

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего
образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И ПРОГРАММИРОВАНИЯ

КАФЕДРА компьютерных технологий и программной инженерии

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

ассистент

Д.А. Кочин

должность, уч. степень,
звание

подпись, дата

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

«АЛГОРИТМИЧЕСКАЯ СИСТЕМА ПОСТА»

по курсу: Теория вычислительных процессов

РАБОТУ ВЫПОЛНИЛА

СТУДЕНТКА ГР. 4031
№

Х.В. Сидиропуло

подпись, дата

инициалы, фамилия

1. Цель работы

Реализовать Машину Поста на высокоуровневом языке программирования.

2. Задание

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

3. Листинг программы

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;

public class Command
{
    private readonly string _left;
    private readonly string _right;
    private readonly HashSet<char> _variables;
    private readonly string _txtCommand;

    public Command(Dictionary<string, string> command, HashSet<char> alphabet,
        HashSet<char> variables, string axioms)
    {
        _left = TranslateLeftCommand(command["left"], alphabet, variables, axioms);
        _right = TranslateRightCommand(command["right"], alphabet, variables);
        _variables = variables;
        _txtCommand = $"{command["left"]} {command["right"]}";
    }

    public string Execute(string input)
    {
        List<string> output = new List<string>();
        int prevStart = 0;

        MatchCollection matches = Regex.Matches(input, _left);
        foreach (Match match in matches)
        {
            int start = match.Index;
            int end = start + match.Length;
            output.Add(input.Substring(prevStart, start - prevStart));
            prevStart = end;
        }

        if (prevStart != 0)
        {
            output.Add(input.Substring(prevStart));
        }
    }
}
```

```

else
{
    return null;
}

var groupValues = Regex.Match(input, _left).Groups;
var groupDict = new Dictionary<string, string>();
foreach (var groupName in groupValues.Cast<Group>().Select(g => g.Name))
{
    if (_variables.Contains(groupName[0]))
    {
        groupDict[groupName] = groupValues[groupName].Value;
    }
}

string result = string.Join(_right, output).FormatWith(groupDict);
Console.WriteLine($"Command executed: {input} -- {_txtCommand} -->
{result}");
return result;
}

public string GetTxtCommand()
{
    return _txtCommand;
}

private string TranslateLeftCommand(string command, HashSet<char> alphabet,
HashSet<char> variables, string axioms)
{
    const string specialSymbols = "!@#$%^&*()[]{},;.:<>?\\'\\|`~=_+";
    var checkedVariables = new HashSet<char>();
    var result = "";

    foreach (char ch in command)
    {
        if (specialSymbols.Contains(ch))
        {
            result += $"\\{ch}";
        }
        else if (alphabet.Contains(ch))
        {
            result += ch;
        }
        else if (variables.Contains(ch))
        {
            if (checkedVariables.Contains(ch))
            {
                result += $"(<{ch}>";
            }
            else
            {
                checkedVariables.Add(ch);
                result += $"(<{ch}>[{axioms}]*";
            }
        }
        else
        {
            throw new ArgumentException("No such character in alphabet");
        }
    }

    return result;
}

```

```

        private string TranslateRightCommand(string command, HashSet<char> alphabet,
        HashSet<char> variables)
        {
            const string specialSymbols = @"\{\}";
            var result = "";

            foreach (char ch in command)
            {
                if (specialSymbols.Contains(ch))
                {
                    result += $"\\{ch}";
                }
                else if (alphabet.Contains(ch))
                {
                    result += ch;
                }
                else if (variables.Contains(ch))
                {
                    result += $"{{{ch}}}";
                }
                else
                {
                    throw new ArgumentException("No such character in alphabet");
                }
            }

            return result;
        }
    }

    public static class StringExtensions
    {
        public static string FormatWith(this string format, Dictionary<string, string>
        args)
        {
            foreach (var arg in args)
            {
                format = format.Replace("{ " + arg.Key + " }", arg.Value);
            }

            return format;
        }
    }

    public class PostMachine
    {
        private readonly HashSet<char> _A;
        private readonly HashSet<char> _X;
        private readonly HashSet<char> _A1;
        private readonly List<Command> _R;
        private readonly string _outPath;

        public PostMachine(HashSet<char> A, HashSet<char> X, HashSet<char> A1,
        List<Command> R, string outPath)
        {
            _A = A;
            _X = X;
            _A1 = A1;
            _R = R;
            _outPath = outPath;
        }

        public string Fit(string input)
        {
            int step = 0;

```

```

Console.WriteLine("Инициализация переменных и чтение файлов:");
Console.WriteLine($"    Алфавит: {{{string.Join(", ", _A)}}}.");
Console.WriteLine($"    Переменные: {{{string.Join(", ", _X)}}}.");
Console.WriteLine($"    Аксиомы: {{{string.Join(", ", _A1)}}}.");
Console.WriteLine($"    Входные данные: \"{input}\".");
Console.WriteLine($"    Правила:");
foreach (var rule in _R)
{
    Console.WriteLine($"        {rule.GetTxtCommand()}");
}
Console.WriteLine("\n    Запуск машины Поста:");

using (StreamWriter outputStream = new StreamWriter(_outPath))
{
    int failedCommandsNum = _R.Count;

    while (failedCommandsNum != 0)
    {
        failedCommandsNum = _R.Count;
        step++;

