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)**

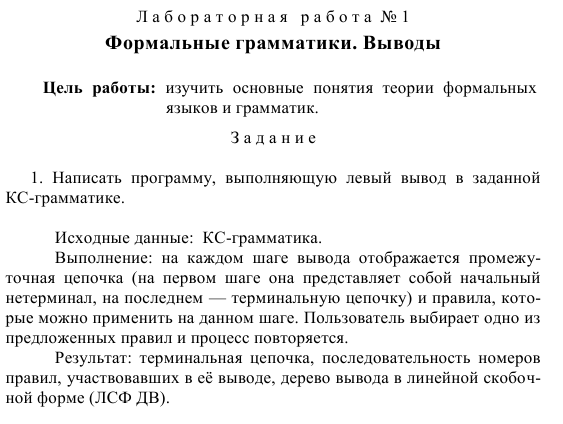
Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №1

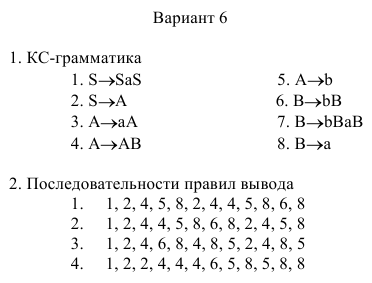
по дисциплине «Теория автоматов и формальных языков»

тема: «Формальные грамматики. Выводы.»

Белгород 2020 г.



Вариант 16



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

Во всех программах используется следующий класс

G.py

**from** random **import** choices

**class** G:

**def** \_\_init\_\_(self, n=set(), a=set(), p=dict(), s=**'S'**):

self.N : set = n

self.A : set = a

self.P : dict = p

self.S : str = s

self.P\_assotioate: list = []

**for** key **in** p:

**for** rule **in** p[key]:

self.P\_assotioate.append((rule[1], key))

self.P\_assotioate.sort()

self.P\_assotioate = [el[1] **for** el **in** self.P\_assotioate]

**def** input\_n(self):

print(**"Введите конечное множество нетерминальных символов"**, end=**": "**)

self.N.clear()

**for** n **in** input().split():

self.N.add(n)

**def** input\_a(self):

print(**"Введите конечное множество терминальных символов"**, end=**": "**)

self.A.clear()

**for** a **in** input().split():

self.A.add(a)

**def** input\_p(self):

self.P.clear()

self.P\_assotioate.clear()

print(**"Введите колличество правил"**, end=**": "**)

count\_p = int(input())

print(**"Введите %d правил:"** % count\_p)

**for** i **in** range(1, count\_p+1):

list\_rule = input().split(**"->"**)

key = list\_rule[0]

value = list\_rule[1]

**if** key **in** self.P.keys():

self.P[key].append([value, i])

**for** \_ **in** self.P[key]:

self.P\_assotioate.append(key)

**else**:

self.P.update({key: [[value, i]]}.copy())

**def** input\_s(self):

print(**"Введите начальный нетерминал"**, end=**": "**)

self.S = input()

**def** input\_all(self):

self.input\_n()

self.input\_a()

self.input\_p()

self.input\_s()

**def** \_get\_non\_term\_rules(self, non\_term):

rule\_list = []

rights = self.P[non\_term]

**for** r **in** rights:

rule\_list.append(str(r[1]) + **". "** + non\_term + **"->"** + r[0])

rule\_list.sort()

**return** rule\_list

**def** \_\_str\_\_(self):

print(**"КС-грамматика: "**)

rule\_list = []

**for** non\_term **in** self.P.keys():

rule\_list += self.\_get\_non\_term\_rules(non\_term)

rule\_list.sort()

**return "\n"**.join(rule\_list)

**def** left\_tree(self, chain, list\_rule):

offset = 0

**for** rule **in** list\_rule:

newChain = chain[:offset]

**for** ch\_i **in** range(offset, len(chain)):

newChain += chain[ch\_i]

**if** chain[ch\_i] **in** self.N:

offset = len(newChain)

newChain += **"("** + rule[0] + **")"**

newChain += chain[ch\_i+1:]

