

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
 «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

**Лабораторная работа № 2**

**Тема: «Эквивалентные преобразования контекстно-свободных грамматик»**

**Дисциплина: «Формальные языки»**

Выполнил:

студент группы ВКБ-21

Котелевец К.А.

Проверил:

доцент Пиневич Е.В.

Ростов-на-Дону

2024

Цель:

* закрепить понятия «эквивалентные грамматики», «приведенная КС-грамматика»;
* сформировать умения и навыки эквивалентных преобразований контекстно-свободных грамматик.

Постановка задачи к лабораторной работе №2:

**Дано:**

Грамматика G = ({a, b, k, {, [, }, ], ⊥},{S, R, T, F, E}, P, S}, где P состоит из следующих пунктов:

1) S -> {R | [R

2) R -> Ra} | Ra] | a | T| F | ɛ

3) F -> {F} | bb

4) T -> [T]

5) E -> k⊥

**Задача:**

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) ε-правил;

г) цепных правил;

д) левой факторизации правил;

е) прямой левой рекурсии.

**Ход работы:**

По данной мне грамматике проведем реализацию эквивалентных преобразований, которые представлены далее:

|  |
| --- |
| А) Удаление бесполезных символов  ВХОД: КС-грамматика G = ({a, b, k, {, [, ], }, ⊥}, {S, R,T,F,E}, P, S}  P: 1) S → {R | [R  2) R → Ra} | Ra] | T | a | F | ɛ  3) F → {F} | bb  4) T → [T]  5)E → k⊥  Выход: G’ = (Vт, V’n, P’, S)  Шаг 1  N0 = ∅  N1 = {S, R, F, E}  N2 = {S, R, F, E} => т.к. N1=N2, то N={S,R,F,E}. После удаления бесполезных символов и правил вывода получим: G’=({a,b,k,{,[,},],⊥},{S,R,F,E},P’,S}  P’: 1) S → {R | [R  2) R → Ra} | Ra] | T | a | F | ɛ  3) F → {F} | bb  4)E → k⊥ |

|  |
| --- |
| Б) Удаление недостижимых символов  ВХОД: КС-грамматика G = ({a, b, k, {, [, ], }, ⊥}, {S, R,T,F,E}, P, S}  P: 1) S → {R | [R  2) R → Ra} | Ra] | T | a | F | ɛ  3) F → {F} | bb  4) T → [T]  5)E → k⊥  Выход: G’ = (Vт, V’n, P’, S)  Шаг 1:  W0 = {S}  W1 = {S, R}  W2 = {S, R, T, F}  W3 = {S, R, T, F} => т.к. W2=W3, то W={S,R,F,T}. Множество недостижым символов Vб = {E}. После удаления недостижимых символов и правил вывода получим: G’=({a,b,k,{,[,},],⊥},{S,R,F,T},P’,S}  P’: 1) S → {R | [R  2) R → Ra} | Ra] | T | a | F | ɛ  3) F → {F} | bb  4)T → [T] |

|  |
| --- |
| В) Удаление e-правил  Шаг 1:  N0 → {R}  N1 → {R} => т.к. N0=N1 => искомое множество построено и N = {R}  Шаг 2:  P’= 1) C -> S | e  2) S → {R | [R  3) R → Ra} | Ra] | T | a | F  4) F → {F} | bb  5) T → [T]  6)E → k⊥ |

|  |
| --- |
| Г) Удаление цепных правил  Шаг 1:   * 1. N0(S) = {S}   N1(S) = {S} => N(S) = {S}   * 1. N0(R) = {R}   N1(R) = {R,T,F}  N2(R)= {R,T,F} => N(R)={R,T,F}   * 1. N0(F) = {F}   N1(F) = {F} => N(F) = {F}   * 1. N0(T) = {T}   N1(T) = {T} => N(T) = {T}   * 1. N0(E) = {E}   N1(E) = {E} => N(E) = {E}  Шаг 2:  G’ = ({a, b, k, {, [, ], }, ⊥}, {S, R,T,F,E}, P’, S}  P’: 1) S → {R | [R  2) R → Ra} | Ra] | [T] | a | {F}| bb | ɛ  3) F → {F} | bb  4) T → [T]  5)E → k⊥ |

