МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №1

по дисциплине: «Теория автоматов и формальных языков»

тема: «Формальные грамматики. Выводы»

Выполнил студент группы ПВ-222

Короткунов Александр Александрович

Проверили

Рязанов Юрий Дмитриевич

Белгород 2024 г.

Цель работы: изучить основные понятия теории формальных языков и грамматик.

Вариант 1

Задание 1. Написать программу, выполняющую левый вывод в заданной КС-грамматике.

```
class Rule:
    def __init__(self, input, output):
        self.input = input
        self.output = output

    def __str__(self):
        return f'{self.input} → {''.join(map(str, self.output))}'
```

```
class Node:
    def __init__(self, value: str, children=None):
        self.value = value
        self.children = children

    def get_tree_representation(self):
        if self.children:
            return f"{self.value}({''.join([x.get_tree_representation()
for x in self.children])})"
        return f"{self.value}"

    def get_formatted_tree_representation(self, tabs=0):
        if self.children:
            return f"\n{'\t' * tabs}{tabs}. {self.value}
{''.join([x.get_formatted_tree_representation(tabs + 1) for x in
self.children])}"
        return f"\n{'\t' * tabs}{tabs}. {self.value}"

    def get_plain_representation(self):
        if self.children:
            return ''.join([x.get_plain_representation() for x in
self.children])
        return self.value

    def find_first_non_terminal_node_left(self):
        return self.find_node_left(lambda x: x.value.isupper())

    def find_non_terminal_node_by_value_left(self, value):
        return self.find_node_left(lambda x: x.value == value)

    def find_first_non_terminal_node_right(self):
        return self.find_node_right(lambda x: x.value.isupper())
```

```
    def find_non_terminal_node_by_value_right(self, value):
        return self.find_node_right(lambda x: x.value == value)

    def find_node_left(self, predicate):
        if self.children:
            stack = []
            for i in self.children:
                stack.append(i.find_node_left(predicate))
            stack = [x for x in stack if x is not None]
            if stack:
                return stack[0]
        elif predicate(self):
            return self
        else:
            return None

    def find_node_right(self, predicate):
        if self.children:
            stack = []
            for i in self.children[::-1]:
                stack.append(i.find_node_right(predicate))
            stack = [x for x in stack if x is not None]
            if stack:
                return stack[0]
        elif predicate(self):
            return self
        else:
            return None

    def __str__(self):
        return str(self.value)
```

```
from Node import Node
from Rule import Rule


class LeftOutput:
    def __init__(self, grammar_rules):
        self.root = Node('S')
        self.grammar_rules = grammar_rules
        self.steps = []
        self.step_index = 1

    def apply(self):
        while True:
            non_terminal_node = self.get_first_non_terminal()
            if not non_terminal_node:
                self.print_end_info()
                break
```

```
            else:
                rules =
self.get_applicable_rules_for_node(non_terminal_node)
                self.print_info(rules)
                rule, rule_index = self.get_rule_from_input(rules)
                self.apply_rule_for(non_terminal_node, rule)
                self.step_index += 1
                self.steps.append(rule_index)

    def get_first_non_terminal(self):
        return self.root.find_first_non_terminal_node()

    def get_applicable_rules_for_node(self, node):
        if not node:
            return None
        return [x for x in self.grammar_rules if x.input == node.value]

    def print_info(self, rules):
        print(f'### Шаг {self.step_index} ###')
        print(f'Промежуточная цепочка:
{self.root.get_plain_representation()}')
        print('Можно применить: ')
        for (i, rule) in enumerate(rules):
            print(f'{i}. {rule}')

    def print_end_info(self):
        print(f'### Шаг {self.step_index} ###')
        print(f'Терминальная цепочка:
{self.root.get_plain_representation()}')
        print(f'Последовательность правил: {' '.join(map(str,
self.steps))}')
        print(f'ЛСФ ДВ: {self.root.get_tree_representation()}')

    @staticmethod
    def get_rule_from_input(rules):
        rule_index = int(input("Выберите правило: "))
        print()
        return rules[rule_index], rule_index

    @staticmethod
    def apply_rule_for(node, rule):
        node.children = [Node(x) for x in rule.output]


def main():
    output = LeftOutput([
        Rule('S', ['a', 'S', 'S', 'a']),
        Rule('S', ['b', 'S', 'b']),
        Rule('S', ['c', 'A', 'b']),
        Rule('A', ['A', 'a']),
```

```
        Rule('A', ['B', 'a']),
        Rule('A', ['']),
        Rule('B', ['b', 'B']),
        Rule('B', ['a', 'A'])
    ])

    output.apply()


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

Задание 2. Выполнить левый (правый) вывод терминальной цепочки в заданной грамматике, построить дерево вывода. Определить, существует ли неэквивалентный вывод полученной цепочки и, если существует, представить его деревом вывода.

