Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Программная инженерия
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №7

Исчисления и абстрактная интерпретация
тема

Преподаватель _____ А. С. Кузнецов
подпись, дата — инициалы, фамилия

Студент КИ23-17/1Б, 032320072 _____ М. А. Мальцев
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2025

1 Цель

Исследование проблем вычислимости без использования абстрактной машины Тьюринга.

2 Задания

В части 1 необходимо произвести программную реализацию вычислителя заданной математической функции для заданных аргументов, причем исключительно средствами примитивной и частичной рекурсии, или формально доказать невозможность этого. Привести примеры выполнения вычислений. В части 2 необходимо, используя метод абстрактной интерпретации, для произвольной программной процедуры с количеством строк кода без комментариев не менее 10, определить знаки всех переменных

Часть 1. Вариант 6.

$f(x) = 2^{x!}$, $x \geq 0$, двойка может задаваться явно или неявно

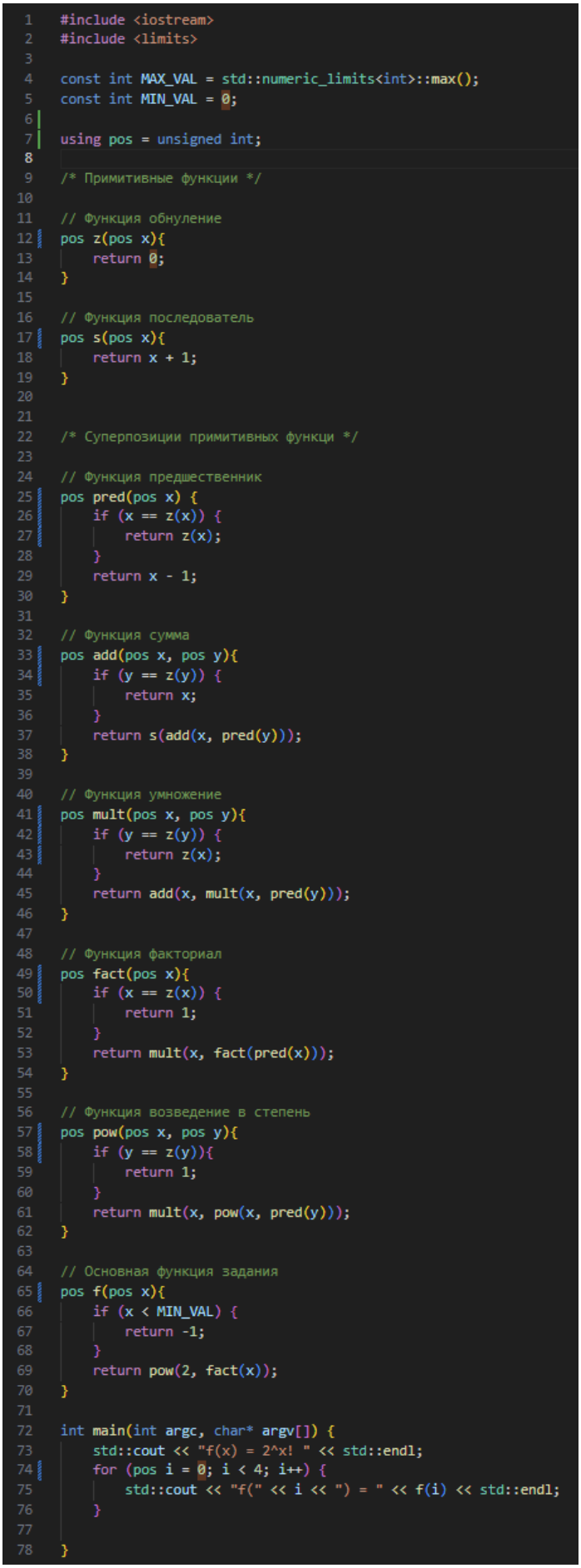
Часть 2.

Программная процедура для абстрактной интерпретации предлагается студентом.