|  |
| --- |
| Д) Удаление левой факторизации правил  Шаг 1:  R → Ra} | Ra] | a | T | F | ɛ  Шаг 2:  R → R(a}| a]) | a | T | F | ɛ  Шаг 3, 4: G’ = ({a, b, k, {, [, ], }, ⊥}, {S, R,T,F,E,R5’}, P’, S}  P’;1) S → {R | [R  2) R → RR’ | T | a | F | ɛ  3) F → {F} | bb  4) T → [T]  5)E → k⊥  6) R’ → a} | a] |

|  |
| --- |
| Е) Удаление левой рекурсии  Шаг 1:  R → Ra} | Ra]  R → a | T | F | ɛ  Шаг 2:  R’ → a}R’ | a]R’  R’ → a} | a]  Шаг 3, 4: G’ = ({a, b, k, {, [, ], }, ⊥}, {S, R,T,F,E,R’}, P’, S}  P’;1) S → {R | [R  2) R → TR’ | aR’ | FR’ | ɛR’  3) F → {F} | bb  4) T → [T]  5)E → k⊥  6) R’ → a}R’ | a]R’ | ɛ |

Грамматика является контекстно-свободной, т.к. слева отправил только нетерминалы, а справа терминалы и нетерминалы. Язык также существует, т.к. стартовый символ присутствует в правилах грамматики.

Для определения является ли грамматика КС или нет по классификации Хомского было реализовано программное средство, в котором также осуществлено эквивалентное преобразование грамматики, направленных на удаление различных правил