```
### Шаг 1 ###
Промежуточная цепочка: S
Можно применить:
0. S → aSSa
1. S → bSb
2. S → cAb
Выберите правило: 2

### Шаг 2 ###
Промежуточная цепочка: cAb
Можно применить:
0. A → Aa
1. A → Ba
2. A →
Выберите правило: 1

### Шаг 3 ###
Промежуточная цепочка: cBab
Можно применить:
0. B → bB
1. B → aA
Выберите правило: 1

### Шаг 4 ###
Промежуточная цепочка: caAab
Можно применить:
0. A → Aa
1. A → Ba
```

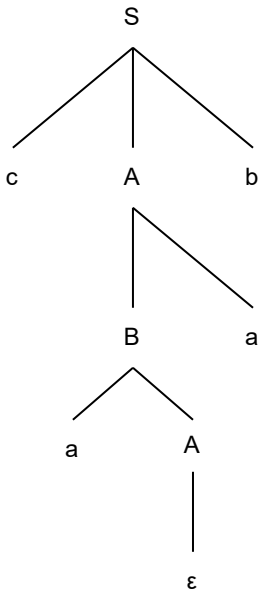
```
2. A →
Выберите правило: 2

### Шаг 5 ###
Терминальная цепочка: caab
Последовательность правил: 2 1 1 2
ЛСФ ДВ: S(cA(B(aA())a)b)
```

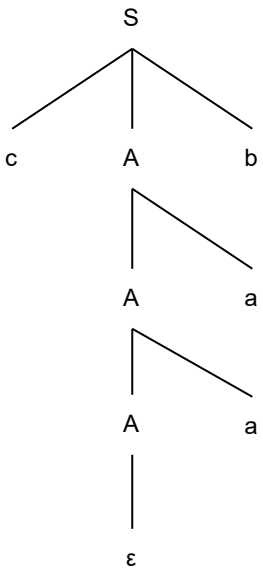
Для данного вывода нашёлся неэквивалетный вывод

```
### Шаг 1 ###
Промежуточная цепочка: S
Можно применить:
0. S → aSSa
1. S → bSb
2. S → cAb
Выберите правило: 2

### Шаг 2 ###
Промежуточная цепочка: cAb
Можно применить:
0. A → Aa
1. A → Ba
2. A →
Выберите правило: 0

### Шаг 3 ###
Промежуточная цепочка: cAab
Можно применить:
0. A → Aa
1. A → Ba
2. A →
Выберите правило: 0

### Шаг 4 ###
Промежуточная цепочка: cAaab
Можно применить:
0. A → Aa
1. A → Ba
2. A →
Выберите правило: 2

### Шаг 5 ###
```

```
Терминальная цепочка: caab
Последовательность правил: 2 0 0 2
ЛСФ ДВ: S(cA(A(A()a)a)b)
```

Деревья вывода:





Задание 3. Написать программу, определяющую, можно ли применить заданную последовательность правил при левом выводе цепочки в заданной КС-грамматике.

```
from operator import indexOf

from Node import Node
from Rule import Rule


class LeftOutput2:
    def __init__(self, grammar_rules):
        self.root = Node('S')
        self.grammar_rules = grammar_rules
        self.steps = []
        self.step_index = 1

    def apply(self):
        for index, rule in enumerate(self.grammar_rules):
            print(f'{index + 1}. {rule}')

        inputs = [int(x) - 1 for x in input('Введите последовательность
шагов: ').split(',')]
        for i in inputs:
            non_terminal_node = self.get_first_non_terminal()
            if not non_terminal_node:
                break
            else:
```

```
                rule = self.grammar_rules[i]
                if not self.can_rule_be_applied_to_node(non_terminal_node,
rule):
                    break
                self.apply_rule_for(non_terminal_node, rule)
                self.step_index += 1
                self.steps.append(i)

        self.print_end_info()

        if self.get_first_non_terminal() is None:
            print('Последовательность можно применить')
        else:
            print('Последовательность невозможно применить')

    @staticmethod
    def can_rule_be_applied_to_node(node, rule):
        return node.value == rule.input

    def get_first_non_terminal(self):
        return self.root.find_first_non_terminal_node()

    def get_applicable_rules_for_node(self, node):
        if not node:
            return None
        return [x for x in self.grammar_rules if x.input == node.value]

    def print_info(self, rules):
        print(f'### Шаг {self.step_index} ###')
        print(f'Промежуточная цепочка:
{self.root.get_plain_representation()}')
        print('Можно применить: ')
        for (i, rule) in [(indexOf(self.grammar_rules, x) + 1, x) for i, x
in enumerate(rules)]:
            print(f'{i}. {rule}')

    def print_end_info(self):
        print(f'Терминальная цепочка:
{self.root.get_plain_representation()}')
        print(f'Последовательность правил: {' '.join(map(str, [x + 1 for x
in self.steps]))}')
        print(f'ЛСФ ДВ: {self.root.get_tree_representation()}')