3 Ход выполнения

3.1 Программная реализация вычислителя функции

Сначала была написана программная реализация на C++ для вычисления заданной математической функции $f(x) = 2^{x!}$. Она показана на рисунке 1.



```cpp
#include <iostream>
#include <limits>

const int MAX_VAL = std::numeric_limits<int>::max();
const int MIN_VAL = 0;

using pos = unsigned int;

/* Примитивные функции */

// Функция обнуление
pos z(pos x){
    return 0;
}

// Функция последователь
pos s(pos x){
    return x + 1;
}


/* Суперпозиции примитивных функци */

// Функция предшественник
pos pred(pos x) {
    if (x == z(x)) {
        return z(x);
    }
    return x - 1;
}

// Функция сумма
pos add(pos x, pos y){
    if (y == z(y)) {
        return x;
    }
    return s(add(x, pred(y)));
}

// Функция умножение
pos mult(pos x, pos y){
    if (y == z(y)) {
        return z(x);
    }
    return add(x, mult(x, pred(y)));
}

// Функция факториал
pos fact(pos x){
    if (x == z(x)) {
        return 1;
    }
    return mult(x, fact(pred(x)));
}

// Функция возведение в степень
pos pow(pos x, pos y){
    if (y == z(y)){
        return 1;
    }
    return mult(x, pow(x, pred(y)));
}

// Основная функция задания
pos f(pos x){
    if (x < MIN_VAL) {
        return -1;
    }
    return pow(2, fact(x));
}

int main(int argc, char* argv[]) {
    std::cout << "f(x) = 2^x! " << std::endl;
    for (pos i = 0; i < 4; i++) {
        std::cout << "f(" << i << ") = " << f(i) << std::endl;
    }

}
```

Рисунок 1 – Программная реализация для 1 части задания

На рисунке 2 показан результат выполнения программы.

```
PS D:\work_space\AutomataTheoryCourse\pw7> .\primitive_recursion.exe
f(x) = 2^x!
f(0) = 2
f(1) = 2
f(2) = 4
f(3) = 64
```

Рисунок 2 – Результат выполнения программы для 1 части задания

3.2 Программная реализация метода абстрактной интерпретации

Далее была написана программная реализация на C++ метода абстрактной интерпретации для самостоятельно заданной программной процедуры. Был определен абстрактный домен из знаков {+ (plus, положительное число), - (minus, отрицательное число), 0 (zero, ноль), U (unknown, неизвестное значение)}. Правила определения знаков показаны на рисунке 3.

