
КАФЕДРА

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ

должность, уч. степень, звание

подпись, дата

инициалы, фамилия

Отчет о лабораторной работе №3

Алгоритмическая система Поста

По дисциплине: Теория вычислительных процессов

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

подпись, дата

инициалы, фамилия

Санкт-Петербург 2024

Цель работы:

Целью работы является построение формальной системы Поста, предназначенной для вычисления заданной арифметической функции, и разработка программы на языке высокого уровня, эмулирующей процесс вычисления на основе выводимости в системе. Программа должна корректно обрабатывать входные данные, включая алфавит, множество переменных, аксиомы и правила вывода, а также предоставлять файл с результатами вычислений.

Постановка задачи:

В данной лабораторной работе требуется:

- Построить формальную систему Поста FSp, реализующую вычисление заданной арифметической функции.
- Написать программу на языке высокого уровня имитирующую (эмулирующую) вычисления на основе выводимости в формальной системе Поста.
- Программа должна работать на любых входных данных из заданного множества.
- Программа должна удовлетворять предъявляемым требованиям.

Требования к программе:

Программа должна корректно обрабатывать ошибочные входные данные;

Контроль однозначности заданных правил проводить не требуется;

Следующие входные данные должны считываться программой из файла:

- A – собственный алфавит
- X – множество переменных
- A1 – множество аксиом
- R – конечное множество продукций (или правил вывода)

Результат работы программы должен содержать:

- Файл с результатами вычисления
- Сообщение на экране о следующих событиях:
 - Вычисление прошло успешно;
 - В ходе вычисления произошла ошибка:
 - Найден символ, не входящий в алфавит;
 - Найдена переменная, не входящая в заданное множество переменных;

Требования к входным данным:

- Все входные данные должны содержаться в одном файле.
- В файле элементы ФС должны быть заданы в следующем виде:
$$A = \{ \langle \text{элемент алфавита 1} \rangle, \langle \text{элемент алфавита 2} \rangle, \dots, \langle \text{элемент алфавита N} \rangle \};$$
$$X = \{ \langle \text{переменная 1} \rangle, \langle \text{переменная 2} \rangle, \dots, \langle \text{переменная K} \rangle \};$$
$$A1 = \{ \langle \text{аксиома 1} \rangle, \langle \text{аксиома 2} \rangle, \dots, \langle \text{аксиома M} \rangle \};$$
$$R = \{ \langle \text{правило 1} \rangle, \langle \text{правило 2} \rangle, \dots, \langle \text{правило L} \rangle \}$$

Требования к выходным данным:

- В файл с результатами вычисления на каждое применение правила выводятся следующие строки:
исходная строка;
применяемое правило;
результат применения правила.

Выполнение задачи:

Вариант:

38.

x_1x_2-y

Построить формальную систему Поста FSp, реализующую вычисление заданной арифметической функции.

Рассмотрим вычисление арифметической функции $x_1 * x_2 - y$, где $x_1 = 3$, $x_2 = 4$, и $y=4$. Задача заключается в создании формальной системы, которая может решать данную функцию для любых значений переменных.

Зададим исходную строку следующим образом:

111*1111-1111=

Алфавит:

$A = \{1, *, -, =\}$

Алфавит содержит символы, используемые для представления чисел, операций умножения и вычитания, а также знак равенства.

Множество переменных:

$X = \{x_1, x_2, y, x\}$

Переменные x_1, x_2, y используются в процессе вычисления.

Множество аксиом:

$A_1 = \{111*1111-1111=\}$

Аксиома содержит минимальный элемент — единицу. Вся строка будет строиться на основе этой единицы.

