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 КС-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 спроектировать 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: Старцев И. Р.
Группа: М8О-201Б-21
Руководитель: Киндинова В. В.

Оценка:
Дата:

Москва. 2023

Практическая работа №1 (1-3 лаб.)

Лабораторные работы №1-2

Формулировка задания:

Спроектировать грамматику для трёх заданных паттернов. Составить на основе разработанных регулярных грамматик конечные автоматы, распознающие эквивалентные им языки.

Спроектируем грамматику для заданного языка:

1. pattern = "192\.168\.1\.\d{1,3}"

Автоматная грамматика:

L(pattern) = L("192\.168\.1\.\d{1,3}") = { 192.168.1.{0...9},
192.168.1.({0...9})^2, 192.168.1.({0...9})^3 }

G(T, V, P, S_0) = G({0, 1, ..., 9, .}, {**S_0, A, B, ..., M**}, {p_1, p_2, ..., p_{14}}, **S_0**)

Правила регулярной грамматики:

p_1: S_0 → 1A

p_2: A → 9B

p_3: B → 2C

p_4: C → .D

p_5: D → 1E

p_6: E → 6F

p_7: F → 8G

p_8: G → .H

p_9: H → 1I

p_{10}: I → .J

p_{11}: J → 0K | 1K | ... | 9K

p_{12}: K → L | 0L | 1L | ... | 9L

p_{13}: L → M | 0M | 1M | ... | 9M

p_{14}: M → ε

Пример цепочек:

S_0 =>^{1} 1**A** =>^{2} 19**B** =>^{3} 192**C** =>^{4} 192.**D** =>^{5} 192.1**E** =>^{6} 192.16**F** =>^{7} 192.168**G** =>^{8} 192.168.**H** =>^{9} 192.168.1**I** =>^{10} 192.168.1.**J** =>^{11} 192.168.1.1**K** =>^{12} 192.168.1.1**L** =>^{13} 192.168.1.1M =>^{14} 192.168.1.1

S_0 =>^{1} 1**A** =>^{2} 19**B** =>^{3} 192**C** =>^{4} 192.**D** =>^{5} 192.1**E** =>^{6} 192.16**F** =>^{7} 192.168**G** =>^{8} 192.168.**H** =>^{9} 192.168.1**I** =>^{10} 192.168.1.**J** =>^{11} 192.168.1.5**K** =>^{12} 192.168.1.51**L** =>^{13} 192.168.1.512M =>^{14} 192.168.512

S_0 =>^{1} 1**A** =>^{2} 19**B** =>^{3} 192**C** =>^{4} 192.**D** =>^{5} 192.1**E** =>^{6} 192.16**F** =>^{7} 192.168**G** =>^{8} 192.168.**H** =>^{9} 192.168.1**I** =>^{10} 192.168.1.**J** =>^{11} 192.168.1.0**K** =>^{12} 192.168.1.04**L** =>^{13} 192.168.1.04M =>^{14} 192.168.04

Конечный автомат:

L(KA) = L(G)

KA = (**Q**, **Σ**, **δ**, S_0, **F**), где

Q = { S_0, **A**, **B**, ..., **M**, q_f }, **Σ** = { 0, 1, ..., 9, .}, S_0 = S_0, **F** = q_f,

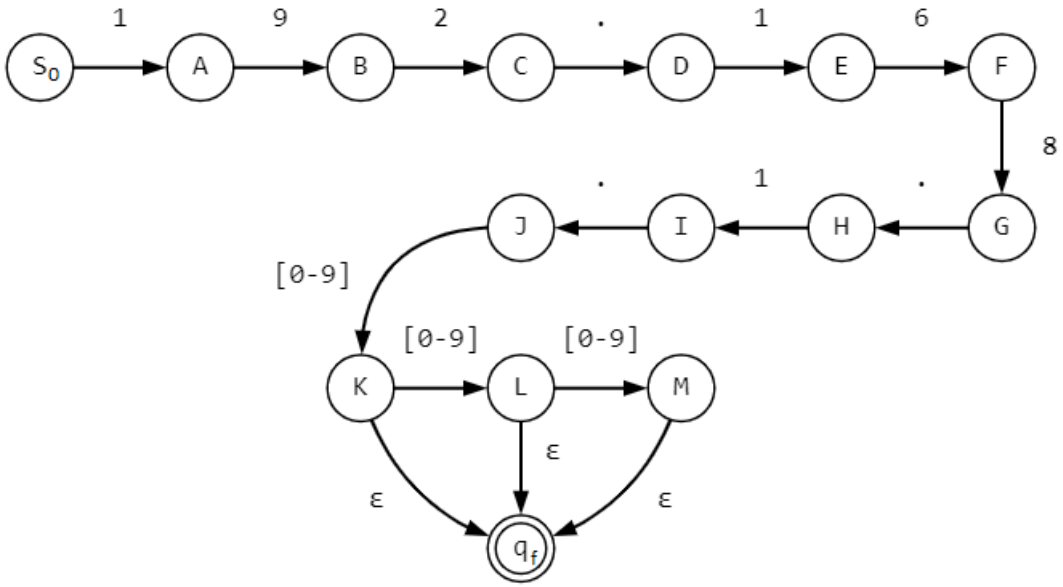
δ = { 1. δ(S_0, 1) = {**A**},
2. δ(**A**, 9) = {**B**},
3. δ(**B**, 2) = {**C**},
4. δ(**C**, .) = {**D**},
5. δ(**D**, 1) = {**E**},
6. δ(**E**, 6) = {**F**},
7. δ(**F**, 8) = {**G**},
8. δ(**G**, .) = {**H**},
9. δ(**H**, 1) = {**I**},
10. δ(**I**, .) = {**J**},
11. δ(**J**, 0) = {**K**},
12. δ(**J**, 1) = {**K**},
...
20. δ(**J**, 9) = {**K**},
21. δ(**K**, ε) = {**L**},
22. δ(**K**, 0) = {**L**},
23. δ(**K**, 1) = {**L**},
...
31. δ(**K**, 9) = {**L**},
32. δ(**L**, ε) = {**M**},
33. δ(**L**, 0) = {**M**},
34. δ(**L**, 1) = {**M**},
...
42. δ(**L**, 9) = {**M**},
43. δ(**M**, ε) = {q_f},
}

Примеры конфигурации КА:

1. (**S_0**, 192.168.1.512) $\vdash^1$ (**A**, 92.168.1.512) $\vdash^2$ (**B**, 2.168.1.512) $\vdash^3$ (**C**, .168.1.512) $\vdash^4$ (**D**, 168.1.512) $\vdash^5$ (**E**, 68.1.512) $\vdash^6$ (**F**, 8.1.512) $\vdash^7$ (**G**, .1.512) $\vdash^8$ (**H**, 1.512) $\vdash^9$ (**I**, .512) $\vdash^{10}$ (**J**, 512) $\vdash^{15}$ (**K**, 12) $\vdash^{23}$ (**L**, 2) $\vdash^{35}$ (**M**, ε) $\vdash^{43}$ (**q_f**, ε)

