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 КИ22-17/1Б, 03221283 _____ А. Н. Воробьев
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2025

Содержание

Содержание..... 2

1 Цель 3

2 Задача 3

3 Ход работы 3

3.1 Первая часть 3

3.2 Вторая часть 5

Вывод 10

1 Цель

Исследование проблем вычислимости без использования абстрактной машины Тьюринга.

2 Задача

Вариант 6.

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.

3 Ход работы

3.1 Первая часть

Программная реализация для вычисления заданной математической функции (f(x) = 2^(x!)) приведена на листинге 1.

Листинг 1 – Программная реализация

```
import sys
sys.setrecursionlimit(100000)

# --- Примитивные функции ---

def zero(x):
    return 0

def succ(x):
    return x + 1

# --- Суперпозиции и примитивные рекурсии ---
```

```
def add(x, y):
    if y == 0:
        return x
    return succ(add(x, y - 1))

def mul(x, y):
    if y == 0:
        return 0
    return add(x, mul(x, y - 1))

def pow_rec(a, b):
    if b == 0:
        return 1
    return mul(a, pow_rec(a, b - 1))

def fact(x):
    if x == 0:
        return 1
    return mul(x, fact(x - 1))

def f(x):
    """Искомая функция f(x) = 2^(x!)"""
    return pow_rec(2, fact(x))

def main():
    # --- Примеры вычислений ---
    for i in range(4):
        print(f"f({i}) = 2^({i}!) = {f(i)}")

if __name__ == "__main__":
    main()
```

Результат выполнения программы можно увидеть на рисунке 1.

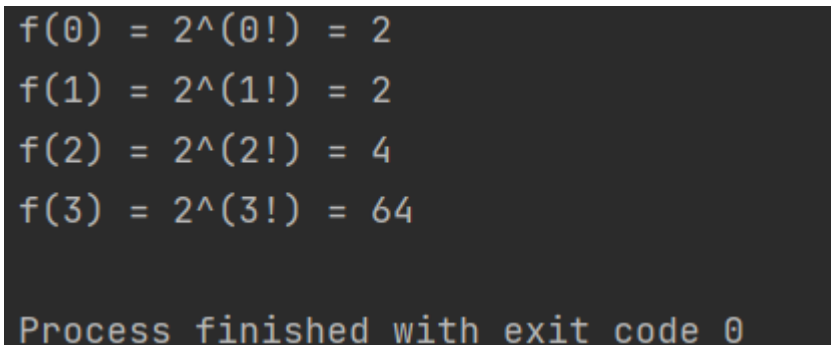

Рисунок 1 – Пример работы

3.2 Вторая часть

Программная реализация вычислителя знаков переменных с использованием метода абстрактной интерпретации представлена на листинге 2.

Листинг 2 – Программная реализация

```
plus = "plus"
minus = "minus"
zero = "zero"
unknown = "unknown"

signs = {}

# --- Проверка и определение знаков чисел ---
def is_digit(value: str):
    try:
        int(value)
        return True
    except ValueError:
        return False

def sign_of_digit(value: int):
    if value > 0:
        return plus
    elif value < 0:
        return minus
```

```
        else:
            return zero


    # --- Абстрактные правила для операций ---
    def combine_signs(x, y, op):
        # Если хоть одно неизвестно
        if x == unknown or y == unknown:
            return unknown

        if op == "+":
            if x == zero:
                return y
            if y == zero:
                return x
            if x == y:
                return x
            return unknown

        if op == "-":
            if y == zero:
                return x
            if x == y:
                return unknown
            if x == plus and y == minus:
                return plus
            if x == minus and y == plus:
                return minus
            return unknown

        if op == "*":
            if x == zero or y == zero:
                return zero
            if (x == plus and y == plus) or (x == minus and y ==
minus):
```

```
                return plus
            if (x == plus and y == minus) or (x == minus and y
== plus):
                return minus
            return unknown

        if op == "/":
            if y == zero:
                return unknown  # деление на ноль не определено
            if x == zero:
                return zero
            if (x == plus and y == plus) or (x == minus and y ==
minus):
                return plus
            if (x == plus and y == minus) or (x == minus and y
== plus):
                return minus
            return unknown

        return unknown


    # --- Рекурсивное вычисление знаков выражений ---
    def determine_sign(expr):
        expr = expr.strip().replace(" ", "")

        # если это число
        if is_digit(expr):
            return sign_of_digit(int(expr))

        # если это просто переменная
        if expr in signs:
            return signs[expr]

        # если в скобках целиком
```

```
        if expr.startswith("(") and expr.endswith(")") and
check_brackets(expr[1:-1]):
            return determine_sign(expr[1:-1])

        # ищем главную операцию (с учётом скобок)
        depth = 0
        main_op = None
        main_index = -1
        for i in range(len(expr) - 1, -1, -1):  # справа налево
(чтобы + и - имели меньший приоритет)
            c = expr[i]
            if c == ')':
                depth += 1
            elif c == '(':
                depth -= 1
            elif depth == 0 and c in "+-*/":
                main_op = c
                main_index = i
                break

        if main_op is None:
            return unknown

        left = expr[:main_index]
        right = expr[main_index + 1:]
        x = determine_sign(left)
        y = determine_sign(right)
        return combine_signs(x, y, main_op)


    def check_brackets(expr):
        """Проверяет сбалансированность скобок"""
        depth = 0
        for c in expr:
            if c == '(':
```

```
                depth += 1
            elif c == ')':
                depth -= 1
                if depth < 0:
                    return False
        return depth == 0


    # --- Основная функция анализа ---
    def analyze(procedure):
        lines = [line.strip() for line in procedure.split("\n")
if line.strip()]
        for line in lines:
            if "=" in line:
                left, right = line.split("=", 1)
                var = left.strip()
                expr = right.strip()
                signs[var] = determine_sign(expr)

        print("\nПроцедура:")
        print(procedure)
        print("\nЗнаки переменных:")
        for var, s in signs.items():
            print(f"{var}: {s}")


    # --- Пример использования ---
    def main():
        # пример процедуры
        procedure = """
    a = 5
    b = -2
    c = a + b
    d = a - b
    e = a * b
```

```
f = d + e
g = f - b
h = g * 2
j = (a - b) * (b + 3)
k = (a + e) / b
    """
    analyze(procedure)


if __name__ == "__main__":
    main()
```

Результаты выполнения для заданных процедур показаны на рисунке 2.

```
Процедура:

a = 5
b = -2
c = a + b
d = a - b
e = a * b
f = d + e
g = f - b
h = g * 2
j = (a - b) * (b + 3)
k = (a + e) / b


Знаки переменных:
a: plus
b: minus
c: unknown
d: plus
e: minus
f: unknown
g: unknown
h: unknown
j: unknown
k: unknown
```

Рисунок 2 – Результат работы

Вывод

Произвели программную реализацию вычислителя заданной математической функции для заданных аргументов, причем исключительно

средствами примитивной и частичной рекурсии, а также используя метод абстрактной интерпретации, определили знаки всех переменных.