**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: **5.2. pattern = 192\.168\.1\.\d{1,3}** Автоматная грамматика:

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

192.168.1.({0-9})2,

192.168.1.({0-9})3

)

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

{192.168.1., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9},

{S0, A, B, C, D},

{p1, p2, p3, p4, p5}, S**0**

)

Правила регулярной грамматики: p1**: S0 →** 192.168.1.**A**

P2**: A →** 0**B |** 1**B | ... |** 9**B** p3**: B →** 0**C |** 1**C | ... |** 9**C | C** p4**: C →** 0**D |** 1**D | ... |** 9**D | D** p5**: D** → ε

Пример цепочек:

**S0** =>1 192.168.1.**A** =>2 192.168.1.1**B** =>3 192.168.1.12**C** =>4 192.168.1.123**D** =>5 192.168.1.123

**S0** =>1 192.168.1.**A** =>2 192.168.1.1**B** =>3 192.168.1.12**C** =>4 192.168.1.12**D** =>5 192.168.1.12

**S0** =>1 192.168.1.**A** =>2 192.168.1.1**B** =>3 192.168.1.1**C** =>4 192.168.1.1**D** =>5 192.168.1.12

Конечный автомат: L(КА) = L(G)

КА = (**Q**, **Σ**, **δ**, **S0**, **F**), где

**Q** = {**S0**, **A**, B, C **qf**}, **Σ** = {0-9, 192.168.1.}, **S0** = **S0**, **F** = **qf**,

**δ** = {  
1. δ(**S0**, 192.168.1.) = {**A**},  
2. δ(**A**, 0) = {**B**},  
…  
11. δ(**A**, 9) = {**B**},  
12. δ(**B**, 0) = {**C**},  
…  
21. δ(**B**, 9) = {**C**},  
22. δ(**B**, ε) = {**qf**},  
23. δ(**C**, 0) = {**D**},  
…  
32. δ(**C**, 9) = {**D**},  
33. δ(**C**, ε) = {**qf**},  
34. δ(**D**, ε) = {**qf**}  
}

Примеры конфигурации КА:

**1.** (**S0**, 192.168.1.1) ⸠1 (**A**, 1) ⸠3 (**B**, ε) ⸠22 (**qf**, ε)

**2.** (**S0**, 192.168.1.12) ⸠1 (**A**, 12) ⸠3 (**B**, 2) ⸠14 (**C**, ε) ⸠33 (**qf**, ε)

**3.** (**S0**, 192.168.1.123) ⸠1 (**A**, 123) ⸠3 (**B**, 23) ⸠14 (**C**, 3) ⸠26 (**D**, ε) ⸠34 (**qf**, ε)

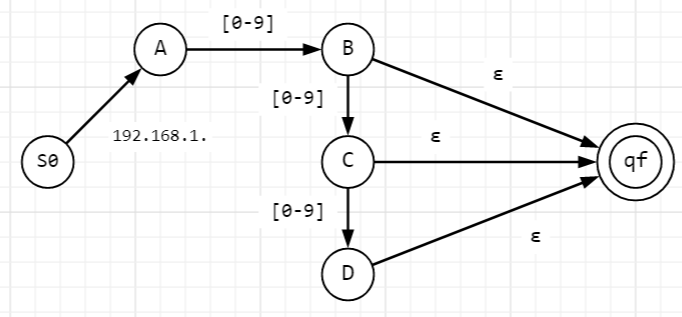


Рисунок 1 — Диаграмма автомата

Программа:

|  |
| --- |
| var ka1 = new FSAutomate(new List<Symbol>() { "S0", "A", "B", "C", "D", "qf" },  new List<Symbol>() { "", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "192.168.1." },  new List<Symbol>() { "qf" },  "S0");  ka1.AddRule("S0", "192.168.1.", "A");  for (int i = 0; i < 10; i++)  ka1.AddRule("A", i.ToString(), "B");  for (int i = 0; i < 10; i++)  ka1.AddRule("B", i.ToString(), "C");  ka1.AddRule("B", "", "qf");  for (int i = 0; i < 10; i++)  ka1.AddRule("C", i.ToString(), "D");  ka1.AddRule("C", "", "qf"); ka1.AddRule("D", "", "qf"); |

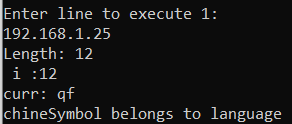
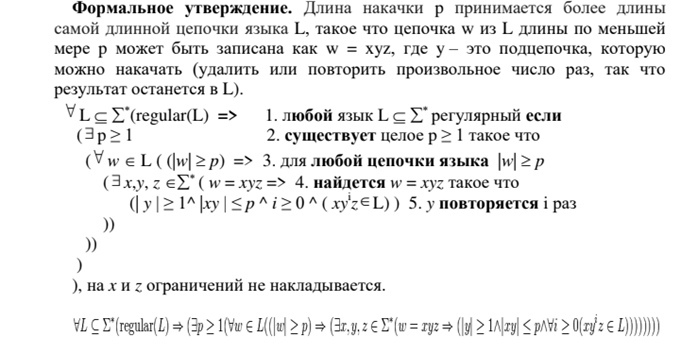


Рисунок 2 — Вывод конечного автомата

Лемма о накачке:



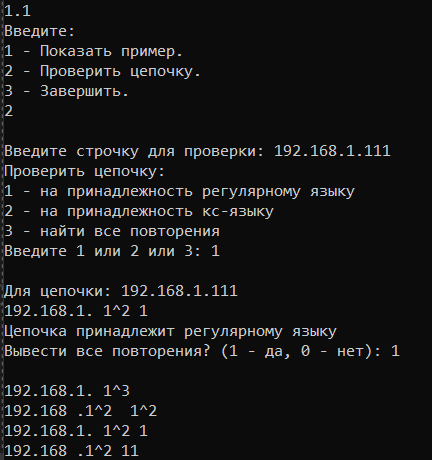


Рисунок 3 – Лемма о накачке

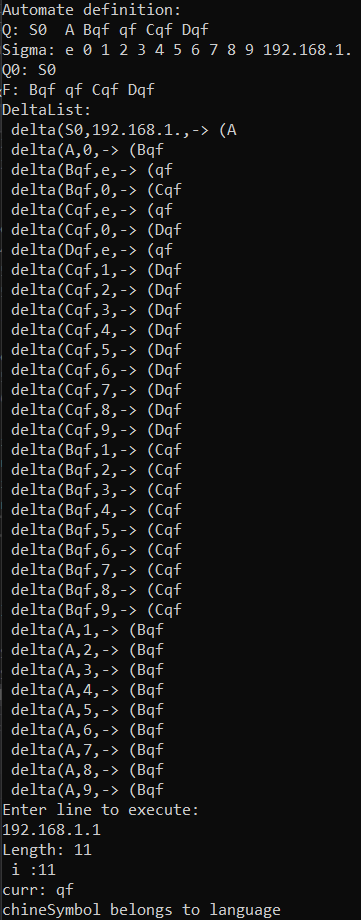
Проектирование ДКА:  


