Московский авиационный институт

(национальный исследовательский институт)

Институт «Компьютерные науки и прикладная математика»

**Лабораторные  работы**

**по курсу**

**«Системы программирования»**

**IV семестр**

1. Спроектировать грамматику по паттерн-модели регулярного языка.

2. Преобразовать спроектированную грамматику в конечный автомат, составить диаграмму переходов КА и  реализовать.

3. Определить свойства КА. Изучить  алгоритм преобразования НДКА в ДКА.

4. Устранить из КС-грамматики  бесполезные символы и ε–правила.

5. Устранить из KС-грамматики цепные правила и устранить левую рекурсию.

6. Определить форму КС-грамматики и сделать ее приведение.

7. Спроектировать МП-автомат для приведенной КС-грамматики.

8. Реализовать МП-автомат для приведенной КС-грамматики.

9. Для LL(k) анализатора построить управляющую таблицу M.

10. Аналитически написать правила вывода для цепочки LL(k) анализатора.

11. Реализовать управляющую таблицу M Для LL(k) анализатора.

12. Построить множество LR(0)-таблиц не содержащих ε-правила.

13. Для LR(k) -грамматики cпроектировать матрицу oblow.

14. Определить функции перехода g(X).

15. Определить функцию переноса-свертки f(u).

16. Для функции перехода g(X) и функции переноса-свертки f(u) спроектировать управляющую таблицу.

*Студент:* Белоносов К.А.*Группа: М8О-208Б-21*

*Руководитель:* Семёнов А. С.

*Оценка:*

*Дата:*

**Москва. 2023**

*Лабораторные работы №1-2*

1. **Pattern = «(\W|^)stock\s{0,3}tips(\W|$)»**

**Автоматная грамматика:**

L(pattern) = L(**«(\W|^)stock\s{0,3}tips(\W|$)»**) = {«stocktips» , «stock tips» , «stock tips» , «stock tips»}

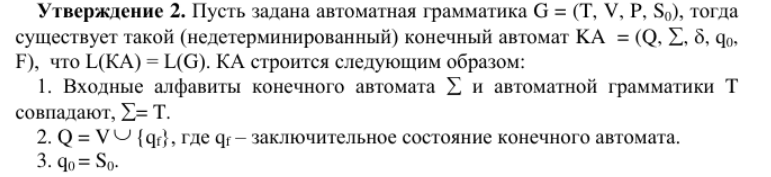
G(T, V, P, S0) = G({s, t, o, c, k, i, p, #, &, }, {S0, S, K, A, ..., I}, {p1, ..., p12}, S0)

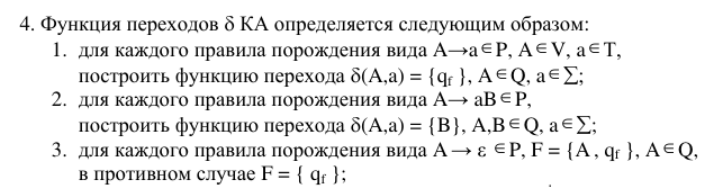
*Правила регулярной грамматики:*

*Пример цепочек:*

1. *#S*
2. *#S*
3. *#S*

*Алгоритм построения КА.*

**

****

**Конечный автомат:**

L(КА) = L(G)

КА = где

Q = { ,S, K, A, B, C, …, I, },

= {s, t, o, c, k, i, p},

=

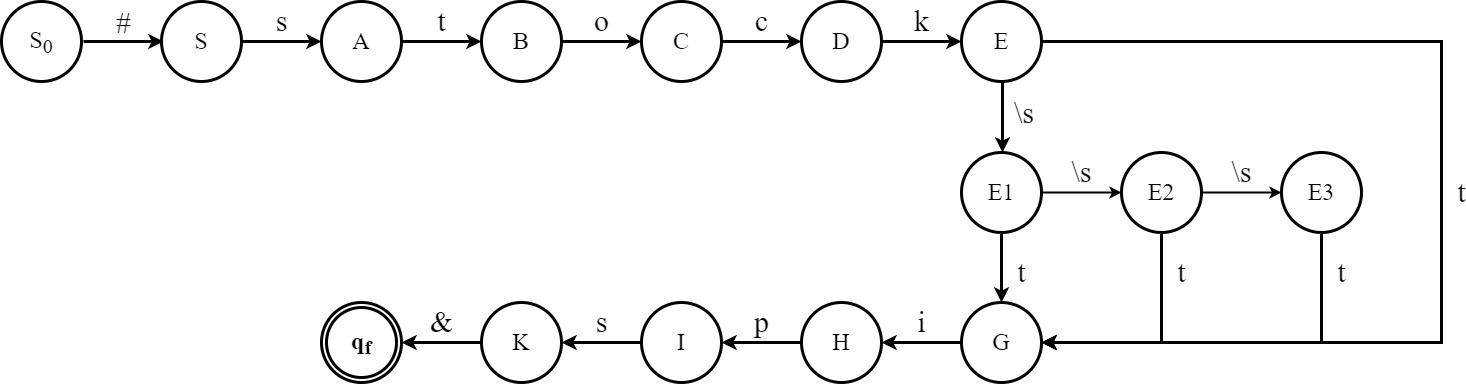
F =

= {

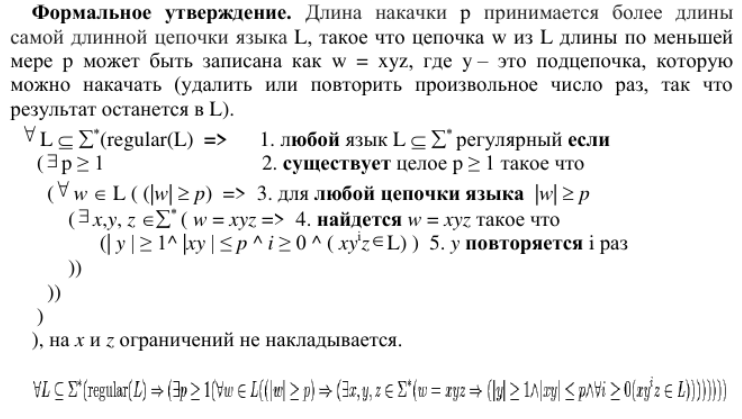
1. (, #) = {S},
2. (S, s) = {A},
3. (A, t) = {B},
4. (B, o) = {C},
5. (C, c) = {D},
6. (D, k) = {E},
7. (E, t) = {G},
8. (E, ) = {E1},
9. (E1, t) = {G},
10. (E1, ) = {E2},
11. (E2, t) = {G},
12. (E2, ) = {E3},
13. (E3, t) = {G},
14. (G, i) = {H},
15. (H, p) = {I},
16. (I, s) = {K},
17. (K, &) = {}

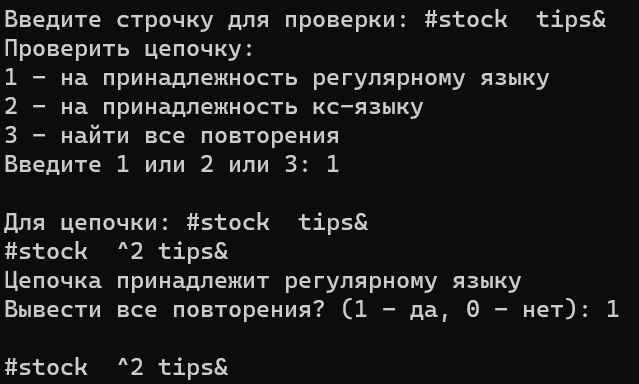
}

*Примеры конфигураций КА:*



*Лемма о накачке:*

**

**

1. **Pattern = «192\.168\.1\.»**

**Автоматная грамматика:**

L(pattern) = L(**«192\.168\.1\.»)** = {«192.168.1.0», «192.168.1.1», …, «192.168.1.255»}

G(T, V, P, S0) = G({1, 2, 3, 4, 5, 6, 7, 8, 9, \.}, {S0, A, B, ..., J, , , M}, {p1, ..., p14}, S0)

*Правила регулярной грамматики:*

1 | 2 | … |

1 | 2 | … | |

1M | 2M | … | | M

*Пример цепочек:*

**Конечный автомат:**

L(КА) = L(G)

КА = где

Q = {S0, A, B, ..., J, , , M, },

= {1, 2, 3, 4, 5, 6, 7, 8, 9, \.},

=

F =

= {

1. (, 1) = {A},
2. (A, 9) = {B},
3. (B, 2) = {C},
4. (C, \.) = {D},
5. (D, 1) = {E},
6. (E, 6) = {F},
7. (F, 8) = {G},
8. (G, \.) = {H},
9. (H, 1) = {I},
10. (I, \.) = {J},
11. (J, 1) = {},
12. (J, 2) = {},