Множество правил вывода:

$R = \{$

- (1) $x_1 * 1x_2 - y = \rightarrow x_1 * x_2 - y = x_1$ - умножение кроме последней итерации
- (2) $x_1 * 1 - y = x_2 \rightarrow =x_1x_2 - y$ - умножение последней итерации, перенос равно
- (3) $=1x - 1y \rightarrow =x - y$ - вычитание кроме последней итерации
- (4) $=1x - 1 \rightarrow =x$ - вычитание последней единицы
- (5) $=1 - 1x \rightarrow =-x$ - учёт отрицательных результатов
- (6) $x_1 * 1 - y = \rightarrow =x_1 - y$ - учёт второго множителя равному единице

$\}$

1:

111*1111-1111=

111*111-1111=111

111*11-1111=111111

111*1-1111=11111111

2:

=111111111111-1111

3:

=11111111111-111

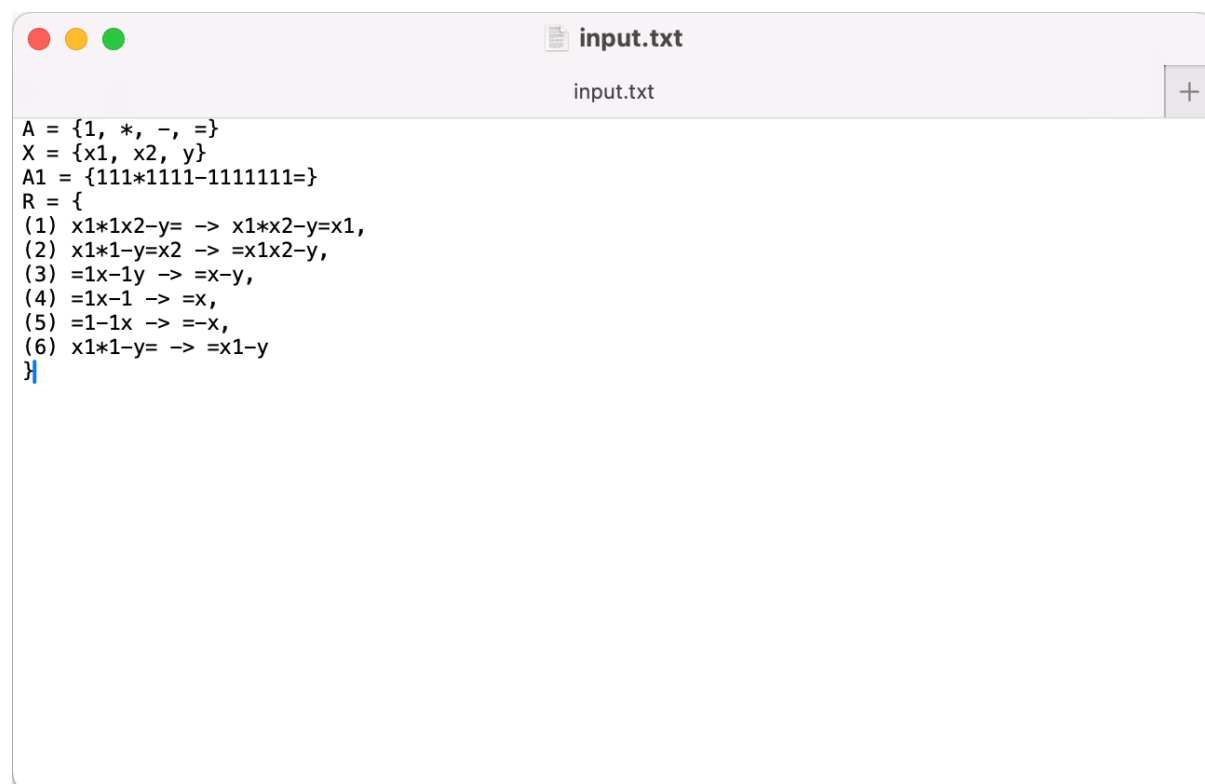
=1111111111-11

=111111111-1

4:

=111111111

Содержимое входного файла согласно заданию:



```
A = {1, *, -, =}  
X = {x1, x2, y}  
A1 = {111*1111-1111111=}  
R = {  
  (1) x1*x2-y= -> x1*x2-y=x1,  
  (2) x1*1-y=x2 -> =x1x2-y,  
  (3) =1x-1y -> =x-y,  
  (4) =1x-1 -> =x,  
  (5) =1-1x -> =-x,  
  (6) x1*1-y= -> =x1-y  
}
```

Рисунок 1 – входной файл

Примеры результатов выполнения:

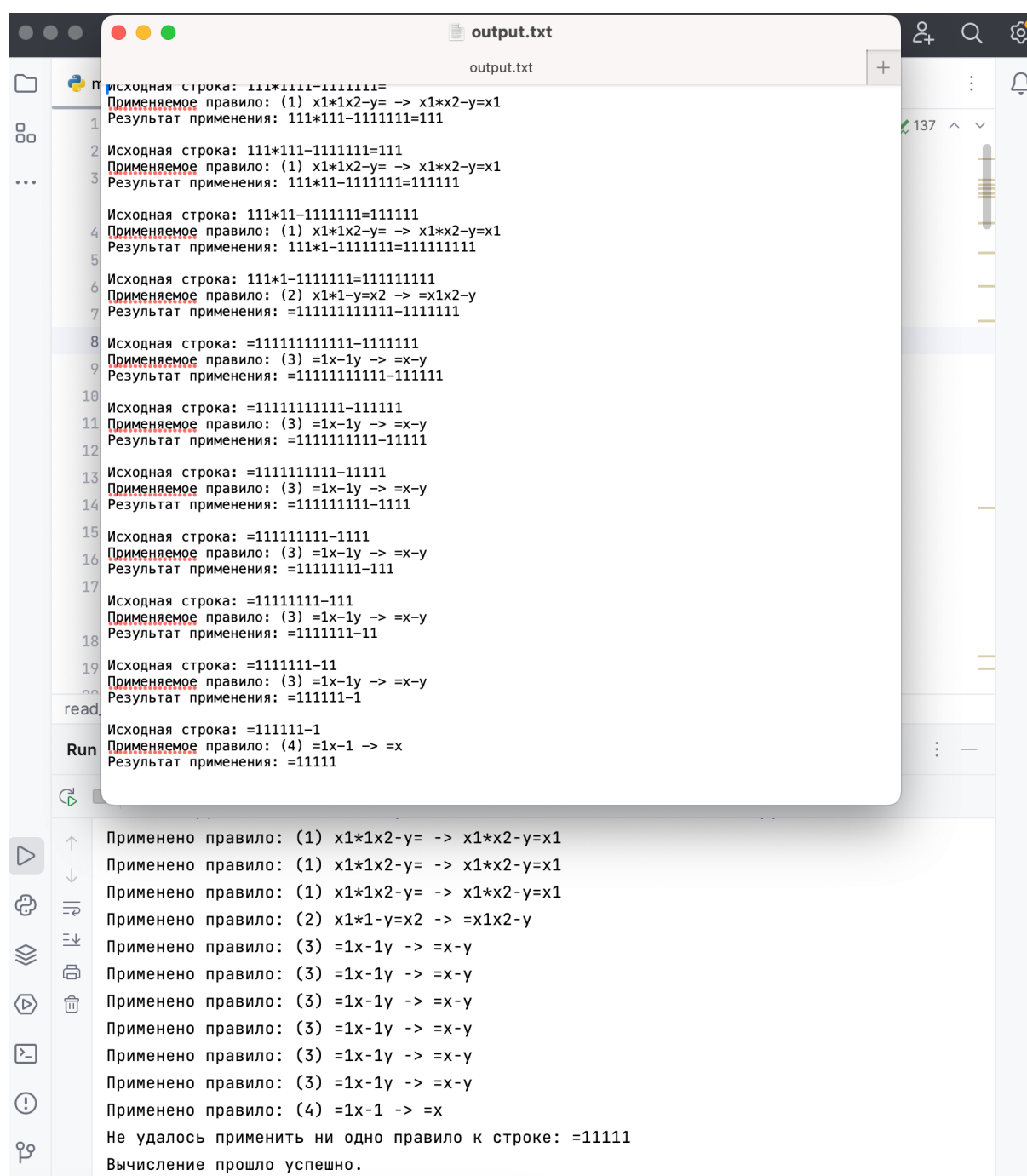


Рисунок 2 – результат выполнения программы

Корректные и некорректные входные данные:

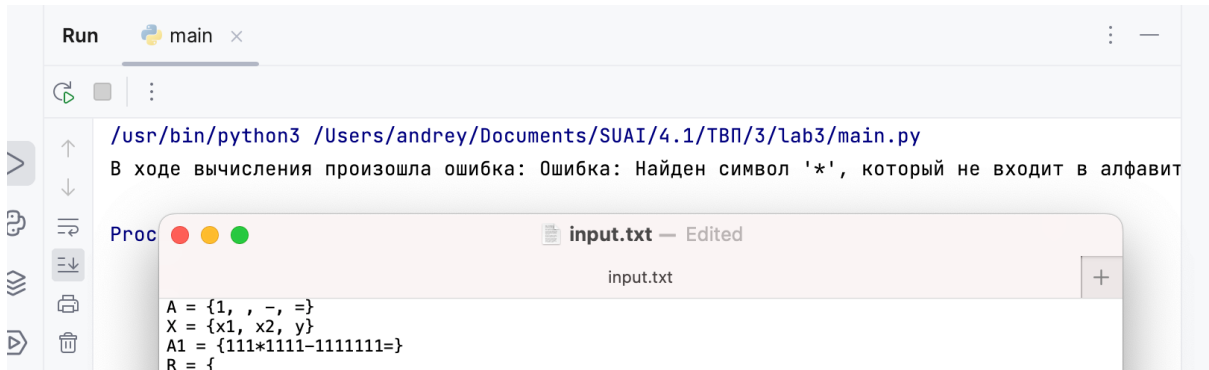


Рисунок 3 – обработка ошибки: отсутствие символа в алфавите

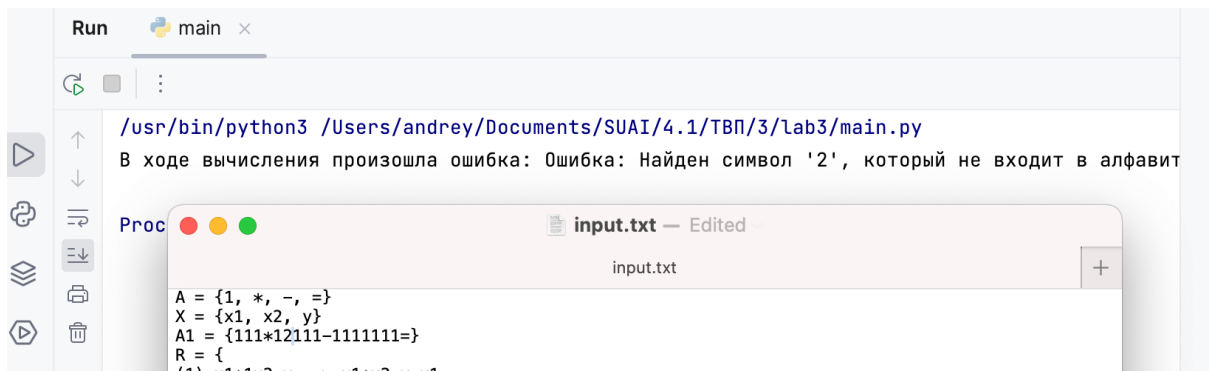


Рисунок 4 – обработка ошибки: символ в аксиоме отсутствует в алфавите