Рисунок 4 – Детерменированный конечный автомат

## 7.1. pattern = "^\+?\d{0,2}\-?\d{4,5}\-?\d{5,6}"

Автоматная грамматика:

L(pattern) = L("^\+?\d{0,2}\-?\d{4,5}\-?\d{5,6}")

G(T, V, P, S0) = G({+, -, 0, …, 9}, {S0, A0, A1, A2, B0, …, B5, C0, …, C6}, {p1, …, p17}, S**0**)

Правила регулярной грамматики:   
p1**: S0 →** +**A0** | **A0**p2: **A0 →** 0**A1** | … | 9**A1** | **A1**p3: **A1 →** 0**A2** | … | 9**A2**p4: **A2 →** -**B0** | **B0**p5: **B0 →** 0**B1** | … | 9**B1**  
…  
p9: **B4 →** 0**B5** | … | 9**B5** | **B5**p10: **B5 →** -**C0** | **C0**p11: **C0 →** 0**C1** | … | 9**C1**…  
p16: **C5 →** 0**C6** | … | 9**C6** | **C6**p17: **C6 →** ε

Пример цепочек:

**S0** =>1 +**A0** =>2 +9**A1** =>3 +91**A2** =>4 +91-**B0** =>5 +91-9**B1** =>6 +91-96**B2** =>7 +91-967**B3** =>8 +91-9678**B4** =>9 +91-96789**B5** =>10 +91-96789**C0** =>11 +91-967896**C1** =>12 +91-9678967**C2** =>13 +91-96789671**C3** =>14 +91-967896710**C4** =>15 +91-9678967101**C5** =>16 +91-9678967101**C6** =>17 +91-9678967101

Конечный автомат: L(КА) = L(G)

КА = (**Q**, **Σ**, **δ**, **S0**, **F**), где

**Q** = { **S0, A0, A1, A2, B0, …, B5, C0, …, C6, qf**},  
**Σ** = {+, -, 0-9}, **S0** = **S0**, **F** = **qf**,

**δ** = {

1. δ(**S0**, +) = {**A0**},

2. δ(**S0**, ε) = {**A0**},

3. δ(**A0**, 0) = {**A1**},

…

13. δ(**A0**, ε) = {**A1**},

14. δ(**A0**, ε) = {**A2**},

15. δ(**A1**, 0) = {**A2**},

…

25. δ(**A2**, -) = {**B0**},

26. δ(**A2**, ε) = {**B0**},

27. δ(**B0**, 0) = {**B1**},

…

77. δ(**B4**, ε) = {**B5**},

78. δ(**B5**, -) = {**C0**},

79. δ(**B5**, ε) = {**C0**},

80. δ(**C0**, 0) = {**C1**},

…

140. δ(**C5**, ε) = {**C6**},

141. δ(**C6**, ε) = {**qf**}

}

Примеры конфигурации КА:

(**S0**,+91-9678967101) ⸠1 (**A0**,91-9678967101) ⸠12 (**A1**,1-9678967101) ⸠14  
(**A2**,-9678967101) ⸠25 (**B0**,9678967101) ⸠36 (**B1**,678967101) ⸠43(**B2**,78967101) ⸠54 (**B3**,8967101) ⸠65 (**B4**,967101)⸠76 (**B5**,967101) ⸠79(**C0**,967101) ⸠89 (**C1**,67101) ⸠96 (**C2**,7101) ⸠107 (**C3**,101) ⸠111 (**C4**,01) ⸠120(**C5**,1) ⸠131 (**C6**,ε) ⸠141 (**qf**,ε)

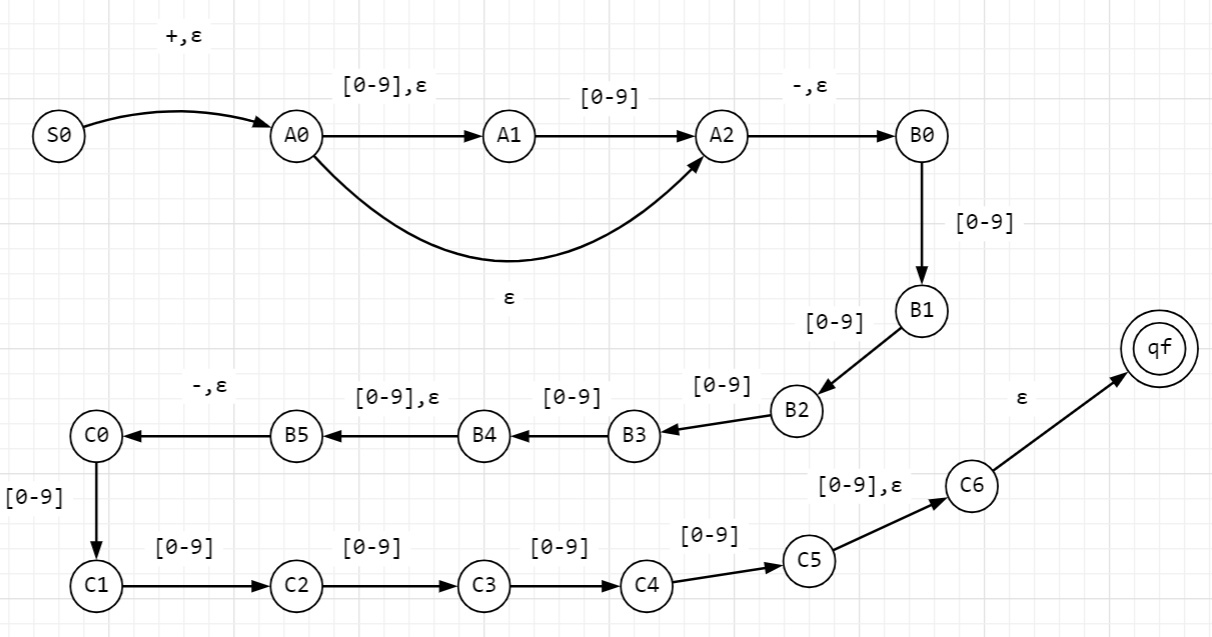


Рисунок 5 – Диаграмма преобразованного атвомата

Программа:

|  |
| --- |
| var ka2 = new FSAutomate(new List<Symbol>() { "S0", "A0", "A1", "A2", "B0", "B1", "B2", "B3", "B4", "B5", "C0", "C1", "C2", "C3", "C4", "C5", "C6", "qf"},  new List<Symbol>() { "", "+", "-", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" },  new List<Symbol>() { "qf" },  "S0");  ka2.AddRule("S0", "+", "A0");  ka2.AddRule("S0", "", "A0");  for (int i = 0; i < 2; i++)  for (int j = 0; j < 10; j++)  ka2.AddRule("A" + i.ToString(), j.ToString(), "A" + (i + 1).ToString());  ka2.AddRule("A0", "", "A1");  ka2.AddRule("A0", "", "A2");  ka2.AddRule("A2", "-", "B0");  ka2.AddRule("A2", "", "B0");  for (int i = 0; i < 5; i++)  for (int j = 0; j < 10; j++)  ka2.AddRule("B" + i.ToString(), j.ToString(), "B" + (i + 1).ToString());  ka2.AddRule("B4", "", "B5");  ka2.AddRule("B5", "-", "C0");  ka2.AddRule("B5", "", "C0");  for (int i = 0; i < 6; i++)  for (int j = 0; j < 10; j++)  ka2.AddRule("C" + i.ToString(), j.ToString(), "C" + (i + 1).ToString());  ka2.AddRule("C5", "", "C6");  ka2.AddRule("C6", "", "qf"); |

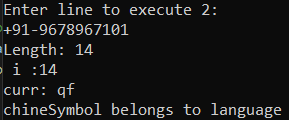


Рисунок 6 – Проверка цепочки на принадлежность к языку

Лемма о накачке:

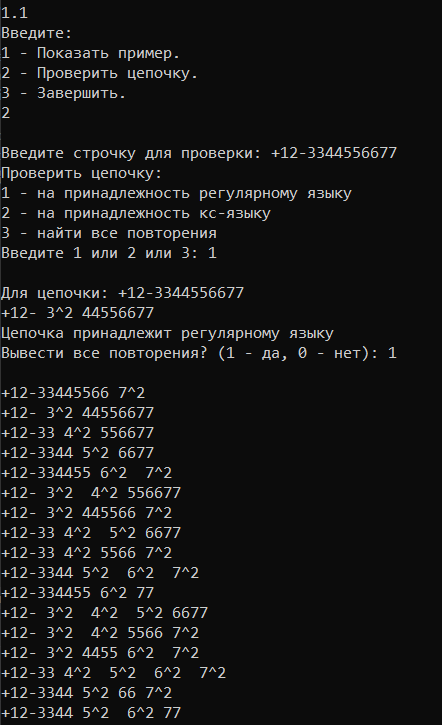


Рисунок 7 – Лемма о накачке для регулярного выражения

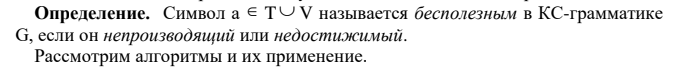
**Практическая работа №2 (4-8 лаб.)**

*Лабораторные работы №4-8*

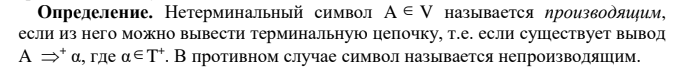
*Заданная грамматика:*

***G =*** ***{( a, b, c, d, f, epsilon, z, r, t, g ),( S, A, B, C, D, R, T, H, G, V ),( A → bC, B → cA, C → dD, D → R, R → f, S → aABT, T → epsilon, H → z, G → r, S → gV, V → Vt, V → t),S}***

**Лабораторная работа №4: Устранить из КС-грамматики бесполезные символы и ε–правила**



1. *Устранение непроизводящих символов*

*14 Построить каноническую форму множества ситуаций.*

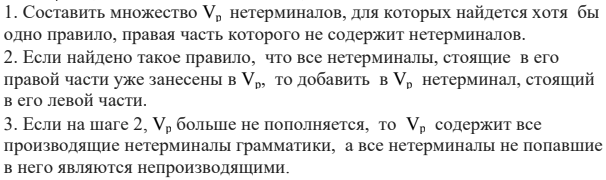
*Построить управляющую таблицу для функции перехода g(х) и действий f(u)*

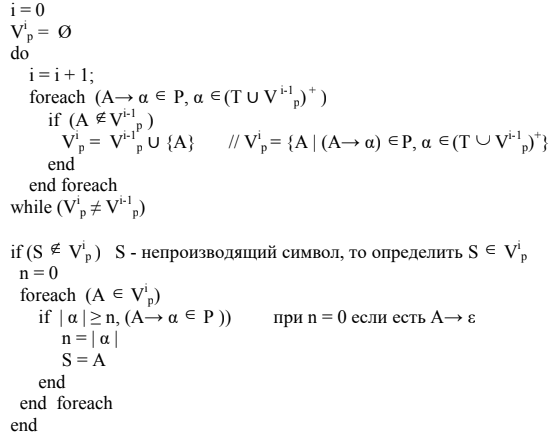
*16.1 LR(0) using g(X), f(a)*

*16.2 LR(0) using g(X), f(a) example*

*16.3 LR(1) using g(X), f(a)*

*16.4 LR(1) using g(X), f(a) example*





1. V1 = {**R, T, H, G, V**} – для этих нетерминалов нашлось хотя бы одно правило, правая часть которого не содержит нетерминалов
2. V2 = {R, T, H, G, V, **D, S**} - если найдено такое правило, что все нетерминалы, стоящие в его правой части уже занесены в V, то добавить в V нетерминал, стоящий в его левой части
3. V3 = {R, T, H, G, V, D, S, **C**}
4. V4 = {R, T, H, G, V, D, S, C, **A**}
5. V5 = {R, T, H, G, V, D, S, C, A, **B**} => все нетерминалы производящие

Тогда P’ = P = {p1, …, p12}

p1: A → bC,

p2: B → cA,

p3: C → dD,

p4: D → R,

p5: R → f,

p6: S → aABT,

p7: T → ε,

p8: H → z,

p9: G → r,

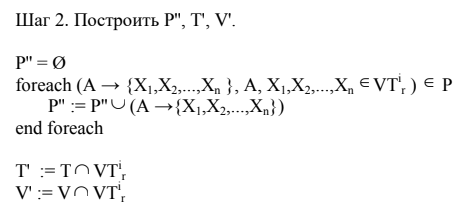
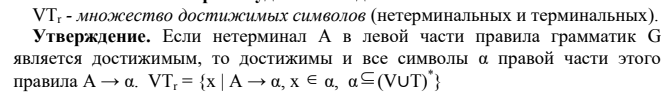
p10: S → gV,

p11: V → Vt,

p12: V → t

G1 = {( a, b, c, d, f, ε, z, r, t, g ),( S, A, B, C, D, R, T, H, G, V ), P’, S}

1. Устранение недостижимых символов (VTr – множество недостижимых символов)



