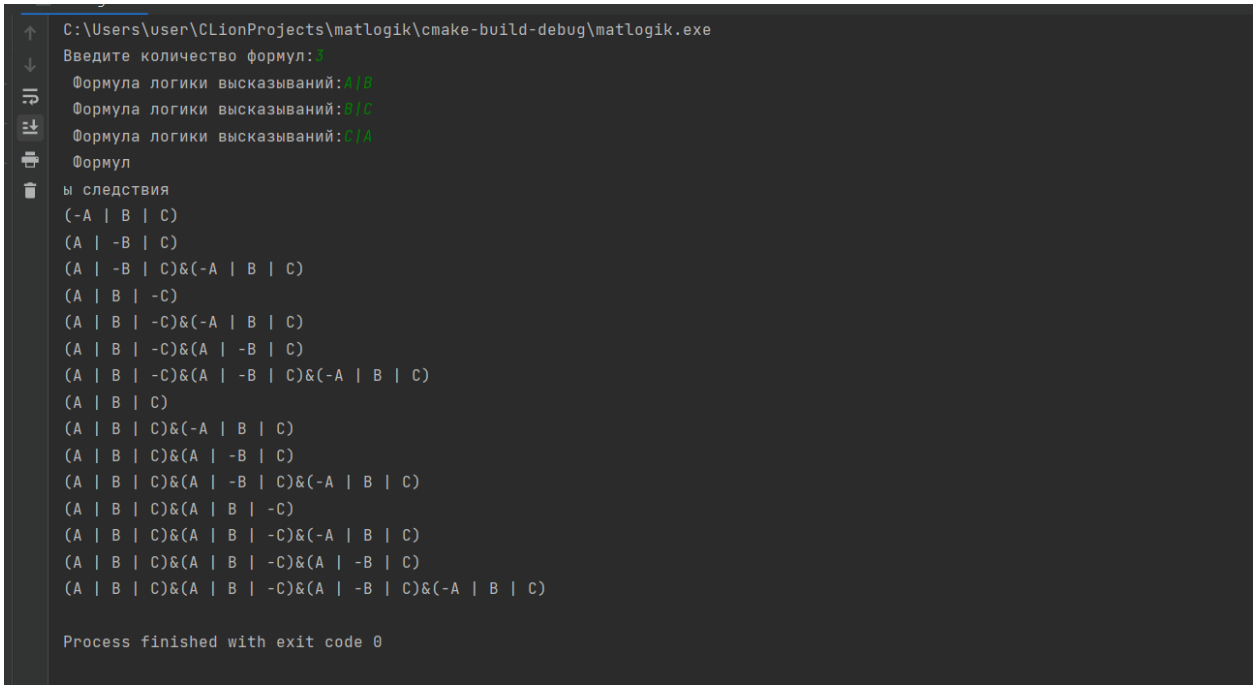
Тестовые данные:



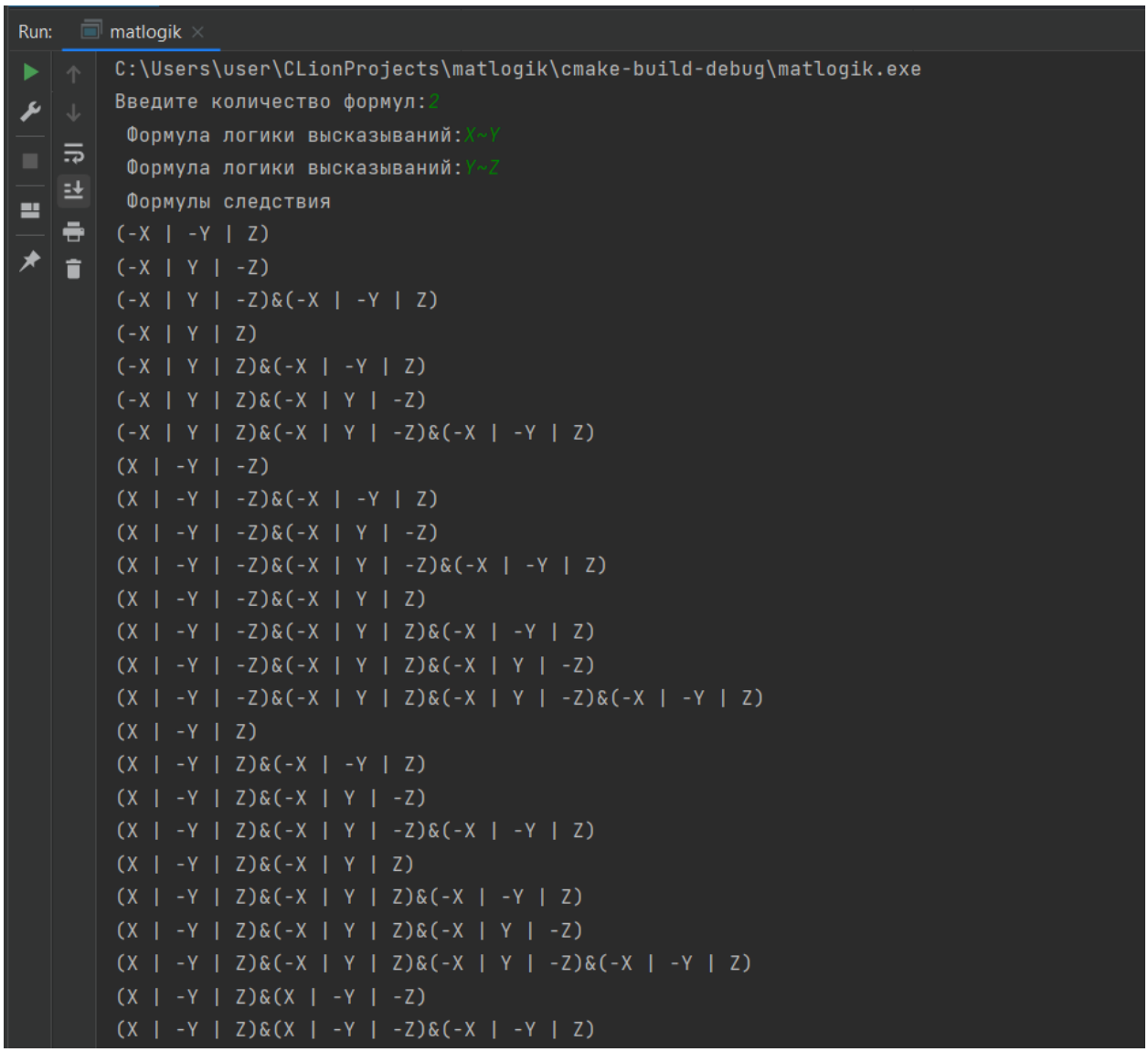


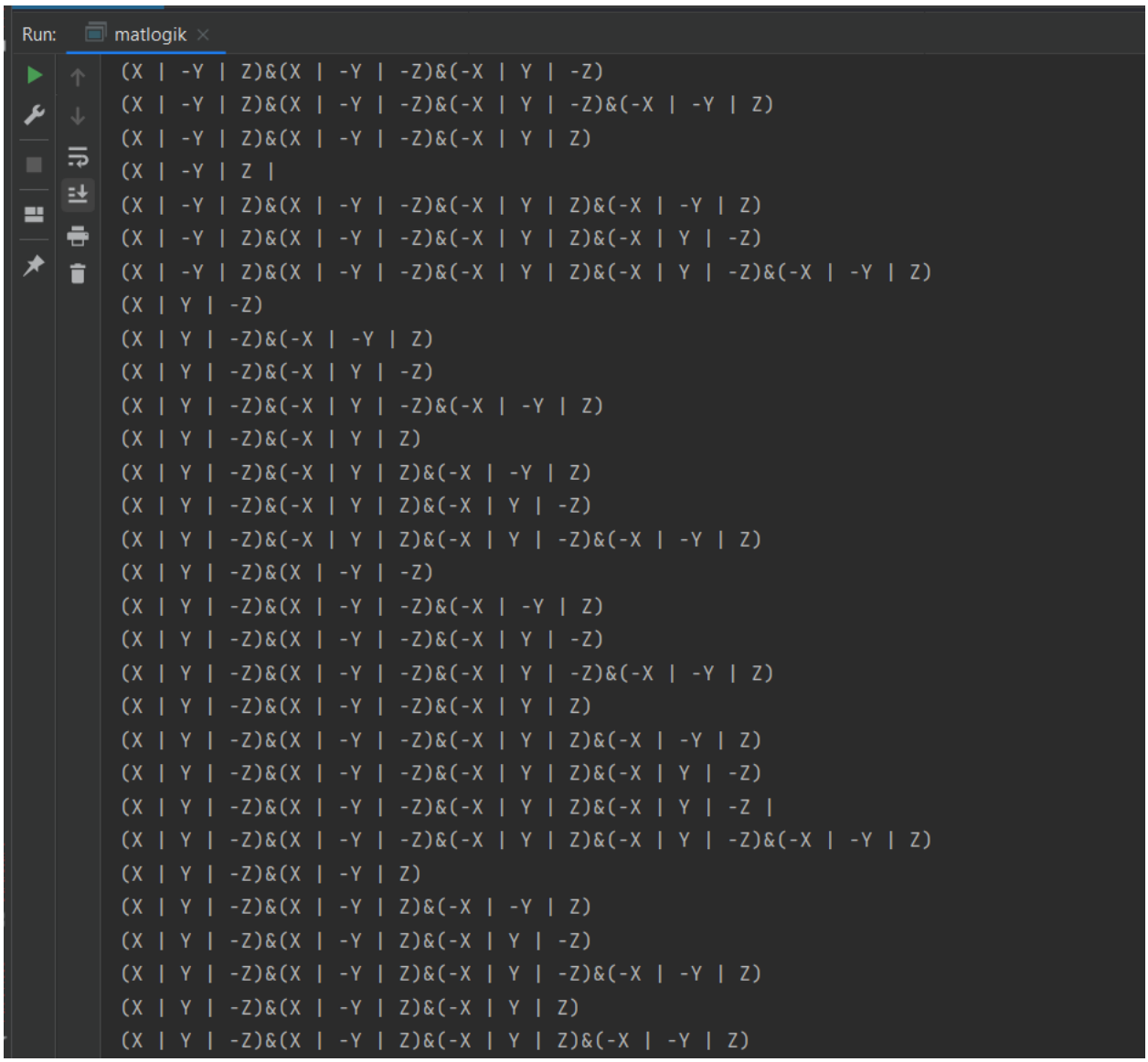


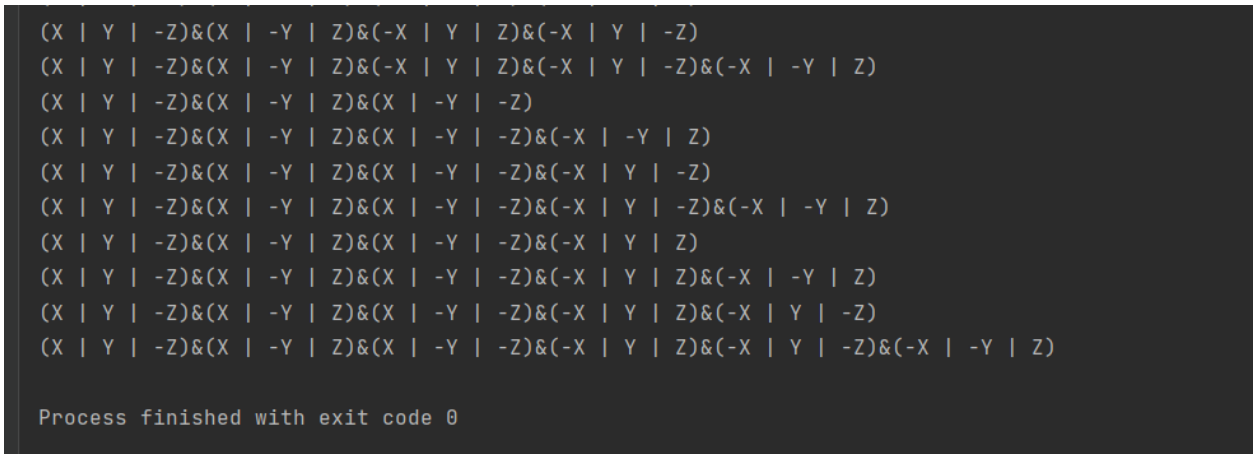




(проверено на прошлой лабораторной работе)


Run:
matlogik
C:\Users\user\CLionProjects\matlogik\cmake-build-debug\matlogik.exe
Введите количество формул:2
Формула логики высказываний:X~Y
Формула логики высказываний:Y~Z
Формулы следствия
(-X | -Y | Z)
(-X | Y | -Z)
(-X | Y | -Z)&(-X | -Y | Z)
(-X | Y | Z)
(-X | Y | Z)&(-X | -Y | Z)
(-X | Y | Z)&(-X | Y | -Z)
(-X | Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | -Y | -Z)
(X | -Y | -Z)&(-X | -Y | Z)
(X | -Y | -Z)&(-X | Y | -Z)
(X | -Y | -Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | -Y | -Z)&(-X | Y | Z)
(X | -Y | -Z)&(-X | Y | Z)&(-X | -Y | Z)
(X | -Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)
(X | -Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | -Y | Z)
(X | -Y | Z)&(-X | -Y | Z)
(X | -Y | Z)&(-X | Y | -Z)
(X | -Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | -Y | Z)&(-X | Y | Z)
(X | -Y | Z)&(-X | Y | Z)&(-X | -Y | Z)