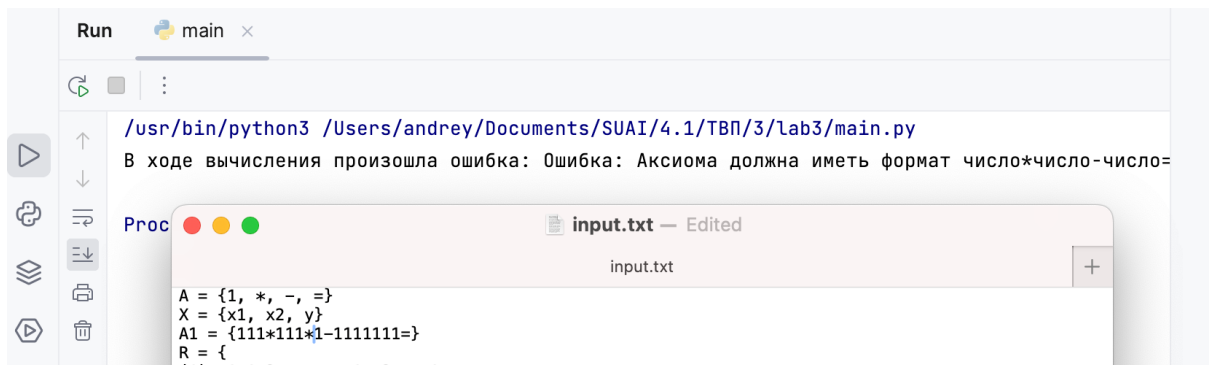


Рисунок 5 – обработка ошибки: неверный формат аксиомы

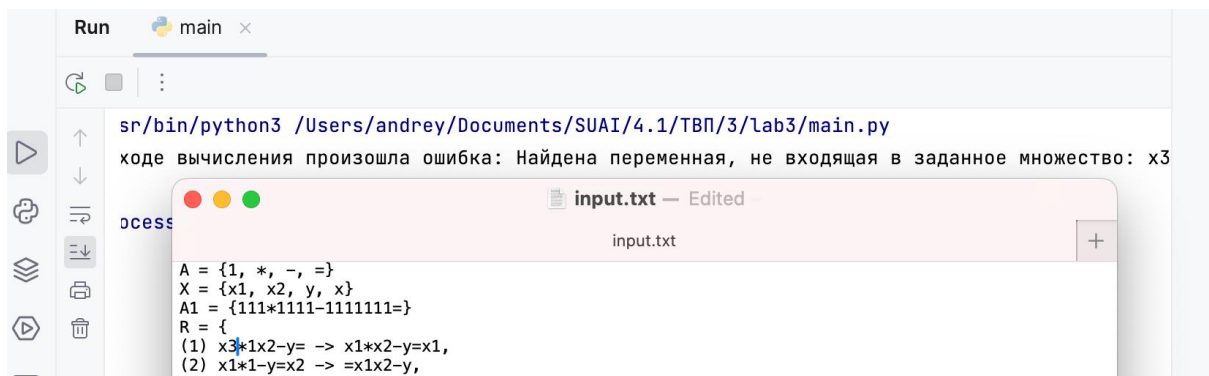


Рисунок 6 – обработка ошибки: неопределенная переменная

