ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Дожность		Рогачев С.А
старший преподаватель	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

ФС Поста

по дисциплине: Теория вычислительных процессов

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 $x_1 * x_2$):

Формальная система Поста FSp, реализующая вычисление функции

$$f(x_1, x_2)=x_1 * x_2$$

Здесь:

- Алфавит: A={1,* ,=,/}
- Множество переменных: X={a,b,c}
- Аксиома: A_1={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()
```

5. 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

};

6. Примеры результатов выполнения

Неверные входные данные

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

```
PS E:\Theory-of-computational-processes\lab3> & C:/Users/1/anaconda3/python.exe e
Error: Rule application error: Symbol '7' not in alphabet A
PS E:\Theory-of-computational-processes\lab3>
```

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

```
PS E:\Theory-of-computational-processes\lab3> & C:/Users/1/anaconda3/python.exe
Error: Rule application error: Symbol 'f' not in alphabet A
PS E:\Theory-of-computational-processes\lab3>
```

Верные входные данные

```
PS E:\Theory-of-computational-processes\lab3> & C:/Users/1/anaconda3/python.ex
Computation completed successfully.
Computation results written to output.txt
Final computed result: 1111
PS E:\Theory-of-computational-processes\lab3>
```

```
PS E:\Theory-of-computational-processes\lab3> & C:/Users/1/anaconda3/python.e
Вычисление завершено успешно.
Результаты вычислений записаны в output.txt
Окончательный результат: 111111
PS E:\Theory-of-computational-processes\lab3>
```

7. 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.