1. VT1 = {S}
2. VT2 = {S, **a, A, B, T, g, V**}
3. VT3 = {S, a, A, B, T, g, V, **b, C, c, ε, t**}
4. VT4 = {S, a, A, B, T, g, V, b, C, c, ε, t, **d, D**}
5. VT5 = {S, a, A, B, T, g, V, b, C, c, ε, t, d, D, **R**}
6. VT6 = {S, a, A, B, T, g, V, b, C, c, ε, t, d, D, R, **f**}

Так, в VT не вошли символы {z, r, H, G}. Тогда P’’ = {p1, …, p10}

p1: A → bC,

p2: B → cA,

p3: C → dD,

p4: D → R,

p5: R → f,

p6: S → aABT,

p7: T → ε,

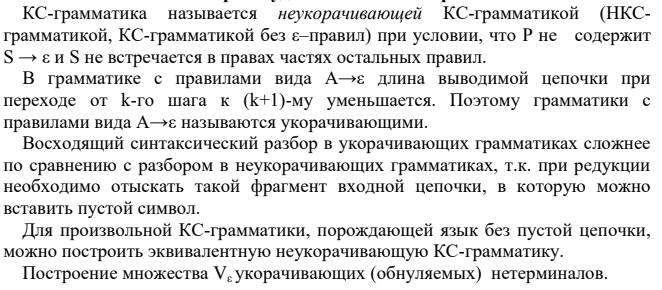
p8: S → gV,

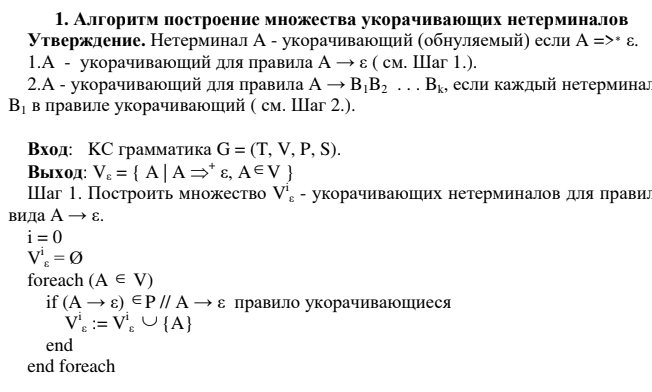
p9: V → Vt,

p10: V → t

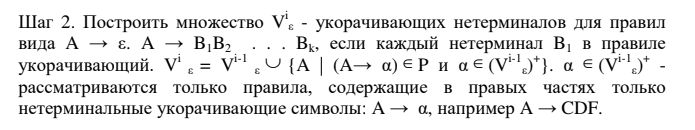
G’ = {( a, b, c, d, f, ε, t, g ),( S, A, B, C, D, R, T, V ), P’’, S}

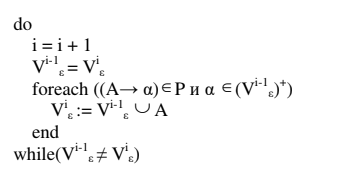
1. Устранить из КС-грамматики ε–правила





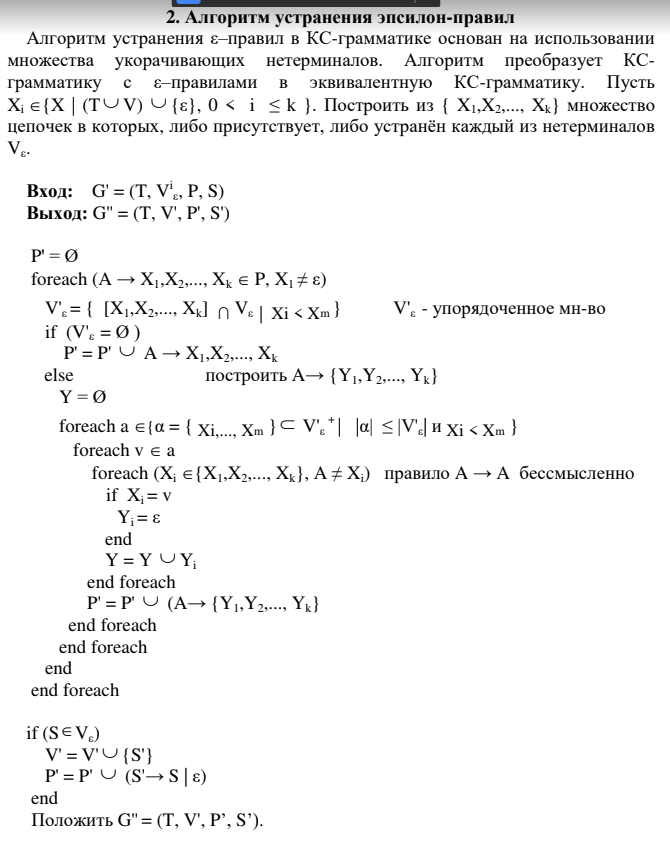
1. V0 = {T} - (множество укорачивающих нетерминалов для правил вида A → ε)





Необходимо построить множество V1 укорачивающих нетерминалов для правил вида A → B1B2…Bk, если каждый нетерминал Bi в правиле укорачивающий. Однако в данном примере таких правил нет.

Итак, множество укорачивающих терминалов: V = {T}.



G’ = {( a, b, c, d, f, ε, t, g ),( S, A, B, C, D, R, V ), P, S}, где P = {p1, …, p9}