2. (**S_0**, 192.168.1.04) $\vdash^1$ (**A**, 92.168.1.04) $\vdash^2$ (**B**, 2.168.1.04) $\vdash^3$ (**C**, .168.1.04) $\vdash^4$ (**D**, 168.1.04) $\vdash^5$ (**E**, 68.1.04) $\vdash^6$ (**F**, 8.1.04) $\vdash^7$ (**G**, .1.04) $\vdash^8$ (**H**, 1.04) $\vdash^9$ (**I**, .04) $\vdash^{10}$ (**J**, 04) $\vdash^{11}$ (**K**, 4) $\vdash^{26}$ (**L**, ε) $\vdash^{32}$ (**M**, ε) $\vdash^{43}$ (**q_f**, ε)

3. (**S_0**, 192.168.1.1) ⊦[1] (**A**, 92.168.1.1) ⊦[2] (**B**, 2.168.1.1) ⊦[3] (**C**, .168.1.1) ⊦[4] (**D**, 168.1.1) ⊦[5] (**E**, 68.1.1) ⊦[6] (**F**, 8.1.1) ⊦[7] (**G**, .1.1) ⊦[8] (**H**, 1.1) ⊦[9] (**I**, .1) ⊦[11] (**J**, 1) ⊦[11] (**K**, ε) ⊦[21] (**L**, ε) ⊦[32] (K, ε) ⊦[43] (**q_f**, ε)



Лемма о накачке:

```
Введите строчку для проверки: 192.168.1.112
Проверить цепочку:
1 - на принадлежность регулярному языку
2 - на принадлежность кс-языку
3 - найти все повторения
Введите 1 или 2 или 3: 1

Для цепочки: 192.168.1.112
192.168.1. 1^2 2
Цепочка принадлежит регулярному языку
Вывести все повторения? (1 - да, 0 - нет): 1

192.168 .1^2 12
192.168.1. 1^2 2
```

2. (\W|^)stock\stips(\W|$)

Автоматная грамматика:

L(pattern) = L("(\W|^)stock\s{0,3}tips(\W|$)") = { stock\stips }

G(T, V, P, S0) = G({0, 1, ..., 9, .}, {S0, A, B, ..., L},

{p1, p2, ..., p12}, S0)

Правила регулярной грамматики:

p1: S0 → sA' | #Q

p2: Q → sA'

p3: A' → tB'

p4: B' → oC'

p5: C' → cD'

p6: D' → kG'

p7: G' → \sF

p8: F → tH'

p9: H' → iI'

p10: I' → pM'

p11: M' → s | sW

p12: W → #

Пример цепочек:

S0 → sA' → stB' → stoC' → stocD' → stockG' → stock F → stock tH' →

stock tiI'→ stock tipM' → stock tips

Конечный автомат:

L(KA) = L(G)

KA = (Q, Σ, δ, S0, F), где

Q = {s, t, o, c, k, i, p, \s, q, #}

Σ = {S0, A', B', C', D', E', F', G', H', M', Q}

S0 = S0

δ = {

1. δ(S0, s) = {A'},

2. δ(A', t) = {B'},

...

5. δ(D', k) = {E'},

6. δ(E', \s) = {F'},

7. δ(F', t) = {G'},

8. δ(G', i) = {H'},

9. δ(H', p) = {M'},

10. δ(M', s) = {q},

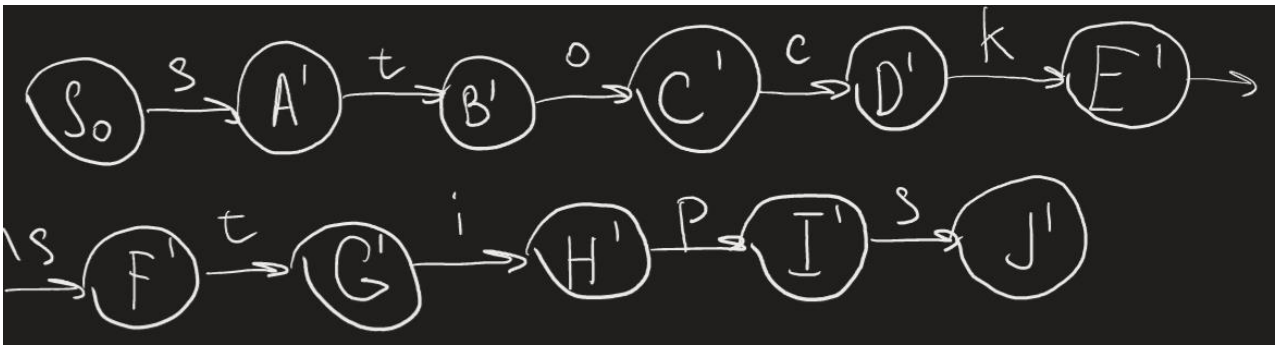
11. δ(Q, #) = {A'}

12. δ(W, #) = {q}

}

*Примеры конфигурации КА:*stock\stips → A'tock\stips → ... → G'\stips → F\stips → H'ips → ... → M's → qfstock\stips → A'tock\stips → ... → G'\stips →

F3\stips → F2tips → H'ips → ... → M's → qfstock\stips → A'tock\stips → ... → G'\stips → F3tips → H'ips → ... → M's → qfstocktips → A'tocktips → ... → G'tips → H'ips → ... → M's → qf



Практическая работа №2 (лабораторные 4-8)

Лабораторная работа №4:

Формулировка задания:

Устранить из КС-грамматики бесполезные символы и ε–правила

Исходная грамматика (вариант 22):

G = {

(a, b, c, d, f, epsilon, r, z, o),

(S, A, B, C, D, M, W, T, J, K),

(S → aAB, A → bC, B → cA, D → M, M → f, C → dDW, W → epsilon, T → r, J → z, B → aK, K → Ko, K → o),

S

}

p_1: S → aAB

p_2: A → bC

p_3: B → cA

p_4: D → M

p_5: M → f

p_6: C → dDW

p_7: W → epsilon

p_8: T → r

p_9: J → z

p_{10}: B → aK

p_{11}: K → Ko

p_{12}: K → o

Устранить из КС-грамматики бесполезные символы.

Устранение непроизводящих символов.

Символ недостижимый, если он не участвует ни в одной цепочке вывода из начального символа грамматики.

Нетерминальный символ A ∈ V называется производящим, если из него можно вывести терминальную цепочку (последовательность терминальных символов, которые могут быть выводимыми в грамматике).

1. V^1_p = {M, W, T, J, K} – для этих нетерминалов нашлось хотя бы одно правило, правая часть которого не содержит нетерминалов

2. V^2_p = {M, W, T, J, K, D, B} – если найдено такое правило, что все нетерминалы, стоящие в его правой части уже занесены в V_p, то добавить в V_p нетерминал, стоящий в его левой части

3. V^3_p = {M, W, T, J, K, D, B, C}

4. V^4_p = {M, W, T, J, K, D, B, C, A}

5. V^5_p = {M, W, T, J, K, D, B, C, A, S}

6. V_p = {M, W, T, J, K, D, B, C, A, S} содержит все производящие нетерминалы грамматики, а все нетерминалы, не попавшие в него, являются непроизводящими ($V - V_p = \emptyset$).

1.Добавлять в P' только правила в правой части которых только производящие символы

P' = {p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}}

p_1: S → aAB

p_2: A → bC

p_3: B → cA

p_4: D → M

p_5: M → f

p_6: C → dDW

p_7: W → epsilon

p_8: T → r

p_9: J → z

p_{10}: B → aK

p_{11}: K → Ko

p_{12}: K → o

G_1 = ({a, b, c, d, f, epsilon, r, z, o}, {S, A, B, C, D, M, W, T, J, K}, P', S)

Устранение недостижимых символов (VT_r – множество недостижимых символов):

Шаг1. Построить множество VT^i_r - достижимых терминалов и не терминалов.