**break**

chain = newChain

**return** chain

**def** get\_non\_terminal\_for\_num\_rule(self, num\_rule) -> str:

**return** self.P\_assotioate[num\_rule - 1]

**def** check\_rules(self, rules):

chain = self.S

**for** rule **in** rules:

non\_term = self.get\_non\_terminal\_for\_num\_rule(rule)

**if** non\_term **in** chain:

current\_rule = list(filter(**lambda** el: el[1]==rule, self.P[non\_term]))[0]

chain = chain.replace(non\_term, current\_rule[0], 1)

**else**:

**return False**

**return True**

**def** check\_left\_rules(self, rules):

chain = self.S

**for** rule **in** rules:

newChain = **""**

flag = **True**

**for** ch **in** chain:

**if** (ch **in** self.N) **and** flag:

current\_rule = list(filter(**lambda** el: el[1] == rule,self.P[ch]))

flag = **False**

**if** current\_rule != []:

newChain+= current\_rule[0][0]

**else**:

**return False**

**else**:

newChain += ch

chain = newChain

**return True**

**def** left\_vivod(self):

print(**"Левый вывод\n"**)

chain = self.S

step = 1

step\_by\_step = []

**while**(**True**):

chain.replace(**"e"**, **""**)

flag = **True**

newChain = **""**

**for** ch **in** chain:

**if** (ch **in** self.N) **and** flag:

print(**"Шаг %d."** % step)

step += 1

print(**"Промежуточная цепочка: %s"** % chain)

flag = **False**

rules = self.\_get\_non\_term\_rules(ch)

print(**"Можно применить правила:"**)

print(**"\n"**.join(rules))

num = int(input())

rule = list(filter(**lambda** el: el[1]==num, self.P[ch]))[0]

print(**"Применяем правило %d"** % rule[1])

step\_by\_step.append(rule)

newChain += rule[0]

**else**:

newChain += ch

print()

**if** (flag):

**break**

chain = newChain

chain.replace(**"e"**, **""**)

print(**"Шаг %d."** % step)

print(**"Терминальная цепочка: %s"** % chain)

print(**"Последовательность правил: %s"** % **' '**.join([str(step[1]) **for** step **in** step\_by\_step]))

print(**"ЛСФ ДВ: "** + self.left\_tree(self.S, step\_by\_step).replace(**"e"**, **""**))

Текст программы

**from** G **import** G

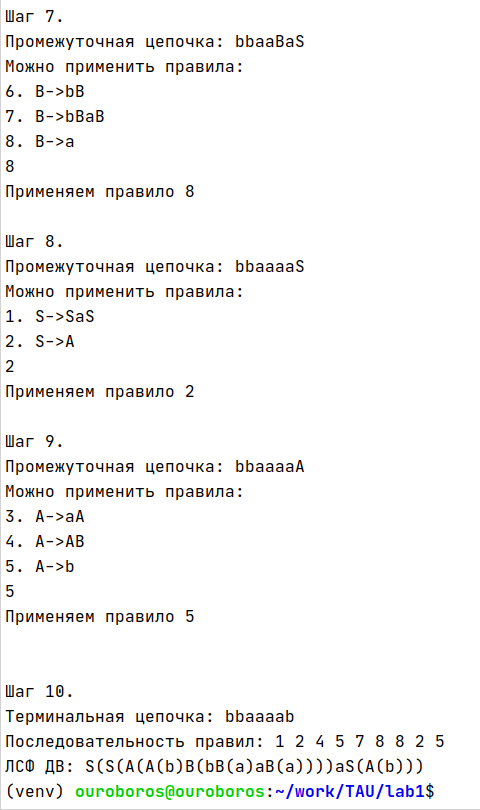
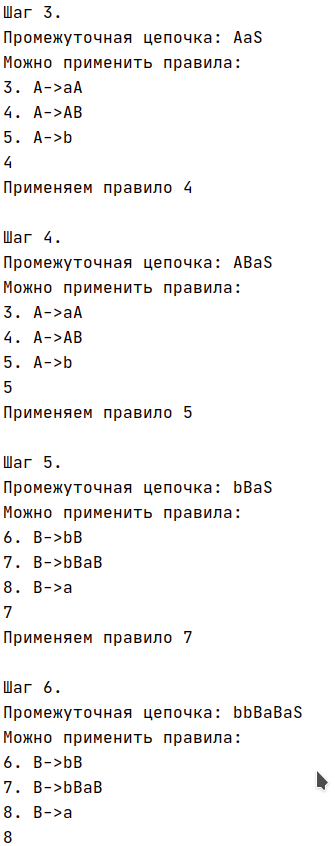
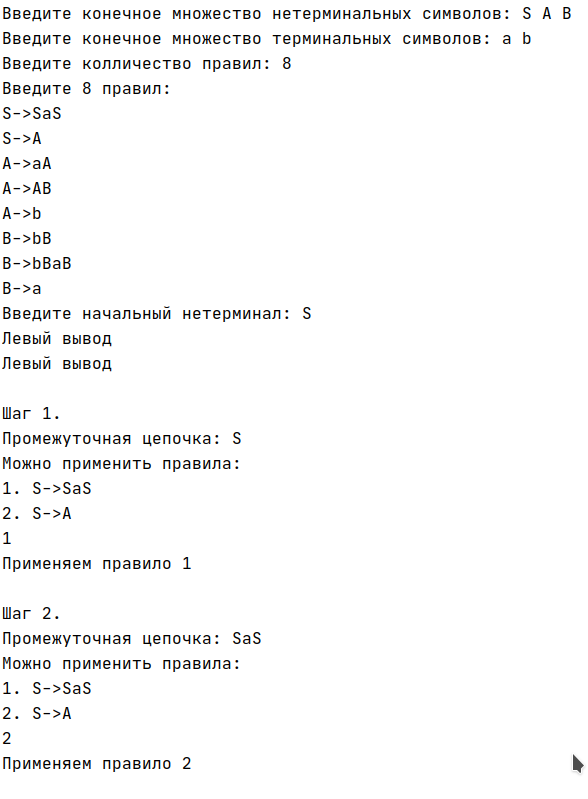
**if** \_\_name\_\_ == **"\_\_main\_\_"**:

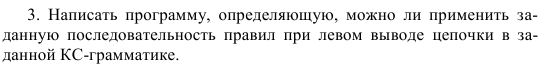
G = G()

G.input\_all()

print(**"Левый вывод"**)

G.left\_vivod()

Результат выполнения программы



Текст программы

**from** G **import** G

**if** \_\_name\_\_ == **"\_\_main\_\_"**:

G = G()

G.input\_all()

print(**"Введите номера правил: "**,end=**""**)

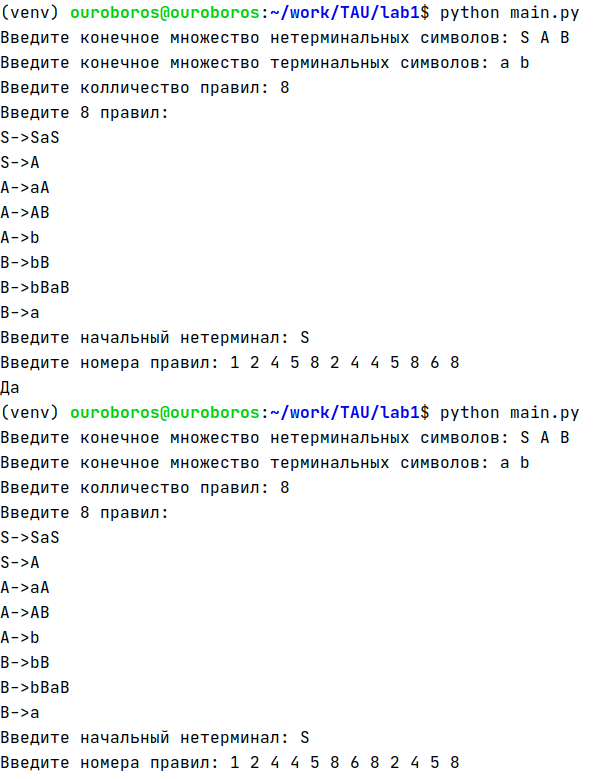
rule\_list = map(int, input().split())