…

1. (J, 8) = {},
2. (J, 9) = {},
3. (, 1) = {},
4. (, 2) = {},

…

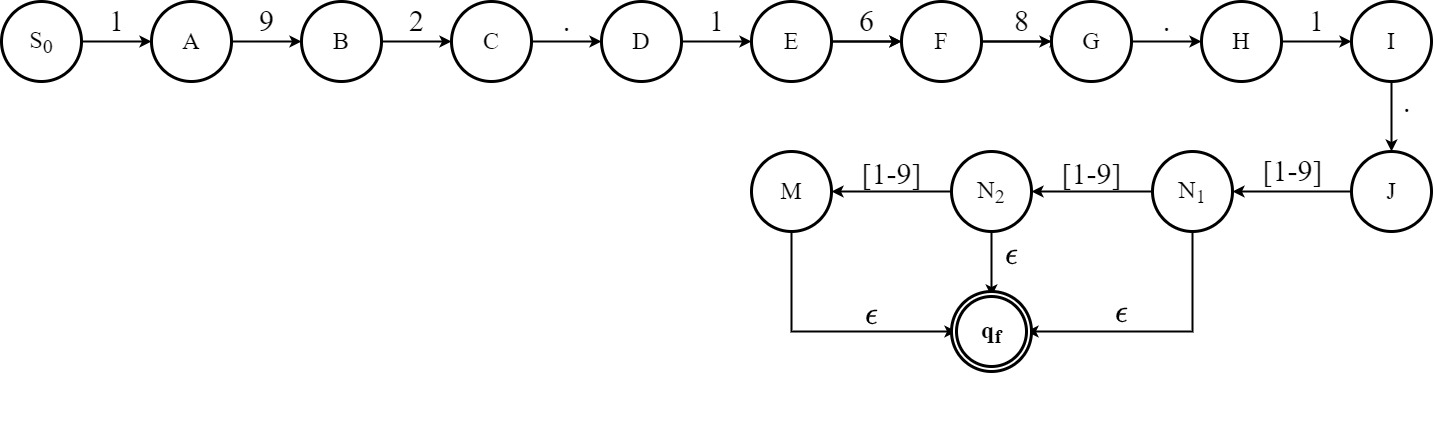
1. (, 8) = {},
2. (, 9) = {},
3. (, ) = {},
4. (, 1) = {M},
5. (, 2) = {M},

…

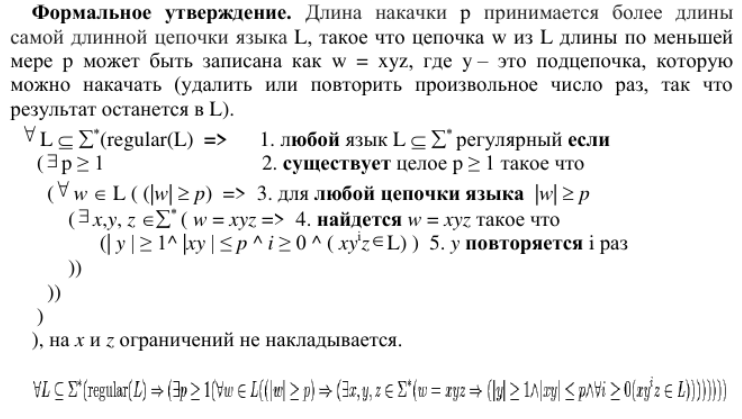
1. (, 8) = {M},
2. (, 9) = {M},
3. (, ) = {},
4. (M, ) = {}

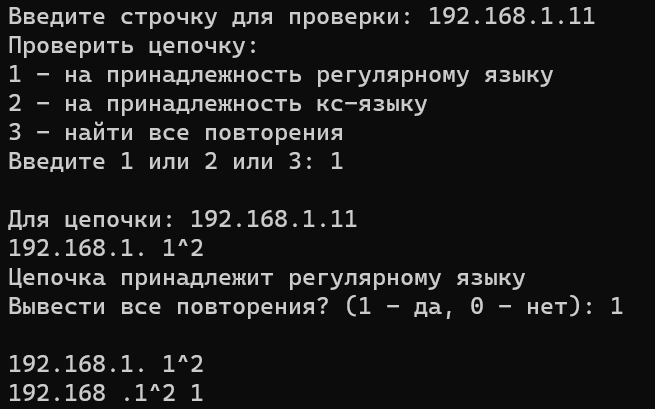
}

*Примеры конфигураций КА:*



*Лемма о накачке:*

**

**

*Лабораторная работа №3*

**Формулировка задания:**

Реализовать конечные автоматы, составленные в ЛР №2

**Код программы:**

FSAutomate[] automates = new FSAutomate[] {

new FSAutomate(

new List<Symbol>() { "S0","A","B","C","D","E","F","G","H","I","J","K","L","M","N","qf" },

new List<Symbol>() { "s", "t", "o", "c", "k", "i", "p", "#", "&", " " },

new List<Symbol>() { "qf" },

"S0"

),

new FSAutomate(

new List<Symbol>() { "S0", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "N1", "N2", "M", "qf" },

new List<Symbol>() { "1", "2", "3", "4", "5", "6", "7", "8", "9", "." },

new List<Symbol>() { "qf" },

"S0"

)

};

// First automate

automates[0].AddRule("S0","#","S");

automates[0].AddRule("S","s","A");

automates[0].AddRule("A","t","B");

automates[0].AddRule("B","o","C");

automates[0].AddRule("C","c","D");

automates[0].AddRule("D","k","E");

automates[0].AddRule("E","t","G");

automates[0].AddRule("E"," ","E1");

automates[0].AddRule("E1","t","G");

automates[0].AddRule("E1"," ","E2");

automates[0].AddRule("E2","t","G");

automates[0].AddRule("E2"," ","E3");

automates[0].AddRule("E3","t","G");

automates[0].AddRule("G","i","H");

automates[0].AddRule("H","p","I");

automates[0].AddRule("I","s","K");

automates[0].AddRule("K","&","qf");

//second automate

automates[1].AddRule("S0", "1", "A");

automates[1].AddRule("A", "9", "B");

automates[1].AddRule("B", "2", "C");

automates[1].AddRule("C", ".", "D");

automates[1].AddRule("D", "1", "E");

automates[1].AddRule("E", "6", "F");

automates[1].AddRule("F", "8", "G");

automates[1].AddRule("G", ".", "H");

automates[1].AddRule("H", "1", "I");

automates[1].AddRule("I", ".", "J");

string[] numbers = { "1", "2", "3", "4", "5", "6", "7", "8", "9" };

foreach (string curNumber in numbers)

{

automates[1].AddRule("J", curNumber, "N1");

automates[1].AddRule("N1", curNumber, "N2");

automates[1].AddRule("N2", curNumber, "M");

}

automates[1].AddRule("N1", "ε", "qf");

automates[1].AddRule("N2", "ε", "qf");

Console.WriteLine("Enter line to execute first:");

automates[0].Execute(Console.ReadLine());

Console.WriteLine("Enter line to execute second:");

automates[1].Execute(Console.ReadLine() + "ε");

**Пример работы:**

*Первый пример:*

Enter line to execute first:

#stocktips&

Length: 11

i :11

curr: qf

chineSymbol belongs to language

Enter line to execute second:

192.168.1.1

Length: 12

i :12

curr: qf

chineSymbol belongs to language

*Второй пример:*

Enter line to execute first:

#stock tips&

Length: 12

i :12

curr: qf

chineSymbol belongs to language

Enter line to execute second:

192.168.1.89

Length: 13

i :13

curr: qf

chineSymbol belongs to language

*Третий пример:*

*Enter line to execute first:*

*#stock tips&*

*Length: 15*

*i :14*

*curr: qf*

*chineSymbol doesn't belong to language*

*Enter line to execute second:*

*192.168.1.155*

*Length: 14*

*i :14*

*curr: qf*

*chineSymbol belongs to language*

*Перевод недетерминированного КА в детерминированный:*

*Automate definition:*

*Q: S0 S A B C D E G H I K qf E1 E2 E3*

*Sigma: s t o c k i p # &*

*Q0: S0*

*F: qf*

*DeltaList:*

*delta(S0,#,-> (S*

*delta(S,s,-> (A*

*delta(A,t,-> (B*

*delta(B,o,-> (C*

*delta(C,c,-> (D*

*delta(D,k,-> (E*

*delta(E,t,-> (G*

*delta(G,i,-> (H*

*delta(H,p,-> (I*

*delta(I,s,-> (K*

*delta(K,&,-> (qf*

*delta(E, ,-> (E1*

*delta(E1,t,-> (G*

*delta(E1, ,-> (E2*

*delta(E2,t,-> (G*

*delta(E2, ,-> (E3*

*delta(E3,t,-> (G*

*Enter line to execute first:*

*#stock tips&*

*Length: 13*

*i :12*

*curr: qf*

*chineSymbol doesn't belong to language*

*Automate definition:*

*Q: S0 A B C D E F G H I J N1 N2 qf*

*Sigma: 1 2 3 4 5 6 7 8 9 .*

*Q0: S0*

*F: qf*

*DeltaList:*

*delta(S0,1,-> (A*

*delta(A,9,-> (B*

*delta(B,2,-> (C*

*delta(C,.,-> (D*

*delta(D,1,-> (E*

*delta(E,6,-> (F*

*delta(F,8,-> (G*

*delta(G,.,-> (H*

*delta(H,1,-> (I*

*delta(I,.,-> (J*

*delta(J,1,-> (N1*

*delta(N1,1,-> (N2*

*delta(N2,1,-> (qf*

*delta(N2,2,-> (qf*

*delta(N2,3,-> (qf*

*delta(N2,4,-> (qf*

*delta(N2,5,-> (qf*

*delta(N2,6,-> (qf*

*delta(N2,7,-> (qf*

*delta(N2,8,-> (qf*

*delta(N2,9,-> (qf*

*delta(N1,2,-> (N2*

*delta(N1,3,-> (N2*

*delta(N1,4,-> (N2*

*delta(N1,5,-> (N2*

*delta(N1,6,-> (N2*

*delta(N1,7,-> (N2*

*delta(N1,8,-> (N2*

*delta(N1,9,-> (N2*

*delta(J,2,-> (N1*

*delta(J,3,-> (N1*

*delta(J,4,-> (N1*

*delta(J,5,-> (N1*

*delta(J,6,-> (N1*

*delta(J,7,-> (N1*

*delta(J,8,-> (N1*

*delta(J,9,-> (N1*

*Enter line to execute second:*

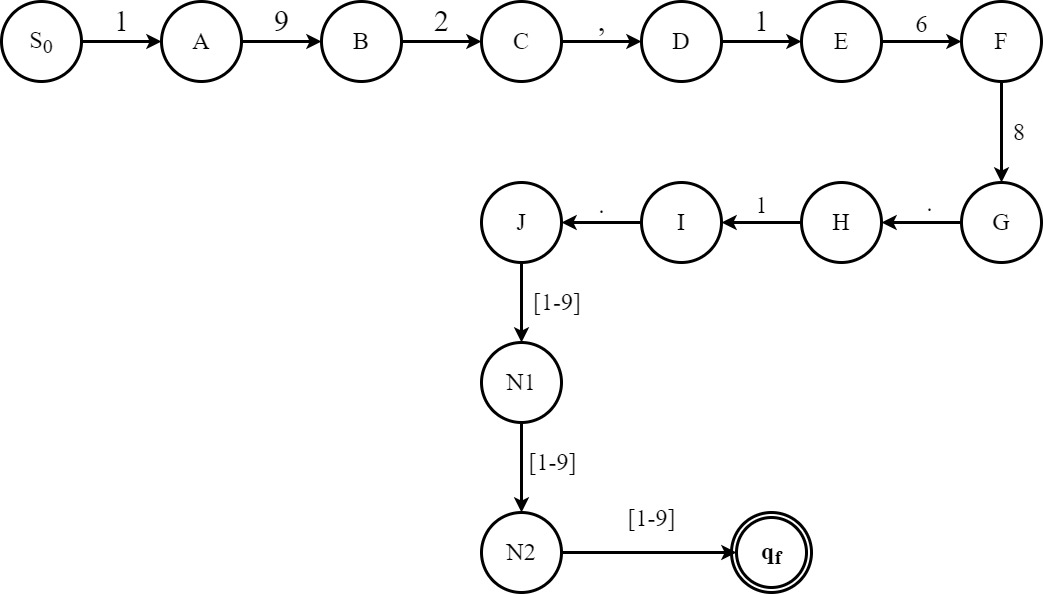
*192.168.1.165*

*Length: 14*

*i :14*

*curr: qf*

*chineSymbol doesn't belong to language*

**

*Лабораторная работа №4:*

**Формулировка задания:**

Устранить из КС-грамматики бесполезные символы и ε–правила

Исходная грамматика (a 2222, K e):

**G** = (**T**, **V**, **P**, **S0**), где

**T** = {a, b, c, d, f, g, , v, o, u};

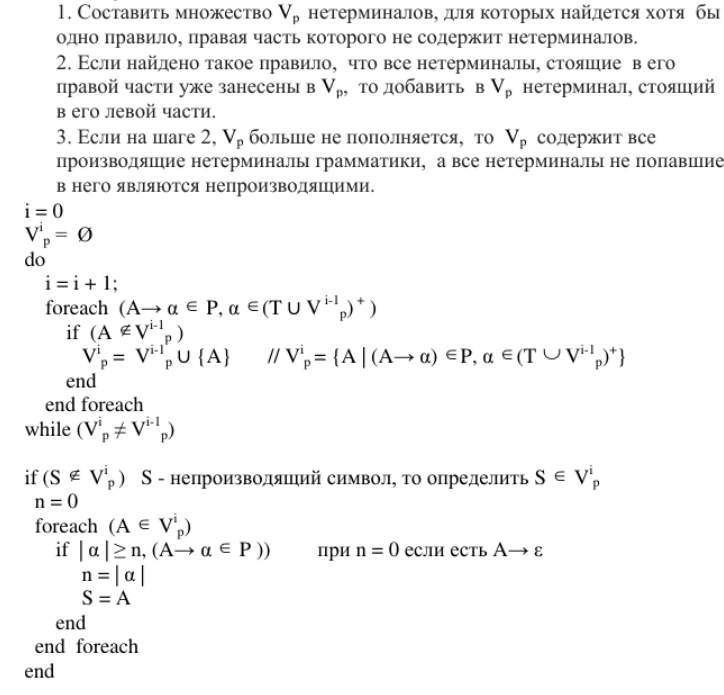
**V** = {S, A, B, C, D, L, R, O, G, E, F, H};

**P** = { S → aAb, A → BCD, C → dfB, D → L, L → g, B → cDR, R → , O → cGv, E → vFc, S → uH, H → Ho, H → o };

**S0** = S.

**Устранить из КС-грамматики бесполезные символы.**

**Устранение непроизводящих символов.**



1. V1p = {L, R, H} – для этих нетерминалов нашлось хотя бы одно правило, правая часть которого не содержит нетерминалов

2. V2p = {L, R, H, D, S, B, C, A} – если найдено такое правило, что все нетерминалы, стоящие в его правой части уже занесены в Vp, то добавить в Vp нетерминал, стоящий в его левой части

3. Vp = {L, R, H, D, S, B, C, A} содержит все производящие нетерминалы грамматики, а все нетерминалы, не попавшие в него, являются непроизводящими (V - Vp = {O, G, F, E}).

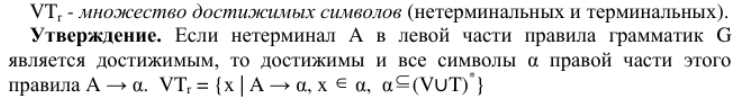


Запишем P' (те правила, в правой части которых только производящие символы):

P' = {}

G1 = ({ɛ, a, b, c, d, f, g, u, o}, {L, R, H, D, S, B, C, A}, P', S)

**Алгоритм устранения недостижимых символов**

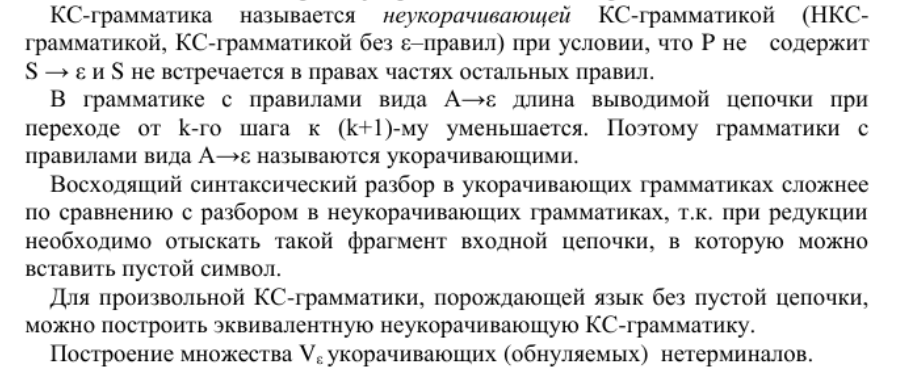




Нет недостижимых символов.

Получаем

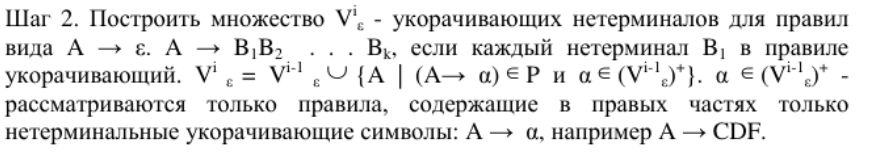
**Устранить из грамматики эпсилон-правила**

****

*Построим множество укорачивающих нетерминалов согласно алгоритму:*



(множество укорачивающих нетерминалов для правил вида A → ε)

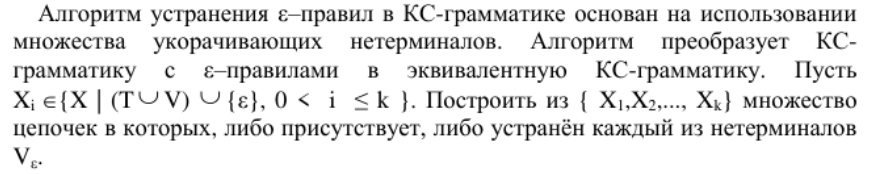


В данном варианте не присутствует укорачивающих правил вида

Таким образом:

– Множество укорачивающих терминалов

*Составим G" на основе алгоритма удаления эпсилон правил:*



, где :