VT^1_r = {S}

VT^2_r = {S, a, A, B}

VT^3_r = {S, a, A, B, b, C, c, K}

VT^4_r = {S, a, A, B, b, C, c, K, o, d, D, W}

VT^5_r = {S, a, A, B, b, C, c, K, o, d, D, W, M, epsilon }

VT^6_r = {S, a, A, B, b, C, c, K, o, d, D, W, M, epsilon, f}

Так, не входящие в VT_r символы {T, J, r, z} недостижимы

Добавим в P'' только правила, состоящие из достижимых символов:

P'' = {p_1, p_2, p_3, p_4, p_5, p_6, p_9, p_{10}, p_{11}, p_{12}}:

p_1: S → aAB

p_2: A → bC

p_3: B → cA

p_4: D → M

p_5: M → f

p_6: C → dDW

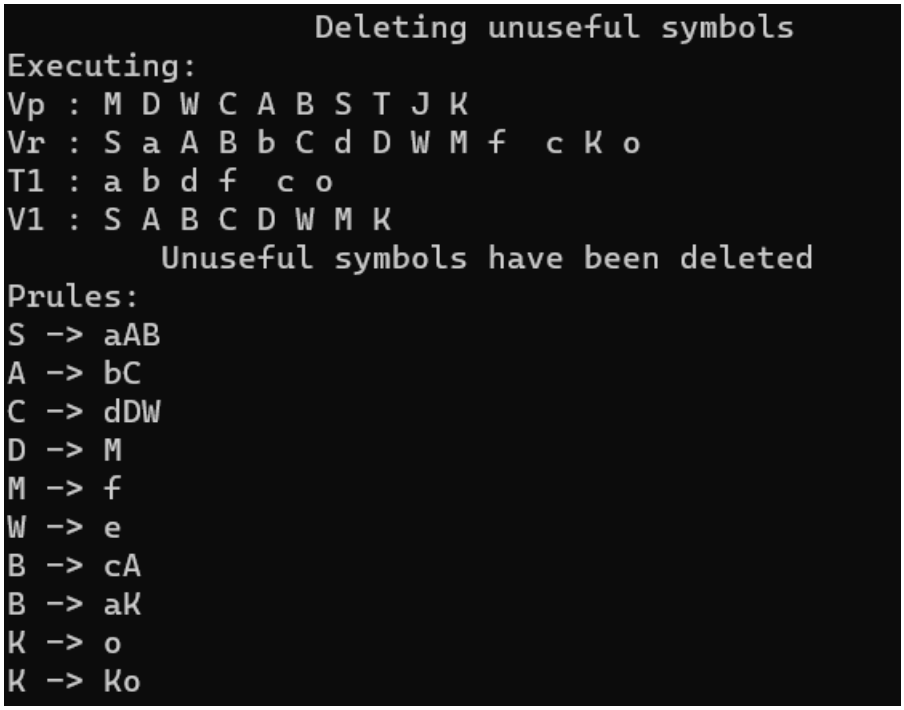
p_7: W → epsilon

p_8: B → aK

p_9: K → Ko

p_{10}: K → o

Получаем G' = ({a, b, c, d, f, epsilon, o}, {S, A, B, C, D, M, W, K}, P'', S)

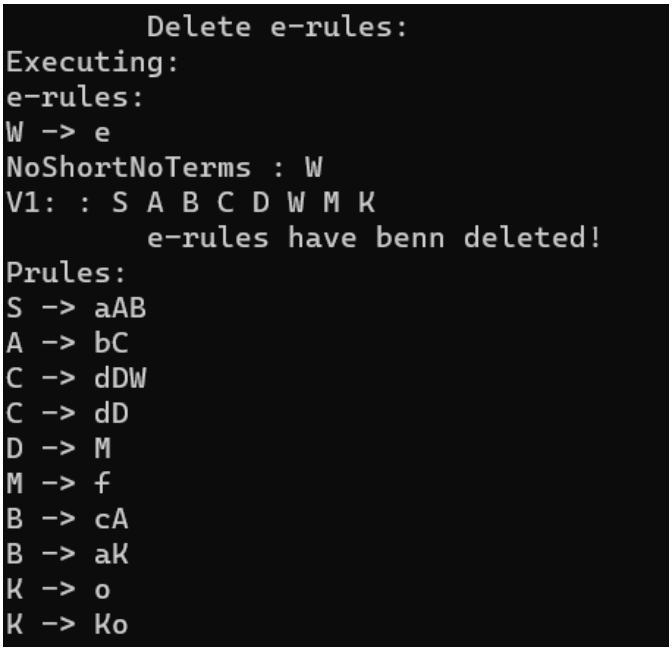


Устранить из КС-грамматики ε-правила.

Составим G" на основе алгоритма удаления эпсилон правил

G' = ({a, b, c, d, f, epsilon, o}, {S, A, B, C, D, M, W, K}, P, S), где P = {S → aAB, A → bC, B → cA, D → M, M → f, C → dDW, W → epsilon, B → aK, K → Ko, K → o}

G" = ({a, b, c, d, f, epsilon, o}, {S, A, B, C, D, M, W, K}, P, S), где P' = {S → aAB, A → bC, B → cA, D → M, M → f, C → dDW, C → dD, B → aK, K → Ko, K → o}



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

Устранить из КС-грамматики цепные правила.

Цепные правила:

D → M, M → f

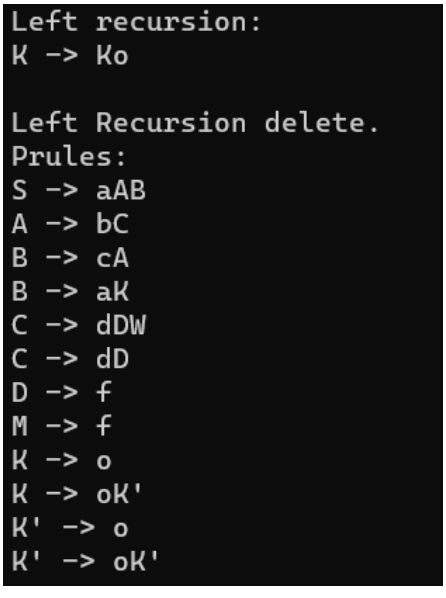
V^1_x = {M}

Правила D → M, M → f поменяем на D → f

Устранить из КС-грамматики левую рекурсию.

Левая рекурсия: K → Ko

Заменим правило K → Ko, на правила: K → o | oK’, K’ → o | oK’



Приведенная грамматика:

```
Normal Grammatic:
T : a b d f c o
V : S A B C D K K'
Prules:
S -> aAB
A -> bC
C -> dD
D -> f
B -> cA
B -> aK
K -> o
K -> oK'
K' -> o
K' -> oK'
Start symbol: S
```

Лабораторная работа №6

Формулировка задания:

Определить форму КС-грамматики и сделать ее приведение

Определение 7. КС грамматика G = (T, V, P, S) называется грамматикой в нормальной форме Грейбах, если в ней нет ε-правил, т.е. правил вида A→ ε, и каждое правило из P отличное от S → ε, имеет вид A → aα, где a ∈T, α⊂ V^*.