p1: A → bC,

p2: B → cA,

p3: C → dD,

p4: D → R,

p5: R → f,

p6: S → aAB,

p7: S → gV,

p8: V → Vt,

p9: V → t

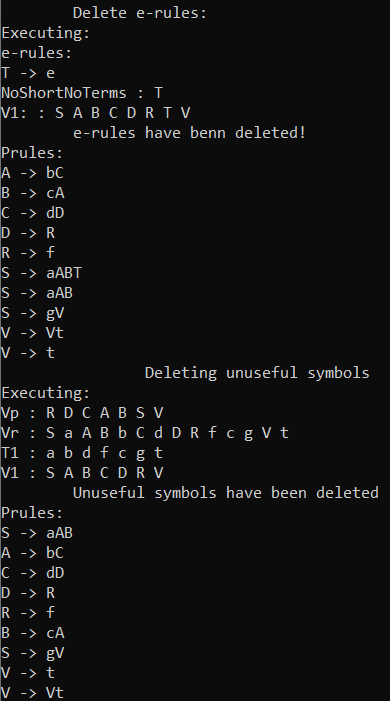
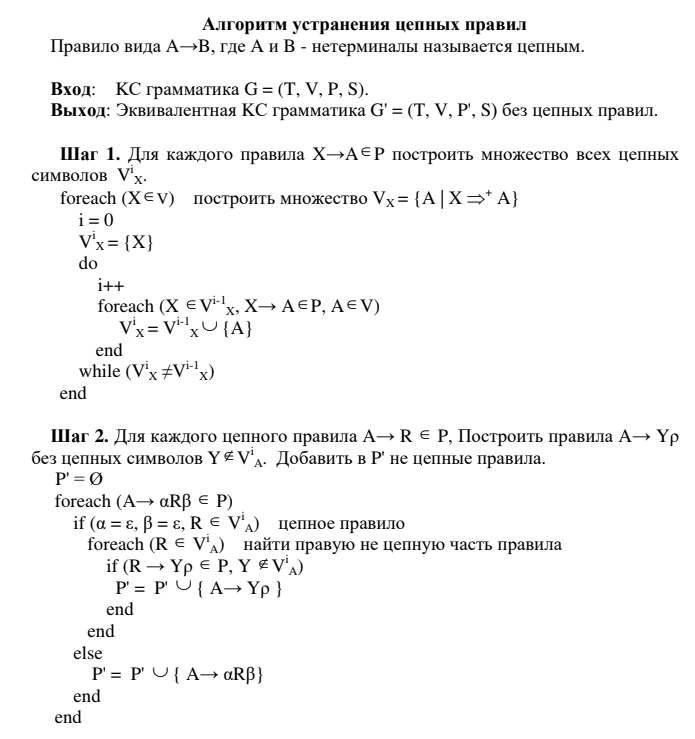


Рисунок 8 – Удаление эпсилон-правил

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

1. Устранить из KС-грамматики цепные правила.



Цепные правила: D → R

V1 = {R}

Правила D → R, R → f, “заменим” на D → f

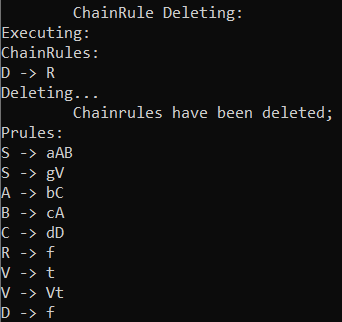
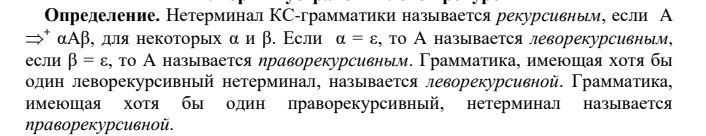
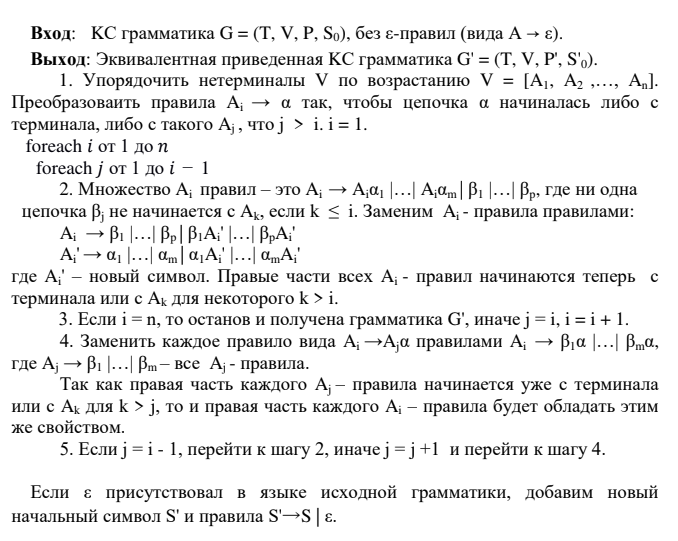


Рисунок 9 – Цепные правила

1. Устранить из KС-грамматики левую рекурсию.



**Алгоритм Устранения непосрественной левой рекурсии**



Левая рекурсия V → Vt

Заменим правило V → Vt, на правила V → tV’, V’ → tV’|t

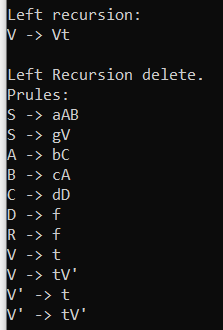


Рисунок 10 – Левая рекурсия

**Приведенная грамматика:**

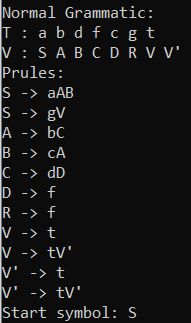
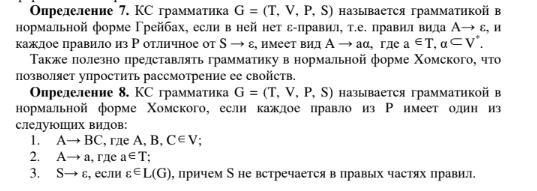


Рисунок 11 – Приведённая грамматика

|  |
| --- |
| var regGr = new Grammar(new List<Symbol>() { "a", "b", "c", "d", "f", "t", "g", "z", "r" },  new List<Symbol>() { "S", "A", "B", "C", "D", "R", "T", "V", "H", "G" },  "S");  regGr.AddRule("A", new List<Symbol>() { "b", "C"});  regGr.AddRule("B", new List<Symbol>() { "c", "A" });  regGr.AddRule("C", new List<Symbol>() { "d", "D" });  regGr.AddRule("D", new List<Symbol>() { "R" });  regGr.AddRule("R", new List<Symbol>() { "f" });  regGr.AddRule("S", new List<Symbol>() { "a", "A", "B", "T" });  regGr.AddRule("T", new List<Symbol>() { "" });  regGr.AddRule("H", new List<Symbol>() { "z" });  regGr.AddRule("G", new List<Symbol>() { "r" });  regGr.AddRule("S", new List<Symbol>() { "g", "V" });  regGr.AddRule("V", new List<Symbol>() { "V", "t" });  regGr.AddRule("V", new List<Symbol>() { "t" });  Console.WriteLine("Grammar:");  regGr.Debug("T", regGr.T);  regGr.Debug("T", regGr.V);  regGr.DebugPrules();  Grammar G1 = regGr.EpsDelete();  G1.DebugPrules();  Grammar G2 = G1.unUsefulDelete();  G2.DebugPrules();  Grammar G3 = G2.ChainRuleDelete();  G3.DebugPrules();  Grammar G4 = G3.LeftRecursDelete\_new6();  G4.DebugPrules();  // G4 - приведенная грамматика  Console.WriteLine("--------------------------------------------");  Console.WriteLine("Normal Grammatic:");  G4.Debug("T", G4.T);  G4.Debug("V", G4.V);  G4.DebugPrules();  Console.Write("Start symbol: "); Console.WriteLine(G4.S0 + "\n"); |

