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: ст. группы ПВ-223
Дмитриев Андрей Александрович

Проверил:
Рязанов Юрий Дмитриевич

Белгород 2024 г.

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

Вариант 2:

1. КС-грамматика

1. S→Aba	5. A→BaB
2. S→bS	6. A→ε
3. S→cAbA	7. B→b
4. A→AaS	8. B→aA

2. Последовательности правил вывода

1. 1, 4, 7, 5, 3, 7, 7, 6, 6, 4, 6, 2, 3, 6, 6
2. 1, 7, 4, 3, 6, 6, 5, 7, 4, 2, 3, 6, 6, 6, 7
3. 1, 4, 5, 7, 7, 4, 6, 2, 1, 6, 7, 3, 6, 6, 7
4. 1, 4, 3, 5, 7, 6, 6, 7, 7, 4, 2, 6, 6, 7, 6

Для выполнения заданий используется единственный класс:

```
class GMode:
    LEFT_OUTPUT = 1
    RIGHT_OUTPUT = 2
    FREE_OUTPUT = 3


class G:
    def __init__(self, n, a, p, s):
        self.N: list = n
        self.A: list = a
        self.P: dict[str, list[tuple[str, int]]] = p
        self.S: str = s
        self.seq: str = s
        self.rules_history: str = ""
        self.tree: str = s

    def clear(self):
        self.seq = self.S
        self.rules_history = ""
        self.tree = self.S

    def to_str_rule(self, non_term, rule):
        return str(rule[1]) + ". " + non_term + "->" + rule[0]

    def show_valid_rule(self, get_non_term):
        res: set[str] = set()
        for undef_term in self.seq:
            non_term = get_non_term()
            if undef_term.isupper() and undef_term in non_term:
                rights = self.P[undef_term]
                for right in rights:
                    res.add(self.to_str_rule(undef_term, right))

        _res = list(res)
        _res.sort()
        for x in _res:
            print(x)

    def get_tree(self):
        res = self.tree
        for i in range(len(self.N)):
            res = res.replace(str(i), self.N[i])

        return res

    def check_on_term(self):
        for x in self.seq:
            if x.isupper():
                return False

        return True

    def define_rule(self, rule_num):
        # rule: tuple[str, str] = tuple[str, str]()
        rule = None
        for rules in self.P.items():
            for pair in rules[1]:
                if pair[1] == rule_num:
                    rule = (rules[0], pair[0])
                    break

            if rule: break

        if not rule: raise Exception("rule not found by num")

        return rule
```

```
    def get_left_non_term(self):
        for x in self.seq:
            if x.isupper():
                return x

        return None

    def get_right_non_term(self):
        temp = list(self.seq)
        temp.reverse()
        for x in temp:
            if x.isupper():
                return x

        return None

    def get_free_non_term(self):
        res = set()
        for x in self.seq:
            if x.isupper():
                res.add(x)

        return None

    def use_rule(self, rule_num, get_non_term):
        rule = self.define_rule(rule_num)

        non_term = get_non_term()
        if not rule[0] in non_term or not non_term:
            raise Exception("non-term not found")

        temp = (
            self.seq
            .replace(rule[0], rule[1], 1)
            .replace("e", "")
        )
        if temp != self.seq:
            self.seq = temp
            self.tree = (
                self.tree
                .replace(rule[0], "%s(%s)" % (self.N.index(rule[0]), rule[1]), 1)
            )
        else:
            raise Exception("rule not use")

        self.rules_history += str(rule_num) + ' '

    def run_process(self, mode: int, seq_of_command: list[int] = ()):
        i = 0
        command = -1
        while command != 0:
            if self.check_on_term():
                print("Терминальная цепочка:", self.seq)
                print("Последовательность правил:", self.rules_history)
                print("ЛСФ ДВ:", self.get_tree())
                break

            print()
            print("Шаг %s" % str(i + 1))
            print("Промежуточная цепочка: %s" % self.seq)
            print("Можно применить правила:")
            if mode == GMode.LEFT_OUTPUT:
                self.show_valid_rule(self.get_left_non_term)
            elif mode == GMode.RIGHT_OUTPUT:
                self.show_valid_rule(self.get_right_non_term)
            elif mode == GMode.RIGHT_OUTPUT:
```

```
                self.show_valid_rule(self.get_free_non_term)
            else:
                raise Exception("GMode not change")

            if i >= len(seq_of_command):
                command = int(input())
            else:
                command = seq_of_command[i]

            if mode == GMode.LEFT_OUTPUT:
                self.use_rule(command, self.get_left_non_term)
            elif mode == GMode.RIGHT_OUTPUT:
                self.use_rule(command, self.get_right_non_term)
            elif mode == GMode.RIGHT_OUTPUT:
                self.use_rule(command, self.get_free_non_term)
            else:
                raise Exception("GMode not change")

            i += 1

    def check_on_term_by_rules(self, rule_nums, get_non_term):
        curr_rule_num = 0
        try:
            for rule_num in rule_nums:
                curr_rule_num = rule_num
                self.use_rule(rule_num, get_non_term)
        except:
            self.rules_history += '~' + str(curr_rule_num)
            return False

        return self.check_on_term()

    def run_check_process(self, mode: int, seq_of_command: list[int] = ()):
        print("КС-грамматика:")
        for non_term in self.P.items():
            for pair in non_term[1]:
                print(self.to_str_rule(non_term[0], pair))

        if len(seq_of_command) == 0:
            seq_of_command = list(map(int, input().split()))

        if mode == GMode.LEFT_OUTPUT:
            res = self.check_on_term_by_rules(seq_of_command, self.get_left_non_term)
        elif mode == GMode.RIGHT_OUTPUT:
            res = self.check_on_term_by_rules(seq_of_command, self.get_right_non_term)
        elif mode == GMode.RIGHT_OUTPUT:
            res = self.check_on_term_by_rules(seq_of_command, self.get_free_non_term)
        else:
            raise Exception("GMode not change")

        print("Введенная последовательность правил:", *seq_of_command)
        print("Примененная последовательность правил:", self.rules_history)
        print("Результат:", "Да" if res else "Нет")
```