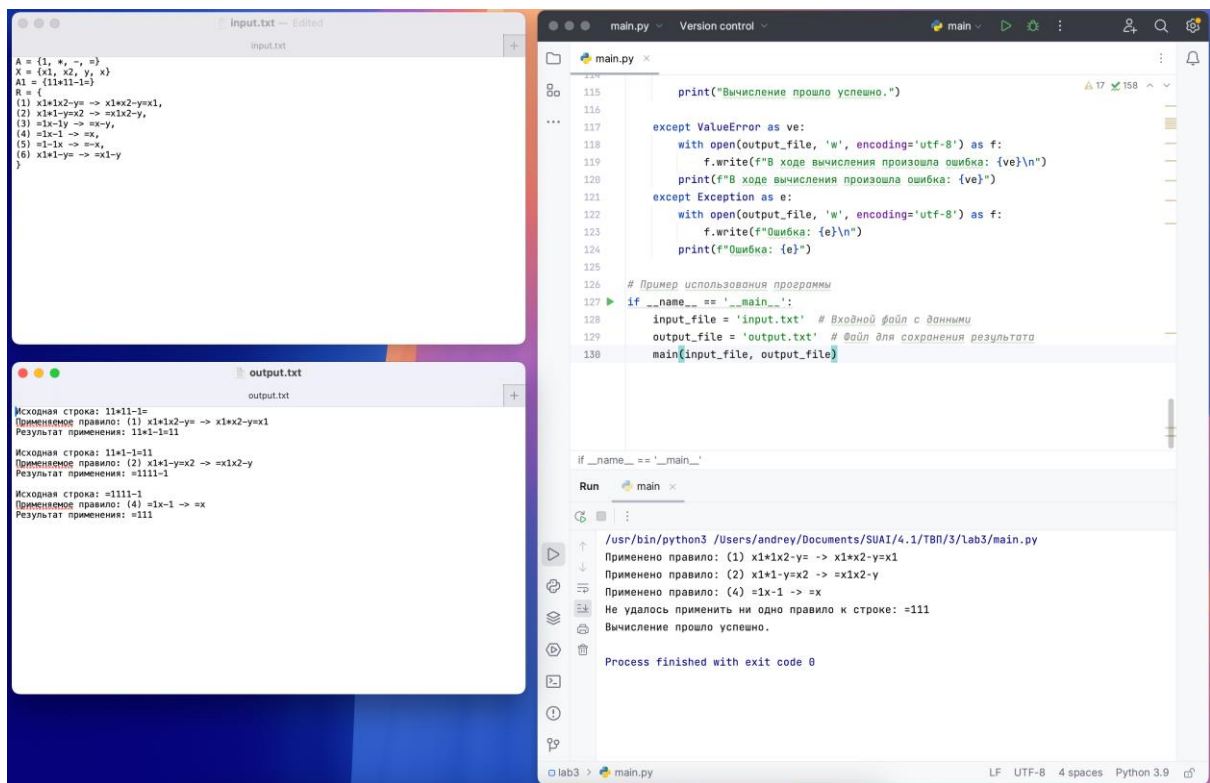
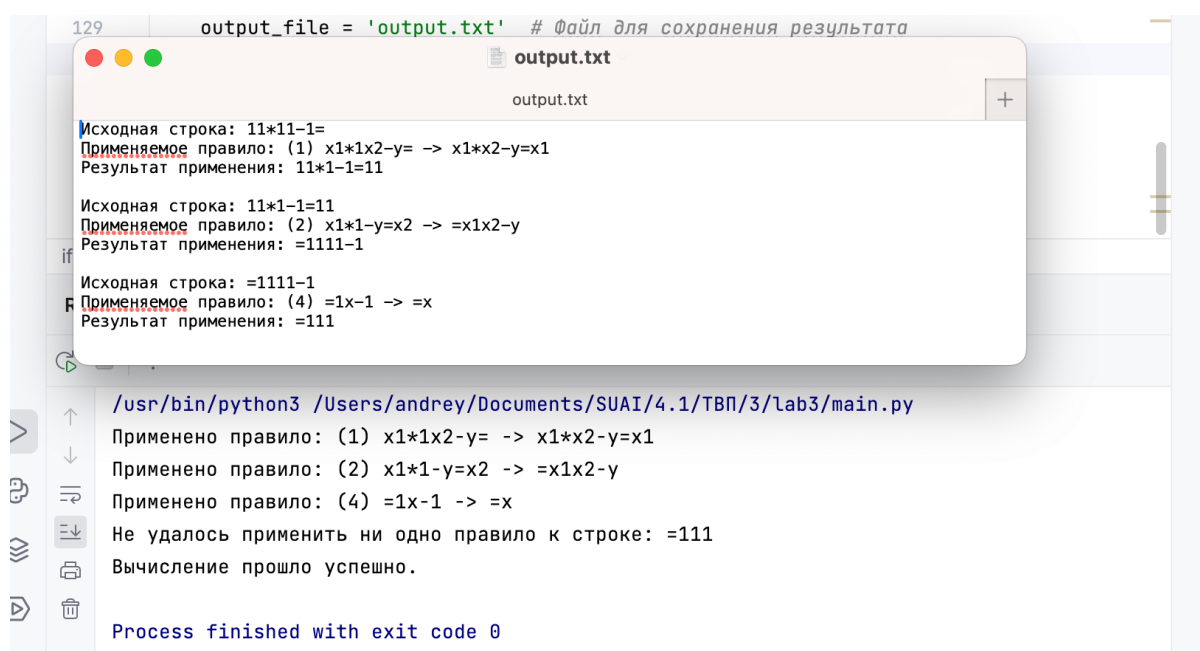


Рисунок 7 – корректные данные и корректное решение

Содержимое выходных файлов и сообщения, выводимые на экран:



The screenshot shows a code editor with a Python script and a terminal window displaying its output. The script is located at `/usr/bin/python3 /Users/andrey/Documents/SUAI/4.1/ТВП/3/lab3/main.py`. The terminal output shows the application of rules to the expression `11*11-1`.

```
output_file = 'output.txt' # Файл для сохранения результата

Исходная строка: 11*11-1=
Применяемое правило: (1) x1*x2-y= -> x1*x2-y=x1
Результат применения: 11*11-1=11

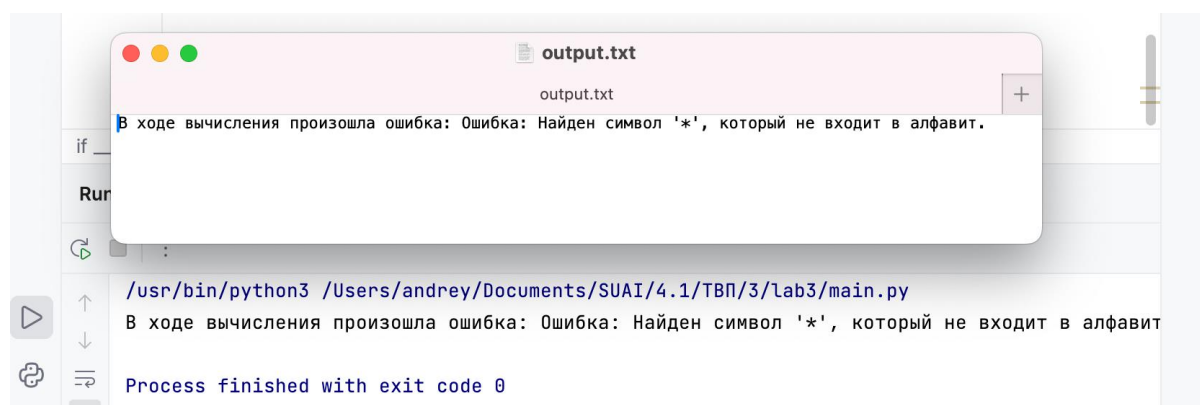
Исходная строка: 11*11-1=11
Применяемое правило: (2) x1*1-y=x2 -> =x1x2-y
Результат применения: =1111-1

Исходная строка: =1111-1
Применяемое правило: (4) =1x-1 -> =x
Результат применения: =111

/usr/bin/python3 /Users/andrey/Documents/SUAI/4.1/ТВП/3/lab3/main.py
Применено правило: (1) x1*x2-y= -> x1*x2-y=x1
Применено правило: (2) x1*1-y=x2 -> =x1x2-y
Применено правило: (4) =1x-1 -> =x
Не удалось применить ни одно правило к строке: =111
Вычисление прошло успешно.

Process finished with exit code 0
```

Рисунок 8 – выходные данные при корректных данных



The screenshot shows the same code editor and terminal window, but with an error message displayed in the terminal. The error message indicates that the symbol '*' is not in the alphabet.

```
output.txt

В ходе вычисления произошла ошибка: Ошибка: Найден символ '*', который не входит в алфавит.

/usr/bin/python3 /Users/andrey/Documents/SUAI/4.1/ТВП/3/lab3/main.py
В ходе вычисления произошла ошибка: Ошибка: Найден символ '*', который не входит в алфавит

Process finished with exit code 0
```

Рисунок 9 – выходные данные при ошибках

Вывод:

В ходе выполнения лабораторной работы была успешно построена формальная система Поста, предназначенная для вычисления заданной арифметической функции. Разработанная программа корректно обрабатывает входные данные и обеспечивает вывод результатов вычислений в соответствии с заданными требованиями. Все аспекты задачи были освоены и реализованы на высоком уровне, что позволило достигнуть поставленных целей.

