ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«ИРКУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ»

(ФГБОУ ВО ИрГУПС)

Факультет: Управление на транспорте и информационные технологии

Кафедра: Информационные системы и защиты информации

Дисциплина: Теория языков программирования и методы трансляции

Лабораторная работа №1

Выполнил: Проверил:

Студент гр. ПИ 1-20-1 Преподаватель

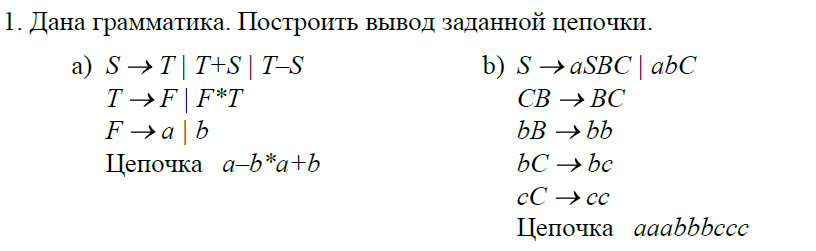
Хабибуллин Р.Д. Черкашин Е.А.

Иркутск 2023

**Лабораторная работа №1. Формальные языки, грамматики и их свойства**

*Задание на лабораторную работу*

1. Дана грамматика. Построить вывод заданной цепочки.



1) Для цепочки a-b\*a+b

S

T - S

T

F

S

T + S

T

F

S

T

F

2) Для цепочки aaabbbccc.

aSBC

aaSBCBC

aaabSBCBCBC

aaababCBCBCBC

aaababbbCCBCBC

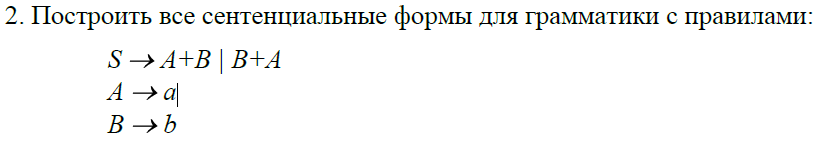
aaababbbccBCBC

aaababbbcccBBC

aaababbbcccBC

aaababbbcccbbC

aaababbbcccbbcc



S (Начальная форма)

A + B (Используем S -> A + B)

a + B (Используем A -> a)

a + b (Используем B -> b)

первая сентенциальная форма: "a + b"

S (Начальная форма)

B + A (Используем S -> B + A)

b + A (Используем B -> b)

b + a (Используем A -> a)

Вторая сентенциальная форма: "b + a"

S (Начальная форма)

A + B (Используем S -> A + B)

a + B (Используем A -> a)

a + a (Используем B -> b)

Третья сентенциальная форма: "a + a"

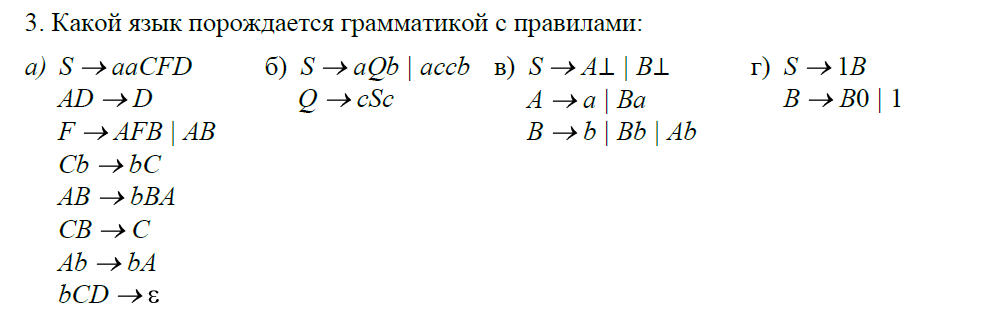
S (Начальная форма)

B + A (Используем S -> B + A)

b + A (Используем B -> b)

b + b (Используем A -> a)