Листинг программы

|  |
| --- |
| import itertools as iter import string from collections import defaultdict from itertools import chain from typing import Iterator, Any, Generator  massT = str(input("Множество терминалов: ")).split() massN = str(input("Множество нетерминалов: ")).split() S = str(input("Введите стартовый символ: ")) n = int(input("Количество правил: ")) print("Введите правила по типу: Aa = Rpppp\n"  "Если из левой части есть несколько переходов, пропишите их через пробел слева\n"  "В качестве пустой цепочки выступает точка (.)\n")   def check\_KS():  *"""  Предлагает пользователю ввести последовательность строк, представляющих контекстно-свободную грамматику.  Анализирует каждую введенную строку и сохраняет разобранное представление в списке списков под названием `rooles`.  Проверяет, присутствуют ли все символы в первой строке каждого ввода в заранее определенном списке под названием `massN`.  Если какие-либо символы отсутствуют, увеличивает счетчик под названием `count`.  Если `count` равно нулю, устанавливает строку результата равной "Введена КС-грамматика" и устанавливает флаг в 1.  В противном случае устанавливает строку результата равной "Введенная грамматика не является КС-грамматикой" и устанавливает флаг в 0.  Возвращает кортеж, содержащий строку результата, флаг и список `rooles`.  """* count = 0  rooles = []  for i in range(n):  roole\_n = str(input()).split()  a = roole\_n[0]  b = []  raw\_count = 0  for j in roole\_n:  if j == '=':  raw\_count = 1  elif raw\_count > 0 and j != '=':  b.append(j)  rooles.append([a, b])  list\_a = list(a)  if not all(x in massN for x in list\_a):  count += 1  if count == 0:  rez, c = 'Введена КС-грамматика', 1  else:  rez, c = "Введенная грамматика не является КС-грамматикой", 0  return rez, c, rooles   def check\_exist(lst, massN, S):  *"""  Проверяет, существует ли данная строка S в списке строк massN.   Параметры:  lst (list): Список кортежей, каждый из которых представляет язык. Каждый кортеж содержит два списка:  первый список представляет нетерминальные символы, второй список - терминальные символы.  massN (list): Список строк, представляющих все возможные терминальные символы.  S (str): Строка, представляющая язык для проверки существования.   Возвращает:  str: Строка, указывающая на существование или несуществование языка. Возвращает 'Язык не существует',  если язык не существует, и 'Язык существует', если язык существует.  """* N0 = []  for i in range(len(lst)):  for j in chain(lst[i][1], lst[i][0]):  [N0.append(x) for x in list(j) if x in massN and x not in N0]  if S not in N0:  return 'Язык не существует'  else:  return "Язык существует"   def del\_useless\_sym(massT, massN, lst):  *"""  Эта функция принимает три параметра: massT, massN и lst.  Она удаляет ненужные символы из переданных списков и возвращает обновленные списки.  Затем определяется вложенная функция cycle\_el(), которая принимает список mass и итерируется по lst,  чтобы найти элементы, имеющие подмножества massT и mass.  Функция инициализирует массив mass значением '.' и вызывает cycle\_el() для обновления массива mass.  Затем вызывается cycle\_el() еще раз для обновления Ni.  Процесс продолжается до тех пор, пока N1 и Ni не будут равны.  Затем функция создает новый список N, фильтруя элементы из massN, которые находятся в Ni.  Если длина списка N не равна 0, функция создает новый список r и итерируется по lst, чтобы найти элементы,  имеющие подмножества Ni, massT или '.'.  """* print('a) бесполезных символов')   def cycle\_el(mass):  mass\_el\_mass = massT + mass  for i in range(len(lst)):  for x in lst[i][1]:  if set(list(x)).issubset(mass\_el\_mass):  if lst[i][0] not in mass:  mass.append(lst[i][0])  return mass   mass = ['.']  N1 = cycle\_el(mass)  Ni = cycle\_el(N1.copy())  while N1 != Ni:  N1 = Ni  Ni = cycle\_el(N1.copy())  N = [element for element in massN if element not in Ni] *# Бесполезные символы* if len(N) != 0:  r = [] *# Будущие новые правила* for i in range(len(lst)):  r0 = []  for j in lst[i][1]:  v = []  [v.append(x) for x in list(j) if x in Ni or x in massT or x == '.']  if ''.join(v) == j:  r0.append(j)  if len(r0) != 0:  r.append([lst[i][0], r0])  else:  r = lst  Ni.remove('.')  