Листинг программы:

```
import re

# Функция для чтения входных данных из файла
def read_input(file_name):
    with open(file_name, 'r', encoding='utf-8') as f:
        data = f.read()

    # Чтение алфавита, множества переменных, аксиом и правил
    A = re.findall(r'A = {(.*?)}', data)[0].split(' ')
    X = re.findall(r'X = {(.*?)}', data)[0].split(' ')
    A1 = re.findall(r'A1 = {(.*?)}', data)[0].split(' ')
    R = re.findall(r'R = {(.*?)}', data, re.DOTALL)[0].split('\n')
    R = [rule.strip() for rule in R if rule.strip() != '']

    return A, X, A1, R

# Функция для проверки строки на наличие символов, не входящих в алфавит
def check_alphabet(input_str, alphabet):
    for char in input_str:
        if char not in alphabet:
            raise ValueError(f"Ошибка: Найден символ '{char}', который не входит в алфавит.")

# Функция для проверки строк на наличие переменных, не входящих в заданное множество
def check_variables(input_str, variables):
    found_vars = re.findall(r'[a-zA-Z]+\d*', input_str) # Найти все переменные
    for var in found_vars:
        if var not in variables:
            raise ValueError(f"Найдена переменная, не входящая в заданное множество: {var}")

# Функция для проверки формата аксиомы
def check_axiom_format(axiom):
    # Проверяем, соответствует ли аксиома формату число*число-число=
    pattern = r'^\d+\*\d+\-\d+=\d$'
    if not re.match(pattern, axiom):
        raise ValueError("Ошибка: Аксиома должна иметь формат число*число-число=.")

# Функция для проверки правил на наличие недопустимых переменных
def check_rules_variables(rules, variables):
    for rule in rules:
        # Извлечение правых частей правил
        parts = rule.split('->')
        for part in parts:
            check_variables(part.strip(), variables) # Проверка на наличие недопустимых переменных

# Функция для применения правил с использованием регулярных выражений
```

```

def apply_rules(start_str, rules):
    current_str = start_str
    result_log = []

    rule_patterns = [
        (r'(\d+)\*1(\d+)-(\d+)=', r'\1*\2-\3=\1'), # Правило 1
        (r'(\d+)\*1-(\d+)=(\d+)', r'=\1\3-\2'), # Правило 2
        (r'=1(\d+)-1(\d+)', r'=\1-\2'), # Правило 3
        (r'=1(\d+)-1', r'=\1'), # Правило 4
        (r'=1-1(\d+)', r'=-\1'), # Правило 5
        (r'(\d+)\*1-(\d+)=', r'=\1-\2'), # Правило 6
    ]

    rule_names = [
        "(1) x1*x2-y= -> x1*x2-y=x1",
        "(2) x1*1-y=x2 -> =x1 x2-y",
        "(3) =1x-1y -> =x-y",
        "(4) =1x-1 -> =x",
        "(5) =1-1x -> =-x",
        "(6) x1*1-y= -> =x1-y",
    ]

    while True:
        applied = False

        for i, (pattern, replacement) in enumerate(rule_patterns):
            match = re.search(pattern, current_str)
            if match:
                result_log.append(f'Исходная строка: {current_str}')
                result_log.append(f'Применяемое правило: {rule_names[i]}')
                current_str = re.sub(pattern, replacement, current_str, count=1)
                result_log.append(f'Результат применения: {current_str}\n')
                applied = True
                print(f'Применено правило: {rule_names[i]}')
                break

        if not applied:
            print(f'Не удалось применить ни одно правило к строке: {current_str}')
            break

    return current_str, result_log

# Основная функция программы
def main(input_file, output_file):
    try:
        # Открываем файл для записи (очищаем его содержимое)
        with open(output_file, 'w', encoding='utf-8') as f:
            f.write("") # Очищаем файл, записывая пустую строку

        # Чтение входных данных
        alphabet, variables, axioms, rules = read_input(input_file)

```

```

# Проверка каждой аксиомы на соответствие алфавиту и множеству переменных
for axiom in axioms:
    check_alphabet(axiom, alphabet)
    check_variables(axiom, variables)
    check_axiom_format(axiom) # Проверка формата аксиомы

# Проверка правил на наличие недопустимых переменных
check_rules_variables(rules, variables)

# Применение правил вывода к аксиоме
current_str, log = apply_rules(axiom, rules)

# Запись результата в файл
with open(output_file, 'a', encoding='utf-8') as f: # Открываем файл в режиме
добавления
    f.write(f"Результаты для аксиомы: {axiom}\n")
    for line in log:
        f.write(line + '\n')

# Проверка: если лог пуст, записать об этом в файл
if not log:
    with open(output_file, 'a', encoding='utf-8') as f:
        f.write(f"Ни одно правило не было применено для аксиомы: {axiom}\n")

print("Вычисление прошло успешно.")

except ValueError as ve:
    with open(output_file, 'w', encoding='utf-8') as f:
        f.write(f"В ходе вычисления произошла ошибка: {ve}\n")
    print(f"В ходе вычисления произошла ошибка: {ve}")
except Exception as e:
    with open(output_file, 'w', encoding='utf-8') as f:
        f.write(f"Ошибка: {e}\n")
    print(f"Ошибка: {e}")

# использование программы
if __name__ == '__main__':
    input_file = 'input.txt' # Входной файл с данными
    output_file = 'output.txt' # Файл для сохранения результата
    main(input_file, output_file)

```