    @staticmethod
    def get_rule_from_input(rules, rule_index):
        return rules[rule_index], rule_index

    @staticmethod
    def apply_rule_for(node, rule):
        node.children = [Node(x) for x in rule.output]
```

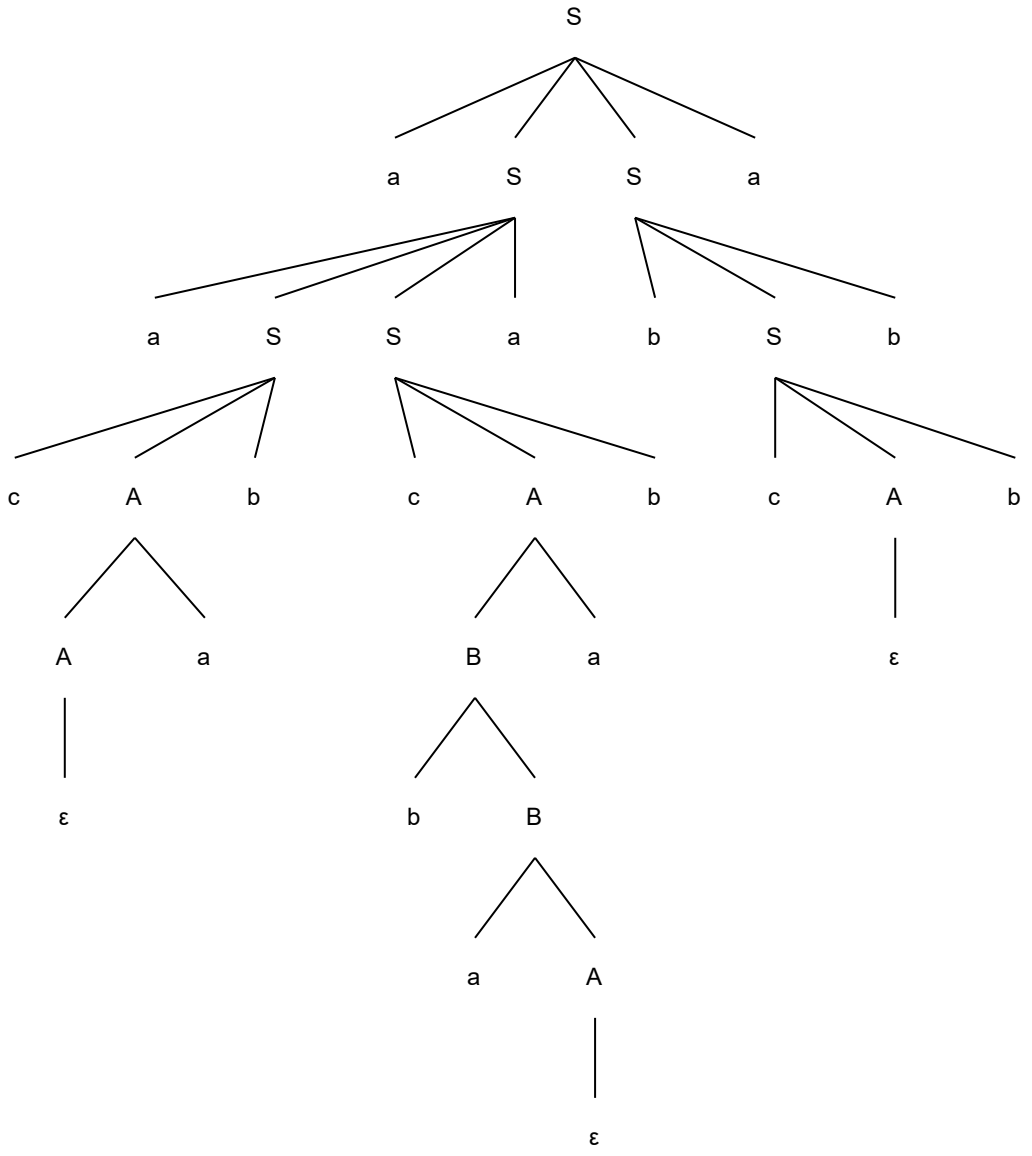
```
def main():
    output = LeftOutput2([
        Rule('S', ['a', 'S', 'S', 'a']),
        Rule('S', ['b', 'S', 'b']),
        Rule('S', ['c', 'A', 'b']),
        Rule('A', ['A', 'a']),
        Rule('A', ['B', 'a']),
        Rule('A', ['']),
        Rule('B', ['b', 'B']),
        Rule('B', ['a', 'A'])
    ])

    output.apply()