Также полезно представлять грамматику в нормальной форме Хомского, что позволяет упростить рассмотрение ее свойств.

Определение 8. КС грамматика G = (T, V, P, S) называется грамматикой в нормальной форме Хомского, если каждое правло из P имеет один из следующих видов:

1. A→ BC, где A, B, C∈V;
2. A→ a, где a∈T;
3. S→ ε, если ε∈L(G), причем S не встречается в правых частях правил.

Приведенная грамматика не является грамматикой в нормальной форме Хомского, так как присутствует правило: **S** → a**AB.**

Приведенная грамматика является грамматикой в нормальной форме Грейбах, так как в ней нет – epsilon-правил, и каждое правило имеет вид **A** → aα, где a∈T, α ⊂ V^*.

Лабораторная работа №7

Формулировка задания: спроектировать МП-автомат для приведённой КС-грамматики.

Так, МП = (Q, Σ, Г, δ, q_0, z_0, F) – это семерка объектов, где

Q – конечное множество состояний устройства управления;
$\sum$ - конечный алфавит входных символов;
Γ - конечный алфавит магазинных символов;
δ - функция переходов, отображает множества $Q\times(\sum \cup \{\varepsilon\})\times\Gamma$ в множество конечных подмножеств множества $Q\times\Gamma^*$, δ: $Q\times(\sum \cup \{\varepsilon\})\times\Gamma \to Q\times\Gamma^*$
q_0 - начальное состояние, $q_0 \in Q$;
z_0 - начальный символ магазина, $z_0 \in \Gamma$;
F - множество заключительных состояний, $F \subseteq Q$.

Приведённая грамматика:

G = (**T**, **V**, **P**, **S**), где

{

T = { a, b, c, d, f, o },

V = { S, A, B, C, D, K, K`},

S_0 = S

P:

p_1: S → aAB

p_2: A → bC

p_3: C → dD

p_4: D → f

p_5: B → cA

p_6: B → aK

p_7: K → o

p_8: K → oK`

p_9: K` → o

p_{10}: K` → oK`

}

Вывод цепочки:

$S_0 =>^1$ aAB $=>^2$ abCB $=>^3$ abdDB $=>^4$ abdfB $=>^5$ abdfcA $=>^2$ abdfcbC $=>^3$ abdfcbdD $=>^4$ abdfcbdf

$S_0 =>^1$ aAB $=>^2$ abCB $=>^3$ abdDB $=>^4$ abdfB $=>^6$ abdfaK $=>^8$ abdfaoK` $=>^{10}$ abdfaooK` $=>^9$ abdfaooo

L(МП) = L(G)

МП = (**Q**, **Σ**, **Γ**, **δ**, **q_0**, **z_0**, **F**):

Q = {q}, **Σ** = T, **Γ** = T ∪ V, **δ** = **δ**, **q_0** = **q_0**, **z_0** = **S_0**, **F** = {q}

МП = (

{q},

{ a, b, c, d, f, o },

{ a, b, c, d, f, o, S, A, B, C, D, K, K` },

δ,

q_0,

S,

{q}

)

Строим функции переходов:

p_1: S → aAB	1: $\delta(q_0, \varepsilon, S) = (q, aAB)$
p_2: A → bC	2: $\delta(q, \varepsilon, A) = (q, bC)$
p_3: C → dD	3: $\delta(q, \varepsilon, C) = (q, dD)$
p_4: D → f	4: $\delta(q, \varepsilon, D) = (q, f)$
p_5: B → cA	5: $\delta(q, \varepsilon, B) = (q, cA)$
p_6: B → aK	6: $\delta(q, \varepsilon, B) = (q, aK)$
p_7: K → o	7: $\delta(q, \varepsilon, K) = (q, o)$
p_8: K → oK`	8: $\delta(q, \varepsilon, K) = (q, oK`)$
p_9: K` → o	9: $\delta(q, \varepsilon, K`) = (q, o)$
p_{10}: K` → oK`	10: $\delta(q, \varepsilon, K`) = (q, oK`)$
	11: $\delta(q, a, a) = (q, \varepsilon)$, $a \in \Sigma$

Последовательность тактов МП-автомата abdfcbdf:

$(q_0, abdfcbdf, S) \vdash^1 (q, abdfcbdf, aAB) \vdash^{11} (q, bdfcbdf, AB) \vdash^2 (q, bdfcbdf, bCB) \vdash^{11} (q, dfcbdf, CB) \vdash^3 (q, dfcbdf, dDB) \vdash^{11} (q, fcbdf, DB) \vdash^4 (q, fcbdf, fB) \vdash^{11} (q, cbdf, B) \vdash^5 (q, cbdf, cA) \vdash^{11} (q, bdf, A) \vdash^2 (q, bdf, bC) \vdash^{11} (q, df, C) \vdash^3 (q, df, dD) \vdash^{11} (q, f, D) \vdash^4 (q, f, f) \vdash^{11} (q, \varepsilon, \varepsilon)$

Лабораторная работа №8

Формулировка задания:

Реализовать спроектированный МП-автомат для приведённой КС-грамматики.

Код программы:

```
        case "7": {
              var CFGrammar = new Grammar(new List<Symbol>() { "a", "b", "c", "d", "f", "o" },
                                          new List<Symbol>() { "S","A","B","C", "D", "K", "K`"
},
                                          "S");

         CFGrammar.AddRule("S",new List<Symbol>() { "a", "A", "B" });
         CFGrammar.AddRule("A",new List<Symbol>() { "b","C" });
         CFGrammar.AddRule("C",new List<Symbol>() { "d","D" });
         CFGrammar.AddRule("D",new List<Symbol>() { "f" });
         CFGrammar.AddRule("B",new List<Symbol>() { "c", "A" });
         CFGrammar.AddRule("B",new List<Symbol>() { "a","K" });
         CFGrammar.AddRule("K",new List<Symbol>() { "o" });
         CFGrammar.AddRule("K",new List<Symbol>() { "o", "K`" });
         CFGrammar.AddRule("K`",new List<Symbol>() { "o" });
         CFGrammar.AddRule("K`",new List<Symbol>() { "o", "K`" });

         Console.Write("Debug KC-Grammar ");
         CFGrammar.DebugPrules();

         var pda = new PDA(CFGrammar);
         pda.Debug();

         Console.WriteLine("\nEnter the line   :");
         Console.WriteLine(pda.Execute(Console.ReadLine()).ToString());
         break;
}
```

```
Debug KC-Grammar Prules:
S -> aAB
A -> bC
C -> dD
D -> f
B -> cA
B -> aK
K -> o
K -> oK`
K` -> o
K` -> oK`
Delta rules:
 delta(q,ε,S) -> (q,a,A,B)
 delta(q,ε,A) -> (q,b,C)
 delta(q,ε,C) -> (q,d,D)
 delta(q,ε,D) -> (q,f)
 delta(q,ε,B) -> (q,c,A)
 delta(q,ε,B) -> (q,a,K)
 delta(q,ε,K) -> (q,o)
 delta(q,ε,K) -> (q,o,K`)
 delta(q,ε,K`) -> (q,o)
 delta(q,ε,K`) -> (q,o,K`)
 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,o,o) -> (q,ε)
 delta(q0,ε,z0) -> (qf,ε)
```

```
Enter the line   :
abdfcf
 step 1
 delta(q,ε,S) -> (q,a,A,B)
 step 1
 delta(q,a,a) -> (q,ε)
 step 2 a  a
 step 3 a
 step 1
 delta(q,ε,A) -> (q,b,C)
 step 1
 delta(q,b,b) -> (q,ε)
 step 2 b  b
 step 3 b
 step 1
 delta(q,ε,C) -> (q,d,D)
 step 1
 delta(q,d,d) -> (q,ε)
 step 2 d  d
 step 3 d
 step 1
 delta(q,ε,D) -> (q,f)
 step 1
 delta(q,f,f) -> (q,ε)
 step 2 f  f
 step 3 f
 step 1
 delta(q,ε,B) -> (q,c,A)
 step 1
 delta(q,c,c) -> (q,ε)
 step 2 c  c
 step 3 c
 step 1
 delta(q,ε,A) -> (q,C,b)
 step 1
 delta(q,ε,C) -> (q,D,d)
 step 1
 delta(q,ε,D) -> (q,f)
 step 1
 delta(q,f,f) -> (q,ε)
 step 2 f  f
 step 3 f
True
```