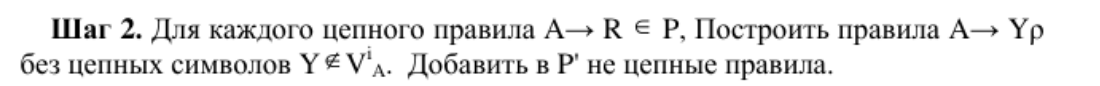
, где :

***Лабораторная работа №5: Устранить из KС-грамматики цепные правила и левую рекурсию***

**Устранить из KС-грамматики цепные правила.**



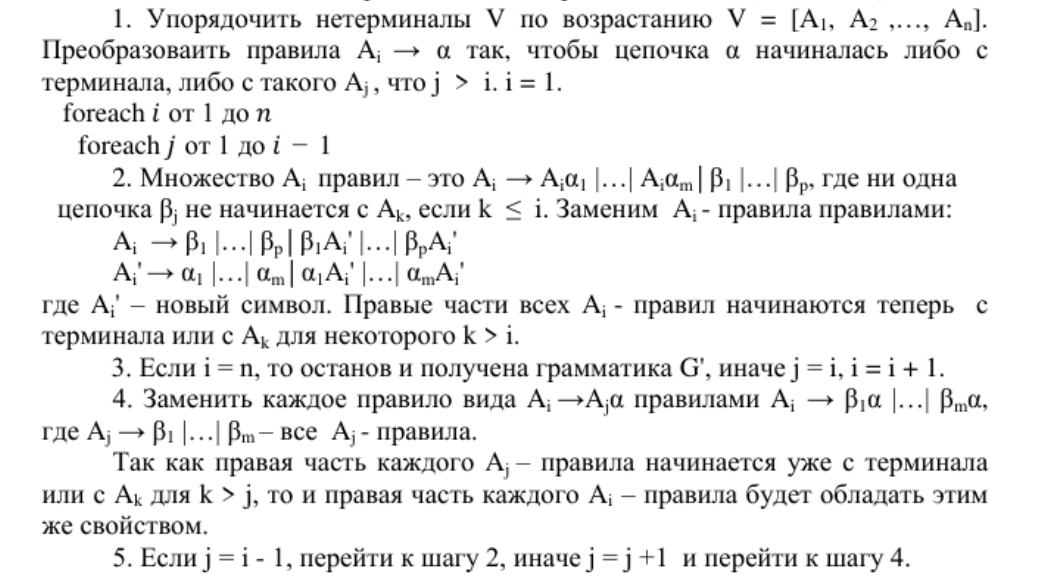
Цепные правила:



Правила «заменим» на

**Устранение из KС-грамматики левой рекурсии.**

Нетерминал КС-грамматики называется рекурсивным, если A αAβ, для некоторых α и β. Если α = ε, то A называется леворекурсивным.



*Шаг 1. Пусть*

*Шаг 2. При i = 6 преобразуем правило , заменим H правило правилом:*

*Обошли все нетерменальные символы*

*Получили правила новой грамматики*

, где :

*Лабораторная работа №6*

**Формулировка задания:**

Определить форму КС-грамматики и сделать ее приведение

Приведем нашу грамматику к Нормальной грамматике Холмского:

1. Уберем все правила где вида: , где k > 2
2. Удалим правил с несколькими терминалами.

**Уберем все длинные цепочки:**

*,*

Введем нетерминал ,

Введем правила:

*,*

Введем нетерминал ,

Введем правила:

*,*

Введем нетерминал ,

Введем правила:

**Удалим правил с несколькими терминалами.**

Правил с несколькими терминалами в данном варианте нет.

Получили нормальную форму Холмского

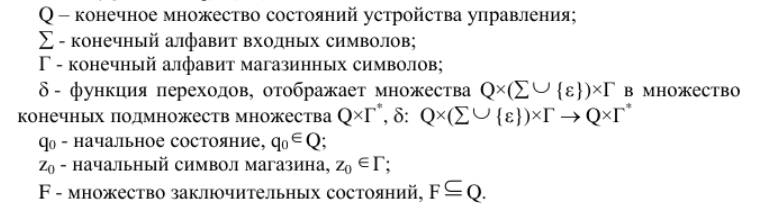
, где :

**Лабораторная работа №7**

**Формулировка задания:**

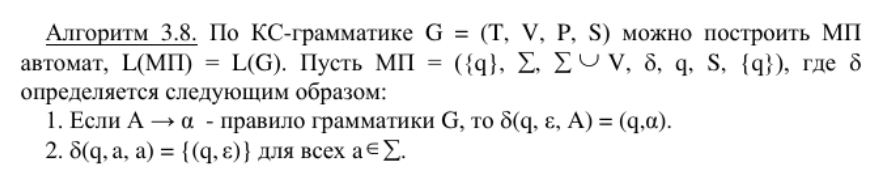
Спроектировать МП-автомат для приведённой КС-грамматики.

Автоматы с магазинной памятью (МП-автоматы) представляют собой модель распознавателей для языков, задаваемых КС-грамматиками. МП-автоматы имеют вспомогательную память, называемую магазином. В магазин можно поместить неограниченное количество символов. В каждый момент времени доступен только верхний символ магазина. Верхний символом магазина будем считать самый левый символ цепочки.

**

Приведённая грамматика:

, где P:



L(МП) = L(G)

**Q =** {q}, **Σ** = **T**, **Г** = **T** U **V**, **δ** = **δ**, **q0** = **q0**, **z0** = **S0**, **F** = {q}

МП = ({q}, , , **δ**, q0, s, {q})

**Определение:** Конфигурацией МП-автомата называется тройка (q, ω, z) ∈ Q×Σ\*×Г\*, где q – текущее состояние управляющего устройства;

ω – необработанная часть входной цепочки (первый символ цепочки ω

находится под входной головкой; если ω = ε, то считается, что вся входная цепочка прочитана);

z – содержимое магазина (самый левый символ цепочки z считается верхним символом магазина; если z = ε, то магазин считается пустым).

Функция перехода **δ** определяется согласно алгоритму:



|  |  |
| --- | --- |
|  |  |

Последовательность тактов МП-автомата

*Лабораторная работа №8*

**Формулировка задания:**

Реализовать спроектированный МП-автомат для приведённой КС-грамматики.

**Код программы:**

var CFGrammar = new Grammar(new List<Symbol>() { "a","b", "c", "d", "f", "g", "u", "o" },

new List<Symbol>() { "S", "S'", "A", "A'", "B", "C", "C'", "D", "H", "H'" },

"S");

CFGrammar.AddRule("S",new List<Symbol>() { "a","S'" });

CFGrammar.AddRule("S'",new List<Symbol>() { "A","b" });

//CFGrammar.AddRule("U",new List<Symbol>() { "\*" });

CFGrammar.AddRule("A",new List<Symbol>() { "B","A'" });

CFGrammar.AddRule("A'",new List<Symbol>() { "C","D" });

CFGrammar.AddRule("C",new List<Symbol>() { "d","C'" });

CFGrammar.AddRule("C'",new List<Symbol>() { "f", "B" });

CFGrammar.AddRule("D",new List<Symbol>() { "g" });

CFGrammar.AddRule("B",new List<Symbol>() { "c", "D" });

CFGrammar.AddRule("S",new List<Symbol>() { "u", "H" });

CFGrammar.AddRule("H",new List<Symbol>() { "o", "H'" });

CFGrammar.AddRule("H",new List<Symbol>() { "o" });

CFGrammar.AddRule("H'",new List<Symbol>() { "o", "H'" });

CFGrammar.AddRule("H'",new List<Symbol>() { "o" });

CFGrammar.DebugPrules();

var pda = new PDA(CFGrammar);

pda.Debug();

Console.Write("Enter chain:\n");

Console.WriteLine(pda.Execute(Console.ReadLine()).ToString());

