ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дожность |  |  |  | [Рогачев С.A](https://pro.guap.ru/inside/profile/911) |
| [старший преподаватель](https://guap.ru/rasp/?p=317) |  | подпись, дата |  | инициалы, фамилия |

|  |  |
| --- | --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3 |  |
| [ФС Поста](https://pro.guap.ru/inside/student/tasks/168449) |  |
| по дисциплине: [Теория вычислительных процессов](https://pro.guap.ru/inside/students/subjects/3391377) |  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4236 |  |  |  | Л. Мвале |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург

2025

1. **Цель работы**

Цель данной лабораторной работы – изучить принципы формальной системы Поста (FSp), понять её структуру и правила вывода, а также приобрести практические навыки построения такой системы для заданной арифметической функции. Дополнительно работа направлена на разработку программы на языке высокого уровня, которая имитирует процесс вычислений в системе Поста, что позволяет закрепить связь между теоретическими концепциями и практическим программированием.

**2. Постановка задачи**

Требуется:

–Построить формальную систему Поста FSpFSpFSp, реализующую вычисление заданной арифметической функции.

–Разработать программу на языке высокого уровня, эмулирующую процесс вывода в данной системе.

–Обеспечить, чтобы программа:

* Считывала все необходимые данные (алфавит, множество переменных, аксиомы и правила продукции) из одного входного файла;
* Корректно применяла правила продукции пошагово до тех пор, пока не останется применимых правил;
* Обрабатывала ошибочные входные данные, включая обнаружение символов, не входящих в алфавит, и переменных, не принадлежащих заданному множеству;
* Выводила результаты каждого шага в файл результатов, включая исходную строку, применённое правило и результат применения правила;
* Отображала на экране сообщения об успешном завершении вычислений или о возникших ошибках.

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

Для твоего кода (умножение x1 ∗x2):

Формальная система Поста FSp, реализующая вычисление функции

f(x1, x2)=x1 ∗x2​

Здесь:

* Алфавит: A={1,∗,=,/}
* Множество переменных: X={a,b,c}
* Аксиома: A1={a ∗b = / =}
* Правила продукции R:

1. a ∗1b = /c =  →  a ∗b = /ca =
2. ∗= /c =  →  /c =
3. /c1 =  →  c =
4. /c =  →  c

* Входные данные (пример):

a=111, b=11

что соответствует 3×2

* Результат вычисления: последовательное применение правил приводит к выводу

111111

(шесть единиц), что эквивалентно 3\*2 = 6

**4. Список программ**

#!/usr/bin/env python3

# post\_simulator.py - Enhanced version with INPUT section support

import sys

import re

def parse\_input\_file(filename):

    """Parse input file with support for INPUT section"""

    try:

        with open(filename, 'r', encoding='utf-8') as f:

            content = f.read()

    except FileNotFoundError:

        raise FileNotFoundError(f"Error: File '{filename}' not found.")

    # Initialize components

    A = set()

    X = set()

    A1 = set()

    R = []

    INPUT = {}

    # Parse each section using regex

    sections = {

        'A': r'A\s\*=\s\*\{([^}]\*)\}',

        'X': r'X\s\*=\s\*\{([^}]\*)\}',

        'A1': r'A1\s\*=\s\*\{([^}]\*)\}',

        'R': r'R\s\*=\s\*\{([^}]\*)\}',

        'INPUT': r'INPUT\s\*=\s\*\{([^}]\*)\}'

    }

    for section, pattern in sections.items():

        match = re.search(pattern, content)

        if match:

            items = [item.strip() for item in match.group(1).split(',') if item.strip()]

            if section == 'A':

                A = set(items)

            elif section == 'X':

                X = set(items)

            elif section == 'A1':

                A1 = set(items)

            elif section == 'R':

                for rule\_str in items:

                    if '->' in rule\_str:

                        lhs, rhs = [part.strip() for part in rule\_str.split('->', 1)]

                        R.append((lhs, rhs))

            elif section == 'INPUT':

                for pair in items:

                    if '=' in pair:

                        var, value = [part.strip() for part in pair.split('=', 1)]

                        INPUT[var] = value

    return A, X, A1, R, INPUT

def substitute\_variables(string, variables):