Таблица сложения

⊕	+	-	0	U
+	+	U	+	U
-	U	-	-	U
0	+	-	0	U
U	U	U	U	U

Таблица вычитания

⊖	+	-	0	U
+	U	+	+	U
-	-	U	-	U
0	-	+	0	U
U	U	U	U	U

Таблица умножения

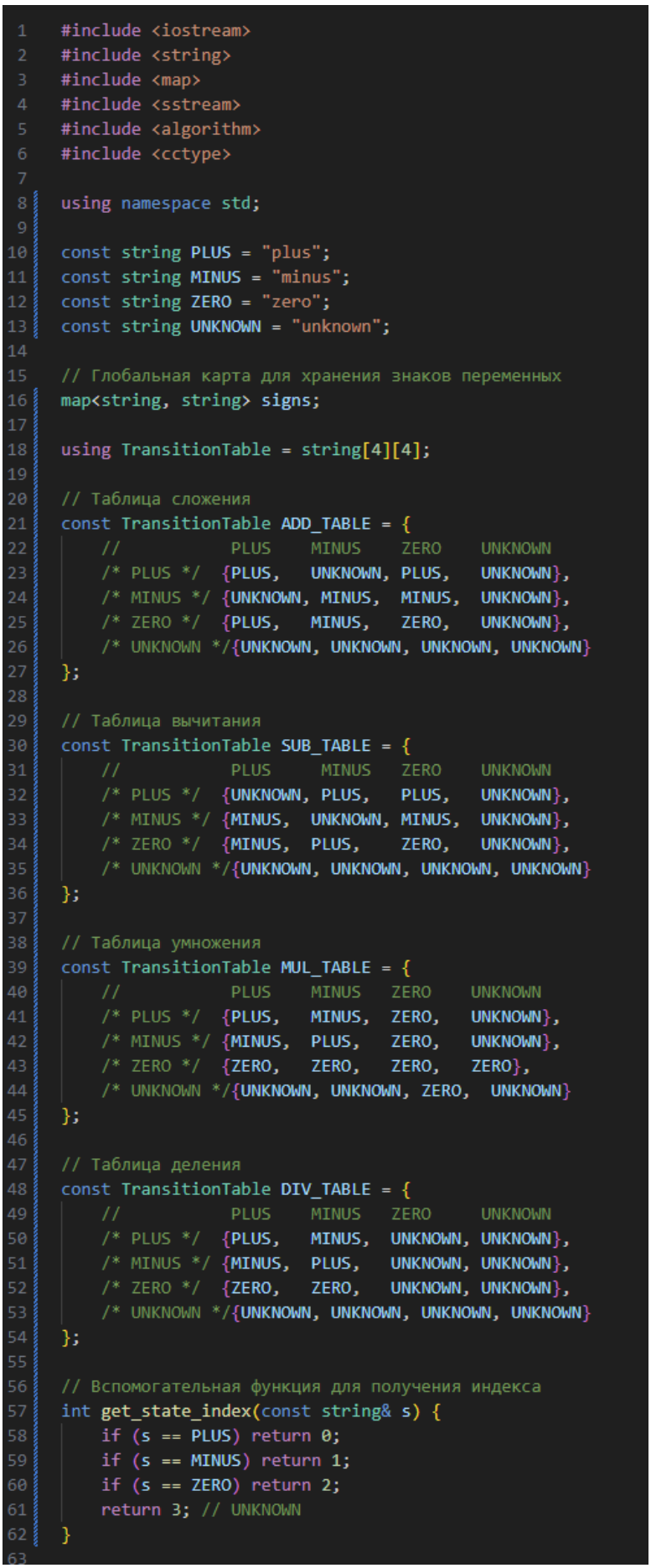
⊗	+	-	0	U
+	+	-	0	U
-	-	+	0	U
0	0	0	0	0
U	U	U	U	U