**Пример работы:**

*Prules:*

*S -> aS'*

*S' -> Ab*

*A -> BA'*

*A' -> CD*

*C -> dC'*

*C' -> fB*

*D -> g*

*B -> cD*

*S -> uH*

*H -> oH'*

*H -> o*

*H' -> oH'*

*H' -> o*

*Delta rules:*

*delta(q,ε,S) -> (q,a,S')*

*delta(q,ε,S') -> (q,A,b)*

*delta(q,ε,A) -> (q,B,A')*

*delta(q,ε,A') -> (q,C,D)*

*delta(q,ε,C) -> (q,d,C')*

*delta(q,ε,C') -> (q,f,B)*

*delta(q,ε,D) -> (q,g)*

*delta(q,ε,B) -> (q,c,D)*

*delta(q,ε,S) -> (q,u,H)*

*delta(q,ε,H) -> (q,o,H')*

*delta(q,ε,H) -> (q,o)*

*delta(q,ε,H') -> (q,o,H')*

*delta(q,ε,H') -> (q,o)*

*delta(q,a,a) -> (q,ε)*

*delta(q,b,b) -> (q,ε)*

*delta(q,c,c) -> (q,ε)*

*delta(q,d,d) -> (q,ε)*

*delta(q,f,f) -> (q,ε)*

*delta(q,g,g) -> (q,ε)*

*delta(q,u,u) -> (q,ε)*

*delta(q,o,o) -> (q,ε)*

*delta(q0,ε,z0) -> (qf,ε)*

*Enter chain:*

*acgdfcggb*

*step 1*

*delta(q,ε,S) -> (q,a,S')*

*step 1*

*delta(q,a,a) -> (q,ε)*

*step 2 a a*

*step 3 a*

*step 1*

*delta(q,ε,S') -> (q,A,b)*

*step 1*

*delta(q,ε,A) -> (q,B,A')*

*step 1*

*delta(q,ε,B) -> (q,c,D)*

*step 1*

*delta(q,c,c) -> (q,ε)*

*step 2 c c*

*step 3 c*

*step 1*

*delta(q,ε,D) -> (q,g)*

*step 1*

*delta(q,g,g) -> (q,ε)*

*step 2 g g*

*step 3 g*

*step 1*

*delta(q,ε,A') -> (q,C,D)*

*step 1*

*delta(q,ε,C) -> (q,d,C')*

*step 1*

*delta(q,d,d) -> (q,ε)*

*step 2 d d*

*step 3 d*

*step 1*

*delta(q,ε,C') -> (q,f,B)*

*step 1*

*delta(q,f,f) -> (q,ε)*

*step 2 f f*

*step 3 f*

*step 1*

*delta(q,ε,B) -> (q,c,D)*

*step 1*

*delta(q,c,c) -> (q,ε)*

*step 2 c c*

*step 3 c*

*step 1*

*delta(q,ε,D) -> (q,g)*

*step 1*

*delta(q,g,g) -> (q,ε)*

*step 2 g g*

*step 3 g*

*step 1*

*delta(q,ε,D) -> (q,g)*

*step 1*

*delta(q,g,g) -> (q,ε)*

*step 2 g g*

*step 3 g*

*step 1*

*delta(q,b,b) -> (q,ε)*

*step 2 b b*

*step 3 b*

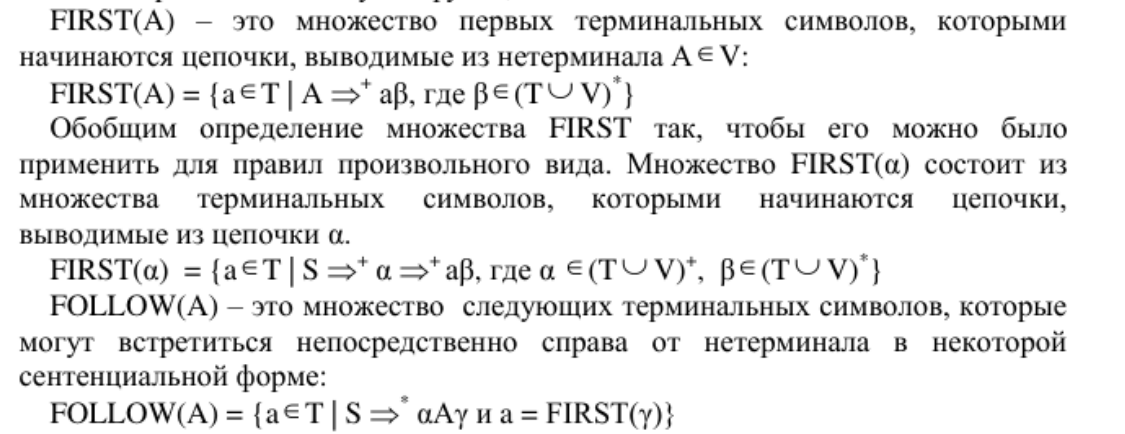
*True*

*Лабораторная работа №9:*

**Формулировка задания:**

Для LL(k) анализатора построить управляющую таблицу M.

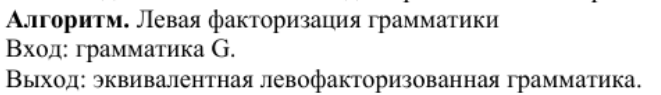
**Определение:** КС-грамматика G = (T, V, P, S) без ε-правил называется простой LL(1) грамматикой (s-грамматикой, разделенной грамматикой), если для каждого v ∈ V все его альтернативы начинаются различными терминальными символами. Единица в названии алгоритма означает, что при чтении анализируемой цепочки, находящейся на входной ленте, входная головка может заглядывать вперед на один символ.

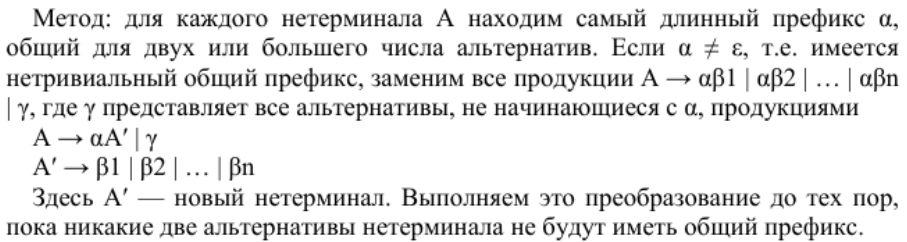
**

*Исходная грамматика:*

, где P:

Преобразуем к LL(1) грамматике





, где P:

**Алгоритм построения управляющей таблицы M для LL(1)-грамматики**

Вход: LL(1)-грамматика G = (T, V, P, S)

Выход: Управляющая таблица M для грамматики G.

Таблица M определяется на множестве (V U T U {}) × (T U {ε}) по правилам:

1. Если A → β – правило вывода грамматики с номером i, то M(А, a) = (β, i) для всех a ≠ ε, принадлежащих множеству FIRST(β). Если ε ∈ FIRST(β), то M(А, b) = (β, i) для всех b ∈ FOLLOW(A).
2. M(a, a) = ВЫБРОС для всех a ∈ T.
3. M(Ʇ, ε) = ДОПУСК.
4. В остальных случаях M(X, a) = ОШИБКА для X(V U T U {}) и a ∈ T U {ε}

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | f | g | u | o |  |
| S | aS`, 1 |  |  |  |  |  | uH, 9 |  |  |
| S` |  |  | Ab, 2 |  |  |  |  |  |  |
| A` |  |  |  | CD, 3 |  |  |  |  |  |
| A |  |  | BA`, 4 |  |  |  |  |  |  |
| C |  |  |  | dC`, 5 |  |  |  |  |  |
| C` |  |  |  |  | fB, 6 |  |  |  |  |
| D |  |  |  |  |  | g, 7 |  |  |  |
| B |  |  | cD, 8 |  |  |  |  |  |  |
| H |  |  |  |  |  |  |  | oT, 10 |  |
| H` |  |  |  |  |  |  |  | oT`, 13 |  |
| T |  |  |  |  |  |  |  | H`, 11 |  |
| T` |  |  |  |  |  |  |  | H`, 11 |  |
| a | ВЫБРОС |  |  |  |  |  |  |  |  |
| b |  | ВЫБРОС |  |  |  |  |  |  |  |
| c |  |  | ВЫБРОС |  |  |  |  |  |  |
| d |  |  |  | ВЫБРОС |  |  |  |  |  |
| f |  |  |  |  | ВЫБРОС |  |  |  |  |
| g |  |  |  |  |  | ВЫБРОС |  |  |  |
| u |  |  |  |  |  |  | ВЫБРОС |  |  |
| o |  |  |  |  |  |  |  | ВЫБРОС |  |
|  |  |  |  |  |  |  |  |  | ДОПУСК |

*Лабораторная работа №10:*

**Формулировка задания:**

Аналитически написать правила вывода для цепочки LL(k) анализатора.

Рассмотрим работу алгоритма для цепочки ()

Шаг 1. Алгоритм находится в начальной конфигурации ((), S0Ʇ, ε), где S0 = K. Значение управляющей таблицы M(K, f) = (A, 1), при этом выполняются следующие действия:

* Заменить верхний символ магазина R цепочкой **V**.
* Не сдвигать читающую головку.
* На выходную ленту поместить номер использованного правила 1.

|  |  |
| --- | --- |
| Текущая конфигурация | Значение М |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

*Лабораторная работа №11:*

**Формулировка задания:**

Реализовать управляющую таблицу M Для LL(k) анализатора.