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

Задание 4. Для каждой последовательности правил (см. варианты заданий п.2) определить, можно ли её применить при левом (правом) выводе терминальной цепочки в заданной КС-грамматике, и, если можно, построить дерево вывода.

```
Введите последовательность шагов: 1, 1, 3, 4, 6, 3, 5, 7, 8, 6, 2, 3, 6
Терминальная цепочка: aacabcbaababcbba
Последовательность правил: 1 1 3 4 6 3 5 7 8 6 2 3 6
ЛСФ ДВ: S(aS(aS(cA(A()a)b)S(cA(B(bB(aA()))a)b)a)S(bS(cA()b)b)a)
Последовательность можно применить

Введите последовательность шагов: 1, 1, 2, 3, 3, 3, 4, 5, 6, 6, 7, 8, 6
Терминальная цепочка: aabcAbbSaSa
Последовательность правил: 1 1 2 3
ЛСФ ДВ: S(aS(aS(bS(cAb)b)Sa)Sa)
Последовательность невозможно применить

Введите последовательность шагов: 1, 3, 3, 5, 7, 4, 6, 8, 6, 2, 3, 6, 6
Терминальная цепочка: acAbSa
Последовательность правил: 1 3
ЛСФ ДВ: S(aS(cAb)Sa)
Последовательность невозможно применить

Введите последовательность шагов: 1, 2, 3, 6, 1, 3, 5, 7, 8, 6, 3, 4, 6
Терминальная цепочка: abcbbacbaabcabaa
Последовательность правил: 1 2 3 6 1 3 5 7 8 6 3 4 6
```

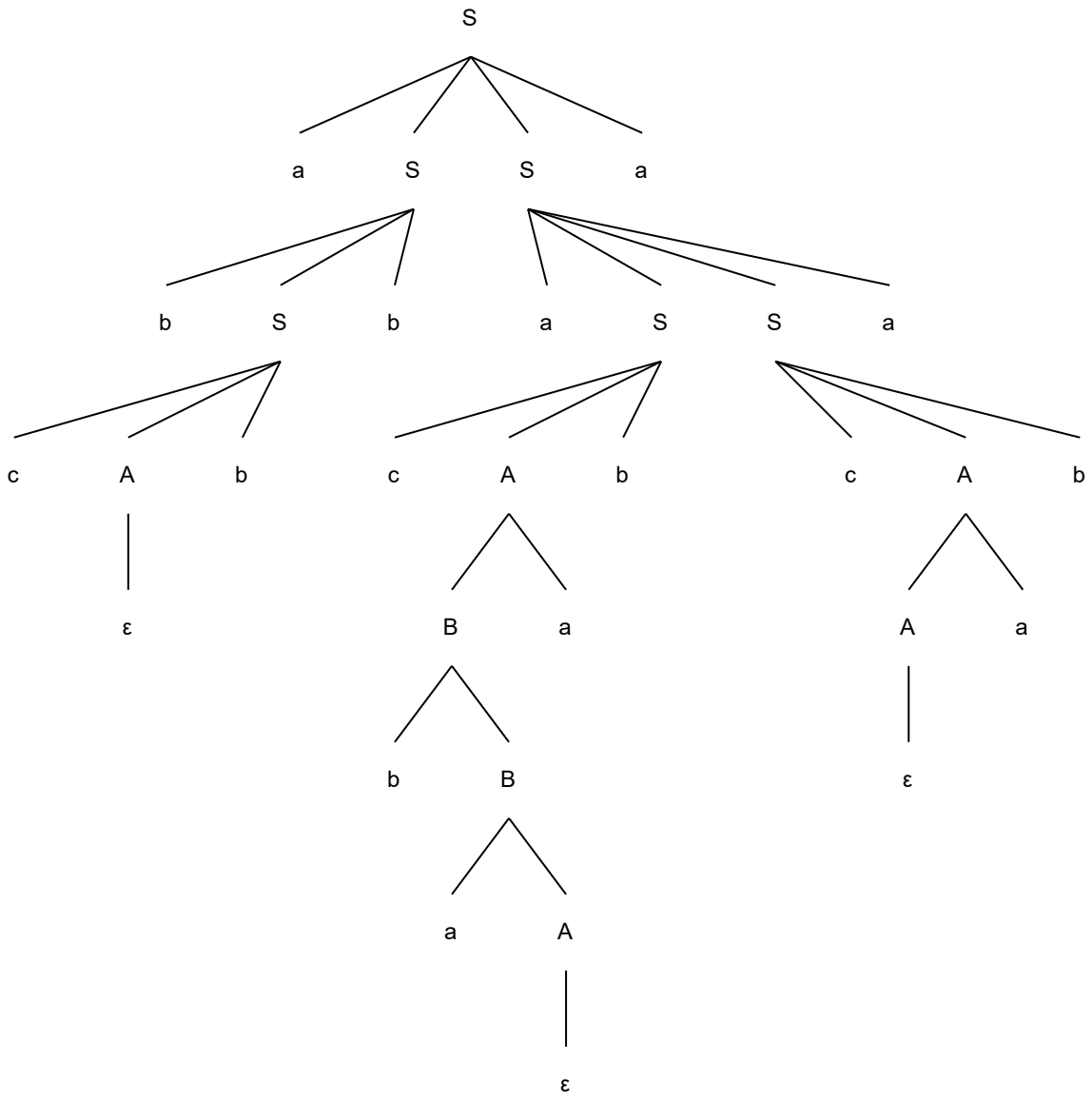
```
ЛСФ ДВ: S(aS(bS(cA()b)b)S(aS(cA(B(bB(aA()))a)b)S(cA(A()a)b)a)a)
Последовательность можно применить
```