Практическая работа №3 (лабораторные 9-11)

Лабораторная работа №9:

Формулировка задания:

Для LL(k) анализатора построить управляющую таблицу M.

LL-грамматика (или грамматика с левой рекурсией) - это тип формальной грамматики, используемый в теории формальных языков и синтаксического анализа. Она названа так, потому что в процессе синтаксического анализа символы обрабатываются слева направо, и она обладает свойством левой рекурсии.

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.

FIRST(A) – это множество первых терминальных символов, которыми начинаются цепочки, выводимые из нетерминала $A \in V$:

$FIRST(A) = \{a \in T \mid A \Rightarrow^{+} a\beta$, где $\beta \in (T \cup V)^{*}\}$

Обобщим определение множества FIRST так, чтобы его можно было применить для правил произвольного вида. Множество FIRST(α) состоит из множества терминальных символов, которыми начинаются цепочки, выводимые из цепочки α.

$FIRST(\alpha) = \{a \in T \mid S \Rightarrow^{+} \alpha \Rightarrow^{+} a\beta$, где $\alpha \in (T \cup V)^{+}$, $\beta \in (T \cup V)^{*}\}$

FOLLOW(A) – это множество следующих терминальных символов, которые могут встретиться непосредственно справа от нетерминала в некоторой сентенциальной форме:

$FOLLOW(A) = \{a \in T \mid S \Rightarrow^{*} \alpha A \gamma$ и $a = FIRST(\gamma)\}$

Магазин содержит цепочку Xa⊥ (см. рис. 1.17), где Xa – цепочка магазинных символов (X - верхний символ магазина), а символ (⊥) – специальный символ, называемый *маркером дна* магазина. Если верхним символом магазина является *маркер дна*, то магазин пуст. Выходная лента содержит цепочку номеров правил π, представляющую собой текущее состояние левого разбора.

Исходная грамматика:

G = (**T**, **V**, **P**, **S**),

{

T = { a, b, c, d, f, o },

V = { S, A, B, C, D, K, K`},

$\mathbf{S_0}$ = S

P {

p_1: S → aAB

p_2: A → bC

p_3: C → dD

p_4: D → f

p_5: B → cA

p_6: B → aK

p_7: K → o

p_8: K → oK`

p_9: K` → o

p_{10}: K` → oK`

}

}

Можно привести грамматику к форме LL(1) путем факторизации. Левая факторизация предполагает переписывание правил производства, когда необходимо выбрать из двух альтернативных продукций для нетерминала A. Вместо того чтобы сделать выбор на основе уже прочитанных символов входного потока, мы можем отложить принятие решения до тех пор, пока не будет прочитано достаточно символов для правильного выбора.

Gф = (**T**, **V**, **P'**, **S**),

{

T = { a, b, c, d, f, o },

V = { S, A, B, C, D, K, K`, T, T` },

$\mathbf{S}_{0}$ = S

P'{

p_1: S → aAB

p_2: A → bC

p_3: C → dD

p_4: D → f

p_5: B → cA

p_6: B → aK

p_7: K → oT

p_8: T → K`

p_9: T → epsilon

p_{10}: K` → oT`

p_{11}: T` → K`

p_{12}: T` → epsilon

}

}

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(A, a) = (β, i) для всех a ≠ ε, принадлежащих множеству FIRST(β). Если ε ∈ FIRST(β), то M(A, b) = (β, i) для всех b ∈ FOLLOW(A).
2. M(a, a) = ВЫБРОС для всех a ∈ **T**.
3. M(⊥, 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**	**o**	**epsilon**
S	(aAB, 1)						
A		(bC, 2)					
B	(aK,6)		(cA, 5)				
C				(dD, 3)			
D					(f,4)		
K						(oT, 7)	
K'						(oT`, 10)	
T						(oT`, 10)	(epsilon, 9)
T`						(oT`, 10)	(epsilon, 11)
a	ВЫБРОС						
b		ВЫБРОС					
c			ВЫБРОС				
d				ВЫБРОС			
f					ВЫБРОС		
o						ВЫБРОС	
⊥							ДОПУСК

Пустые клетки в таблице означают ОШИБКУ.

Аналитичекое представление для таблицы M:

Правило грамматики	Множество	Значение M
p_1: S → aAB	FIRST(S) = {a}	M(S, a) = aAB, 1
p_2: A → bC	FIRST(A) = {b}	M(A, b) = bC, 2
p_3: C → dD	FIRST(C) = {d}	M(C, d) = dD, 3
p_4: D → f	FIRST(D) = {f}	M(D, f) = f, 4
p_5: B → cA	FIRST(B) = {c}	M(B, c) = cA, 5
p_6: B → aK	FIRST(B) = {a}	M(B, a) = aK, 6
p_7: K → oT	FIRST(K) = {o}	M(K, o) = oT, 7
p_8: T → K`	FIRST(T) = {o}	M(T, o) = K`, 8
p_9: T → epsilon	FOLLOW(T) = {epsilon}	M(T, epsilon) = epsilon, 9
p_{10}: K` → oT`	FIRST(K`) = {o}	M(K`, o) = oT`, 10
p_{11}: T` → K`	FIRST(T`) = {o}	M(T`, o) = K`, 11
p_{12}: T` → epsilon	FOLLOW(T`) = {epsilon}	M(T`, epsilon) = epsilon, 12

Лабораторная работа №10:

Формулировка задания:

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

Шаг 1. Алгоритм находится в начальной конфигурации ((abdfaoo), $S_0\perp$, epsilon), где S_0 = S

Значение управляющей таблицы M(S, a) = (aAB, 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.