**Код программы:**

*var LL = new Grammar(new List<Symbol>() { "a", "b", "c", "d", "f", "g", "u", "o", "ε" },*

*new List<Symbol>() { "S","S'","A", "A'", "C", "C'", "B", "H", "H'", "D", "T", "T'" },*

*"S");*

*LL.AddRule("S",new List<Symbol>() { "a", "S'"});*

*LL.AddRule("S'",new List<Symbol>() { "A", "b" });*

*LL.AddRule("A'",new List<Symbol>() { "C", "D" });*

*LL.AddRule("A",new List<Symbol>() { "B", "A'" });*

*LL.AddRule("C",new List<Symbol>() { "d", "C'" });*

*LL.AddRule("C'",new List<Symbol>() { "f", "B" });*

*LL.AddRule("D",new List<Symbol>() { "g" });*

*LL.AddRule("B",new List<Symbol>() { "c", "D" });*

*LL.AddRule("S",new List<Symbol>() { "u", "H" });*

*LL.AddRule("H",new List<Symbol>() { "o", "T" });*

*LL.AddRule("T",new List<Symbol>() { "H'" });*

*LL.AddRule("T",new List<Symbol>() { "ε" });*

*LL.AddRule("H'",new List<Symbol>() { "o", "T'" });*

*LL.AddRule("T'",new List<Symbol>() { "H'" });*

*LL.AddRule("T'",new List<Symbol>() { "ε" });*

*var parser = new LLParser(LL);*

*Console.WriteLine("Введите строку: ");*

*string stringChain = Console.ReadLine();*

*var chain = new List<Symbol> { };*

*foreach (var x in stringChain)*

*chain.Add(new Symbol(x.ToString()));*

*if (parser.Parse(chain)) {*

*Console.WriteLine("Допуск. Цепочка символов = L(G).");*

*Console.WriteLine(parser.OutputConfigure);*

*} else {*

*Console.WriteLine("Не допуск. Цепочка символов не = L(G).");*

*}*

**Результат работы программы:**

*Создадим таблицу. Сначала создадим по столбцу для каждого из этих терминалов:*

*a, b, c, d, f, g, u, o, ε,*

*Также создаем строку для Эпсилон*

*Рассмотрим нетерминал S*

*Первый символ правила S -> aS' - a*

*Это правило заносим в таблицу на пересечении строки нетерминала S и столбца терминала a*

*Первый символ правила S -> uH - u*

*Это правило заносим в таблицу на пересечении строки нетерминала S и столбца терминала u*

*Рассмотрим нетерминал S'*

*Первый символ правила S' -> Ab - c*

*Это правило заносим в таблицу на пересечении строки нетерминала S' и столбца терминала c*

*Рассмотрим нетерминал A*

*Первый символ правила A -> BA' - c*

*Это правило заносим в таблицу на пересечении строки нетерминала A и столбца терминала c*

*Рассмотрим нетерминал A'*

*Первый символ правила A' -> CD - d*

*Это правило заносим в таблицу на пересечении строки нетерминала A' и столбца терминала d*

*Рассмотрим нетерминал C*

*Первый символ правила C -> dC' - d*

*Это правило заносим в таблицу на пересечении строки нетерминала C и столбца терминала d*

*Рассмотрим нетерминал C'*

*Первый символ правила C' -> fB - f*

*Это правило заносим в таблицу на пересечении строки нетерминала C' и столбца терминала f*

*Рассмотрим нетерминал B*

*Первый символ правила B -> cD - c*

*Это правило заносим в таблицу на пересечении строки нетерминала B и столбца терминала c*

*Рассмотрим нетерминал H*

*Первый символ правила H -> oT - o*

*Это правило заносим в таблицу на пересечении строки нетерминала H и столбца терминала o*

*Рассмотрим нетерминал H'*

*Первый символ правила H' -> oT' - o*

*Это правило заносим в таблицу на пересечении строки нетерминала H' и столбца терминала o*

*Рассмотрим нетерминал D*

*Первый символ правила D -> g - g*

*Это правило заносим в таблицу на пересечении строки нетерминала D и столбца терминала g*

*Рассмотрим нетерминал T*

*Первый символ правила T -> H' - o*

*Это правило заносим в таблицу на пересечении строки нетерминала T и столбца терминала o*

*Первый символ правила T -> ε - ε*

*Это правило заносим в таблицу на пересечении строки нетерминала T и столбца терминала ε*

*Рассмотрим нетерминал T'*

*Первый символ правила T' -> H' - o*

*Это правило заносим в таблицу на пересечении строки нетерминала T' и столбца терминала o*

*Первый символ правила T' -> ε - ε*

*Это правило заносим в таблицу на пересечении строки нетерминала T' и столбца терминала ε*

*Введите строку:*

*acgdfcggb*

*Допуск. Цепочка символов = L(G).*

*12487356877*

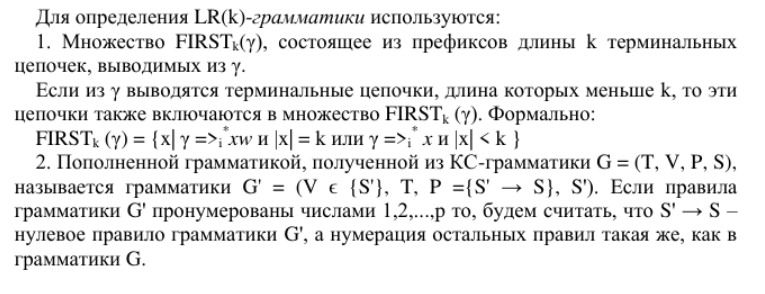
**Практическая работа №4 (лабораторные 12-16)**

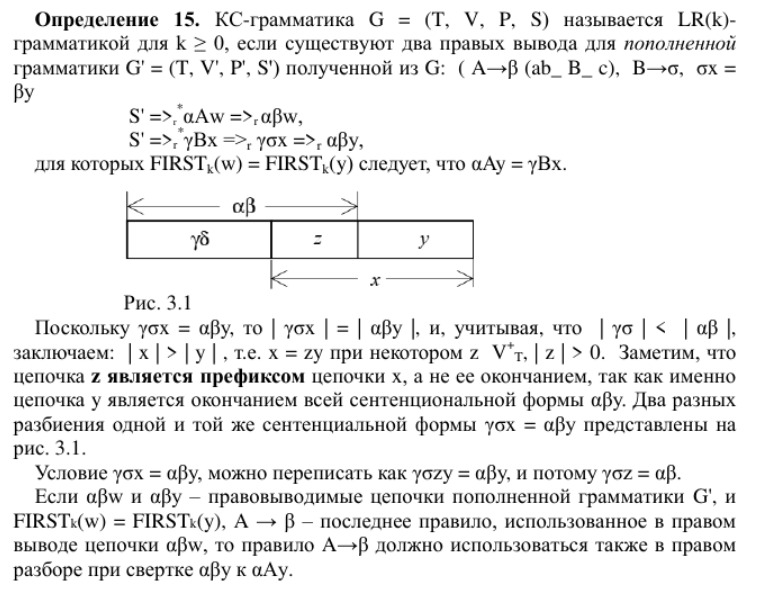
**Формулировка задания:**

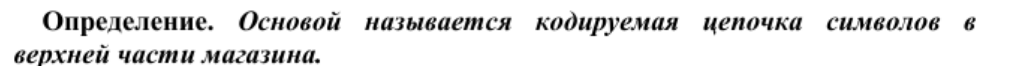
Построить управляющую таблицу M для LR(k)-грамматики, написать правило вывода выделенной строки. Описать работу алгоритма LR(k) анализатора. Построить LR(k) анализатор на основе грамматического вхождения.

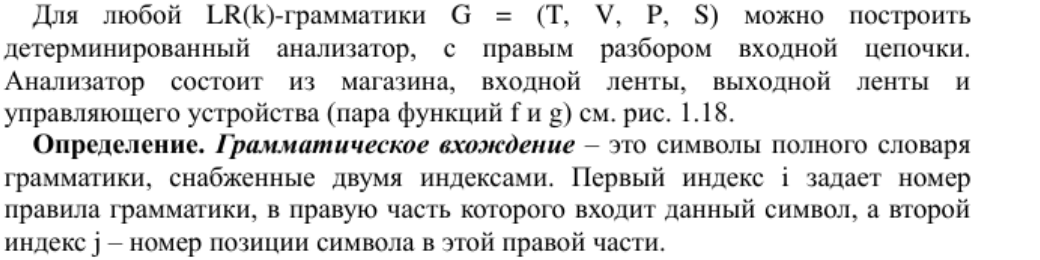
Синтаксический LR-анализатор анализирует входную цепочку слева направо (L), и строит правый (R) вывод грамматики.

Грамматики, для которых можно построить детерминированный восходящий анализатор, называются LR(k)-грамматиками (входная цепочка читается слева (Left) направо, выходом анализатора является правый (Right) разбор, k – число символов входной цепочки, на которое можно “заглянуть” вперёд для выделения основы).