Дерево вывода для последовательности `1, 1, 3, 4, 6, 3, 5, 7, 8, 6, 2, 3, 6`



Результат: `aacabcbaababcbba`

Дерево вывода для последовательности `1, 2, 3, 6, 1, 3, 5, 7, 8, 6, 3, 4, 6`



Результат: `abcbbacbaabcabaa`

Задание 5. Написать программу, определяющую, можно ли применить заданную последовательность правил при выводе цепочки в заданной КС-грамматике.

```
from Node import Node
from Rule import Rule


class Task3:
    def __init__(self, grammar_rules):
        self.root = Node('S')
        self.grammar_rules = grammar_rules
        self.steps = []
        self.step_index = 1

    def apply(self):
        for index, rule in enumerate(self.grammar_rules):
            print(f'{index + 1}. {rule}')
```

```
        inputs = [int(x) - 1 for x in input('Введите последовательность
шагов: ').split(',')]

        for i in inputs:
            rule = self.grammar_rules[i]
            node = self.get_node_that_fits_rule(rule)
            if node is None:
                print('Последовательность невозможно применить')
                return
            else:
                self.apply_rule_for(node, rule)
                self.step_index += 1
                self.steps.append(i)

        self.print_end_info()

        print('Последовательность можно применить')

    def get_node_that_fits_rule(self, rule):