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

```
if __name__ == "__main__":
    g = G(
        n=['S', 'A', 'B'],
        a=['a', 'b'],
        p={
            'S': [("Aba", 1), ("bS", 2), ("cAbA", 3)],
            'A': [("AaS", 4), ("BaB", 5), ("e", 6)],
            'B': [("b", 7), ("aA", 8)]
        },
```

```
        s='S'
    )

    g.run_process(GMode.LEFT_OUTPUT)
```

```
    def get_left_non_term(self):
        for x in self.seq:
            if x.isupper():
                return x

        return None
```

Результат выполнения:

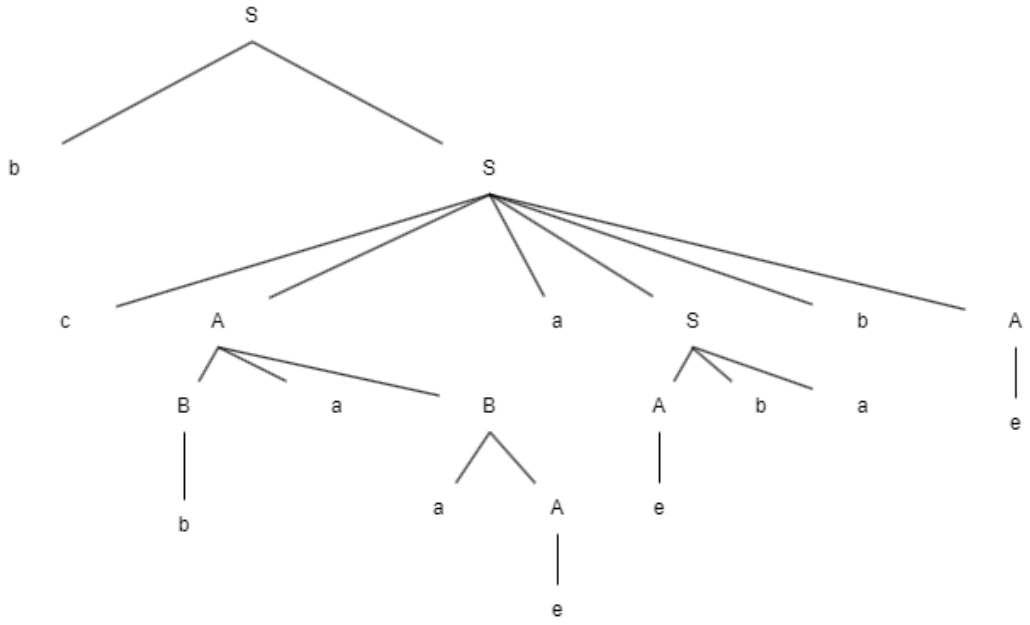
Шаг 1 Промежуточная цепочка: S Можно применить правила: 1. S->Aba 2. S->bS 3. S->cAbA 3 Шаг 2 Промежуточная цепочка: cAbA Можно применить правила: 4. A->AaS 5. A->BaB 6. A->e 5 Шаг 3 Промежуточная цепочка: cBaBbA Можно применить правила: 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA 7	Шаг 4 Промежуточная цепочка: cbaBbA Можно применить правила: 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA 8 Шаг 5 Промежуточная цепочка: cbaaAbA Можно применить правила: 4. A->AaS 5. A->BaB 6. A->e 6 Шаг 6 Промежуточная цепочка: cbaabA Можно применить правила: 4. A->AaS 5. A->BaB 6. A->e 6 Терминальная цепочка: cbaab Последовательность правил: 3 5 7 8 6 6 ЛСФ ДВ: S(cA(B(b)aB(aA(e)))bA(e))

Задание 2.

Выполнить левый (правый) вывод терминальной цепочки в заданной грамматике (см. варианты заданий п.1), построить дерево вывода. Определить, существует ли неэквивалентный вывод полученной цепочки и, если существует, представить его деревом вывода.

Терминальная цепочка: bcbaaabab
Последовательность правил: 2 3 4 5 7 8 6 1 6 6

ЛСФ ДВ: S(bS(cA(A(B(b)aB(aA(e)))aS(A(e)ba))bA(e)))



Доказательство наличия неэквивалентных выводов:
«B» одназначно можно перевести в терминалы «a» и «b». Можно рассмотреть близкие по значению правила «A->AaS» (1), «A->BaB» (2). В 1-ой цепочке следует заменить «A» на «e», во 2ой заменим «B» на «a». Далее преобразуем 2-ю цепочку, изменяя «B», и получим терминальную цепочку, в 1-ой цепочке из «S» следует последовательности имеющие несколько терминалов, что не будет эквивалентно 2-ой строчке. Следует сделать вывод, что **вывод неэквивалентным не является.**

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

```
if __name__ == "__main__":
    g = G(
        n=['S', 'A', 'B'],
        a=['a', 'b'],
        p={
            'S': [("Aba", 1), ("bS", 2), ("cAbA", 3)],
            'A': [("AaS", 4), ("BaB", 5), ("e", 6)],
            'B': [("b", 7), ("aA", 8)]
        },
        s='S'
    )

    g.run_check_process(GMode.LEFT_OUTPUT)
```

Результат выполнения:

КС-грамматика:
1. S->Aba
2. S->bS
3. S->cAbA
4. A->AaS
5. A->BaB
6. A->e

7. B->b
8. B->aA
2 3 4 5 7 8 6 1 6 6
Введенная последовательность правил: 2 3 4 5 7 8 6 1 6 6
Примененная последовательность правил: 2 3 4 5 7 8 6 1 6 6
Результат: Да