Таблица деления

⊘	+	-	0	U
+	+	-	U	U
-	-	+	U	U
0	0	0	U	U
U	U	U	U	U

Рисунок 3 – Таблицы правила определения знаков

Реализация показана на рисунке 4, 5, 6 и 7.



```cpp
#include <iostream>
#include <string>
#include <map>
#include <sstream>
#include <algorithm>
#include <cctype>

using namespace std;

const string PLUS = "plus";
const string MINUS = "minus";
const string ZERO = "zero";
const string UNKNOWN = "unknown";

// Глобальная карта для хранения знаков переменных
map<string, string> signs;

using TransitionTable = string[4][4];

// Таблица сложения
const TransitionTable ADD_TABLE = {
    //           PLUS     MINUS    ZERO     UNKNOWN
    /* PLUS */  {PLUS,    UNKNOWN, PLUS,    UNKNOWN},
    /* MINUS */ {UNKNOWN, MINUS,   MINUS,   UNKNOWN},
    /* ZERO */  {PLUS,    MINUS,   ZERO,    UNKNOWN},
    /* UNKNOWN */{UNKNOWN, UNKNOWN, UNKNOWN, UNKNOWN}
};

// Таблица вычитания
const TransitionTable SUB_TABLE = {
    //           PLUS     MINUS    ZERO     UNKNOWN
    /* PLUS */  {UNKNOWN, PLUS,    PLUS,    UNKNOWN},
    /* MINUS */ {MINUS,   UNKNOWN, MINUS,   UNKNOWN},
    /* ZERO */  {MINUS,   PLUS,    ZERO,    UNKNOWN},
    /* UNKNOWN */{UNKNOWN, UNKNOWN, UNKNOWN, UNKNOWN}
};

// Таблица умножения
const TransitionTable MUL_TABLE = {
    //           PLUS     MINUS    ZERO     UNKNOWN
    /* PLUS */  {PLUS,    MINUS,   ZERO,    UNKNOWN},
    /* MINUS */ {MINUS,   PLUS,    ZERO,    UNKNOWN},
    /* ZERO */  {ZERO,    ZERO,    ZERO,    ZERO},
    /* UNKNOWN */{UNKNOWN, UNKNOWN, ZERO,   UNKNOWN}
};

// Таблица деления
const TransitionTable DIV_TABLE = {
    //           PLUS     MINUS    ZERO     UNKNOWN
    /* PLUS */  {PLUS,    MINUS,   UNKNOWN, UNKNOWN},
    /* MINUS */ {MINUS,   PLUS,    UNKNOWN, UNKNOWN},
    /* ZERO */  {ZERO,    ZERO,    UNKNOWN, UNKNOWN},
    /* UNKNOWN */{UNKNOWN, UNKNOWN, UNKNOWN, UNKNOWN}
};

// Вспомогательная функция для получения индекса
int get_state_index(const string& s) {
    if (s == PLUS) return 0;
    if (s == MINUS) return 1;
    if (s == ZERO) return 2;
    return 3; // UNKNOWN
}
```

Рисунок 4 – Первая часть программной реализации для 2 части задания

```
// Абстрактные правила для операций
string abstract_combine(const string& x, const string& y, char op) {
    int idx_x = get_state_index(x);
    int idx_y = get_state_index(y);

    switch (op) {
        case '+':
            return ADD_TABLE[idx_x][idx_y];
        case '-':
            return SUB_TABLE[idx_x][idx_y];
        case '*':
            return MUL_TABLE[idx_x][idx_y];
        case '/':
            return DIV_TABLE[idx_x][idx_y];
        default:
            return UNKNOWN;
    }
}


/* Вспомогательные функции для работы со строками и числами */

// Определяет знак целого числа
string get_sign_of_digit(int value) {
    if (value > 0) {
        return PLUS;
    } else if (value < 0) {
        return MINUS;
    } else {
        return ZERO;
    }
}

// Проверяет, является ли строка числом (с учетом знака)
bool is_digit(const string& value) {
    if (value.empty()) {
        return false;
    }
    string s = value;
    size_t start = 0;
    if (s[0] == '+' || s[0] == '-') {
        start = 1;
    }
    return !s.substr(start).empty() && all_of(s.begin() + start, s.end(), ::isdigit);
}

// Проверяет сбалансированность скобок
bool check_brackets(const string& expr) {
    int depth = 0;
    for (char c : expr) {
        if (c == '(') {
            depth++;
        } else if (c == ')') {
            depth--;
        }
        if (depth < 0) {
            return false;
        }
    }
    return depth == 0;
}
```