Четвертая сентенциальная форма: "b + b"

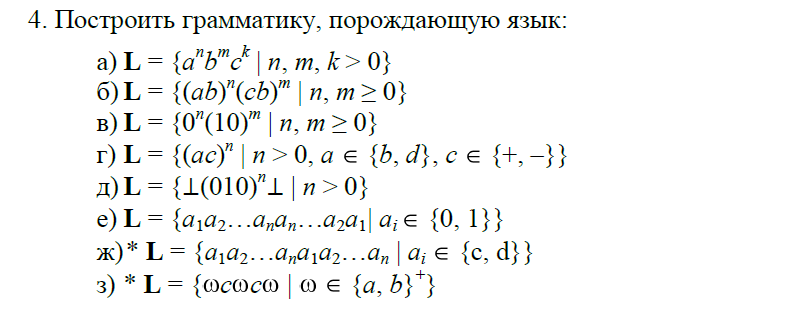


А) Этот язык можно описать как набор строк, начинающихся с двух 'a' и содержащих разные комбинации символов 'C', 'D', 'A', 'F', 'B', 'b'.

Б) Этот язык можно описать как набор строк, начинающихся с 'a', с одной или более сериями символов 'c', и заканчивающихся символом 'b'.

В) Этот язык можно описать как набор строк, состоящих из комбинаций символов "a" и "b", за которыми следует символ "⊥", и эти строки могут начинаться как строка, порождаемая A или B.

Г) Этот язык можно описать как язык бинарных чисел, начинающихся с "1".



а) L = {a^n b^m c^k | n, m, k > 0}

Грамматика:

S -> aSc | X X -> bXc | ε

б) L = {(ab)^n (cb)^m | n, m ≥ 0}

Грамматика:

S -> T | ε T -> abTcb | ε

в) L = {0^n (10)^m | n, m ≥ 0}

Грамматика:

S -> 0S | X X -> 10X | ε

г) L = {ωcωcω | ω ∈ {a, b}+}

Грамматика:

S -> aSa | bSb | cSc | ε

д) L = {(ac)^n | n > 0, a ∈ {b, d}, c ∈ {+, -}}

Грамматика:

S -> bS | dS | X X -> acX | ε

е) L = {⊥(010)^n⊥ | n > 0}

Грамматика:

S -> ⊥010S⊥ | ε

ж) L = {a1a2…anan…a2a1 | ai ∈ {0, 1}}

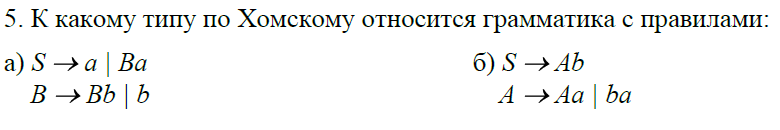
Грамматика:

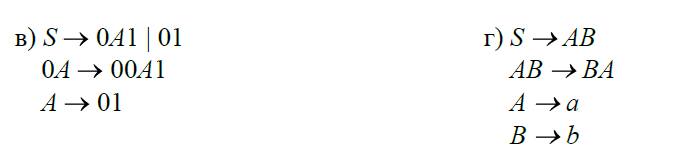
S -> 01S10 | ε

з) L = {a1a2…ana1a2…an | ai ∈ {c, d}}

Грамматика:

S -> cSc | dSd | ε



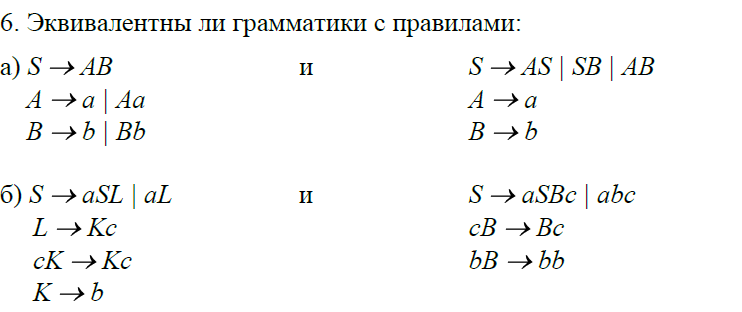


а) контекстно-свободные грамматики

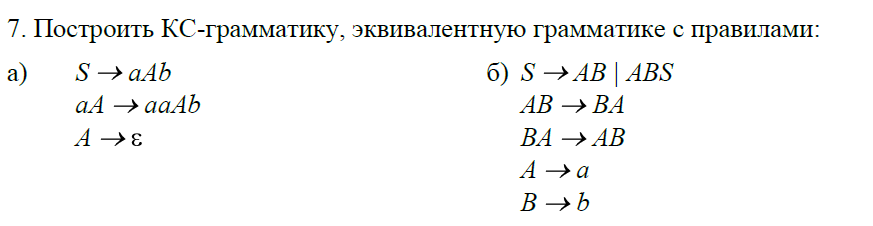
б) контекстно-свободные грамматики

в) контекстно-свободные грамматики

г) контекстно-свободные грамматики



А) не являются эквивалентными.

Б) являются эквивалентными. 

А) S -> aAb | ε

A -> aA | ε

B -> b

Б) S -> AB

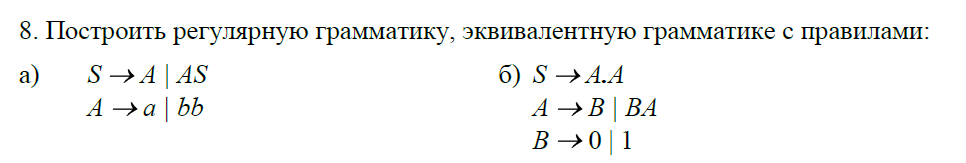
S -> ABS

A -> a

B -> b

A -> BA

B -> AB



А) S -> A | AS | ε

A -> a | bb

Б) S -> B.A | BA.A | ε

A -> B | BA

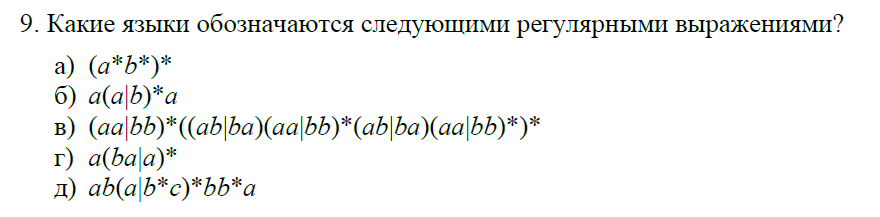
B -> 0 | 1

упростить

S -> A | B | ε

A -> B | BA

B -> 0 | 1



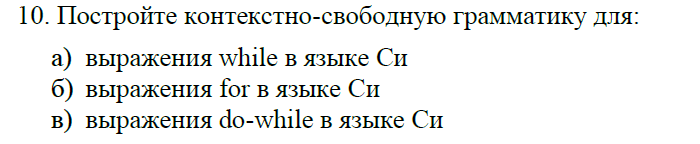
А) язык всех возможных комбинаций символов "a" и "b".

Б) Это регулярное выражение описывает язык, в котором строки начинаются и заканчиваются символом "a", а между ними может быть произвольное количество символов "a" и "b".

В) В общем случае, это язык строк, в которых блоки "aa" и "bb" чередуются с блоками "ab" и "ba".

Г) Это язык строк, которые начинаются с "a" и могут содержать произвольное количество "ba" или "a" после него.

Д) Это язык строк, которые начинаются с "ab", заканчиваются на "bb" и могут содержать произвольное количество символов "a" и "b" между ними.