**Лабораторная работа №6: Определить форму КС-грамматики и сделать ее приведение.**



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

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

**Лабораторная работа №7: Спроектировать МП-автомат для приведенной КС-грамматики.**

Приведённая грамматика:

G = (T, V, P, S), где

T = {a, b, c, d, f, t, g}, V = {S, A, B, C, D, R, V, V’}, S0 = S,

P = {

p1: A → bC,

p2: B → cA,

p3: C → dD,

p4: D → f,

p5: R → f,

p6: S → aAB,

p7: S → gV,

p8: V → t,

p9: V → tV’,

p10: V’ → t,

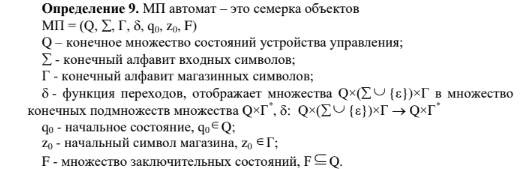
p11: V’ → tV’

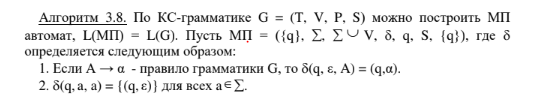
}

Цепочки вывода:

S =>7 gV =>8 gt

S =>6 aAB =>1 abCB =>3 abdDB =>4 abdfB =>2 abdfcA =>1 abdfcbC =>3 abdfcbdD =>4 abdfcbdf





L(МП) = L(G)

MП = (Q, Σ, Г, δ, q0, z0, F):

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

МП = ({q}, {a, b, c, d, f, t, g}, {a, b, c, d, f, t, g, S, A, B, C, D, R, V, V’}, δ, q0, S, {q})

δ = {

1. δ(q, ε, A) = {(q, bC)},

2. δ(q, ε, B) = {(q, cA)},

3. δ(q, ε, C) = {(q, dD)},

4. δ(q, ε, D) = {(q, f)},

5. δ(q, ε, R) = {(q, f)},

6. δ(q0, ε, S) = {(q, aAB)},

7. δ(q0, ε, S) = {(q, gV)},

8. δ(q, ε, V) = {(q, t)},

9. δ(q, ε, V) = {(q, tV’)},

10. δ(q, ε, V’) = {(q, t)},

11. δ(q, ε, V’) = {(q, tV’)},

12. δ(q, a, a) = {(q, ε)}, ∀ a ∈ Σ

}

Последовательность тактов МП-автомата для цепочки abdfcbdf:

(q0, abdfcbdf, S) ⸠6 (q, abdfcbdf, aAB) ⸠12 (q, bdfcbdf, AB) ⸠1 (q, bdfcbdf, bCB) ⸠12 (q, dfcbdf, CB) ⸠3 (q, dfcbdf, dDB) ⸠12 (q, fcbdf, DB) ⸠4 (q, fcbdf, fB) ⸠12 (q, cbdf, B) ⸠2 (q, cbdf, cA) ⸠12 (q, bdf, A) ⸠1 (q, bdf, bC) ⸠12 (q, df, C) ⸠3 (q, df, dD) ⸠12 (q, f, D) ⸠4 (q, f, f) ⸠12 (q, ε, ε)

**Лабораторная работа №8: Реализовать МП-автомат для приведенной КС-грамматики.**

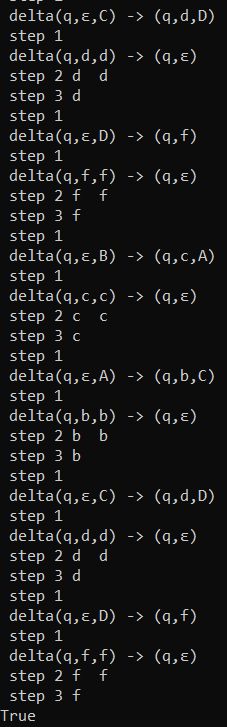
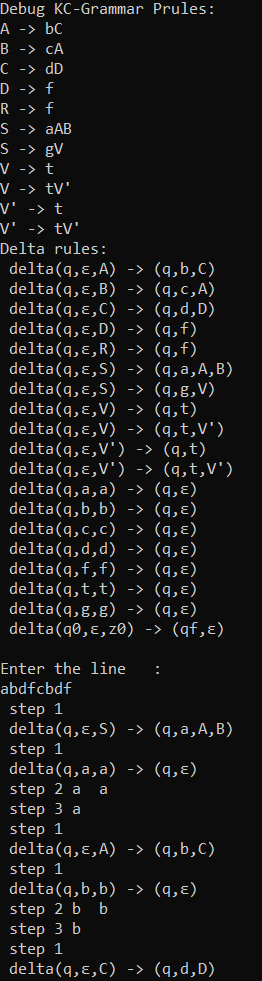


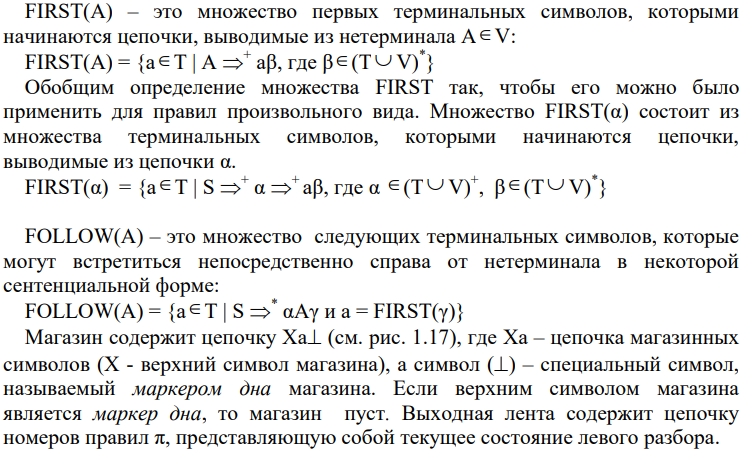
Рисунок 12 – МП-автомат

|  |
| --- |
| var CFGrammar = new Grammar(new List<Symbol>() { "a", "b", "c", "d", "f", "t", "g" },  new List<Symbol>() { "S", "A", "B", "C", "D", "R", "V", "V'" },  "S");  CFGrammar.AddRule("A", new List<Symbol>() { "b", "C" });  CFGrammar.AddRule("B", new List<Symbol>() { "c", "A" });  CFGrammar.AddRule("C", new List<Symbol>() { "d", "D" });  CFGrammar.AddRule("D", new List<Symbol>() { "f" });  CFGrammar.AddRule("R", new List<Symbol>() { "f" });  CFGrammar.AddRule("S", new List<Symbol>() { "a", "A", "B" });  CFGrammar.AddRule("S", new List<Symbol>() { "g", "V" });  CFGrammar.AddRule("V", new List<Symbol>() { "t" });  CFGrammar.AddRule("V", new List<Symbol>() { "t", "V'" });  CFGrammar.AddRule("V'", new List<Symbol>() { "t" });  CFGrammar.AddRule("V'", new List<Symbol>() { "t", "V'" });  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()); |