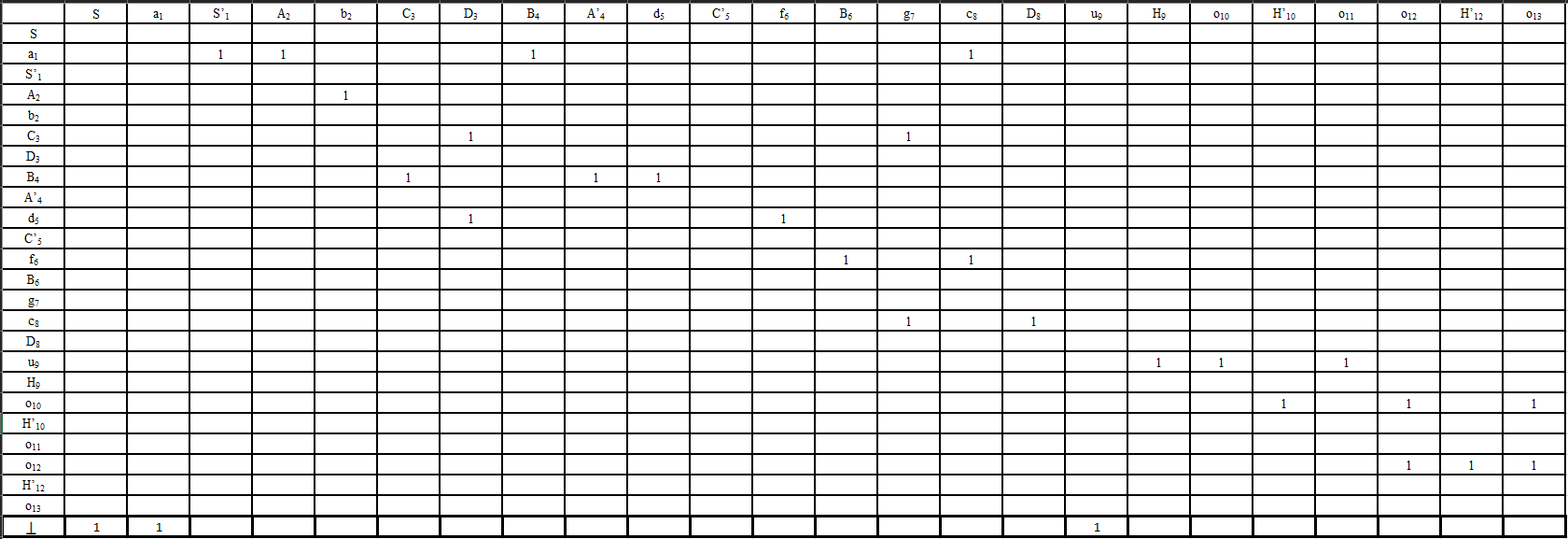


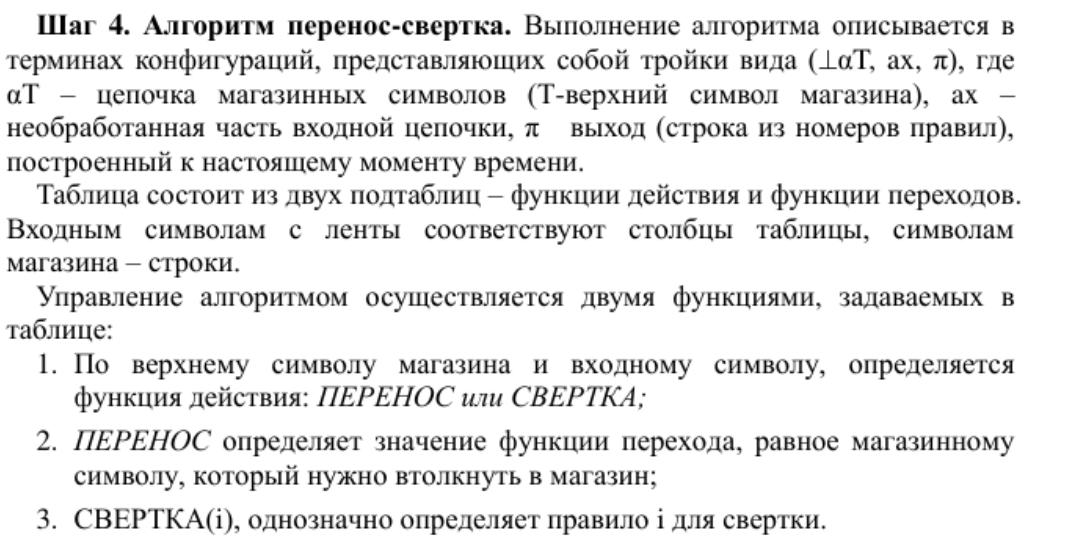
**Существует два способа построения LR(k) анализаторов:**

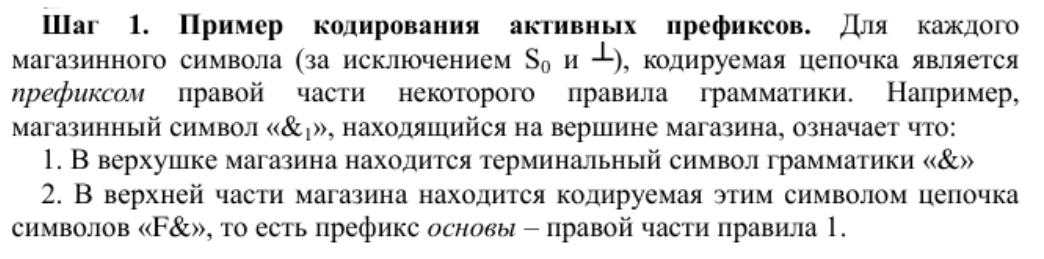
1. На основе активных префиксов (построения расширенного магазинного алфавита) и отношения OBLOW;
2. На основе LR(0)-ситуаций и функций CLOSURE и GOTO.

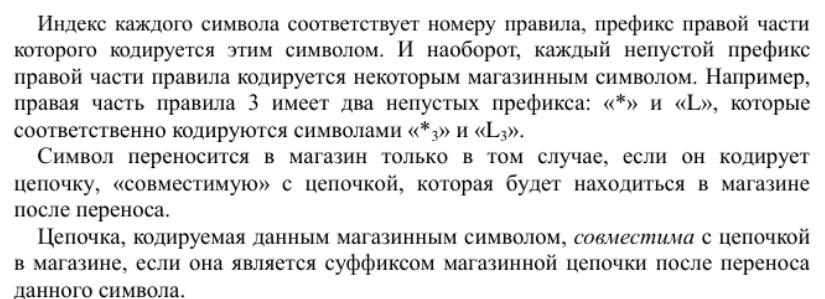
Построим двумя способами LR(k) анализатор для заданной грамматики:

, где P:







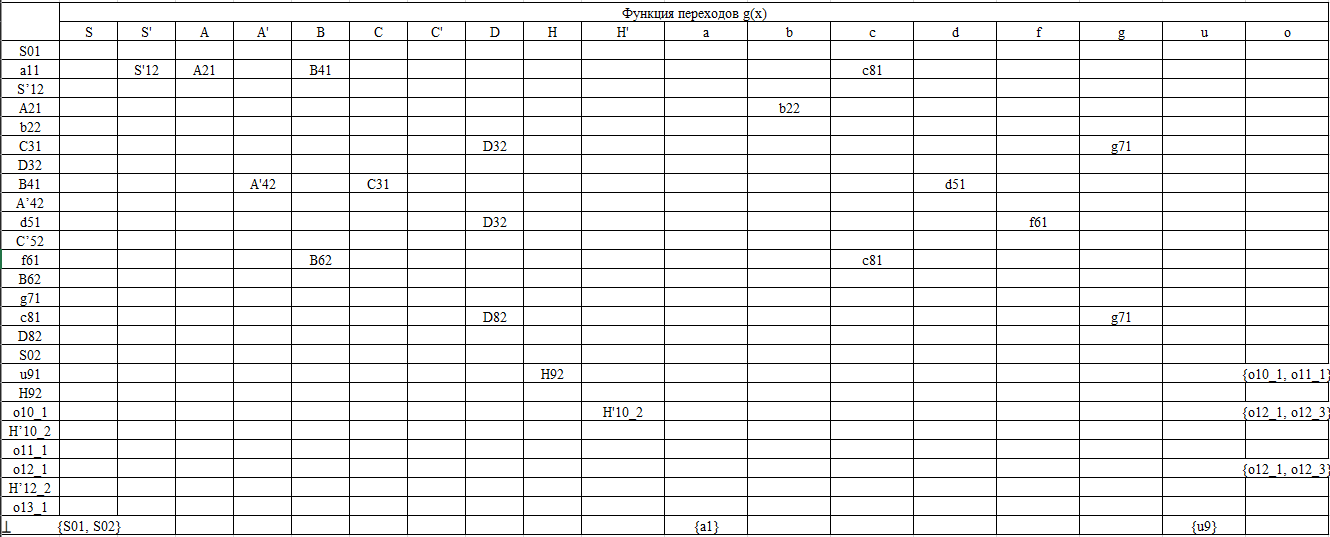


|  |  |  |  |
| --- | --- | --- | --- |
| Символ грамматики | Магазинный символ | Кодируемая цепочка | Операции |
| S0 =S | S0 | S | Д |
| S’ | S’1 | aS’1 | С1 |
| A’ | A’4 | BA’ | C4 |
| A | A2 | A | П |
| С | С3 | C | П |
| С’ | C’5 | dC’5 | С5 |
| D | D8  D3 | cD  CD | C8  C3 |
| B | B4 | B | П |
| H | H9 | uH | C9 |
| H’ | H10  H12 | oH’  oH’ | C10  C12 |
| a | a1 | a | П |
| b | b2 | Ab | C2 |
| c | c8 | c | П |
| d | d5 | d | П |
| f | f6 | f | П |
| g | g7 | g | С7 |
| u | u9 | u | П |
| o | o10  o11  o12  o13 | o  o  o  o | П  С11  П  С13 |

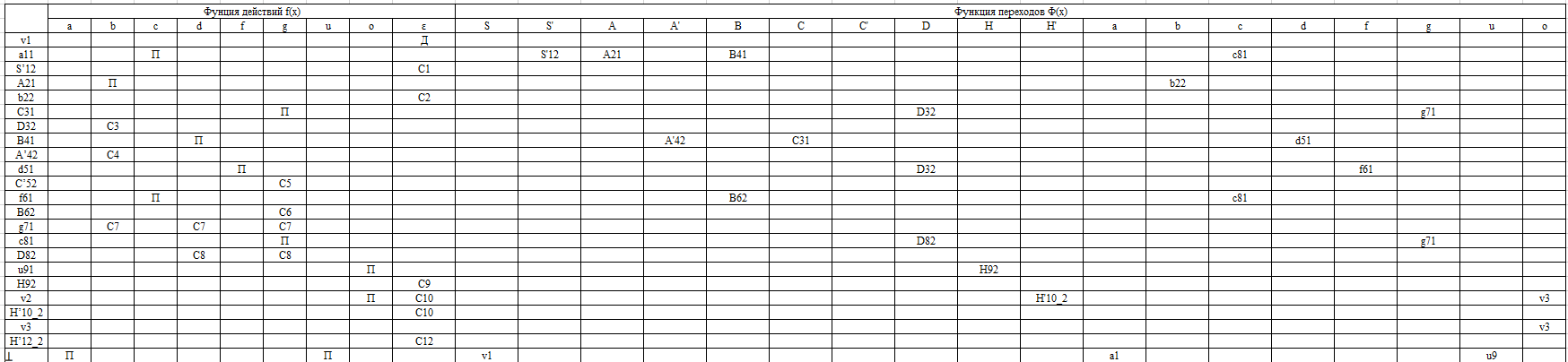
**Основа** – кодируемая цепочка символов в верхней части магазина.