        return self.root.find_non_terminal_node_by_value_left(rule.input)

    def print_end_info(self):
        print(f'Терминальная цепочка:
{self.root.get_plain_representation()}')
        print(f'Последовательность правил: {' '.join(map(str, [x + 1 for x
in self.steps]))}')
        print(f'ЛСФ ДВ: {self.root.get_tree_representation()}')

    @staticmethod
    def get_rule_from_input(rules, rule_index):
        return rules[rule_index], rule_index

    @staticmethod
    def apply_rule_for(node, rule):
        node.children = [Node(x) for x in rule.output]


def main():
    output = Task3([
        Rule('S', ['a', 'S', 'S', 'a']),
        Rule('S', ['b', 'S', 'b']),
        Rule('S', ['c', 'A', 'b']),
        Rule('A', ['A', 'a']),
        Rule('A', ['B', 'a']),
        Rule('A', ['']),
        Rule('B', ['b', 'B']),
        Rule('B', ['a', 'A'])
    ])
```

```
    output.apply()


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

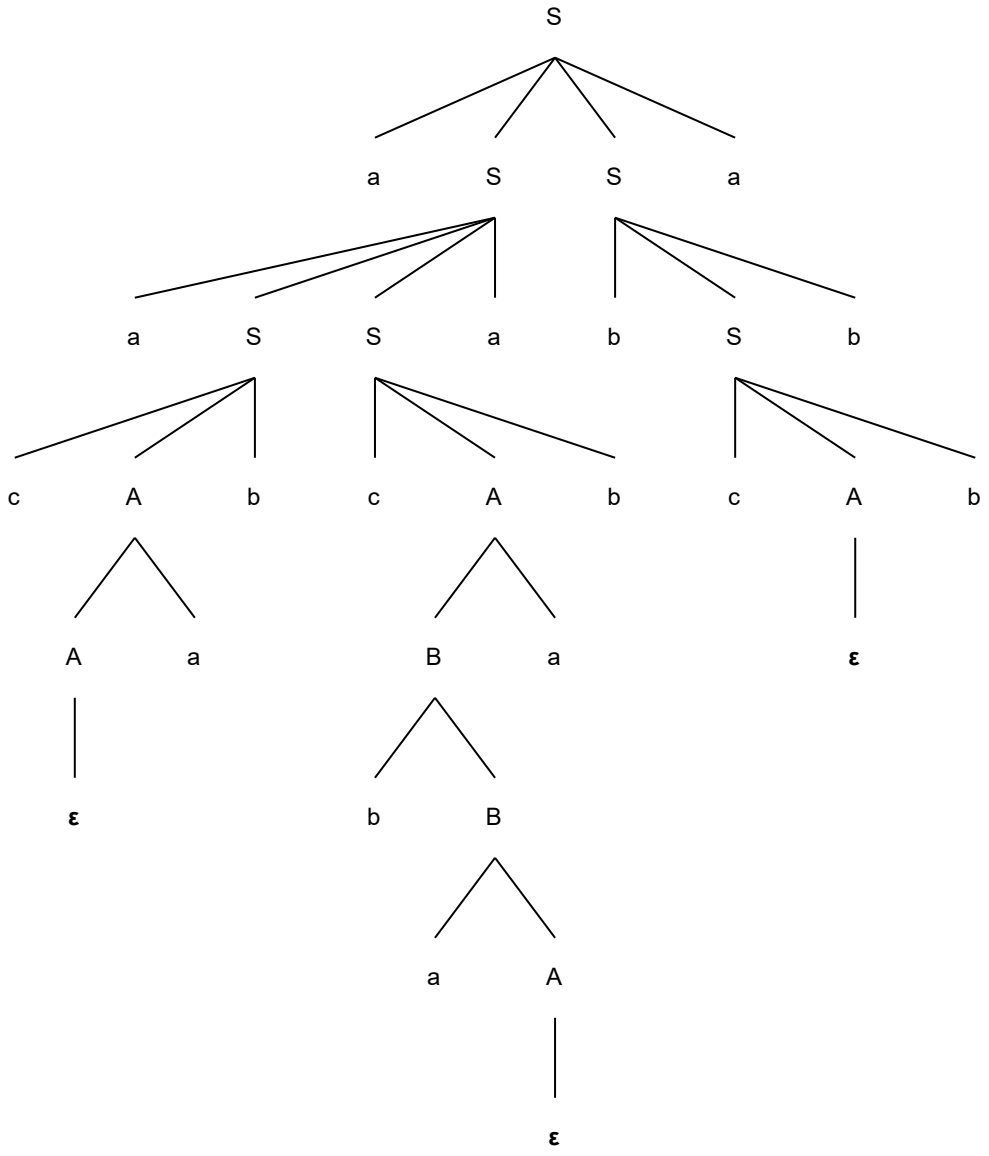
Задание 6. Для каждой последовательности правил определить, можно ли её применить при выводе терминальной цепочки в заданной КС-грамматике, и, если можно, построить дерево вывода и записать эквивалентные левый и правый вывод.

Последовательность `1, 1, 3, 4, 6, 3, 5, 7, 8, 6, 2, 3, 6`

Вывод программы