А) while\_expression -> while ( condition ) statement

condition -> expression

statement -> compound\_statement | other\_statement

compound\_statement -> { statement\_list }

statement\_list -> statement | statement statement\_list

other\_statement -> /

expression -> /

б) for\_expression -> for ( for\_init ; for\_condition ; for\_update ) statement

for\_init -> /\* контекстно-свободная грамматика для инициализации в Си \*/

for\_condition -> /\* контекстно-свободная грамматика для условия в Си \*/

for\_update -> /\* контекстно-свободная грамматика для обновления в Си \*/

statement -> compound\_statement | other\_statement

compound\_statement -> { statement\_list }

statement\_list -> statement | statement statement\_list

other\_statement -> /\* другие возможные операторы внутри for \*/

в) do\_while\_expression -> do statement while ( condition ) ;

condition -> expression

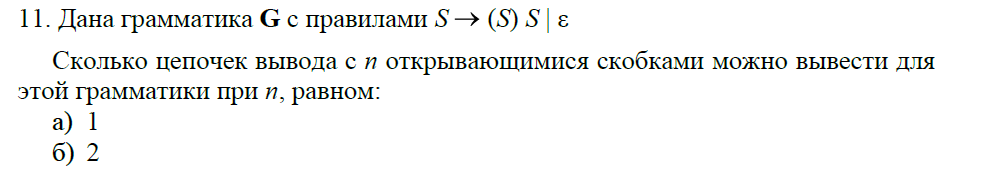
statement -> compound\_statement | other\_statement

compound\_statement -> { statement\_list }

statement\_list -> statement | statement statement\_list

other\_statement -> /\* другие возможные операторы внутри do-while \*/

expression -> /\* контекстно-свободная грамматика для выражения в Си \*/





а) При n = 1, есть только одна возможная цепочка вывода: "()".

б) При n = 2, мы можем построить следующие цепочки:

"()"()

"()"ε

ε()

Всего 3 цепочки вывода.

в) При n = 3, мы можем построить следующие цепочки (это будет более трудоемким для ручного расчета):

"()"()()

"()"()"()"

"()"ε()

"()"εε

ε()()

ε()"()"

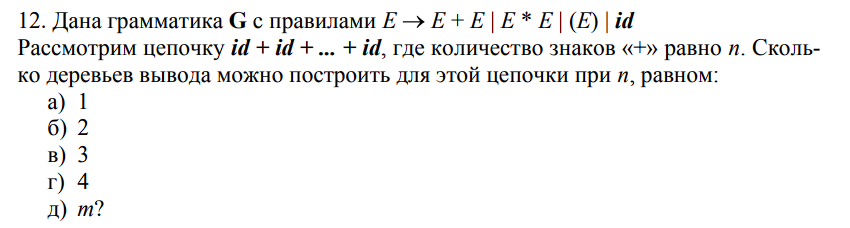
Всего 6 цепочек вывода.

г) При n = 4, количество цепочек будет больше, и их можно построить аналогичным образом. Однако для ручного расчета их количество будет заметно увеличиваться.

д) Для общего случая с n открывающимися скобками, количество цепочек вывода можно рассчитать как число Каталана C(n), которое определяется рекурсивно и имеет формулу:

C(n) = (2n)! / [(n+1)! \* n!]

Итак, для m открывающихся скобок, количество цепочек вывода будет равно C(m).



а) При n = 1, есть только один возможный способ построения дерева вывода:

E

/|\

id + id

б) При n = 2, мы можем построить два различных дерева вывода:

Дерево вывода 1:

E

/|\

E + E

/|\ |\

id + id id

Дерево вывода 2:

E

/|\

E + E

/| |\

id + id id

в) При n = 3, есть несколько различных деревьев вывода. Можно построить следующие:

Дерево вывода 1:

E

/|\

E + E

/|\ |\

E + E id

/| |\

id + id id

Дерево вывода 2:

E

/|\

E + E

/| |\

E + E id

/|\ |

id + id id

Дерево вывода 3:

E

/|\

E + E

/| \

E + E

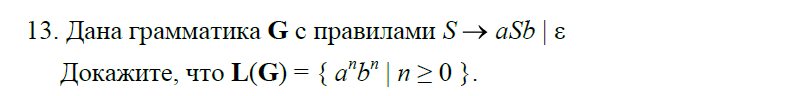
/| /|\

id id + id

|\

id

д) Для общего случая с n знаками "+", количество деревьев вывода можно рассчитать рекурсивно, но количество деревьев будет расти экспоненциально с увеличением n. Выражение для точного числа деревьев будет сложным и зависеть от n.



