

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

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

**(ДГТУ)**

**Лабораторная работа № 1**

**Тема: «Распознавание типов формальных языков и грамматик»**

**Дисциплина: «Формальные языки»**

Выполнил:

студент группы ВКБ-21

Котелевец К.А.

Проверил:

доцент Пиневич Е.В.

Ростов-на-Дону

2024

Цель:

* закрепить понятия “алфавит”, “цепочка”, “формальная грамматика”, “формальный язык”, “выводимость цепочек”, “эквивалентная грамматика”;
* сформировать умения и навыки распознавания типов формальных языков и грамматик по классификации Хомского, построения эквивалентных грамматик.

Постановка задачи к лабораторной работе №1:

При выполнении лабораторной работы следует реализовать следующие действия:

1. составить грамматику, порождающую формальный язык, заданный в соответствии с вариантом;
2. определить тип формальной грамматики и языка по классификации Хомского;
3. Разработать программное средство, распознающее тип введенной пользователем грамматики по классификации Хомского.

* Алфавитом *V* называется конечное множество символов.
* Цепочкой α в алфавите *V* называется любая конечная последовательность символов этого алфавита.
* Формальной грамматикой называется четверка вида: *G* = (*VT, VN* , *P*, *S*), где *VN* - конечное множество нетерминальных символов грамматики (обычно прописные латинские буквы); *VT* - множество терминальных символов грамматики (обычно строчные латинские буквы, цифры, и т.п.),   
  *VT* ∩*VN =*∅; *Р* – множество правил вывода грамматики, являющееся конечным подмножеством множества (*VT* ∪ *VN*)*+* × (*VT* ∪ *VN*)*\**; элемент (α*,* β) множества *Р* называется правилом вывода и записывается в виде α→β (читается: «из цепочки α выводится цепочкаβ»); *S -* начальный символ грамматики, *S* ∈*VN*.
* Языком, порожденным грамматикой *G* = (*VT, VN,* *P*, *S*),называется множество всех цепочек в алфавите *VT*, которые выводимы из начального символа грамматики *S* c помощью правил множества *Р*, т.е. множество *L*(*G*) = {α ∈*VT*\* | *S* ⇒\*α}.

Для определения типов грамматик по классификации Хомского было реализовано программное средство, в основе которого были заложены следующие операции:

Листинг программы

|  |
| --- |
| **import** **sys**  **import** **re**  # неограниченная грамматика  **def** **unlimited**(rules, dataSymbols):  **for** rule **in** rules:  # если слева нету нетерминала, то возвращаем false  **if** re.search(«[«+dataSymbols[«nonterminal»]+»]», rule[**0**]) == None:  **return** False  **return** True  # контекстно-зависимая грамматика  **def** **contextSensitive**(rules, dataSymbols):  **for** rule **in** rules:  **for** i **in** range(len(rule) - **1**):  # левая часть больше правой  **if** len(rule[**0**]) > len(rule[i + **1**]):  **return** False  **return** True  # контекстно-свободная грамматика  **def** **contextFree**(rules, dataSymbols):  **for** rule **in** rules:  # нету только нетерминала слева  **if** re.match(«^[«+dataSymbols[«nonterminal»]+»]$», rule[**0**]) == None:  **return** False  **return** True  # регулярная грамматика  **def** **regular**(rules, dataSymbols):  rightRegular = **1**  **for** rule **in** rules:  **if** re.match(«^[«+dataSymbols[«nonterminal»]+»]$», rule[**0**]) == None:  **return** False  **else**:  **for** i **in** range(len(rule) - **1**):  # не леволинейная и была праволинейной  **if** re.match(«^[«+dataSymbols[«nonterminal»]+»]?[«+dataSymbols[«terminal»]+»]+$», rule[i+**1**]) == None **or** rightRegular == **2**:  # не праволинейная и была леволинейной  **if** re.match(«^[«+dataSymbols[«terminal»]+»]+[«+dataSymbols[«nonterminal»]+»]?$», rule[i+**1**]) == None **or** rightRegular == **0**:  **return** False  # праволинейная  **else**:  rightRegular = **2**  # леволинейная  **else**:  rightRegular = **0**  **return** True  **print**(«Классификация Хомского для грамматики G = (T, N, P, S)»)  **print**(«Символ 'e' зарезервирован для пустого символа!»)  dataSymbols = {}  dataSymbols[«terminal»] = re.sub(r»(\[\[\]**\\**\/\^\$\.\|\?\\*\+\(\)\{\}])», r»**\\**\1», input(«Введите терминальные символы T: «).replace(« «, ««)) + «e»  dataSymbols[«nonterminal»] = input(«Введите нетерминальные символы N: «).replace(« «, ««)  dataSymbols[«alphabet»] = dataSymbols[«nonterminal»] + dataSymbols[«terminal»]  count = int(input(«Введите количество правил: «))  **print**(«Знак следует обозначается '->', знак или '|'«)  productionRules = []  pattern = r»^[« + dataSymbols[«alphabet»] + «]+ \*-> \*» + «(?:[« + dataSymbols[«alphabet»] + «]+)(?:(?: \***\\**| \*)(?:[« + dataSymbols[«alphabet»] + «]+))\*$»  **for** i **in** range(count):  inputString = input(«Введите правило номер « + str(i + **1**) +»: «)  # проверка корректности ввода  **if** re.match(pattern, inputString):  productionRules.append(re.split(r» \*-> \*| \*\| \*», inputString))  **else**:  **print**(«Правило введено не верно!»)  sys.exit()  dataSymbols[«start symbol»] = input(«Введите начальный символ: «)  # сравнение первого символа первого правила с начальным символом  **if** productionRules[**0**][**0**] == dataSymbols[«start symbol»]:  # проверка на тип 0  **if** unlimited(productionRules, dataSymbols):  typeChomsky = **0**  # проверка на тип 3 - регулярность  **if** regular(productionRules, dataSymbols):  typeChomsky = **3**  # если не регулярная, проверяем на тип 2 - КС  **elif** contextFree(productionRules, dataSymbols):  typeChomsky = **2**  # если не КС, проверяем на тип 1 - КЗ  **elif** contextSensitive(productionRules, dataSymbols):  typeChomsky = **1**  **else**:  **print**(«Грамматика не подходит ни под один тип!»)  sys.exit()  **else**:  **print**(«Первый символ не соответствует начальному!»)  sys.exit()  typesChomsky = {  **0** : «тип 0 - неограниченная»,  **1** : «тип 1 - контекстно-зависимая»,  **2** : «тип 2 - контекстно-свободная»,  **3** : «тип 3 - регулярная»  }  **print**(«**\n\t**Данная грамматика имеет « + typesChomsky[typeChomsky] + «**\n**») |