return Ni, r   def no\_way\_sym(lst):  *"""  Находит недостижимые символы в заданном списке правил.   Параметры:  - lst (list): Список правил, где каждое правило представляет собой кортеж, содержащий символ и список символов.   Возвращает:  - T (list): Список символов, которые достижимы из начального символа.  - Ni (list): Список символов, которые недостижимы.  - r (list): Список правил, где каждое правило представляет собой кортеж, содержащий символ и список символов.   Функция принимает список правил, где каждое правило представляет собой кортеж, содержащий символ и список символов.  Она находит символы, не достижимые из начального символа, выполняя алгоритм обнаружения циклов. Затем определяет  достижимые символы и возвращает их вместе с недостижимыми символами и правилами, содержащими недостижимые символы.   Пример:  lst = [('S', [('A', 'B'), ('C', 'D')]), ('A', [('A', 'A'), ('B', 'B')]), ('B', [('C', 'C'), ('D', 'D')]), ('C', [('A', 'A'), ('B', 'B')]), ('D', [('C', 'C'), ('D', 'D')])]  no\_way\_sym(lst)  (['A', 'C'], ['B', 'D'], [('A', [('A', 'A'), ('B', 'B')]), ('C', [('A', 'A'), ('B', 'B')])])  """* print('б) недостижимых символов')   def cycle\_el(mass):  for i in range(len(lst)):  if lst[i][0] in mass:  [mass.append(x) for j in lst[i][1] for x in list(j) if x in massN and x not in mass]  return mass   mass = [lst[0][0]]  N1 = cycle\_el(mass)  Ni = cycle\_el(N1.copy())  while N1 != Ni:  N1 = Ni  Ni = cycle\_el(N1.copy())  N = [element for element in massN if element not in Ni] *# Бесполезные символы* if len(N) != 0:  r = [] *# Будущие новые правила* for i in range(len(lst)):  if lst[i][0] in Ni:  r.append(lst[i])  else:  r = lst  T = []  for i in range(len(r)):  [T.append(x) for j in lst[i][1] for x in list(j) if x in massT and x not in T]  return T, Ni, r   def del\_eps\_rooles(massN, lst, S):  *"""  Функция для удаления эпсилон-правил из заданного грамматического разбора.   Параметры:  - massN: список, содержащий нетерминальные символы  - lst: список, представляющий правила грамматики  - S: начальный символ   Возвращает:  - massN: обновленный список нетерминальных символов  - r: обновленные правила грамматики после удаления эпсилон-правил  - S\_new: обновленный начальный символ после удаления эпсилон-правил  """* print('в) е-правил')   def cycle\_el(mass):  for i in range(len(lst)):  for x in lst[i][1]:  if set(list(x)).issubset(mass):  if lst[i][0] not in mass:  mass.append(lst[i][0])  return mass   mass = ['.']  N1 = cycle\_el(mass)  Ni = cycle\_el(N1.copy())  while N1 != Ni:  N1 = Ni  Ni = cycle\_el(N1.copy())  if S in Ni:  c = 0  i = 0  S\_new = list(string.ascii\_uppercase)  while c == 0:  if S\_new[i] not in Ni:  S\_new = S\_new[i]  c = 1  else:  i += 1  massN.append(S\_new)  *# Формируем новые правила* r = [[S\_new, [S, '.']]]  for i in range(len(lst)):  r0 = []  *# Если левая часть есть в списке Ni (правила переходят в е)* if lst[i][0] in Ni:  *# Пробегаемся по правым частям* for j in range(len(lst[i][1])):  if (any(ele in lst[i][1][j] for ele in Ni)) and (lst[i][1][j] != '.'):  *# Добавляем ориг. в итоговый список* r0.append(lst[i][1][j])  *# Рассматриваем правило как список элементов* el = list(lst[i][1][j])  ind = [x for x in range(len(el)) if el[x] in Ni] *# Индексы элементов из Ni в el  # Формируем список комбинаций индексов, которые позже удалим* combin = []  for k in range(1, len(ind) + 1):  for comb in iter.combinations(ind, k):  combin.append(comb)  *# Удаляем элементы поиндексно и добавляем результат в итоговый список* for m in range(len(combin)):  orig\_prav = list(lst[i][1][j])  for x in reversed(combin[m]):  orig\_prav.pop(x)  r0.append(''.join(orig\_prav))  elif lst[i][1][j] != '.':  r0.append(lst[i][1][j])  r.append([lst[i][0], r0])  else:  r.append(lst[i])  else:  r = lst  S\_new = S  return massN, r, S\_new   def zip\_rooles(massN, lst):  *"""  Функция для генерации списка правил без правил-epsilon.   Параметры:  massN (list): Список терминалов.  lst (list): Список правил, где каждое правило представлено кортежем, где первый элемент - нетерминал, а второй элемент - список терминалов/нетерминалов.   