**Практическая работа №3 (9-10 лаб.)**

**Лабораторная работа №9: 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:

G = (T, V, P, S), где

T = {a, b, c, d, f, t, g}, V = {S, A, B, C, D, R, V, V’}, S0 = S,

P = {

p1: A → bC,

p2: B → cA,

p3: C → dD,

p4: D → f,

p5: R → f,

p6: S → aAB,

p7: S → gV,

p8: V → t,

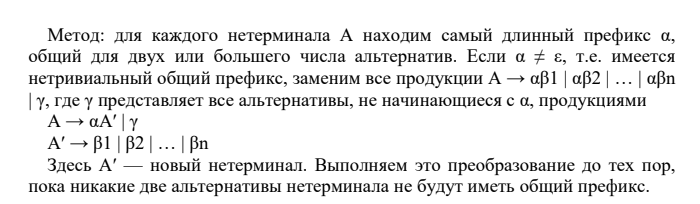
p9: V → tV’,

p10: V’ → t,

p11: V’ → tV’

}

Сделаем приведение к LL(1) для построения управляющей таблицы, проведя факторизацию



Gф = (T, V’, P’, S), где

T = {a, b, c, d, f, t, g}, V’ = {S, A, B, C, D, R, V, V’, T}, S0 = S,

P’ = {

p1: A → bC,

p2: B → cA,

p3: C → dD,

p4: D → f,

p5: R → f,

p6: S → aAB,

p7: S → gV,

p8: V → tT,

p9: V’ → tT,

p10: T → V’,

p11: T → ε

}

**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 **{Ʇ}**) × (**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 {Ʇ}) и a ∈ T U {ε}

Таблица 1 — Управляющая таблица для LL(1)-грамматики

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **a** | **b** | **c** | **d** | **f** | **t** | **g** | **ε** |
| **S** | (aAB, 6) |  |  |  |  |  | (gv, 7) |  |
| **A** |  | (bC, 1) |  |  |  |  |  |  |
| **B** |  |  | (cA, 2) |  |  |  |  |  |
| **C** |  |  |  | (dD, 3) |  |  |  |  |
| **D** |  |  |  |  | (f, 4) |  |  |  |
| **R** |  |  |  |  | (f, 5) |  |  |  |
| **V** |  |  |  |  |  | (tT, 8) |  |  |
| **V’** |  |  |  |  |  | (tT, 9) |  |  |
| **T** |  |  |  |  |  | (V’, 10) |  | (ε, 11) |
| **a** | ВЫБРОС |  |  |  |  |  |  |  |
| **b** |  | ВЫБРОС |  |  |  |  |  |  |
| **c** |  |  | ВЫБРОС |  |  |  |  |  |
| **d** |  |  |  | ВЫБРОС |  |  |  |  |
| **f** |  |  |  |  | ВЫБРОС |  |  |  |
| **t** |  |  |  |  |  | ВЫБРОС |  |  |
| **g** |  |  |  |  |  |  | ВЫБРОС |  |
| **Ʇ** |  |  |  |  |  |  |  | ДОПУСК |

Пустые клетки в таблице означают ОШИБКУ.

Аналитичекое представление для таблицы М:

Таблица 2 — Аналитическое представление управляющей таблицы

|  |  |  |
| --- | --- | --- |
| **Правило грамматики** | **Множество** | **Значение М** |
| p1: A → bC | FIRST(A) = {b} | M(A, b) = (bC, 1) |
| p2: B → cA | FIRST(B) = {c} | M(B, c) = (cA, 2) |
| p3: C → dD | FIRST(C) = {d} | M(C, d) = (dD, 3) |
| p4: D → f | FIRST(D) = {f} | M(D, f) = (f, 4) |
| p5: R → f | FIRST(R) = {f} | M(R, f) = (f, 5) |
| p6: S → aAB | FIRST(S) = {a} | M(S, a) = (aAB, 6) |
| p7: S → gV | FIRST(S) = {g} | M(S, g) = (gV, 7) |
| p8: V → tT | FIRST(V) = {t} | M(V, t) = (tT, 8) |
| p9: V’ → tT | FIRST(V’) = {t} | M(V’, t) = (tT, 9) |
| p10: T → V’ | FIRST(T) = {t} | M(T, t) = (V’, 10) |
| p11: T → ε | FIRST(T) = {ε} | M(T, ε) = (ε, 11) |

**Лабораторная работа №10: Аналитически написать правила вывода для цепочки LL(k) анализатора.**

Шаг 1. Алгоритм находится в начальной конфигурации (abdfcbdf, S0Ʇ, ε), где S0 = S

Значение управляющей таблицы M(K, f) = (A, B, 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:

Таблица 3 — Конфигурации цепочек

|  |  |
| --- | --- |
| Текущая конфигурация | Значение М |
| (abdfcbdf, SꞱ, ε) ⸠ | M(S, a) = (aAB, 6) |
| (abdfcbdf, aABꞱ, 6) ⸠ | M(a, a) = ВЫБРОС |
| (bdfcbdf, ABꞱ, 6) ⸠ | M(A, b) = (bC, 1) |
| (bdfcbdf, bCBꞱ, 61) ⸠ | M(b, b) = ВЫБРОС |
| (dfcbdf, CBꞱ, 61) ⸠ | M(C, d) = (dD, 3) |
| (dfcbdf, dDBꞱ, 613) ⸠ | M(d, d) = ВЫБРОС |
| (fcbdf, DBꞱ, 613) ⸠ | M(D, f) = (f, 4) |
| (fcbdf, fBꞱ, 6134) ⸠ | M(f, f) = ВЫБРОС |
| (cbdf, BꞱ, 6134) ⸠ | M(B, c) = (cA, 2) |
| (cbdf, cAꞱ, 61342) ⸠ | M(c, c) = ВЫБРОС |
| (bdf, AꞱ, 61342) ⸠ | M(A, b) = (bC, 1) |
| (bdf, bCꞱ, 613421) ⸠ | M(b, b) = ВЫБРОС |
| (df, CꞱ, 613421) ⸠ | M(C, d) = (dD, 3) |
| (df, dDꞱ, 6134213) ⸠ | M(d, d) = ВЫБРОС |
| (f, DꞱ, 6134213) ⸠ | M(D, f) = (f, 4) |
| (f, fꞱ, 61342134) ⸠ | M(f, f) = ВЫБРОС |
| (ε, Ʇ, 61342134) ⸠ | M(Ʇ, ε) = ДОПУСК |