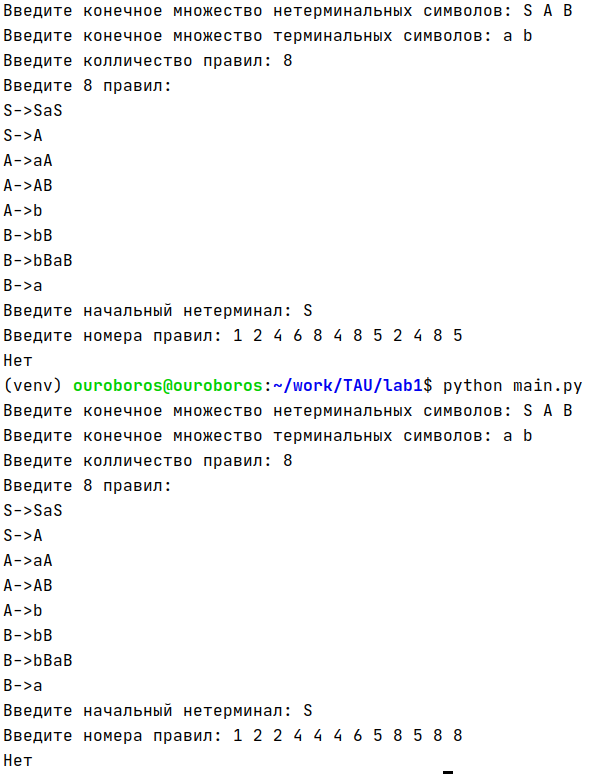
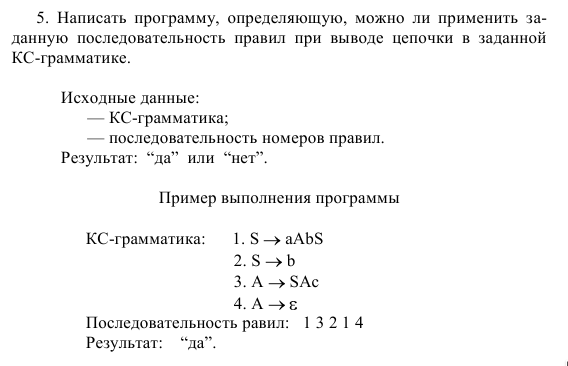
**if** G.check\_left\_rules(rule\_list):

print(**"Да"**)

**else**:

print(**"Нет"**)

Результат работы



Текст программы

**from** G **import** G

**if** \_\_name\_\_ == **"\_\_main\_\_"**:

G = G()

G.input\_all()

print(**"Введите номера правил: "**,end=**""**)

rule\_list = map(int, input().split())

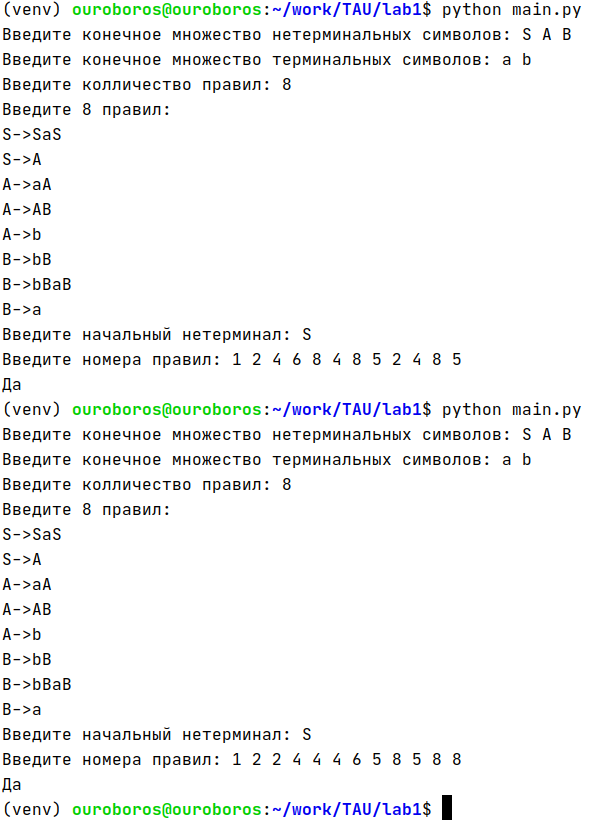
**if** G.check\_rules(rule\_list):

print(**"Да"**)

**else**:

print(**"Нет"**)

Результат работы



Вывод

Был получен опыт работы с КС грамматикой и разнообразными выводами терминальных цепочек.