(X | -Y | Z)&(-X | Y | Z)&(-X | Y | -Z)
(X | -Y | Z)&(-X | Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | -Y | Z)&(X | -Y | -Z)
(X | -Y | Z)&(X | -Y | -Z)&(-X | -Y | Z)


Run:
matlogik
(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | -Z)
(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | Z)
(X | -Y | Z |
(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | -Y | Z)
(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)
(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)
(X | Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(-X | Y | -Z)
(X | Y | -Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(-X | Y | Z)
(X | Y | -Z)&(-X | Y | Z)&(-X | -Y | Z)
(X | Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)
(X | Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | -Z)
(X | Y | -Z)&(X | -Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | -Z)&(-X | Y | -Z)
(X | Y | -Z)&(X | -Y | -Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | -Z)&(-X | Y | Z)
(X | Y | -Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)
(X | Y | -Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | Y | -Z |
(X | Y | -Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | Z)
(X | Y | -Z)&(X | -Y | Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | Z)&(-X | Y | -Z)
(X | Y | -Z)&(X | -Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | Z)&(-X | Y | Z)
(X | Y | -Z)&(X | -Y | Z)&(-X | Y | Z)&(-X | -Y | Z)


(X | Y | -Z)&(X | -Y | Z)&(-X | Y | Z)&(-X | Y | -Z)
(X | Y | -Z)&(X | -Y | Z)&(-X | Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | Z)&(X | -Y | -Z)
(X | Y | -Z)&(X | -Y | Z)&(X | -Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | -Z)
(X | Y | -Z)&(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | -Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | Z)
(X | Y | -Z)&(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | -Y | Z)
(X | Y | -Z)&(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)
(X | Y | -Z)&(X | -Y | Z)&(X | -Y | -Z)&(-X | Y | Z)&(-X | Y | -Z)&(-X | -Y | Z)
Process finished with exit code 0