На рисунке 1 представлен пример работы программы

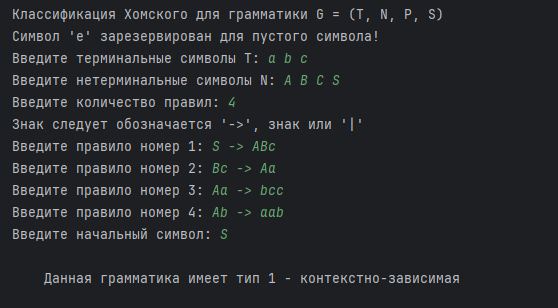


Рисунок 1 – пример работы программы

В варианте был дан такой формальный язык, представленный на рисунке 2.



Рисунок 2 – Вариант моего формального языка

Для данного языка можно составить такую грамматику:

T: {c, d}  
N: {A, S}  
S: {S}  
P: 1) S -> cAcA | dAdA 2) A -> dA | cA | E

*Тип данной грамматики*: контекстно-свободный (тип 2), т.к. слева имеются неограниченное количество N и неограниченное количество справа N, T.

*Тип языка* также контекстно-свободный

Для данного языка была составлена программа, на рисунке 3 показан пример ее работы

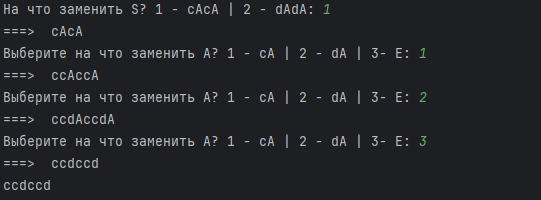


Рисунок 3 – пример работы программы

|  |
| --- |
| **def** **generate\_string**():  string = "S"  **print**("На что заменить S? 1 - cAcA | 2 - dAdA")  choice = input()  **if** choice == "1":  string = string.replace("S", "cAcA", **1**)  **print**("===> ", string)  **elif** choice == "2":  string = string.replace("S","dAdA", **1**)  **print**("===> ", string)  **else**:  **print**("Неправильный ввод")  **while** "A" **in** string:  choice = input("Выберите на что заменить A? 1 - cA | 2 - dA | 3- E: ")  **if** choice == '1':  string = string.replace("A", "cA")  **elif** choice == '2':  string = string.replace("A", "dA")  **elif** choice == '3':  string = string.replace("A", "")  **else**:  **print**("Неправильный ввод")  **print**("===> ", string)  **return** string  **print**(generate\_string()) |

**Вывод:** в ходе лабораторной работы я научился определять типы грамматик по классификации Хомского и разработал программный продукт, определяющий тип соответствующей грамматики.