Пусть w принадлежит L(G), то есть w может быть выведено из грамматики G. Это означает, что существует вывод w из начального символа S в грамматике G. Рассмотрим структуру этого вывода:

Первое правило грамматики S -> aSb генерирует один символ 'a' в начале строки.

Затем следует вывод строки S (включая промежуточные выводы), который также начинается с 'a' и заканчивается 'b'.

Поскольку внутри S также могут быть символы 'a' и 'b', процесс рекурсивно продолжается.

Наконец, после всех рекурсивных вызовов S, строка заканчивается символом 'b'.

Таким образом, процесс вывода создает строки вида "anbn", где n - количество рекурсивных вызовов S. Таким образом, L(G) содержит все строки вида "anbn" для n ≥ 0.

Теперь докажем обратное включение. Пусть w принадлежит {anbn | n ≥ 0}, то есть w имеет вид "anbn" для некоторого n ≥ 0. Мы должны показать, что w может быть выведено из грамматики G.

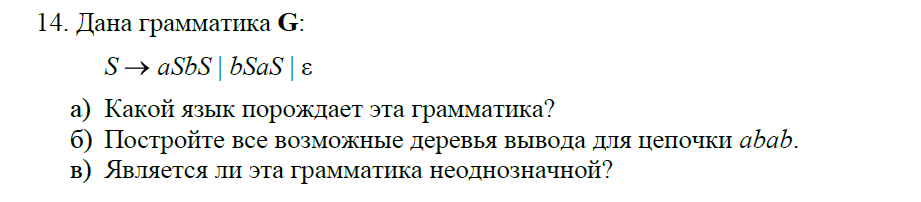
Мы можем построить вывод w следующим образом:

Начнем с вывода строки S -> ε, который приводит к пустой строке.

Затем n раз применим правило S -> aSb, чтобы добавить "anbn" в вывод.

Таким образом, мы показали, что w может быть выведено из грамматики G.

С учетом обоих включений мы можем заключить, что L(G) = {anbn | n ≥ 0}, что и требовалось доказать.



а) Эта грамматика G порождает язык строк, состоящих из символов 'a' и 'b', где количество символов 'a' совпадает с количеством символов 'b' или является пустой строкой. Формально, язык, порождаемый этой грамматикой, можно записать как {anbn | n ≥ 0}.

б)

Первый возможный вывод:

S

/ | \

a S b

|

S

/ | \

a S b

Copy code

S

/ | \

b S a

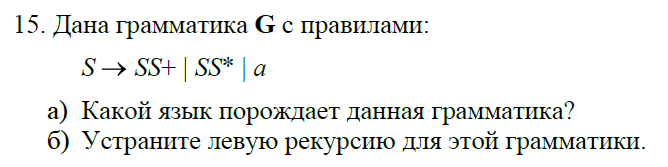
|

S

/ | \

b S a

в) Грамматика G является неоднозначной, так как существует цепочка (в данном случае "abab"), для которой существует более одного дерева вывода.



Эта грамматика порождает язык, состоящий из строк, которые могут содержать один или несколько символов 'a', а также символы '+' или '\*'.

б)

Исходная грамматика:

S -> SS+ | SS\* | a

Шаг 1: Факторизация правил:

S -> aS'

S' -> +S' | \*S' | ε

Теперь грамматика не содержит левой рекурсии, и она эквивалентна исходной грамматике.