Шаг 2. Получаем следующие конфигурации:

Текущая конфигурация	Значение M
(abdfaoo, S⊥, epsilon) ⊦	M(S, a) = aAB, 1
(abdfaoo, aAB⊥, 1) ⊦	M(a, a) = ВЫБРОС
(bdfaoo, AB⊥, 1) ⊦	M(A, b) = bC, 2
(bdfaoo, bCB⊥, 12) ⊦	M(b, b) = ВЫБРОС
(dfaoo, CB⊥, 12) ⊦	M(C, d) = dD, 3
(dfaoo, dDB⊥, 123) ⊦	M(d, d) = ВЫБРОС
(faoo, DB⊥, 123) ⊦	M(D, f) = f, 4
(faoo, fB⊥, 1234) ⊦	M(f, f) = ВЫБРОС
(aoo, B⊥, 1234) ⊦	M(B, a) = aK, 6
(aoo, aK⊥, 12346)	M(a, a) = ВЫБРОС
(oo, K⊥, 12346)	M(K, o) = oT, 7
(oo, oT⊥, 123467)	M(o, o) = ВЫБРОС
(o, T⊥, 123467)	M(T, o) = K`, 8
(o, K`⊥, 1234678)	M(K`, o) = oT`, 10
(o, oT`⊥, 123467810)	M(o, o) = ВЫБРОС
(epsilon, T`⊥, 123467810)	M(T`, epsilon) = epsilon, 12
(epsilon, ⊥, 12346781012)	M(epsilon, ⊥) = ДОПУСК

Лабораторная работа №11:

Формулировка задания:

Реализовать управляющую таблицу M Для LL(k) анализатора.

Код программы:

```
case "9.1": {
            var LL = new Grammar(new List<Symbol>() { "a","b","c","d","f", "o", " " },
                                 new List<Symbol>() { "S", "A", "B", "C", "D", "K", "K`",
"T", "T`" },
                                 "S");

            LL.AddRule("S",new List<Symbol>() { "a","A","B" });
            LL.AddRule("A", new List<Symbol>() { "b", "C" });
            LL.AddRule("C",new List<Symbol>() { "d", "D" });
            LL.AddRule("D", new List<Symbol>() { "f" });
            LL.AddRule("B", new List<Symbol>() { "c", "A" });
            LL.AddRule("B", new List<Symbol>() { "a", "K" });
            LL.AddRule("K", new List<Symbol>() { "o", "T" });
            LL.AddRule("T", new List<Symbol>() { "K`" });
            LL.AddRule("T", new List<Symbol>() { " " });
            LL.AddRule("K`", new List<Symbol>() { "o", "T`" });
            LL.AddRule("T`", new List<Symbol>() { "K`" });
            LL.AddRule("T`", new List<Symbol>() { " " });
```

```
                    var parser = new LLParser(LL);
            Console.WriteLine("Пример вводимых строк: (i+i), (i+*i)");
            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).");
            }
            break;

        }
```

Результат работы программы:

```
Также создаем строку для Эпсилон
Рассмотрим нетерминал S
   Первый символ правила S -> aAB - a
   Это правило заносим в таблицу на пересечении строки нетерминала S и столбца терминала a

Рассмотрим нетерминал A
   Первый символ правила A -> bC - b
   Это правило заносим в таблицу на пересечении строки нетерминала A и столбца терминала b

Рассмотрим нетерминал B
   Первый символ правила B -> cA - c
   Это правило заносим в таблицу на пересечении строки нетерминала B и столбца терминала c

   Первый символ правила B -> aK - a
   Это правило заносим в таблицу на пересечении строки нетерминала B и столбца терминала a

Рассмотрим нетерминал C
   Первый символ правила C -> dD - d
   Это правило заносим в таблицу на пересечении строки нетерминала C и столбца терминала d

Рассмотрим нетерминал D
   Первый символ правила D -> f - f
   Это правило заносим в таблицу на пересечении строки нетерминала D и столбца терминала f

Рассмотрим нетерминал K
   Первый символ правила K -> oT - o
   Это правило заносим в таблицу на пересечении строки нетерминала K и столбца терминала o

Рассмотрим нетерминал K`
   Первый символ правила K` -> oT` - o
   Это правило заносим в таблицу на пересечении строки нетерминала K` и столбца терминала o

Рассмотрим нетерминал T
   Первый символ правила T -> K` - o
   Это правило заносим в таблицу на пересечении строки нетерминала T и столбца терминала o

   Первый символ правила T ->   -
   Это правило заносим в таблицу на пересечении строки нетерминала T и столбца терминала

Рассмотрим нетерминал T`
   Первый символ правила T` -> K` - o
   Это правило заносим в таблицу на пересечении строки нетерминала T` и столбца терминала o

   Первый символ правила T` ->   -
   Это правило заносим в таблицу на пересечении строки нетерминала T` и столбца терминала

Пример вводимых строк: (i+i), (i+*i)
Введите строку:
abdfao
Допуск. Цепочка символов = L(G).
```

Практическая работа №4 (лабораторные 12-16)

Формулировка задания:

Построить множество LR(0)-таблиц не содержащих ε-правила. Определить функции перехода g(X). Определить функцию переноса-свертки f(u). Для функции перехода g(X) и функции переноса-свертки f(u) спроектировать управляющую таблицу.

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

1. На основе активных префиксов (построения расширенного магазинного алфавита) и отношения OBLOW;
2. Построение SL(0) анализатора на основе LR(0)-ситуаций, функций замыкания CLOSURE и перехода GOTO;

Построим вторым способом LR(k) анализатор для заданной грамматики:

II. Построение SL(0) анализатора на основе LR(0)-ситуаций, функций замыкания CLOSURE и перехода GOTO.

Шаг 1. Построение управляющей таблицы M = [f(u), g(x)];

1.Пронумеровать правила продукций. Построить пополненную грамматику G' для исходной грамматики G.

2.Построить множества ситуаций и построение КА для множества ситуаций каждое множество ситуаций - состояние КА;

3.Построить отношение действий f(u) на основе КА.

Шаг 3. Применить алгоритм перенос-свёртка.

Определение. LR(0) ситуация - это правило грамматики с точкой в некоторой позиции правой части, например $[A \to w_1 \bullet w_2]$, если $A \to w_1 w_2$ – правило КС-грамматики.

Пример. Для правила $S \to (S)$ можно получить 4 ситуации:

$[S \to \bullet (S)]$

$[S \to (\bullet S)]$

$[S \to (S\bullet)]$

$[S \to (S)\bullet]$

Замечание. LR(0)-ситуация не содержит аванцепочку u, поэтому при ее записи можно опускать квадратные скобки.

G = (**T**, **V**, **P**, **S**), где

T = { a, b, c, d, f, o }, **V** = { S, A, B, C, D, K, K`}, $\mathbf{S_0}$ = S

P {

p_1: S → aAB

p_2: A → bC

p_3: C → dD

p_4: D → f

p_5: B → cA

p_6: B → aK

p_7: K → o

p_8: K → oK`

p_9: K` → o

p_{10}: K` → oK`

}

Шаг 1. Определение ситуаций и построение конечного автомата

Пусть I – множество LR(0)-ситуаций КС-грамматики G. Тогда назовем замыканием множества I множество ситуаций CLOSURE(I), построенное по следующим правилам:

1. Включить в CLOSURE(I) все ситуации из I.

2. Если ситуация A → α •Bβ уже включена в CLOSURE(I) и B → γ - правило грамматики, то добавить в множество CLOSURE(I) ситуацию B → •γ при условии, что там ее еще нет.

- Наличие ситуации A → α •Bβ в множестве CLOSURE(I) говорит о том, что в некоторый момент разбора может встретиться во входном потоке анализатора подстрока, выводимая из Bβ.
- Если в грамматике имеется правило B → γ, то также может встретиться во входном потоке анализатора подстрока, выводимая из γ, следовательно, в CLOSURE(I) нужно включить ситуацию B → •γ.

