МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра прикладной математики

Эквивалентные преобразования контекстно-свободных грамматик

ОТЧЕТ

По лабораторной работе

по дисциплине

Формальные языки и алгоритмы

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жевнерчук Д.В.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Зырянов Е.А.

22-ПМ-1

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание

[Задание на лабораторную работу: 3](#__RefHeading___Toc2097_1008937971)

[Теоретическая справка 4](#__RefHeading___Toc2099_100893797)

[Реализация](#__RefHeading___Toc2101_100893797) 12

[Результат](#__RefHeading___Toc2103_100893797) 19

[Список литературы](#__RefHeading___Toc2105_100893797) 24

[Приложение](#__RefHeading___Toc2419_100893797) 25

# Задание на лабораторную работу:

1. Разработать программное средство, автоматизирующее процесс эквивалентного преобразования КС-грамматик. Программное средство должно выполнять следующие функции:

1) организация ввода грамматики и проверка ее на принадлежность к классу КС-грамматик;

2) проверка существования языка КС-грамматики;

3) реализация эквивалентных преобразований грамматики, направленных на удаление:

а) бесполезных символов;

б) недостижимых символов;

в) e-правил;

г) цепных правил;

д) левой факторизации правил;

е) прямой левой рекурсии.

Цель работы:

1) Закрепить понятия «эквивалентные грамматики», «приведенная КС-грамматика».

2) Сформировать умения и навыки эквивалентных преобразований контекстно-свободных грамматик.

# Теоретическая справка

**Определение 4.1.** КС-грамматика называется приведенной, если она не имеет циклов, -правил и бесполезных символов.

Рассмотрим основные алгоритмы приведения КС-грамматик.

Перед всеми другими исследованиями и преобразованиями КС-грамматик выполняется проверка существования языка грамматики.

**Алгоритм 4.1. Проверка существования языка грамматики**

Вход: КС-грамматика .

Выход: заключение о существовании или отсутствии языка грамматики.

Определим множество нетерминалов, порождающих терминальные строки .

Шаг 1. Положить *N*0=Ø.

Шаг 2. Вычислить и

Шаг 3. Если , то положить *i*=*i*+1 и перейти к пункту 2, иначе считать .

Если , то выдать сообщение о том, что язык грамматики существует, иначе сообщить об отсутствии языка.

**Пример 4.1.** Дана грамматика , где множество правил : . Построим последовательность приближений множества *N*:

*N*0 = Ø;

*N*1 = {*A*, *B*};

*N*2 = {*S*, *A*, *B*};

*N*3 = {*S*, *A*, *B*}.

Т.к. *N*2=*N*3, то *N*= {*S*, *A*, *B*}, следовательно, язык грамматики существует, потому что начальный символ .

**Определение 4.2.** Бесполезными символами грамматики называют:

а) нетерминалы, не порождающие терминальных строк, т.е. множество символов

б) недостижимые нетерминалы, порождающие терминальные строки, т.е. множество символов

в) недостижимые терминалы, т.е. множество символов

## Алгоритм 4.2. Устранение нетерминалов, не порождающих

## терминальных строк

### Вход: КС-грамматика .

Выход: КС-грамматика , такая, что и для всех существуют выводы , где .

Шаг 1. Определить множество нетерминалов, порождающих терминальные строки, с помощью алгоритма 4.1.

Шаг 2. Вычислить , где - это множество правил, содержащих бесполезные нетерминалы .

**Пример 4.2.** Дана грамматика с правилами

Преобразуем ее в эквивалентную грамматику по алгоритму 4.2:

*N*0 = Ø;

*N*1 = {*S*, *B*, *C*};

*N*2 = {*S*, *B*, *C*}.

Т.к. *N*1= *N*2, то *N* = {*S*, *B*, *C*}. После удаления бесполезных нетерминалов и правил вывода, получим грамматику с правилами

#### Алгоритм 4.3. Устранение недостижимых символов

### Вход: КС-грамматика .

Выход: КС-грамматика , такая, что и для всех существует вывод , где .

##### Определим множество достижимых символов *Z* грамматики *G*, т.е. множество

Шаг 1. Положить

Шаг 2. Вычислить очередное приближение следующим образом:

Шаг 3. Если то положить *i*:=*i*+1 и перейти к шагу 2, иначе считать .

Шаг 4. Вычислить , где - это множество правил, содержащих недостижимые символы .

**Пример 4.3.** Данаграмматика с правилами

Преобразуем ее в эквивалентную грамматику по алгоритму 4.3:

*W*0 = {*S*};

*W*1 = {*S*, *a*, *b*};

*W*2 = {*S*, *a*, *b*}.

Т.к. *W*1=*W*2, то *W*={*S*, *a*, *b*}. Множество недостижимых символов Тогда после удаления недостижимых символов, получим грамматику с правилом

###### **Алгоритм 4.4. Устранение -правил**

### Вход: КС-грамматика .

Выход: Эквивалентная КС-грамматика без -правил для всех нетерминальных символов, кроме начального, который не должен встречаться в правых частях правил грамматики.