Магазинный алфавит построен таким образом, что для каждого магазинного символа (за исключением S0 и ⊥), кодируемая им цепочка является префиксом правой части некоторого правила грамматики.

Построим матрицу функции g(x):



Построим матрицу функции f(x) и объединим с g(x)



Программа:

public void Example\_LR1(){

grammar.AddRule("S", new List<Symbol> { new Symbol("a"), new Symbol("S'") });

grammar.AddRule("S'", new List<Symbol> { new Symbol("A"), new Symbol("b") });

grammar.AddRule("A'", new List<Symbol> { new Symbol("C"), new Symbol("D") });

grammar.AddRule("A", new List<Symbol> { new Symbol("B"), new Symbol("A'") });

grammar.AddRule("C", new List<Symbol> { new Symbol("d"), new Symbol("C'") });

grammar.AddRule("C'", new List<Symbol> { new Symbol("f"), new Symbol("B") });

grammar.AddRule("D", new List<Symbol> { new Symbol("g") });

grammar.AddRule("B", new List<Symbol> { new Symbol("c"), new Symbol("D") });

grammar.AddRule("S", new List<Symbol> { new Symbol("u"), new Symbol("H") });

grammar.AddRule("H", new List<Symbol> { new Symbol("o"), new Symbol("H'") });

grammar.AddRule("H", new List<Symbol> { new Symbol("o") });

grammar.AddRule("H'", new List<Symbol> { new Symbol("o"), new Symbol("H'") });

grammar.AddRule("H'", new List<Symbol> { new Symbol("o")});

grammar.T.AddRange(new List<Symbol>{

new Symbol("a"), new Symbol("b"), new Symbol("c"),

new Symbol("d"), new Symbol("f"), new Symbol("g"),

new Symbol("u"), new Symbol("o")

});

grammar.V.AddRange(new List<Symbol>{

new Symbol("S"), new Symbol("S'"), new Symbol("A"), new Symbol("A'"),

new Symbol("B"), new Symbol("C"), new Symbol("C'"), new Symbol("D"),

new Symbol("H'"), new Symbol("H")

});

Пример вывода программы:

КС - грамматика :

Алфавит нетерминальных символов: SS'AA'BCC'DH'H

Алфавит терминальных символов: abcdfguo

Правила:

(1) S -> aS'

(2) S' -> Ab

(3) A' -> CD

(4) A -> BA'

(5) C -> dC'

(6) C' -> fB

(7) D -> g

(8) B -> cD

(9) S -> uH

(10) H -> oH'

(11) H -> o

(12) H' -> oH'

(13) H' -> o

Начальный нетерминал: S

Для продолжения нажмите <Enter>

Delete e-rules:

Executing:

e-rules:

NoShortNoTerms :

V1: : S S' A A' B C C' D H' H

e-rules have benn deleted!

Пополненная грамматика:

КС - грамматика :

Алфавит нетерминальных символов: SS'AA'BCC'DH'HП

Алфавит терминальных символов: abcdfguo$

Правила:

(0) П -> S

(1) S -> aS'

(2) S' -> Ab

(3) A' -> CD

(4) A -> BA'

(5) C -> dC'

(6) C' -> fB

(7) D -> g

(8) B -> cD

(9) S -> uH

(10) H -> oH'

(11) H -> o

(12) H' -> oH'

(13) H' -> o

Начальный нетерминал: S

Для продолжения нажмите <Enter>

Полученная матрица отношения OBLOW:

S01 a11S'12 A21 b22 C31 D32 B41A'42 d51C'52 f61 B62 g71 c81 D82 u91 H92o101H'102o111o121H'122o131

^ 1 1 1

S01

a11 1 1 1 1

S'12

A21 1

b22

C31 1 1

D32

B41 1 1 1

A'42

d51 1 1

C'52

f61 1 1

B62

g71

c81 1 1

D82

u91 1 1 1

H92

o101 1 1 1

H'102

o111

o121 1 1 1

H'122

o131

Для продолжения нажмите <Enter>

Полученная матрица для функции переходов g(X):

a b c d f g u o S S' A A' B C C' D H' H

^00 a11 u91 S01

a11 c81 S'12 A21 B41

u91 o101o111

H92

S01

c81 g71 D82

S'12

A21 b22

B41 d51 A'42 C31

b22

C31 g71

D32

g71

D32

d51 f61 C'52

A'42

f61 c81 B62

C'52

B62

D82

o101o111 o121o131

H'102

H92

H'102

H'122

Для продолжения нажмите <Enter>

Полученная матрицы для функции действий f(X):

a b c d f g u o $

^00 ПЕРЕНОС ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ПЕРЕНОС ОШИБКА ОШИБКА

a11 ОШИБКА ОШИБКА ПЕРЕНОС ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА

u91 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ПЕРЕНОС ОШИБКА

S01 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ДОПУСК

c81 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ПЕРЕНОС ОШИБКА ОШИБКА ОШИБКА

S'12 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА СВЕРТКА 1

A21 ОШИБКА ПЕРЕНОС ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА

B41 ОШИБКА ОШИБКА ОШИБКА ПЕРЕНОС ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА

b22 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА СВЕРТКА 2

C31 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ПЕРЕНОС ОШИБКА ОШИБКА ОШИБКА

g71 ОШИБКА СВЕРТКА 7 ОШИБКА СВЕРТКА 7 ОШИБКА СВЕРТКА 7 ОШИБКА ОШИБКА ОШИБКА

D32 ОШИБКА СВЕРТКА 3 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА

d51 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ПЕРЕНОС ОШИБКА ОШИБКА ОШИБКА ОШИБКА

A'42 ОШИБКА СВЕРТКА 4 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА

f61 ОШИБКА ОШИБКА ПЕРЕНОС ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА

C'52 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА СВЕРТКА 5 ОШИБКА ОШИБКА ОШИБКА

B62 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА СВЕРТКА 6 ОШИБКА ОШИБКА ОШИБКА

D82 ОШИБКА ОШИБКА ОШИБКА СВЕРТКА 8 ОШИБКА СВЕРТКА 8 ОШИБКА ОШИБКА ОШИБКА

o101o111 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ПЕРЕНОССВЕРТКА 10

H92 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА СВЕРТКА 9

H'102 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКАСВЕРТКА 10

H'122 ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКА ОШИБКАСВЕРТКА 12

Для продолжения нажмите <Enter>

Введите цепочку: acgdfcggb

Входная цепочка acgdfcggb$ распознана.

Результат правого вывода: 7 8 7 8 6 5 7 3 4 2 1

Вы хотите продолжить (да/нет)? – нет

Примеры разбора цепочки:

(⊥,acgdfcggb⊥,ɛ) ⸠ (⊥a,cgdfcggb⊥,ɛ) ⸠ (⊥ac, gdfcggb⊥,ɛ) ⸠ (⊥acg, dfcggb⊥,ɛ) ⸠ (⊥acD, dfcggb⊥,7) ⸠ (⊥acD, dfcggb⊥,7) ⸠ (⊥aB, dfcggb⊥,78) ⸠ (⊥aBd, fcggb⊥,78) ⸠ (⊥aBdf, cggb⊥,78) ⸠ (⊥aBdfc, ggb⊥,78) ⸠ (⊥aBdfc, ggb⊥,78) ⸠ (⊥aBdfcg, gb⊥,78) ⸠ (⊥aBdfcD, gb⊥,787) ⸠ (⊥aBdfB, gb⊥,7878) ⸠ (⊥aBdC’, gb⊥,78786) ⸠ (⊥aBC, gb⊥,787865) ⸠ (⊥aBCg, b⊥,787865) ⸠ (⊥aBCD, b⊥,7878657) ⸠ (⊥aBA’, b⊥,78786573) ⸠ (⊥aA, b⊥,787865734) ⸠ (⊥aAb, ⊥,787865734) ⸠ (⊥aS’, ⊥,7878657342) ⸠ (⊥aS’, ⊥,7878657342) ⸠ (⊥S, ⊥,78786573421) ⸠ ДОПУСК

Диаграмма