3. Повторять правило 2, до тех пор, пока в CLOSURE(I) нельзя будет включить новую ситуацию.

Пополненная грамматика G содержит еще одно правило: S' → S

G = (**T**, **V**, **P**, **S**), где

T = { a, b, c, d, f, o }, **V** = { S, A, B, C, D, K, K`}, $\mathbf{S_0}$ = S

P {

p_0: S’ → S

p_1: S → aAB

p_2: A → bC

p_3: C → dD

p_4: D → f

p_5: B → cA

p_6: B → aK

p_7: K → o

p_8: K → oK`

p_9: K` → o

p_{10}: K` → oK`

}

CLOSURE(S'→ •S) ={S'→ •S, S → •aAB}

I_0= {S'→ •S,S → •aAB}

Если *I*-множество ситуаций, допустимых для некоторого активного префикса γ, то GOTO(I,X) – это множество ситуаций, допустимых для активного префикса γ*X*.

Аргументами функции GOTO(*I*,*X*) являются множество ситуаций *I* и символ грамматики *X*.

Определение. Функция GOTO(*I*, *X*) определяется как замыкание множества всех ситуаций [$A \rightarrow \alpha X \bullet \beta$], таких что [$A \rightarrow \alpha \bullet X \beta$] $\in I$

Функция переходов GOTO

I_0 = CLOSURE(S'→ •S) ={S'→ •S, S → •aAB}
GOTO(I_0, a) = {S → a•AB, A → •bC} = I_1
GOTO(I_0, S) = {S'→ S•} = I_2
GOTO(I_1, b) = {A → b•C, C → •dD} = I_3
GOTO(I_1, A) = {S → aA•B, B → •cA, B → •aK} = I_4
GOTO(I_3, d) = {C → d•D, D → •f} = I_5
GOTO(I_3, C) = {A → bC•} = I_6
GOTO(I_4, a) = {B → a•K, K → •o, K → •oK`} = I_7
GOTO(I_4, c) = {B → c•A, A → •bC} = I_8
GOTO(I_4, B) = {S → aAB•} = I_9
GOTO(I_5, f) = {D → f•} = I_{10}
GOTO(I_5, D) = {C → dD•} = I_{11}
GOTO(I_7, o) = {K → o•, K → o•K`, K` → •o, K` → •oK`} = I_{12}
GOTO(I_7, K) = {B → aK•} = I_{13}
GOTO(I_8, A) = {B → cA•} = I_{14}
GOTO(I_{12}, o) = {K` → o•, K` → o•K`, K` → •o, K` → •oK`} = I_{15}
GOTO(I_{12}, K`) = {K → oK`•} = I_{16}
GOTO(I_{15}, K`) = {K` → oK`•} = I_{17}

Каноническая форма множества ситуаций

Построение канонической системы множеств *LR(0)*– ситуаций:

1 φ = ∅

2. Включить в φ множество I_0 = CLOSURE([S'→ • S]), которое в начале «не отмечено».

3. Если множество ситуаций I, входящее в систему , «не отмечено», то:

- отметить множество I;
- вычислить для каждого символа X ∈ (V U Σ) значение I' = GOTO(I,X);

- если множество I' ≠ ∅ и еще не включено в φ, то включить его в систему множеств как «неотмеченное» множество.

4. Повторять шаг 3, пока все множества ситуаций системы φ не будут отмечены.

φ = {

I_0 = {S'→ •S, S → •aAB},
I_1 = {S → a•AB, A → •bC},
I_2 = {S'→ S•},
I_3 = {A → b•C, C → •dD},
I_4 = {S → aA•B, B → •cA, B → •aK}
I_5 = {C → d•D, D → •f},
I_6 = {A → bC•},
I_7 = {B → a•K, K → •o, K → •oK`},
I_8 = {B → c•A, A → •bC},
I_9 = {S → aAB•},
I_{10} = {D → f•},
I_{11} = {C → dD•},
I_{12} = {K → o•, K → o•K`, K` → •o, K` → •oK`},
I_{13} = {B → aK•},
I_{14} = {B → cA•},
I_{15} = {K` → o•, K` → o•K`, K` → •o, K` → •oK`},
I_{16} = {K → oK`•},
I_{17} = {K` → oK`•}
}

Переход в состояние, которому на диаграмме соответствует лист (вершина, не имеющая исходящих дуг) однозначно определяет операцию Свёртки (i) по правилу i с переходом в новое состояние определяемое переходм на КА левым не териналом правила i, все остальные переходы операцию Переноса .

Шаг 2. Построение управляющей таблицы. Алгоритм построения управляющей таблицы M для LR(0)-грамматик основывается на рассмотрении пар грамматических вхождений, которые могут быть представлены соседними магазинными символами в процессе разбора допустимых цепочек.

1. Если операция[s_m, a_i] = Перенос(s), синтаксический анализатор выполняет перенос в стек очередного состояния s и его конфигурация становится

(s_0s_1... s_ms, a_{i+1}... a_n\$)

Символ a_i хранить в стеке не нужно (он может быть восстановлен из s). Текущим входным символом становится a_{i+1}.

2. Если операция[s_m, a_i] = Свертка(i) правила p_i: A→β, синтаксический анализатор выполняет свертку в два шага и его конфигурация становится

(s_0s_1... $s_{m-r}s$, a_ia_{i+1}... a_n\$)

здесь r - длина β, а s = GOTO[s_{m-r}, A].

2.1. Определяется правило для свертки i и левый нетерминал: p_i: A→ α.

2.2. Синтаксический анализатор снимает r символов состояний с вершины стека, что переносит на вершину стека состояние s_{m-r}. При свертке текущий входной символ не изменяется. (Удаляется из верхней части магазина |α| символов в соответствии с правилом C(i), где i - номер правила, A→ α, |α| - длина правой части правила).

2.3. После чего на вершину стека помещается s, запись из GOTO[s_{m-r}, A]. (По правилу для свертки i и левый нетерминал: p_i: A→ α , определяется по таблице переходов состояние, которое должно быть занесено в стек)

Последовательность символов грамматики X_{m-r+1}...X_m всегда соответствует α, правой части продукции свертки.

3. Если операция[s_m, a_i] = допуск синтаксический анализ завершается.

4.Если операция[s_m, a_i] = ошибка синтаксический анализатор вызывает программу восстановление после ошибки.

Управляющая таблица

I	f(u)							g(x)						
	a	b	c	d	f	o	⊥	S	A	B	C	D	K	K’
0	П,1							2						
1		П,3							4					
2							Д							
3				П,5							6			
4	П,7		П,8							9				
5					П,10							11		
6	С,2	С,2	С,2	С,2	С,2	С,2	С,2							
7						П,12							13	
8		П,3							14					
9	С,1	С,1	С,1	С,1	С,1	С,1	С,1							

10	**C,4**	**C,4**	**C,4**	**C,4**	**C,4**	**C,4**	**C,4**							
11	**C,3**	**C,3**	**C,3**	**C,3**	**C,3**	**C,3**	**C,3**							
12	**C,7**	**C,7**	**C,7**	**C,7**	**C,7**	**П,15**	**C,7**							**16**
13	**C,6**	**C,6**	**C,6**	**C,6**	**C,6**	**C,6**	**C,6**							
14	**C,5**	**C,5**	**C,5**	**C,5**	**C,5**	**C,5**	**C,5**							
15	**C,9**	**C,9**	**C,9**	**C,9**	**C,9**	**П,15**	**C,9**							**17**
16	**C,8**	**C,8**	**C,8**	**C,8**	**C,8**	**C,8**	**C,8**							
17	**C,10**	**C,10**	**C,10**	**C,10**	**C,10**	**C,10**	**C,10**							

Шаг 3. Применение алгоритма «перенос-свёртка» для разбора цепочки символов на ленте.