Шаг 1. В исходной грамматике найти -порождающие нетерминальные символы , такие, что .

* 1. Положить .
  2. Вычислить .
  3. Если , то положить *i*:=*i*+1 и перейти к пункту 1.2, иначе считать .

Шаг 2. Из множества правил исходной грамматики перенести во множество все правила, за исключением -правил, т.е. для всех

Шаг 3. Пополнить множество правилами, которые получаются из каждого правила этого множества путем исключения всевозможных комбинаций -порождающих нетерминалов в правой части. Полученные при этом -правила во множество не включать.

Шаг 4. Если , то , где ; иначе

**Пример 4.4.** Дана грамматика с правилами . Преобразуем ее в эквивалентную грамматику по алгоритму 4.4.

Шаг 1. *N*0 = {*A*, *B*};

*N*1 = {*S*, *A*, *B*};

*N*2 = {*S*, *A*, *B*}.

Т.к. *N*1 = *N*2, то искомое множество построено и *N* = {*S*, *A*, *B*}.

Шаг 2, 3. Множество 1) ; 2) ; 3)

Шаг 4. Т.к. , то введем новый нетерминал *С* и пополним множество правилом вида . Результирующая грамматика будет иметь вид: с правилами .

#### Алгоритм 4.5. Устранение цепных правил

### Вход: КС-грамматика .

Выход: Эквивалентная КС-грамматика без цепных правил, т.е. правил вида , где .

Шаг 1. Для каждого нетерминала вычислить множество выводимых из него нетерминалов, т.е. множество где

* 1. Положить
  2. Вычислить
  3. Если , то положить *i*:=*i*+1 и перейти к пункту 1.2, иначе считать

Шаг 2. Построить множество так: если не является цепным правилом , то включить в правило для каждого , такого, что .

**Пример 4.5.** Грамматика с правилами . Преобразуем ее в эквивалентную грамматику по алгоритму 4.5.

Шаг 1.

Т.к. , то

Т.к. , то

Т.к. , то

Шаг 2. Преобразовав правила вывода грамматики, получим грамматику с правилами

.

###### **Алгоритм 4.6. Устранение левой факторизации правил**

### Вход: КС-грамматика .

Выход: Эквивалентная КС-грамматика без одинаковых префиксов в правых частях правил, определяющих нетерминалы.

Шаг 1. Записать все правила для нетерминала , имеющие одинаковые префиксы , в виде одного правила с альтернативами:

Шаг 2. Вынести за скобки влево префикс в каждой строке-альтернативе:

Шаг 3. Обозначить новым нетерминалом выражение, оставшееся в скобках:

Шаг 4. Пополнить множество нетерминалов новым нетерминалом и заменить правила, подвергшиеся факторизации, новыми правилами для и .

Шаг 5. Повторить шаги 1-4 для всех нетерминалов грамматики, для которых это возможно и необходимо.

**Пример 4.6.** Дана грамматика с правилами . Преобразуем ее в эквивалентную грамматику по алгоритму 4.6:

Шаг 1. .

Шаг 2. .

Шаг 3,4. Пополнив множество нетерминалов новым нетерминалом *С* и заменив правила, подвергшиеся факторизации, получим грамматику с правилами

# Алгоритм 4.7. Устранение прямой левой рекурсии

### Вход: КС-грамматика .

Выход: Эквивалентная КС-грамматика без прямой левой рекурсии, т.е. без правил вида

Шаг 1. Вывести из грамматики все правила для рекурсивного нетерминала :

Шаг 2. Внести новый нетерминал так, чтобы он описывал любой «хвост» строки, порождаемой рекурсивным нетерминалом :

Шаг 3. Заменить в рекурсивном правиле для правую часть, используя новый нетерминал и все нерекурсивные правила для так, чтобы генерируемый язык не изменился:

Шаг 4. Пополнить множество нетерминалов грамматики новым нетерминалом . Пополнить множество правил грамматики правилами, полученными на шаге 3.

Шаг 5. Повторить действия шагов 1-4 для всех рекурсивных нетерминалов грамматики, после чего полученные множества нетерминалов и правил принять в качестве и

**Пример 4.7.** Данаграмматика с правилами . После устранения прямой левой рекурсии получим эквивалентную грамматику с правилами

# Реализация

G=({S, R, T, X, Y}, {a, b, p, g, y}, P, S)

P:

1) S → R | T;