Рисунок 5 – Вторая часть программной реализации для 2 части задания

```
// Рекурсивное вычисление знаков выражений
string determine_sign(const string& expr) {
    string clean_expr = expr;
    clean_expr.erase(remove_if(clean_expr.begin(), clean_expr.end(), ::isspace), clean_expr.end());

    if (clean_expr.empty()) {
        return UNKNOWN;
    }

    if (is_digit(clean_expr)) {
        try {
            return get_sign_of_digit(stoi(clean_expr));
        } catch (...) {
            return UNKNOWN;
        }
    }

    if (signs.count(clean_expr)) {
        return signs[clean_expr];
    }

    if (clean_expr.front() == '(' && clean_expr.back() == ')' && check_brackets(clean_expr.substr(1, clean_expr.length() - 2))) {
        return determine_sign(clean_expr.substr(1, clean_expr.length() - 2));
    }

    int depth = 0;

    string operators = "+-*/";
    char main_operator = '\0';
    int index_of_main = -1;

    for (int i = clean_expr.length() - 1; i >= 0; --i) {
        char c = clean_expr[i];

        if (c == ')') {
            depth++;
        } else if (c == '(') {
            depth--;
        } else if (depth == 0 && operators.find(c) != string::npos) {
            bool is_unary = false;

            if (i == 0) {
                is_unary = true;
            }
            else {
                char prev = clean_expr[i - 1];
                if (operators.find(prev) != string::npos || prev == '(') {
                    is_unary = true;
                }
            }

            if (is_unary) {
                continue;
            }

            bool current_is_low_prio = (c == '+' || c == '-');
            bool existing_is_high_prio = (main_operator == '*' || main_operator == '/');

            if (main_operator == '\0') {
                main_operator = c;
                index_of_main = i;
            } else {
                if (current_is_low_prio) {
                    main_operator = c;
                    index_of_main = i;
                    break;
                }
            }
        }
    }

    if (main_operator == '\0' || index_of_main == -1) {
        return UNKNOWN;
    }

    string left = clean_expr.substr(0, index_of_main);
    string right = clean_expr.substr(index_of_main + 1);

    string x = determine_sign(left);
    string y = determine_sign(right);

    return abstract_combine(x, y, main_operator);
}
```

Рисунок 6 – Третья часть программной реализации для 2 части задания

```
// Основная функция анализа
void analyze(const string& procedure) {
    signs.clear();
    stringstream ss(procedure);
    string line;

    while (getline(ss, line, '\n')) {
        line.erase(remove_if(line.begin(), line.end(), ::isspace), line.end());
        if (line.empty() || line.find('=') == string::npos) {
            continue;
        }

        size_t eq_pos = line.find('=');
        string left = line.substr(0, eq_pos);
        string right = line.substr(eq_pos + 1);

        string var = left;
        string expr = right;

        signs[var] = determine_sign(expr);
    }

    cout << "\nProcedure:" << procedure << endl;
    cout << "Signs of variables:" << endl;
    for (const auto& pair : signs) {
        cout << pair.first << ": " << pair.second << endl;
    }
}

int main() {
    string procedure = R"(
a = 1
b = -1
c = a + b
d = a - b
e = d * d + a
f = 3
f = (f + e) * (b - d)
g = c + c
h = b - (0 - f)
j = -2 / -2
k = (h - j) / (b - 0)
)";
    analyze(procedure);
    return 0;
}
```

Рисунок 7 – Четвертая часть программной реализации для 2 части задания

Составленная программная процедура и результат выполнения программы показаны на рисунке 8.

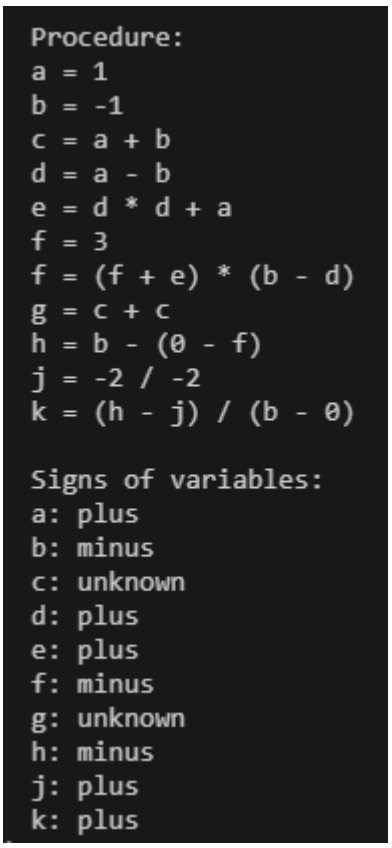


Рисунок 8 – Результат выполнения программы для 2 части задания

4 Выводы

В ходе данной практической работы были исследованы проблемы вычислимости без использования абстрактной машины Тьюринга. Были написаны программные реализации вычислителя функции с помощью средств примитивной рекурсии и метода абстрактной интерпретации.