Возвращает:  list: Список правил без правил-epsilon.  """* print('г) цепных правил')   def cycle\_el(mass):  for i in range(len(lst)):  if lst[i][0] in mass:  [mass.append(j) for j in lst[i][1] if j in massN and j not in mass]  return mass   *# Формируем общий список Ni для каждого нетерминала* mass\_Ni = []  for i in range(len(lst)):  mass = [lst[i][0]]  N1 = cycle\_el(mass)  Ni = cycle\_el(N1.copy())  while N1 != Ni:  N1 = Ni  Ni = cycle\_el(N1.copy())  mass\_Ni.append([lst[i][0], Ni.copy()])  *# Формируем новый список правил без цепных правил* r = []  for i in range(len(mass\_Ni)):  r0 = []  for j in range(len(lst)):  if lst[j][0] in mass\_Ni[i][1]:  r01 = [el for el in lst[j][1] if el not in massN]  r0.extend(r01)  r.append([mass\_Ni[i][0], r0])  return r   def left\_factorize(grammar: dict[str, list[str]]) -> dict[str, list[str]]:  *"""  Эта функция выполняет левую факторизацию по данной грамматике  Параметры:  grammar: словарь, представляющий грамматику, где ключи - это нетерминальные символы,  а значения - списки производственных правил.  Возвращает:  Left-factorized грамматику  """* print('д) левой факторизации правил')  updated\_grammar = {}   for non\_terminal in grammar:  productions = grammar[non\_terminal]  common\_prefixes = {}   for production in productions:  prefix = production[0]  if prefix not in common\_prefixes:  common\_prefixes[prefix] = []  common\_prefixes[prefix].append(production)   new\_productions = []   for prefix in common\_prefixes:  if len(common\_prefixes[prefix]) > 1:  new\_non\_terminal = non\_terminal + "'"  updated\_grammar[new\_non\_terminal] = [production[1:] for production in common\_prefixes[prefix]]  new\_productions.append(prefix + new\_non\_terminal)  else:  new\_productions.extend(common\_prefixes[prefix])   updated\_grammar[non\_terminal] = new\_productions   grammar.update(updated\_grammar)  return grammar   def remove\_left\_recursion(grammar):  print('е) левой рекурсии')  new\_grammar = {}  non\_terminals = list(grammar.keys())   for A in non\_terminals:  A\_rules = grammar[A]  alpha\_rules = [rule for rule in A\_rules if rule.startswith(A)]  beta\_rules = [rule for rule in A\_rules if not rule.startswith(A)]   if not alpha\_rules:  new\_grammar[A] = A\_rules  continue   A\_prime = A + "'"  new\_rules\_A = []  new\_rules\_A\_prime = []   for beta\_rule in beta\_rules:  new\_rule = beta\_rule + A\_prime  new\_rules\_A.append(new\_rule)   for alpha\_rule in alpha\_rules:  new\_rule = alpha\_rule[len(A):] + A\_prime  new\_rules\_A\_prime.append(new\_rule)   new\_rules\_A\_prime.append('.') *# точка (.) представляет собой пустую цепочку* new\_grammar[A] = new\_rules\_A  new\_grammar[A\_prime] = new\_rules\_A\_prime   return new\_grammar   rez, c, rooles = check\_KS() print('1)', rez) if c == 1:  rez = check\_exist(rooles, massN, S)  print('2)', rez)  print('Исходная грамматика:') print('G = (', massT, ',', massN, ', P,', S, ')') print("\n".join(f"{i[0]} -> {' '.join(i[1])}" for i in rooles))  print('Эквивалентное преобразование грамматики посредству удаления:')  massN1, rooles1 = del\_useless\_sym(massT, massN, rooles) print('G = (', massT, ',', massN1, ', P,', S, ')') print("\n".join(f"{i[0]} -> {' '.join(i[1])}" for i in rooles1))  massT1, massN1, rooles1 = no\_way\_sym(rooles) print('G = (', massT1, ',', massN1, ', P,', S, ')') print("\n".join(f"{i[0]} -> {' '.join(i[1])}" for i in rooles1))  massN1, rooles1, S\_new = del\_eps\_rooles(massN, rooles, S) print('G = (', massT, ',', massN1, ', P,', S\_new, ')') print("\n".join(f"{i[0]} -> {' '.join(i[1])}" for i in rooles1))  rooles1 = zip\_rooles(massN1, rooles1) print('G = (', massT, ',', massN1, ', P,', S\_new, ')') print("\n".join(f"{i[0]} -> {' '.join(i[1])}" for i in rooles1))  rooles1 = left\_factorize({item[0]: item[1] for item in rooles}) print(type(rooles1)) print("\n".join(f"{key} -> {value}" for key, value in rooles1.items()))  grammar = {item[0]: item[1] for item in rooles} new\_grammar = remove\_left\_recursion(grammar) new\_grammar = {key: value for key, value in new\_grammar.items()} print("\n".join(f"{key} -> {value}".rstrip("|") for key, value in new\_grammar.items())) |