        Console.WriteLine($"            Шаг {step}:");

        foreach (Command comm in _R)
        {
            string commRes = comm.Execute(input);
            if (commRes != null)
            {
                Console.WriteLine($"                Применяется правило
{comm.GetTxtCommand()}.");
                Console.WriteLine($"                Срабатывает для подстроки
\"{commRes}\".");
                Console.WriteLine($"                Результат: \"{commRes}\".");
                Console.WriteLine();
                outputStream.WriteLine($"Шаг {step}:");
                outputStream.WriteLine($" {input} -- {comm.GetTxtCommand()} -->
{commRes}");

                input = commRes;
                break;
            }
            else
            {
                failedCommandsNum--;
            }
        }
    }

    Console.WriteLine($"        Вывод результатов:");
    Console.WriteLine($"        Выводится промежуточный результат для каждого
примененного правила:");
    Console.WriteLine($"        {input}.");
    Console.WriteLine($"        Выводится окончательный результат: Final result:
{input}.");

    return input;
}

}

class Program
{
    static void Main()
    {
        HashSet<char> alph =
        GetAlphabet(Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
        "C:\\Users\\User\\source\\repos\\TVP LR 3\\TVP LR 3\\alphabet.txt"));
    }
}

```

```

        HashSet<char> variables =
GetVariables(Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"C:\\Users\\User\\source\\repos\\TVP LR 3\\TVP LR 3\\alphabet_variables.txt"));
        HashSet<char> axioms =
GetAxioms(Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"C:\\Users\\User\\source\\repos\\TVP LR 3\\TVP LR 3\\axioms.txt"));
        string input = GetInput(Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"C:\\Users\\User\\source\\repos\\TVP LR 3\\TVP LR 3\\input.txt"));
        List<Command> commands =
GetCommands(Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"C:\\Users\\User\\source\\repos\\TVP LR 3\\TVP LR 3\\rule_inference.txt"), alph,
variables, axioms);

        PostMachine pMach = new PostMachine(alph, variables, axioms, commands,
Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
"C:\\Users\\User\\source\\repos\\TVP LR 3\\TVP LR 3\\output.txt"));
        string result = pMach.Fit(input);

        Console.WriteLine($"-----\nFinal result: {result}");
    }

    static HashSet<char> GetAlphabet(string path)
    {
        AssertFileExists(path);
        string alphabet;
        using (StreamReader reader = new StreamReader(path))
        {
            alphabet = reader.ReadLine();
        }
        return new HashSet<char>(alphabet.Split(' ').Select(c => c[0]));
    }

    static HashSet<char> GetVariables(string path)
    {
        AssertFileExists(path);
        string variables;
        using (StreamReader reader = new StreamReader(path))
        {
            variables = reader.ReadLine();
        }

        return new HashSet<char>(variables.Split(' ').Select(c => c[0]));
    }

    static HashSet<char> GetAxioms(string path)
    {
        AssertFileExists(path);
        string axioms;
        using (StreamReader reader = new StreamReader(path))
        {
            axioms = reader.ReadLine();
        }

        return new HashSet<char>(axioms.Split(' ').Select(c => c[0]));
    }

    static List<Command> GetCommands(string path, HashSet<char> alphabet,
HashSet<char> variables, HashSet<char> axioms)
    {
        AssertFileExists(path);
        List<Command> commands = new List<Command>();
        string[] commandText = File.ReadAllLines(path);

        foreach (string txt in commandText)
        {

```

```

        string[] parts = txt.Split(' ');
        Dictionary<string, string> commDict = new Dictionary<string, string>
        {
            {"left", parts[0]},
            {"right", parts[1]}
        };
        Command comm = new Command(commDict, alphabet, variables,
string.Join("", axioms));
        commands.Add(comm);
    }

    return commands;
}

static string GetInput(string path)
{
    AssertFileExists(path);
    using (StreamReader reader = new StreamReader(path))
    {
        return reader.ReadLine();
    }
}

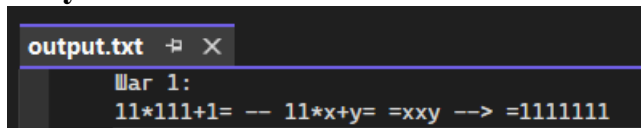
static void AssertFileExists(string path)
{
    if (!File.Exists(path))
    {
        throw new FileNotFoundException($"File not found: {path}");
    }
}
}

```

4. Содержимое входных файлов:

alphabet.txt	1 + * =
alphabet_variables.txt	x y
axioms.txt	1
rule_inference.txt	11*x+y= =xxy
input.txt	11*111+1=

5. Результат выполнения



```

output.txt
Mar 1:
11*111+1= -- 11*x+y= =xxy --> =111111

```

Инициализация переменных и чтение файлов:

Алфавит: {1, +, *, =}.

Переменные: {x, y}.

Аксиомы: {1}.

Входные данные: "11*111+1=".

Правила:

$11*x+y = xxy$

Запуск машины Поста:

Шаг 1:

Command executed: 11*111+1= -- 11*x+y= xxy --> =1111111

Применяется правило $11*x+y = xxy$.

Срабатывает для подстроки "=1111111".

Результат: "=1111111".

Шаг 2:

Вывод результатов:

Выводится промежуточный результат для каждого примененного правила:
=1111111.

Выводится окончательный результат: Final result: =1111111.

Final result: =1111111