

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

подпись, дата

А. С. Кузнецов

инициалы, фамилия

Студент КИ23-17/1Б, 032320072

номер группы, зачетной книжки

подпись, дата

М. А. Мальцев

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.

```

1  #include <iostream>
2  #include <limits>
3
4  const int MAX_VAL = std::numeric_limits<int>::max();
5  const int MIN_VAL = 0;
6
7  /* Примитивные функции */
8
9  // Функция обнуление
10 int z(int x){
11     return 0;
12 }
13
14 // Функция последователь
15 int s(int x){
16     return x + 1;
17 }
18
19
20 /* Суперпозиции примитивных функци */
21
22 // Функция предшественник
23 int pred(int x) {
24     if (x == 0) {
25         return 0;
26     }
27     return x - 1;
28 }
29
30 // Функция сумма
31 int add(int x, int y){
32     if (y == 0) {
33         return x;
34     }
35     return s(add(x, pred(y)));
36 }
37
38 // Функция умножение
39 int mult(int x, int y){
40     if (y == 0) {
41         return 0;
42     }
43     return add(x, mult(x, pred(y)));
44 }
45
46 // Функция факториал
47 int fact(int x){
48     if (x == 0) {
49         return 1;
50     }
51     return mult(x, fact(pred(x)));
52 }
53
54 // Функция возведение в степень
55 int pow(int x, int y){
56     if (y == 0){
57         return 1;
58     }
59     return mult(x, pow(x, pred(y)));
60 }
61
62 // Основная функция задания
63 int f(int x){
64     if (x < MIN_VAL) {
65         return -1;
66     }
67     return pow(2, fact(x));
68 }
69
70 int main(int argc, char* argv[]){
71     std::cout << "f(x) = 2^x!" << std::endl;
72     for (int i = 0; i < 4; i++) {
73         std::cout << "f(" << i << ") = " << f(i) << std::endl;
74     }
75 }
76

```

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. Реализация показана на рисунке 3, 4, 5 и 6.

```
1 #include <iostream>
2 #include <string>
3 #include <map>
4 #include <sstream>
5 #include <algorithm>
6 #include <cctype>
7
8 const std::string PLUS = "plus";
9 const std::string MINUS = "minus";
10 const std::string ZERO = "zero";
11 const std::string UNKNOWN = "unknown";
12
13 // Глобальная карта для хранения знаков переменных
14 std::map<std::string, std::string> signs;
15
16
17 // Абстрактные правила для операций
18 std::string abstract_combine(const std::string& x, const std::string& y, char op) {
19     if (x == UNKNOWN || y == UNKNOWN) {
20         return UNKNOWN;
21     }
22
23     if (op == '+') {
24         if (x == y) return x;
25         if (x == ZERO) return y;
26         if (y == ZERO) return x;
27         return UNKNOWN;
28     }
29
30     if (op == '-') {
31         if (x == PLUS && y == MINUS) return PLUS;
32         if (x == MINUS && y == PLUS) return MINUS;
33         if (y == ZERO) return x;
34         if (x == ZERO && y == PLUS) return MINUS;
35         if (x == ZERO && y == MINUS) return PLUS;
36         if (x == y) return ZERO;
37         return UNKNOWN;
38     }
39
40     if (op == '*') {
41         if (x == y) return PLUS;
42         if (x == ZERO || y == ZERO) return ZERO;
43         return MINUS;
44     }
45
46     if (op == '/') {
47         if (x == y) return PLUS;
48         if (y == ZERO) return UNKNOWN;
49         if (x == ZERO) return ZERO;
50         return MINUS;
51     }
52
53     return UNKNOWN;
54 }
```

Рисунок 3 – Первая часть программной реализации для 2 части задания

```
56
57     /* Вспомогательные функции для работы со строками и числами */
58
59     // Определяет знак целого числа
60     std::string get_sign_of_digit(int value) {
61         if (value > 0) {
62             return PLUS;
63         } else if (value < 0) {
64             return MINUS;
65         } else {
66             return ZERO;
67         }
68     }
69
70     // Проверяет, является ли строка числом (с учетом знака)
71     bool is_digit(const std::string& value) {
72         if (value.empty()) {
73             return false;
74         }
75         std::string s = value;
76         size_t start = 0;
77         if (s[0] == '+' || s[0] == '-') {
78             start = 1;
79         }
80         return !s.substr(start).empty() && std::all_of(s.begin() + start, s.end(), ::isdigit);
81     }
82
83     // Проверяет сбалансированность скобок
84     bool check_brackets(const std::string& expr) {
85         int depth = 0;
86         for (char c : expr) {
87             if (c == '(') {
88                 depth++;
89             } else if (c == ')') {
90                 depth--;
91             }
92             if (depth < 0) {
93                 return false;
94             }
95         }
96         return depth == 0;
97     }
98 }
```

