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-222

Гоголев Владимир Владимирович

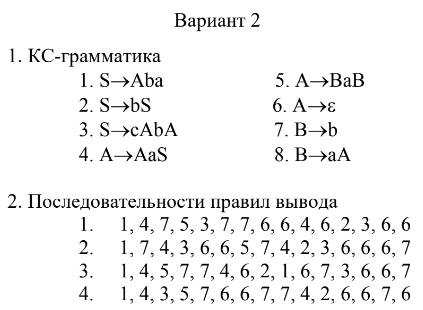
Проверил:

Рязанов Юрий Дмитриевич

Белгород 2024 г.

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

Вариант 2:



Для выполнения заданий используется единственный класс:

class Mode:  
 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 == Mode.LEFT\_OUTPUT:

self.show\_valid\_rule(self.get\_left\_non\_term)

elif mode == Mode.RIGHT\_OUTPUT:

self.show\_valid\_rule(self.get\_right\_non\_term)

elif mode == Mode.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 == Mode.LEFT\_OUTPUT:

self.use\_rule(command, self.get\_left\_non\_term)

elif mode == Mode.RIGHT\_OUTPUT:

self.use\_rule(command, self.get\_right\_non\_term)

elif mode == Mode.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.**

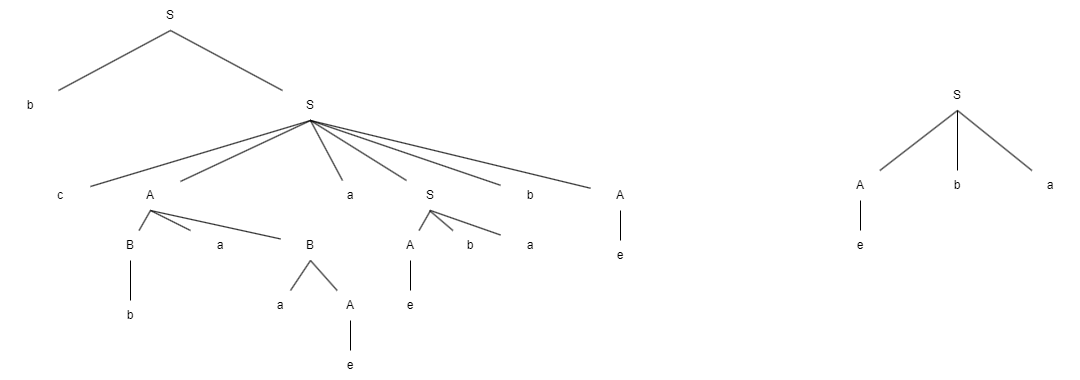
Выполнить левый (правый) вывод терминальной цепочки в заданной грамматике (см. варианты заданий п.1), построить дерево вывода. Определить, существует ли неэквивалентный вывод полученной цепочки и, если существует, представить его деревом вывода.

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)))

Дерево вывода:



Доказательство наличия неэквивалентных выводов:

Цепочка aaaaaaaaaba.

Применим последовательности правил: 1, 5, 8, 5, 8, 6, 8, 6, 8, 5, 8, 6, 8, 6 - получим дерево: S(A(B(aA(B(aA(e))aB(aA(e))))aB(aA(B(aA(e))aB(aA(e)))))ba)

Применим последовательности правил: 1, 5, 8, 5, 8, 5, 8, 6, 8, 6, 8, 6, 8, 6 - получим дерево: S(A(B(aA(B(aA(B(aA(e))aB(aA(e))))aB(aA(e))))aB(aA(e)))ba)

Выводы неэквивалентны, так как деревья неравны при одинаковой терминальной цепочке.

**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 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 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 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 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 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 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

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