    """Substitute variables with their values"""

    result = string

    for var, value in variables.items():

        result = result.replace(var, value)

    return result

def validate\_string(s, A, X):

    """Validate that all characters in string are in alphabet A"""

    for char in s:

        if char not in A:

            return False, f"Symbol '{char}' not in alphabet A"

    return True, ""

def apply\_rule(current\_string, rule, A, X):

    """Try to apply a rule to the current string"""

    lhs, rhs = rule

    # Try to match the rule at every position

    for i in range(len(current\_string)):

        substitutions = {}

        lhs\_index = 0

        current\_index = i

        match = True

        while lhs\_index < len(lhs) and current\_index < len(current\_string):

            if lhs[lhs\_index] in X:

                # Variable - capture everything until next constant

                var = lhs[lhs\_index]

                start = current\_index

                # Look for the end of this variable (next constant in pattern or end)

                # Find next constant in lhs after this variable

                next\_const = None

                for j in range(lhs\_index + 1, len(lhs)):

                    if lhs[j] not in X:

                        next\_const = lhs[j]

                        break

                if next\_const:

                    idx = current\_string.find(next\_const, current\_index)

                    if idx == -1:

                        match = False

                        break

                    value = current\_string[start:idx]

                    current\_index = idx

                else:

                    # Variable goes till end

                    value = current\_string[start:]

                    current\_index = len(current\_string)

                value = current\_string[start:current\_index]

                substitutions[var] = value

                lhs\_index += 1

            else:

                # Constant - must match exactly

                if current\_string[current\_index] != lhs[lhs\_index]:

                    match = False

                    break

                current\_index += 1

                lhs\_index += 1

        # Check if we fully matched the pattern

        if match and lhs\_index == len(lhs):

            # Substitute variables in the right side

            new\_part = rhs

            for var, value in substitutions.items():

                new\_part = new\_part.replace(var, value)

            # Validate the result

            valid, error = validate\_string(new\_part, A, X)

            if not valid:

                raise ValueError(f"Rule application error: {error}")

            # Create the new string

            new\_string = current\_string[:i] + new\_part + current\_string[current\_index:]

            return new\_string, substitution

    return current\_string, None  # Return current\_string instead of None

def main():

    if len(sys.argv) != 2:

        print("Usage: python post\_simulator.py <input\_file>")

        return

    try:

        # Parse input file

        A, X, A1, R, INPUT = parse\_input\_file(sys.argv[1])

        # --- Validation ---

        if not A1:

            print("Error: No axioms found")

            return

        axiom\_template = next(iter(A1))

        # Validate that axiom only contains symbols from alphabet or variables

        for ch in axiom\_template:

            if ch not in A and ch not in X:

                print(f"Error: Symbol '{ch}' in axiom not in alphabet A or variables X")

                return

        # Validate INPUT variables are all declared in X

        for var in INPUT.keys():

            if var not in X:

                print(f"Error: Variable '{var}' in INPUT not in declared set X")

                return

        # Substitute variables if INPUT section exists

        current\_string = substitute\_variables(axiom\_template, INPUT) if INPUT else axiom\_template

        step = 0

        max\_steps = 1000

        output\_filename = "output.txt"

        final\_result = ""  # Store last seen /…= value

        with open(output\_filename, "w", encoding='utf-8') as output\_file:

            output\_file.write(f"Initial string: {current\_string}\n\n")

            while step < max\_steps:

                rule\_applied = False

                for rule in R:

                    try:

                        new\_string, substitutions = apply\_rule(current\_string, rule, A, X)

                    except ValueError as e:

                        # Symbol not in alphabet detected in rule application

                        print(f"Error: {e}")

                        output\_file.write(f"Error: {e}\n")

                        return

                    if substitutions is not None:

                        # Update final\_result if /…= pattern exists

                        match = re.search(r'/([1]+)=', new\_string)

                        if match:

                            final\_result = match.group(1)

                        # Write step

                        output\_file.write(f"Step {step + 1}:\n")

                        output\_file.write(f"Original string: {current\_string}\n")

                        output\_file.write(f"Applied rule: {rule[0]} -> {rule[1]}\n")