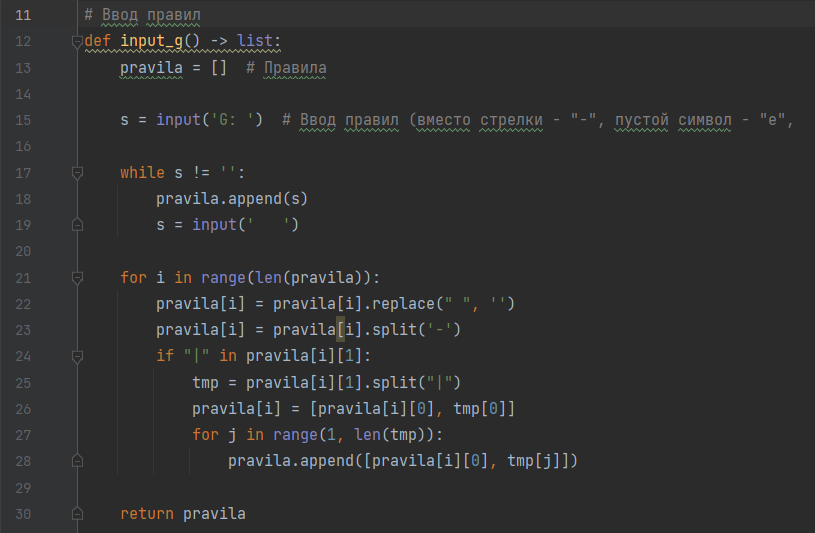
2) R → pX | paR | paT | e;

3) T → Tg | g;

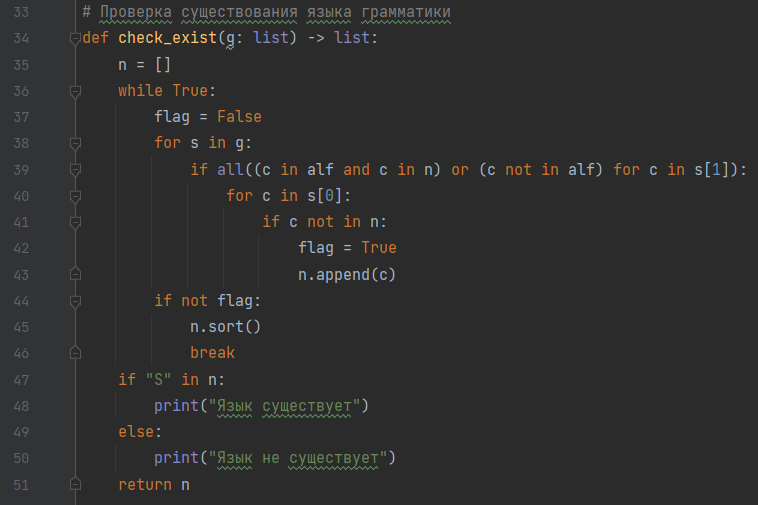
4) X → aXb

5) Y → aYa | y;

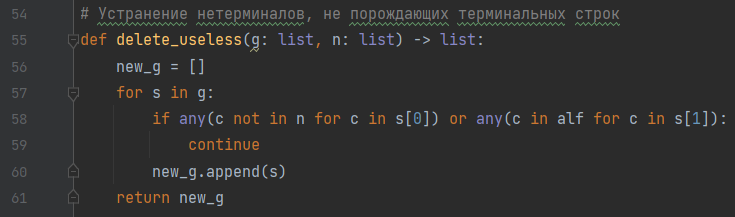
На рисунке 1 представлен фрагмент кода для ввода правил:



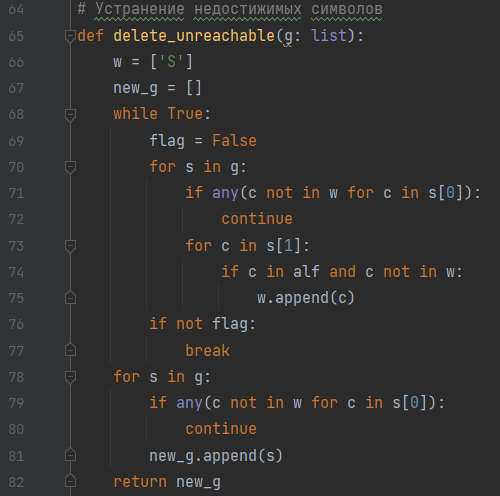
На рисунке 2 представлен фрагмент кода функции , которая проверяет существование языка грамматики:



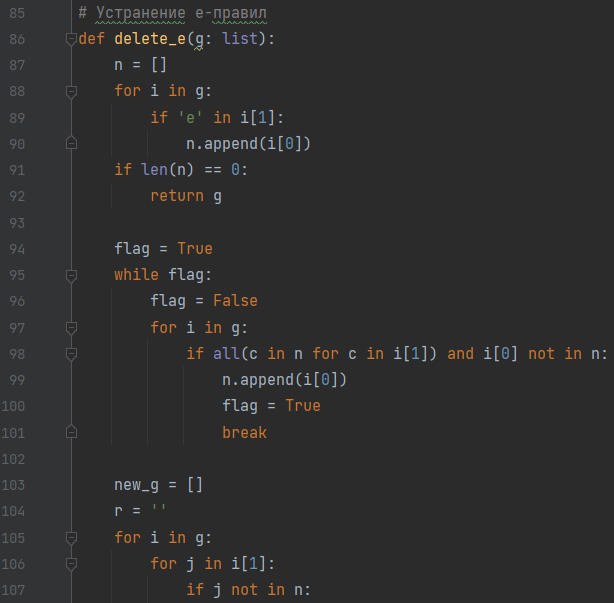
На рисунке 3 представлен фрагмент кода функции, которая устраняет бесполезные символы:

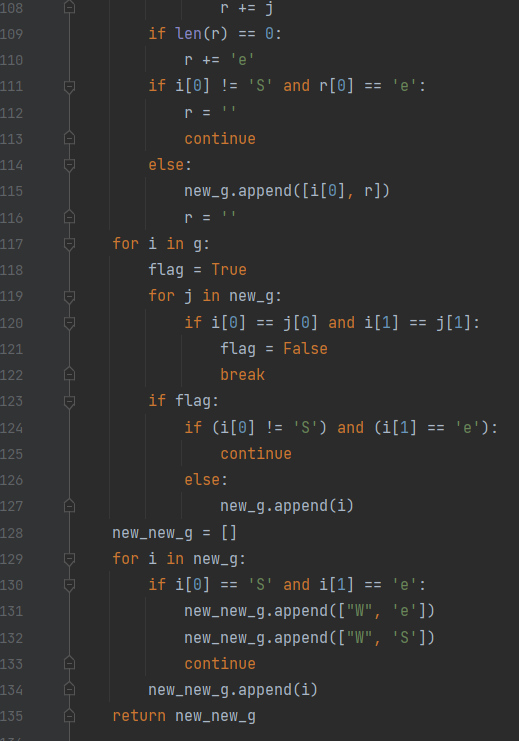


На рисунке 4 представлен фрагмент кода функции, которая устраняет недостижимые символы:

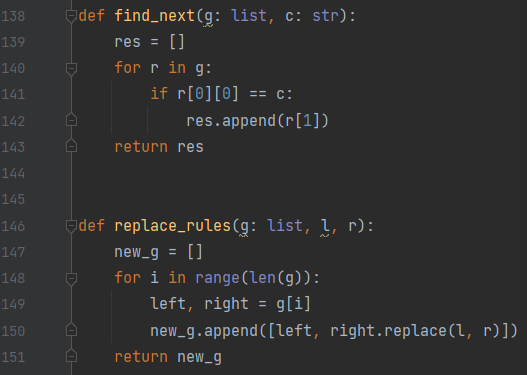


На рисунке 5 и 6 представлен фрагмент кода функции, которая устраняет е-правила:

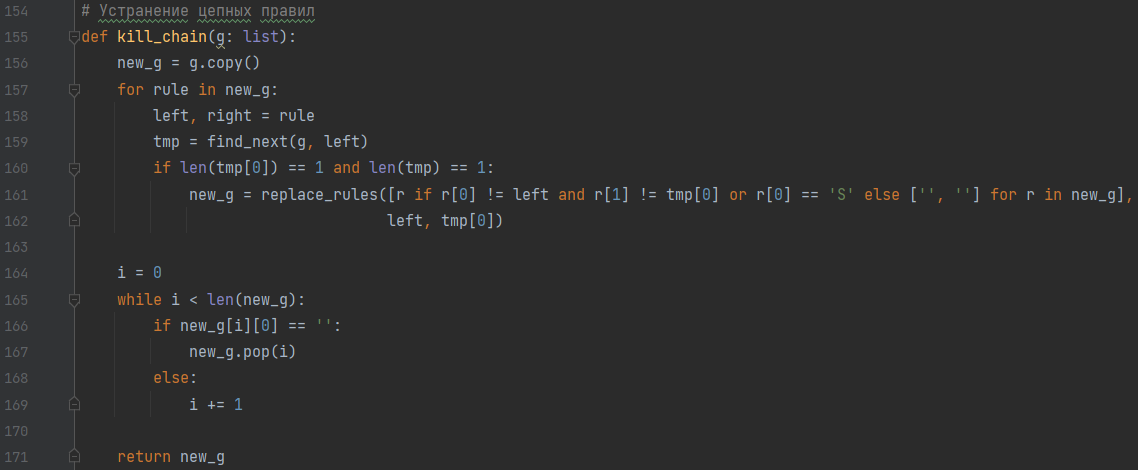




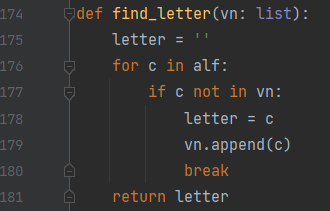
На рисунке 7 представлен фрагмент кода вспомогательных функций, которые:



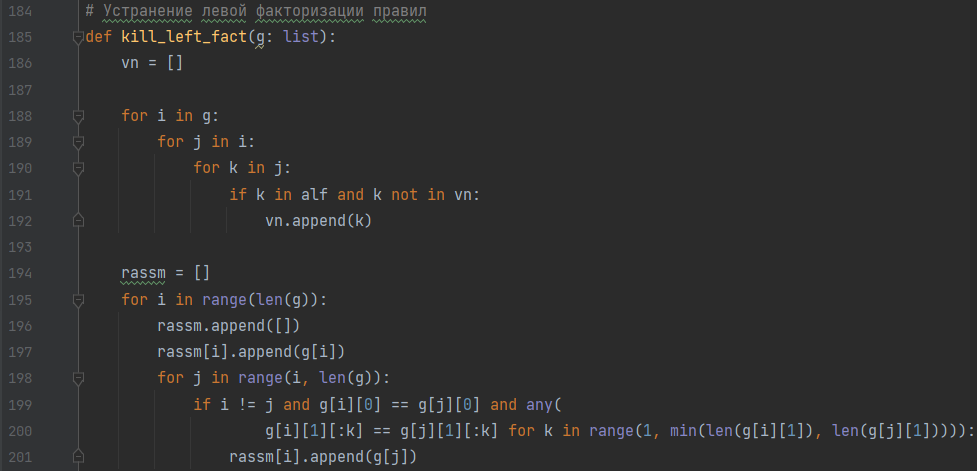
На рисунке 8 представлен фрагмент кода функции, которая устраняет цепные правила:

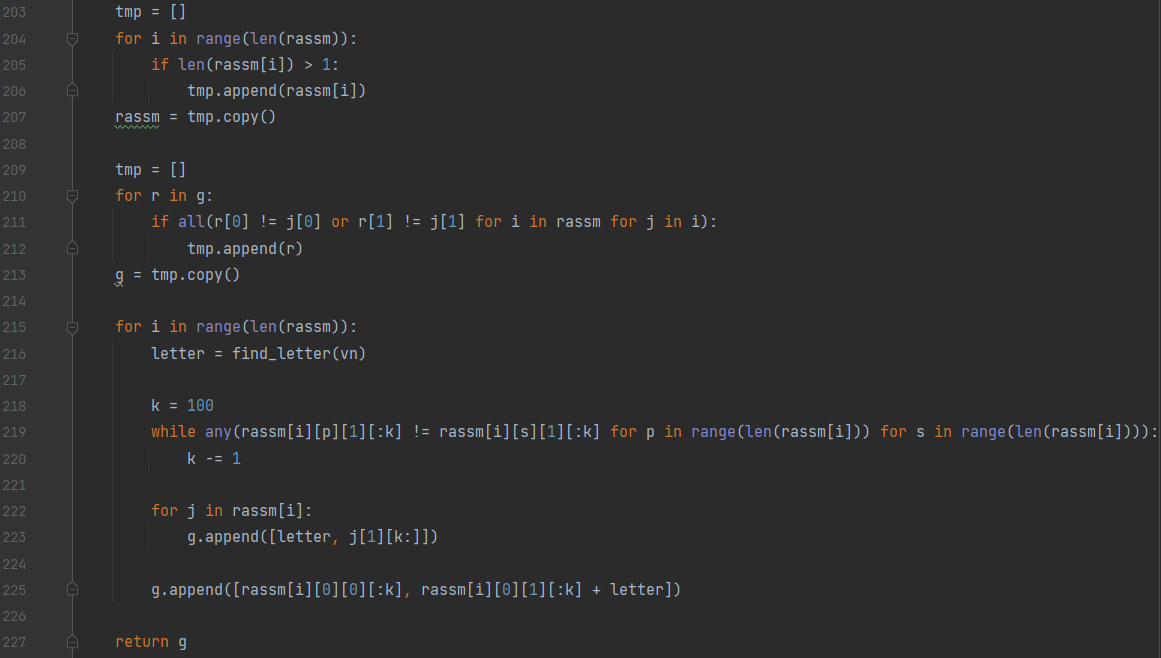


На рисунке 9 представлен фрагмент кода вспомогательной функции, которая :

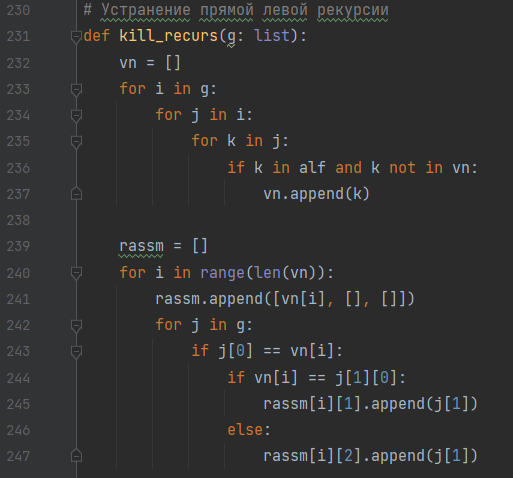


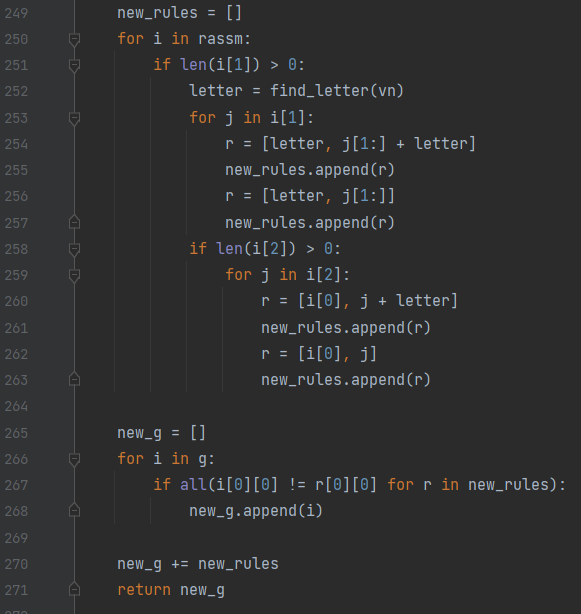
На рисунке 10 и 11 представлен фрагмент кода функции, которая устраняет левую факторизацию правил:





На рисунке 12 представлен фрагмент кода функции, которая устраняет прямую левую рекурсию:

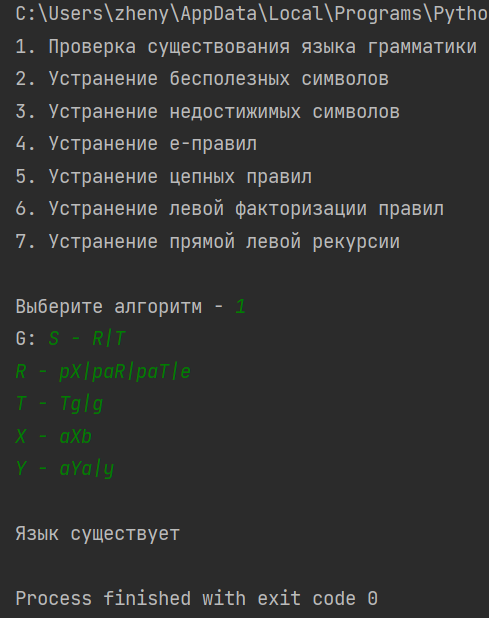




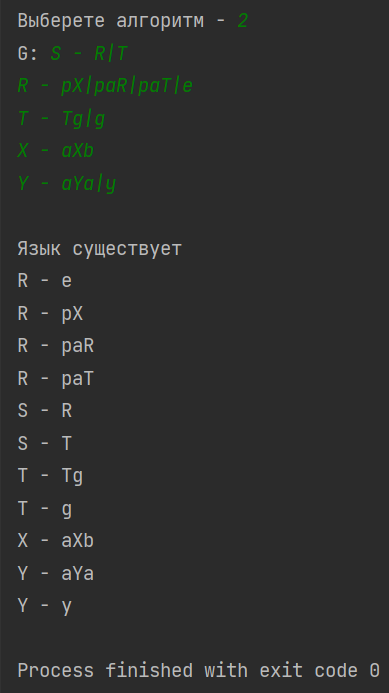
# Результат

Проверка работы программы была выполнена на примере исходной грамматики.

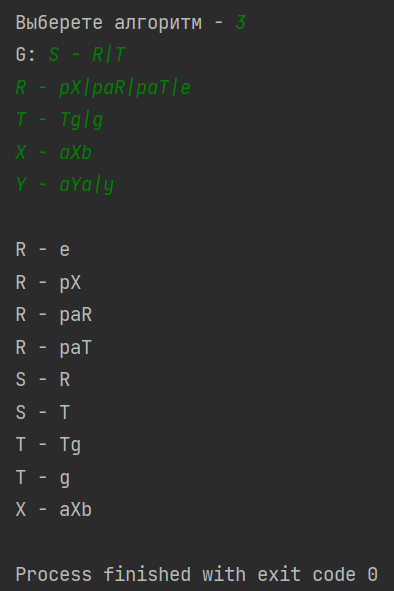
1) Алгоритм проверки существования языка грамматики:



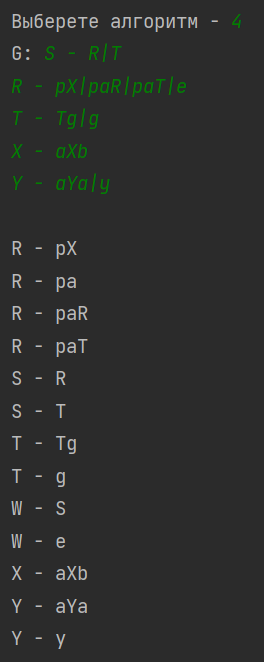
2) Алгоритм устранения бесполезных символов:



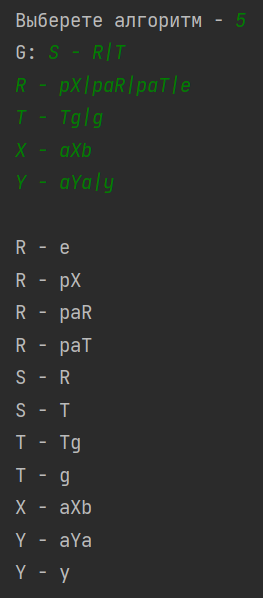
3) Алгоритм устранения недостижимых символов:



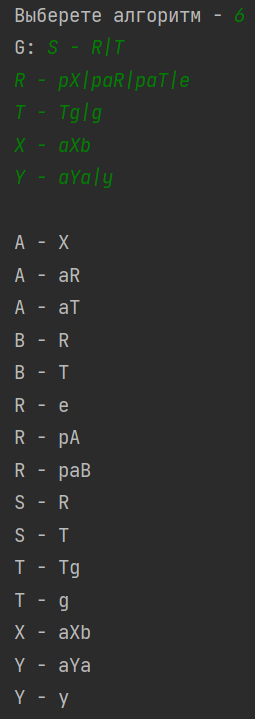
4) Алгоритм устранения е-правил:



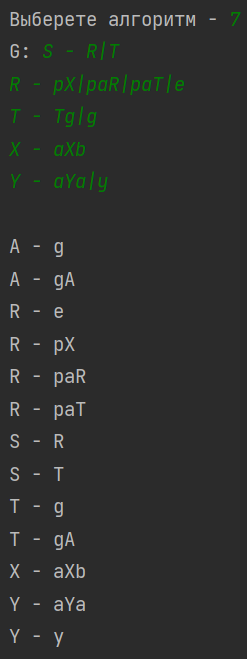
5) Алгоритм устранения цепных правил:



6) Алгоритм устранения левой факторизации правил:



7) Алгоритм устранения прямой левой рекурсии:



Вывод: Закрепили понятия «эквивалентные грамматики», «приведенная КС-грамматика». Сформировали умения и навыки эквивалентных преобразований контекстно-свободных грамматик.

# Список литературы

1. Пентус А. Е., Пентус М. Р. Теория формальных языков. Учебное пособие. - 2013.С.80.

2. Волкова И. А., Вылиток А. А., Руденко Т. В. Формальные грамматики и языки. Элементы теории трансляции. – 2009. С.115.

# Приложение

alf = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
  
  
def print\_g(g: list):  
 g.sort()  
 for i in g:  
 print(i[0], '-', i[1])  
  
  
def is\_first(g: list) -> bool:  
 return all(len(g[i][0]) <= len(g[i][1]) for i in range(len(g))) #Если левая часть длиннее, то это не первый тип  
  
  
def is\_two(g: list) -> bool:  
 return all([all([j in alf for j in i[0]]) for i in g]) #Если в левой части есть маленькая буква, то это не второй тип  
  
  
def input\_g() -> list:  
 pravila = [] # Правила  
  
 s = input('G: ') # Ввод правил (вместо стрелки - "-", пустой символ - "e",  
  
 while s != '':  
 pravila.append(s)  
 s = input(' ')  
  
 for i in range(len(pravila)):  
 pravila[i] = pravila[i].replace(" ", '')  
 pravila[i] = pravila[i].split('-')  
 if "|" in pravila[i][1]:  
 tmp = pravila[i][1].split("|")  
 pravila[i] = [pravila[i][0], tmp[0]]  
 for j in range(1, len(tmp)):  
 pravila.append([pravila[i][0], tmp[j]])  
  
 return pravila  
  
  
def check\_exist(g: list) -> list:  
 n = []  
 for i in range(len(g)):  
 for s in g:  
 if any((c in alf and c in n) or (c not in alf) for c in s[1]):  
 if s[0][0] not in n:  
 n.append(s[0][0])  
 if "S" in n:  
 print("Язык существует")  
 else:  
 print("Язык не существует")  
 return n  
  
  
def delete\_useless(g: list, n: list) -> list:  
 new\_g = []  
 for s in g:  
 if s[0][0] in n:  
 new\_g.append(s)  
 return new\_g  
  
  
def delete\_unreachable(g: list) -> list:  
 w = ['S']  
 new\_g = []  
 while True:  
 flag = False  
 for s in g:  
 if any(c not in w for c in s[0]):  
 continue  
 for c in s[1]:  
 if c in alf and c not in w:  
 w.append(c)  
 if not flag:  
 break  
 for s in g:  
 if any(c not in w for c in s[0]):  
 continue  
 new\_g.append(s)  
 return new\_g  
  
  
def delete\_e(g: list) -> list:  
 n = []  
 for i in g:  
 if 'e' in i[1]:  
 n.append(i[0])  
 if len(n) == 0:  
 return g  
  
 flag = True  
 while flag:  
 flag = False  
 for i in g:  
 if all(c in n for c in i[1]) and i[0] not in n:  
 n.append(i[0])  
 flag = True  
 break  
  
 new\_g = []  
 r = ''  
 for i in g:  
 for j in i[1]:  
 if j not in n:  
 r += j  
 if len(r) == 0:  
 r += 'e'  
 if i[0] != 'S' and r[0] == 'e':  
 r = ''  
 continue  
 else:  
 new\_g.append([i[0], r])  
 r = ''  
 for i in g:  
 flag = True  
 for j in new\_g:  
 if i[0] == j[0] and i[1] == j[1]:  
 flag = False  
 break  
 if flag:  
 if (i[0] != 'S') and (i[1] == 'e'):  
 continue  
 else:  
 new\_g.append(i)  
 new\_new\_g = []  
 for i in new\_g:  
 if i[0] == 'S' and i[1] == 'e':  
 new\_new\_g.append(["W", 'e'])  
 new\_new\_g.append(["W", 'S'])  
 continue  
 new\_new\_g.append(i)  
 return new\_new\_g  
  