Рисунок 4 – Вторая часть программной реализации для 2 части задания

```

98 // Рекурсивное вычисление знаков выражений
99 std::string determine_sign(const std::string& expr) {
100    std::string clean_expr = expr;
101    clean_expr.erase(std::remove_if(clean_expr.begin(), clean_expr.end(), ::isspace), clean_expr.end());
102
103    if (clean_expr.empty()) {
104        return UNKNOWN;
105    }
106
107    if (is_digit(clean_expr)) {
108        try {
109            return get_sign_of_digit(std::stoi(clean_expr));
110        } catch (...) {
111            return UNKNOWN;
112        }
113    }
114
115    if (signs.count(clean_expr)) {
116        return signs[clean_expr];
117    }
118
119    if (clean_expr.front() == '(' && clean_expr.back() == ')' && check_brackets(clean_expr.substr(1, clean_expr.length() - 2))) {
120        return determine_sign(clean_expr.substr(1, clean_expr.length() - 2));
121    }
122
123    int depth = 0;
124
125    std::string operators = "+-*/";
126    char main_operator = '\0';
127    int index_of_main = -1;
128
129    for (int i = clean_expr.length() - 1; i >= 0; --i) {
130        char c = clean_expr[i];
131
132        if (c == ')') {
133            depth++;
134        } else if (c == '(') {
135            depth--;
136        } else if (depth == 0 && operators.find(c) != std::string::npos) {
137            bool is_unary = false;
138
139            if (i == 0) {
140                is_unary = true;
141            }
142            else {
143                char prev = clean_expr[i - 1];
144                if (operators.find(prev) != std::string::npos || prev == '(') {
145                    is_unary = true;
146                }
147            }
148
149            if (is_unary) {
150                continue;
151            }
152
153            bool current_is_low_prio = (c == '+' || c == '-');
154            bool existing_is_high_prio = (main_operator == '*' || main_operator == '/');
155
156            if (main_operator == '\0') {
157                main_operator = c;
158                index_of_main = i;
159            } else {
160                if (current_is_low_prio) {
161                    main_operator = c;
162                    index_of_main = i;
163                    break;
164                }
165            }
166        }
167    }
168
169    if (main_operator == '\0' || index_of_main == -1) {
170        return UNKNOWN;
171    }
172
173    std::string left = clean_expr.substr(0, index_of_main);
174    std::string right = clean_expr.substr(index_of_main + 1);
175
176    std::string x = determine_sign(left);
177    std::string y = determine_sign(right);
178
179    return abstract_combine(x, y, main_operator);
180}
181

```

Рисунок 5 – Третья часть программной реализации для 2 части задания

```

182 // Основная функция анализа
183 void analyze(const std::string& procedure) {
184     signs.clear();
185     std::stringstream ss(procedure);
186     std::string line;
187
188     while (std::getline(ss, line, '\n')) {
189         line.erase(std::remove_if(line.begin(), line.end(), ::isspace), line.end());
190         if (line.empty() || line.find('=') == std::string::npos) {
191             continue;
192         }
193
194         size_t eq_pos = line.find('=');
195         std::string left = line.substr(0, eq_pos);
196         std::string right = line.substr(eq_pos + 1);
197
198         std::string var = left;
199         std::string expr = right;
200
201         signs[var] = determine_sign(expr);
202     }
203
204     std::cout << "\nProcedure:" << procedure << std::endl;
205     std::cout << "Signs of variables:" << std::endl;
206     for (const auto& pair : signs) {
207         std::cout << pair.first << ": " << pair.second << std::endl;
208     }
209 }
210
211 int main() {
212     std::string procedure = R"(

213     a = 1
214     b = -1
215     c = a + b
216     d = a - b
217     e = d * d + a
218     f = 3
219     f = (f + e) * (b - d)
220     g = c + c
221     h = b - (0 - f)
222     j = -2 / -2
223     k = (h - j) / (b - 0)
224 );
225     analyze(procedure);
226     return 0;
227 }
```

Рисунок 6 – Четвертая часть программной реализации для 2 части задания

Составленная программная процедура и результат выполнения программы показаны на рисунке 7.

```
Procedure:  
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)  
  
Signs of variables:  
a: plus  
b: minus  
c: unknown  
d: plus  
e: plus  
f: minus  
g: unknown  
h: minus  
j: plus  
k: plus
```

Рисунок 7 – 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.