                        output\_file.write(f"Result: {new\_string}\n\n")

                        current\_string = new\_string

                        rule\_applied = True

                        step += 1

                        break

                if not rule\_applied:

                    output\_file.write("Computation completed successfully. No more rules apply.\n")

                    print("Computation completed successfully.")

                    break

            else:

                output\_file.write("Computation stopped: Maximum step limit reached.\n")

                print("Warning: Maximum step limit reached")

            output\_file.write(f"Final result: {final\_result}\n")

        print(f"Computation results written to {output\_filename}")

        print(f"Final computed result: {final\_result}")

    except Exception as e:

        print(f"Unexpected error: {e}")

        import traceback

        traceback.print\_exc()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

1. **Содержимое входного файла**

A ={1, \*, =, /};

X ={a, b, c};

A1 ={a\*b=/=};

INPUT ={a=11, b=11};

R ={

a\*1b=/c= -> a\*b=/ca=,

\*=/c= -> /c=,

/c1= -> c=,

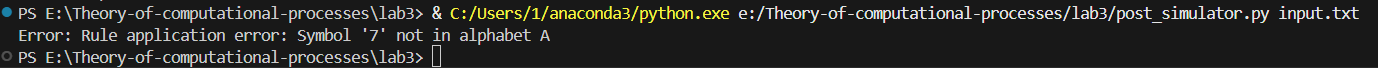
/c= -> c

};

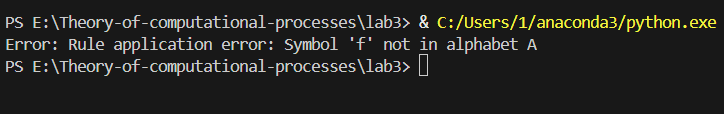
1. **Примеры результатов выполнения**

Неверные входные данные

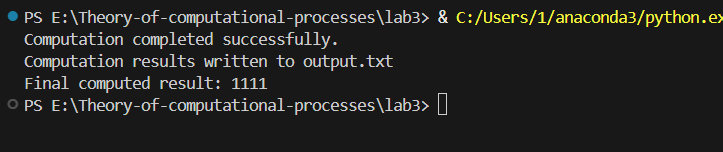
INPUT ={a=71, b=11};

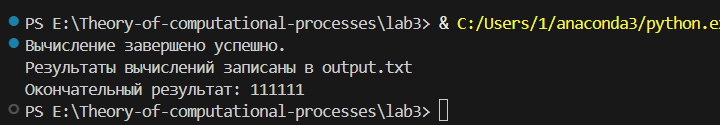


INPUT ={a=f1, b=11};



Верные входные данные





1. **Содержимое выходных файлов**

Output.txt

Начальная строка: 11\*111=/=

Шаг 1:

Исходная строка: 11\*111=/=

Применено правило: a\*1b=/c= -> a\*b=/ca=

Результат: 11\*11=/11=

Шаг 2:

Исходная строка: 11\*11=/11=

Применено правило: a\*1b=/c= -> a\*b=/ca=

Результат: 11\*1=/1111=

Шаг 3:

Исходная строка: 11\*1=/1111=

Применено правило: a\*1b=/c= -> a\*b=/ca=

Результат: 11\*=/111111=

Шаг 4:

Исходная строка: 11\*=/111111=

Применено правило: \*=/c= -> /c=

Результат: 11/111111=

Шаг 5:

Исходная строка: 11/111111=

Применено правило: /c= -> c

Результат: 11111111

Вычисление завершено успешно. Правила больше не применимы.

Итоговый результат: 111111

**Выводы**

В ходе выполнения лабораторной работы была успешно построена формальная система Поста, реализующая вычисление заданной арифметической функции. Разработана программа, моделирующая процесс применения правил в системе, работающая с произвольными входными данными из заданных множеств. Программа корректно обрабатывает правильные входные данные и выявляет ошибочные, формируя подробный журнал шагов вычислений и выводя понятные сообщения о ходе выполнения. Это позволило глубже понять теоретические основы формальной системы Поста и приобрести практические навыки моделирования и имитации формальных систем с помощью средств программирования.