  
def find\_next(g: list, c: str) -> list:  
 res = []  
 for r in g:  
 if r[0][0] == c:  
 res.append(r[1])  
 return res  
  
  
def replace\_rules(g: list, l, r) -> list:  
 new\_g = []  
 for i in range(len(g)):  
 left, right = g[i]  
 new\_g.append([left, right.replace(l, r)])  
 return new\_g  
  
  
def delete\_chain(g: list) -> list:  
 new\_g = g.copy()  
 for rule in new\_g:  
 left, right = rule  
 tmp = find\_next(g, left)  
 if len(tmp[0]) == 1 and len(tmp) == 1:  
 new\_g = replace\_rules([r if r[0] != left and r[1] != tmp[0] or r[0] == 'S' else ['', ''] for r in new\_g],  
 left, tmp[0])  
  
 i = 0  
 while i < len(new\_g):  
 if new\_g[i][0] == '':  
 new\_g.pop(i)  
 else:  
 i += 1  
  
 return new\_g  
  
  
def find\_letter(vn: list) -> str:  
 letter = ''  
 for c in alf:  
 if c not in vn:  
 letter = c  
 vn.append(c)  
 break  
 return letter  
  
  
def delete\_left\_fact(g: list) -> list:  
 vn = []  
  
 for i in g:  
 for j in i:  
 for k in j:  
 if k in alf and k not in vn:  
 vn.append(k)  
  
 rassm = []  
 for i in range(len(g)):  
 rassm.append([])  
 rassm[i].append(g[i])  
 for j in range(i, len(g)):  
 if i != j and g[i][0] == g[j][0] and any(  
 g[i][1][:k] == g[j][1][:k] for k in range(1, min(len(g[i][1]), len(g[j][1])))):  
 rassm[i].append(g[j])  
  
 tmp = []  
 for i in range(len(rassm)):  
 if len(rassm[i]) > 1:  
 tmp.append(rassm[i])  
 rassm = tmp.copy()  
  
 tmp = []  
 for r in g:  
 if all(r[0] != j[0] or r[1] != j[1] for i in rassm for j in i):  
 tmp.append(r)  
 g = tmp.copy()  
  
 for i in range(len(rassm)):  
 letter = find\_letter(vn)  
  
 k = 100  
 while any(rassm[i][p][1][:k] != rassm[i][s][1][:k] for p in range(len(rassm[i])) for s in range(len(rassm[i]))):  
 k -= 1  
  
 for j in rassm[i]:  
 g.append([letter, j[1][k:]])  
  
 g.append([rassm[i][0][0][:k], rassm[i][0][1][:k] + letter])  
  
 return g  
  
  
def delete\_left\_recurs(g: list) -> list:  
 vn = []  
 for i in g:  
 for j in i:  
 for k in j:  
 if k in alf and k not in vn:  
 vn.append(k)  
  
 rassm = []  
 for i in range(len(vn)):  
 rassm.append([vn[i], [], []])  
 for j in g:  
 if j[0] == vn[i]:  
 if vn[i] == j[1][0]:  
 rassm[i][1].append(j[1])  
 else:  
 rassm[i][2].append(j[1])  
  
 new\_rules = []  
 for i in rassm:  
 if len(i[1]) > 0:  
 letter = find\_letter(vn)  
 for j in i[1]:  
 r = [letter, j[1:] + letter]  
 new\_rules.append(r)  
 r = [letter, j[1:]]  
 new\_rules.append(r)  
 if len(i[2]) > 0:  
 for j in i[2]:  
 r = [i[0], j + letter]  
 new\_rules.append(r)  
 r = [i[0], j]  
 new\_rules.append(r)  
  
 new\_g = []  
 for i in g:  
 if all(i[0][0] != r[0][0] for r in new\_rules):  
 new\_g.append(i)  
  
 new\_g += new\_rules  
 return new\_g  
  
  
print("1. Проверка существования языка грамматики\n"  
 "2. Устранение нетерминалов, не порождающих терминальных строк\n"  
 "3. Устранение недостижимых символов\n"  
 "4. Устранение е-правил\n"  
 "5. Устранение цепных правил\n"  
 "6. Устранение левой факторизации правил\n"  
 "7. Устранение прямой левой рекурсии\n")  
  
x = int(input("Выберете алгоритм - "))  
  
g = input\_g()  
  
if is\_first(g) and is\_two(g):  
 f = []  
 if x == 1:  
 f = check\_exist(g)  
 elif x == 2:  
 f = delete\_useless(g, check\_exist(g))  
 elif x == 3:  
 f = delete\_unreachable(g)  
 elif x == 4:  
 f = delete\_e(g)  
 elif x == 5:  
 f = delete\_chain(g)  
 elif x == 6:  
 f = delete\_left\_fact(g)  
 else:  
 f = delete\_left\_recurs(g)  
  
 if x != 1:  
 print\_g(f)  
else:  
 print("Не второй тип")