Работа алгоритма описывается в терминах конфигураций, представляющих собой тройки вида (αT, ax, π), где αT – цепочка магазинных символов, T-верхний символ магазина, T кодирует некий префикс цепочки (символ состояния), ax – необработанная часть входной цепочки, π – выход, построенный к настоящему моменту времени.

Распознавание цепочки abdfaoo:
(0, abdfaoo⊥, ε) $\vdash^{П,1}$ (0 1, bdfaoo⊥, ε) $\vdash^{П,3}$ (0 1 3, dfaoo⊥, ε) $\vdash^{П,5}$ (0 1 3 5, faoo⊥, ε) $\vdash^{П,10}$ (0 1 3 5 10, aoo⊥, ε) $\vdash^{C,4}$ (0 1 3 5 11, aoo⊥, 4) $\vdash^{C,3}$ (0 1 3 6, aoo⊥, 4 3) $\vdash^{C,2}$ (0 1 4, aoo⊥, **4 3 2**) $\vdash^{П,7}$ (0 1 4 7, oo⊥, 4 3 2) $\vdash^{П,12}$ (0 1 4 7 12, o⊥, 4 3 2) $\vdash^{П,15}$ (0 1 4 7 12 15, ⊥, 4 3 2) $\vdash^{C,9}$ (0 1 4 7 12 16, ⊥, 4 3 2 9) $\vdash^{C,8}$ (0 1 4 7 13, ⊥, 4 3 2 9 8) $\vdash^{C,6}$ (0 1 4 9, ⊥, 4 3 2 9 8 6) $\vdash^{C,1}$ (0 2, ⊥, 4 3 2 9 8 6 1) $\vdash^{Д}$

Получили правый разбор входной цепочки: 1 6 8 9 2 3 4
Для проверки построим цепочку по заданному правому выводу: S =>[1] aAB =>[6] aAaK =>[8] aAaoK` =>[9] aAaoo =>[2] abCaoo =>[3] abdDaoo =>[4] abdfaoo

```
Prules:
S' -> S
S -> aAB
A -> bC
C -> dD
D -> f
B -> cA
B -> aK
K -> o
K -> oK`
K` -> o
K` -> oK`
Canonical set of states phi
I_0 = CLOSURE( S' -> .S )
S' -> .S
S -> .aAB
I_1 = CLOSURE( S -> a.AB )
S -> a.AB
A -> .bC
I_2 = CLOSURE( S' -> S. )
S' -> S.
I_3 = CLOSURE( A -> b.C )
A -> b.C
C -> .dD
I_4 = CLOSURE( S -> aA.B )
S -> aA.B
B -> .cA
B -> .aK
I_5 = CLOSURE( C -> d.D )
C -> d.D
D -> .f
I_6 = CLOSURE( A -> bC. )
A -> bC.
I_7 = CLOSURE( B -> a.K )
B -> a.K
K -> .o
K -> .oK`
I_8 = CLOSURE( B -> c.A )
B -> c.A
A -> .bC
I_9 = CLOSURE( S -> aAB. )
S -> aAB.
I_10 = CLOSURE( D -> f. )
D -> f.
I_11 = CLOSURE( C -> dD. )
C -> dD.
```

```
I_12 = CLOSURE( K -> o. )
K -> o.
K -> o.K`
K` -> .o
K` -> .oK`
I_13 = CLOSURE( B -> aK. )
B -> aK.
I_14 = CLOSURE( B -> cA. )
B -> cA.
I_15 = CLOSURE( K` -> o. )
K` -> o.
K` -> o.K`
K` -> .o
K` -> .oK`
I_16 = CLOSURE( K -> oK`. )
K -> oK`.
I_17 = CLOSURE( K` -> oK`. )
K` -> oK`.
LR-automate

Automate definition:
Q: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Sigma: a b c d f o $ S A B C D K K` S'
Q0: 0
F: 2 6 9 10 11 12 13 14 15 16 17
DeltaList:
 delta(0,a,-> (1
 delta(0,S,-> (2
 delta(1,b,-> (3
 delta(1,A,-> (4
 delta(3,d,-> (5
 delta(3,C,-> (6
 delta(4,a,-> (7
 delta(4,c,-> (8
 delta(4,B,-> (9
 delta(5,f,-> (10
 delta(5,D,-> (11
 delta(7,o,-> (12
 delta(7,K,-> (13
 delta(8,b,-> (3
 delta(8,A,-> (14
 delta(12,o,-> (15
 delta(12,K`,-> (16
 delta(15,o,-> (15
 delta(15,K`,-> (17
```

```
 Введите строку:

abdfaoo

 Введена строка: abdfaoo

Строка допущена
Вывод: 1 6 8 9 2 3 4
```

Contol table M

	a	b	c	d	f	o	$	S	A	B	C	D	K	K`
0	S 1							2						
1		S 3							4					
2							A							
3				S 5							6			
4	S 7		S 8							9				
5					S10							11		
6	R 2	R 2	R 2	R 2	R 2	R 2	R 2							
7						S12							13	
8		S 3							14					
9	R 1	R 1	R 1	R 1	R 1	R 1	R 1							
10	R 4	R 4	R 4	R 4	R 4	R 4	R 4							
11	R 3	R 3	R 3	R 3	R 3	R 3	R 3							
12	R 7	R 7	R 7	R 7	R 7	S15	R 7							16
13	R 6	R 6	R 6	R 6	R 6	R 6	R 6							
14	R 5	R 5	R 5	R 5	R 5	R 5	R 5							
15	R 9	R 9	R 9	R 9	R 9	S15	R 9							17
16	R 8	R 8	R 8	R 8	R 8	R 8	R 8							
17	R10	R10	R10	R10	R10	R10	R10							

Код программы:

```
case "14":

            var LR0Grammar = new SLRGrammar(new List<Symbol>() { "a", "b", "c", "d", "f", "o"
},
                                new List<Symbol>() { "S", "A", "B", "C", "D", "K", "K`" },
                                new List<Production>(),
                                "S");

            LR0Grammar.AddRule("S",new List<Symbol>() { "a", "A", "B" });
            LR0Grammar.AddRule("A",new List<Symbol>() { "b", "C" });
            LR0Grammar.AddRule("C",new List<Symbol>() { "d", "D" });
            LR0Grammar.AddRule("D",new List<Symbol>() { "f" });
            LR0Grammar.AddRule("B",new List<Symbol>() { "c", "A" });
            LR0Grammar.AddRule("B", new List<Symbol>() { "a", "K" });
            LR0Grammar.AddRule("K", new List<Symbol>() { "o" });
            LR0Grammar.AddRule("K", new List<Symbol>() { "o", "K`" });
            LR0Grammar.AddRule("K`", new List<Symbol>() { "o" });
            LR0Grammar.AddRule("K`", new List<Symbol>() { "o", "K`" });

            LR0Grammar.Construct();
            LR0Grammar.Inference();
                  break;
```