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.

```
if __name__ == "__main__":
    g = G(
        n=['S', 'A', 'B'],
        a=['a', 'b'],
        p={
            'S': [("Aba", 1), ("bS", 2), ("cAbA", 3)],
            'A': [("AaS", 4), ("BaB", 5), ("e", 6)],
            'B': [("b", 7), ("aA", 8)]
        },
        s='S'
    )

    g.run_check_process(GMode.LEFT_OUTPUT,[1,4,7,5,3,7,7,6,6,4,6,2,3,6,6])
    g.clear()
    print()
    g.run_check_process(GMode.LEFT_OUTPUT,[1,7,4,3,6,6,5,7,4,2,3,6,6,6,7])
    g.clear()
    print()
    g.run_check_process(GMode.LEFT_OUTPUT,[1,4,5,7,7,4,6,2,1,6,7,3,6,6,7])
    g.clear()
    print()
    g.run_check_process(GMode.LEFT_OUTPUT,[1,4,3,5,7,6,6,7,7,4,2,6,6,7,6])
```

Результат выполнения:

КС-грамматика: 1. S->Aba 2. S->bS 3. S->cAbA 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA Введенная последовательность правил: 1 4 7 5 3 7 7 6 6 4 6 2 3 6 6 Примененная последовательность правил: 1 4 ~7 Результат: Нет	КС-грамматика: 1. S->Aba 2. S->bS 3. S->cAbA 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA Введенная последовательность правил: 1 4 5 7 7 4 6 2 1 6 7 3 6 6 7 Примененная последовательность правил: 1 4 5 7 7 ~4 Результат: Нет
КС-грамматика: 1. S->Aba 2. S->bS	КС-грамматика: 1. S->Aba 2. S->bS

3. S->cAbA 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA Введенная последовательность правил: 1 7 4 3 6 6 5 7 4 2 3 6 6 6 7 Примененная последовательность правил: 1 ~7 Результат: Нет	3. S->cAbA 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA Введенная последовательность правил: 1 4 3 5 7 6 6 7 7 4 2 6 6 7 6 Примененная последовательность правил: 1 4 ~3 Результат: Нет

Задание 5.

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

```
if __name__ == "__main__":
    g = G(
        n=['S', 'A', 'B'],
        a=['a', 'b'],
        p={
            'S': [("Aba", 1), ("bS", 2), ("cAbA", 3)],
            'A': [("AaS", 4), ("BaB", 5), ("e", 6)],
            'B': [("b", 7), ("aA", 8)]
        },
        s='S'
    )

    g.run_check_process(GMode.FREE_OUTPUT)
```

```
    def get_free_non_term(self):
        res = set()
        for x in self.seq:
            if x.isupper():
                res.add(x)

        return None
```

Результат выполнения:

КС-грамматика: 1. S->Aba 2. S->bS 3. S->cAbA 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA 2 3 4 5 7 8 6 1 6 6 Введенная последовательность правил: 2 3 4 5 7 8 6 1 6 6 Примененная последовательность правил: 2 3 4 5 7 8 6 1 6 6 Результат: Да

Задание 6.
Для каждой последовательности правил (см. варианты заданий п.2) 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.

КС-грамматика: 1. S->Aba 2. S->bS 3. S->cAbA 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA Введенная последовательность правил: 1 4 7 5 3 7 7 6 6 4 6 2 3 6 6 Примененная последовательность правил: 1 4 ~7 Результат: Нет	КС-грамматика: 1. S->Aba 2. S->bS 3. S->cAbA 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA Введенная последовательность правил: 1 4 5 7 7 4 6 2 1 6 7 3 6 6 7 Примененная последовательность правил: 1 4 5 7 7 ~4 Результат: Нет
КС-грамматика: 1. S->Aba 2. S->bS 3. S->cAbA 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA Введенная последовательность правил: 1 7 4 3 6 6 5 7 4 2 3 6 6 6 7 Примененная последовательность правил: 1 ~7 Результат: Нет	КС-грамматика: 1. S->Aba 2. S->bS 3. S->cAbA 4. A->AaS 5. A->BaB 6. A->e 7. B->b 8. B->aA Введенная последовательность правил: 1 4 3 5 7 6 6 7 7 4 2 6 6 7 6 Примененная последовательность правил: **1 4 3 5 7 6 6 7 ~7** Результат: Нет

Последовательность правил не привели к терминальной цепочке, поэтому деревья вывода не построены.

Вывод: в ходе работы изучены основные понятия теории формальных языков и грамматик. Написана программа для взаимодействия с выводом цепочек и проверкой на терминальность.