```
Введите последовательность шагов: 1, 1, 3, 4, 6, 3, 5, 7, 8, 6, 2, 3, 6
Терминальная цепочка: aacabcbaababcbba
Последовательность правил: 1 1 3 4 6 3 5 7 8 6 2 3 6
ЛСФ ДВ: S(aS(aS(cA(A()a)b)S(cA(B(bB(aA()))a)b)a)S(bS(cA()b)b)a)
Последовательность можно применить
```

Дерево вывода



Эквивалентный левый вывод

```
Терминальная цепочка: aacabcbaababcbba
Последовательность правил: 1 1 3 4 6 3 5 7 8 6 2 3 6
ЛСФ ДВ: S(aS(aS(cA(A()a)b)S(cA(B(bB(aA()))a)b)a)S(bS(cA()b)b)a)
```

Эквивалентный правый вывод

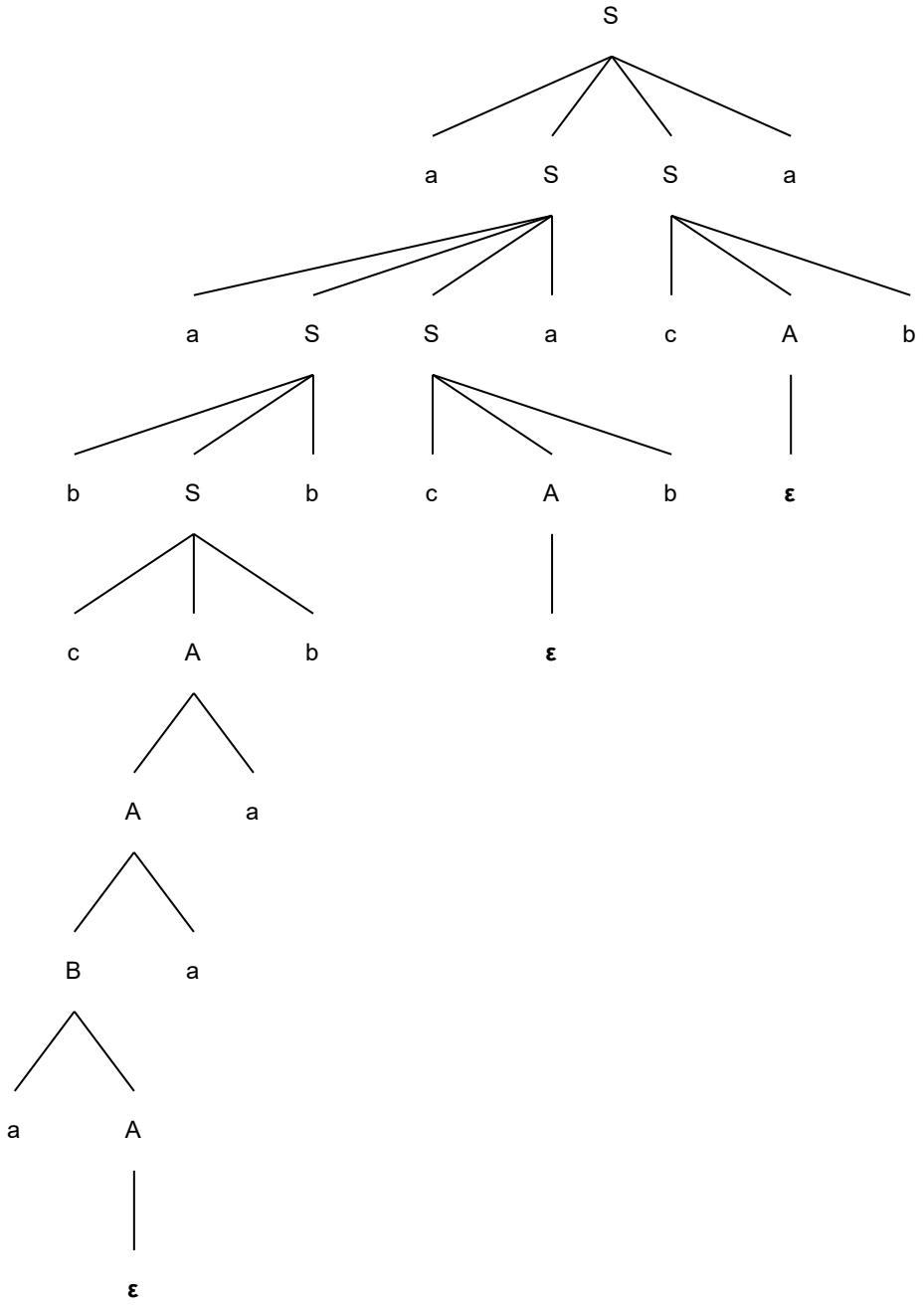
```
Терминальная цепочка: aacabcbaababcbba
Последовательность правил: 1 2 3 6 1 3 5 7 8 6 3 4 6
ЛСФ ДВ: S(aS(aS(cA(A()a)b)S(cA(B(bB(aA()))a)b)a)S(bS(cA()b)b)a)
```

Последовательность `1, 1, 2, 3, 3, 3, 4, 5, 6, 6, 7, 8, 6`

Вывод программы

```
Введите последовательность шагов: 1, 1, 2, 3, 3, 3, 4, 5, 6, 6, 7, 8, 6
Терминальная цепочка: aabcbaaabbcbacba
Последовательность правил: 1 1 2 3 3 3 4 5 6 6 7 8 6
ЛСФ ДВ: S(aS(aS(bS(cA(A(B(bB(aA()))a)a)b)b)S(cA()b)a)S(cA()b)a)
Последовательность можно применить
```

Дерево вывода



Эквивалентный левый вывод