На рисунках 1 и 2 представлен пример работы программы

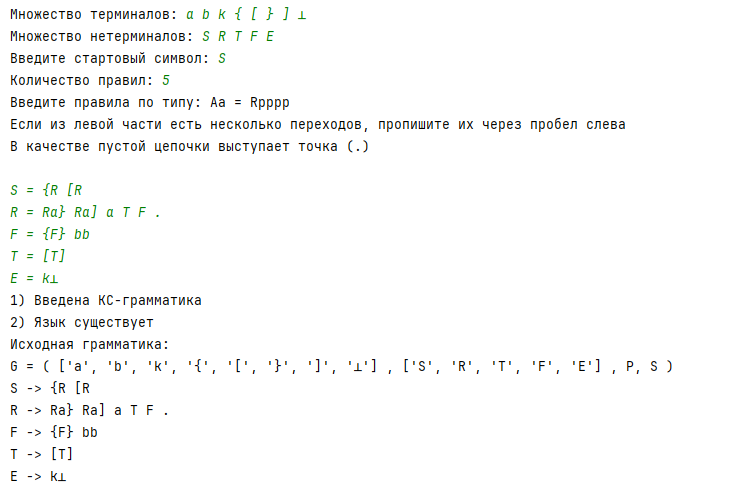


Рисунок 1 – пример работы программы

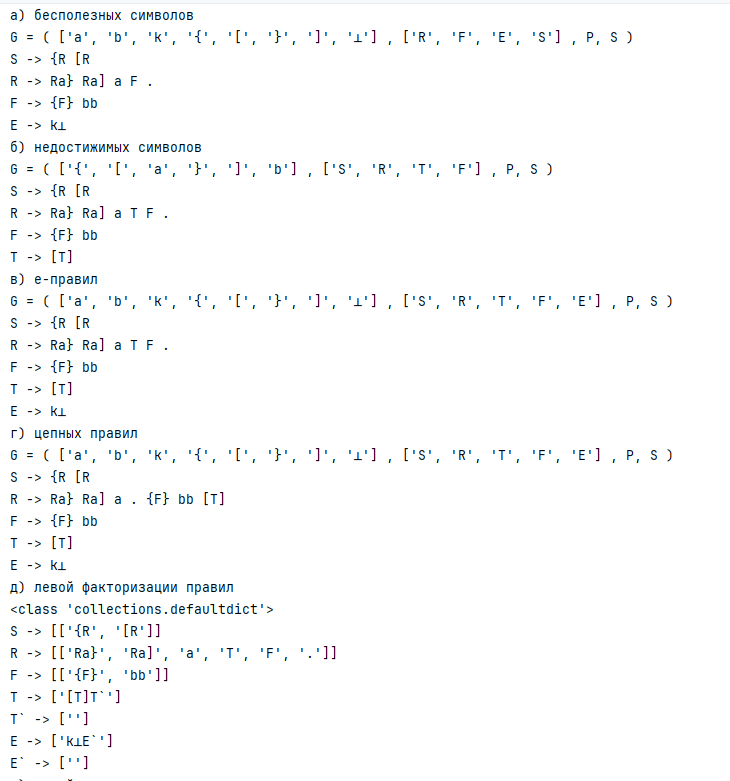
****

Рисунок 2 – пример работы программы

**Вывод:** в ходе лабораторной работы я научился определять – является ли грамматика контекстно свободной, а также сформировал умения и навыки эквивалентных преобразований контекстно-свободных грамматик.