```
Терминальная цепочка: aabcbaaabbcbacba
Последовательность правил: 1 1 2 3 4 5 7 8 6 3 6 3 6
ЛСФ ДВ: S(aS(aS(bS(cA(A(B(bB(aA()))a)a)b)b)S(cA()b)a)S(cA()b)a)
```

Эквивалентный правый вывод

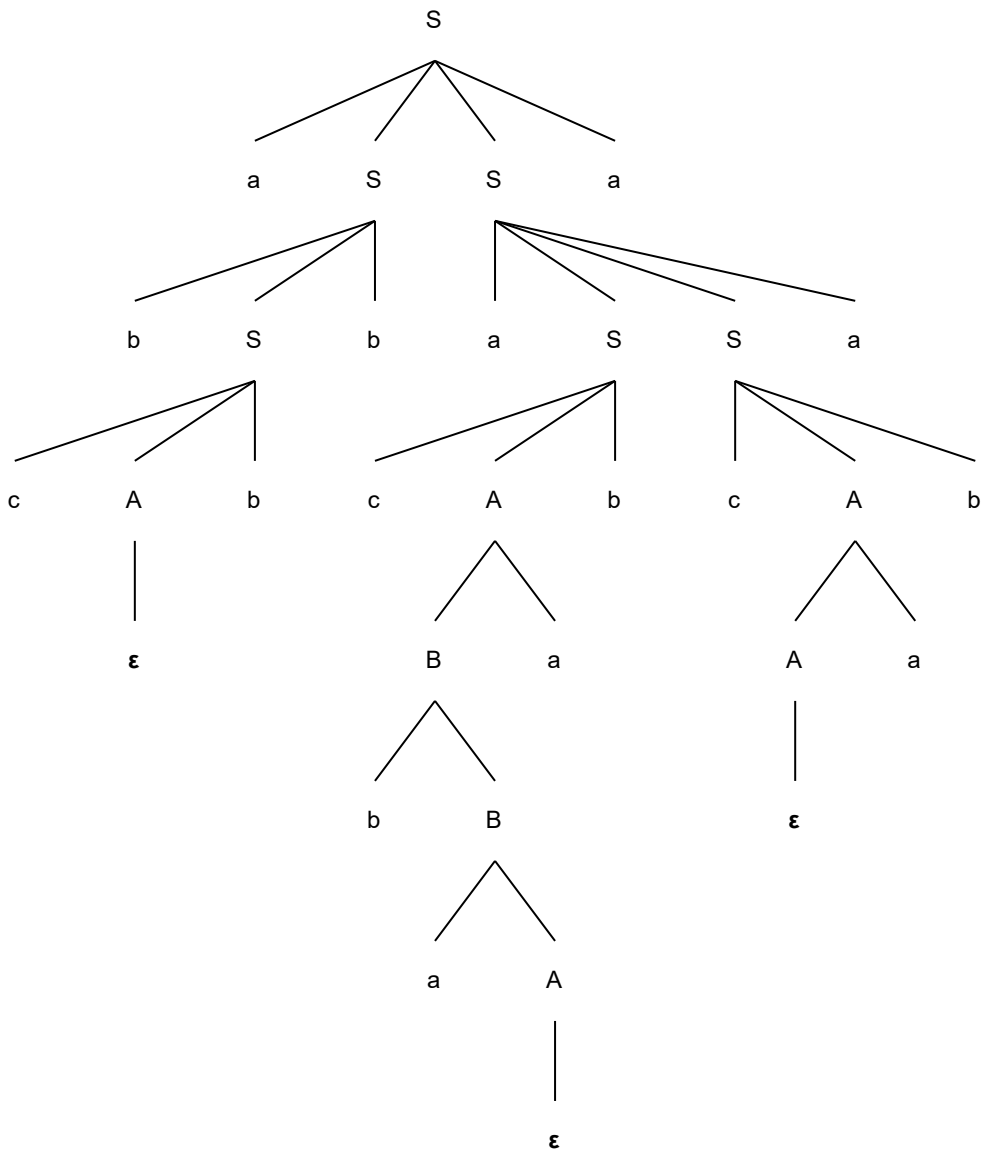
```
Терминальная цепочка: aabcbaaabbcbacba
Последовательность правил: 1 3 6 1 3 6 2 3 5 7 8 4 6
ЛСФ ДВ: S(aS(aS(bS(cA(B(bB(aA(A()a)))a)b)b)S(cA()b)a)S(cA()b)a)
```

Последовательность `1, 2, 3, 6, 1, 3, 5, 7, 8, 6, 3, 4, 6`

Вывод программы

```
Введите последовательность шагов: 1, 2, 3, 6, 1, 3, 5, 7, 8, 6, 3, 4, 6
Терминальная цепочка: abcbbacbaabcabaa
Последовательность правил: 1 2 3 6 1 3 5 7 8 6 3 4 6
ЛСФ ДВ: S(aS(bS(cA()b)b)S(aS(cA(B(bB(aA()))a)b)S(cA(A()a)b)a)a)
Последовательность можно применить
```

Дерево вывода



Эквивалентный левый вывод

```
Терминальная цепочка: abcbbacbaabcabaa
Последовательность правил: 1 2 3 6 1 3 5 7 8 6 3 4 6
ЛСФ ДВ: S(aS(bS(cA()b)b)S(aS(cA(B(bB(aA()))a)b)S(cA(A()a)b)a)a)
```

Эквивалентный правый вывод

```
Терминальная цепочка: abcbbacbaabcabaa
Последовательность правил: 1 1 3 4 6 3 5 7 8 6 2 3 6
ЛСФ ДВ: S(aS(bS(cA()b)b)S(aS(cA(B(bB(aA()))a)b)S(cA(A()a)b)a)a)
```

Вывод: изучили основные понятия теории формальных языков и грамматик.