**Лабораторная работа №11: Реализовать управляющую таблицу M Для LL(k) анализатора.**

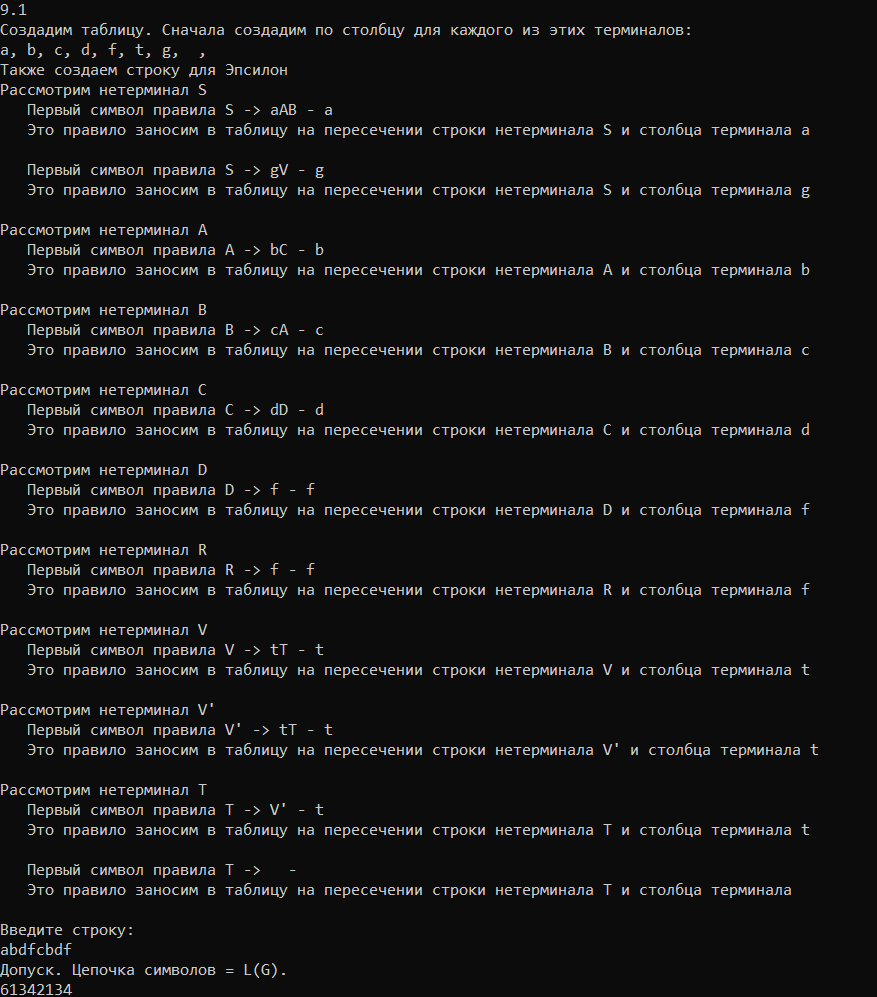
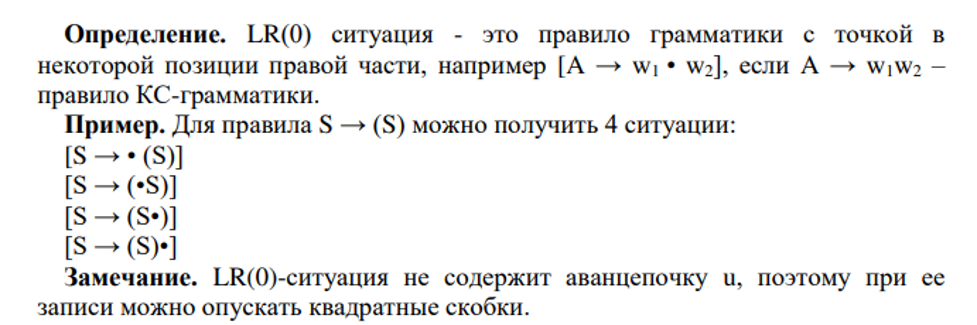
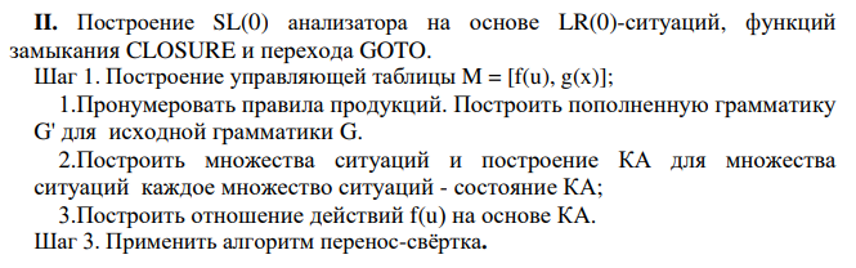


Рисунок 13 – Построение управляющей таблицы для LL(K)-анализатора

|  |
| --- |
| var LL = new Grammar(new List<Symbol>() { "a", "b", "c", "d", "f", "t", "g", " " },  new List<Symbol>() { "S", "A", "B", "C", "D", "R", "V", "V'", "T" },  "S");  LL.AddRule("A", new List<Symbol>() { "b", "C" });  LL.AddRule("B", new List<Symbol>() { "c", "A" });  LL.AddRule("C", new List<Symbol>() { "d", "D" });  LL.AddRule("D", new List<Symbol>() { "f" });  LL.AddRule("R", new List<Symbol>() { "f" });  LL.AddRule("S", new List<Symbol>() { "a", "A", "B" });  LL.AddRule("S", new List<Symbol>() { "g", "V" });  LL.AddRule("V", new List<Symbol>() { "t", "T" });  LL.AddRule("V'", new List<Symbol>() { "t", "T" });  LL.AddRule("T", new List<Symbol>() { "V'" });  LL.AddRule("T", new List<Symbol>() { " " });  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).");  } |

**Практическая работа №4 (12-16 лаб.)**





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:

G = (T, V, P, S), где

T = {a, b, c, d, f, t, g}, V = {S, A, B, C, D, R, V, V’}, S0 = S,

P = {

p1: A → bC,

p2: B → cA,

p3: C → dD,

p4: D → f,

p5: R → f,

p6: S → aAB,

p7: S → gV,

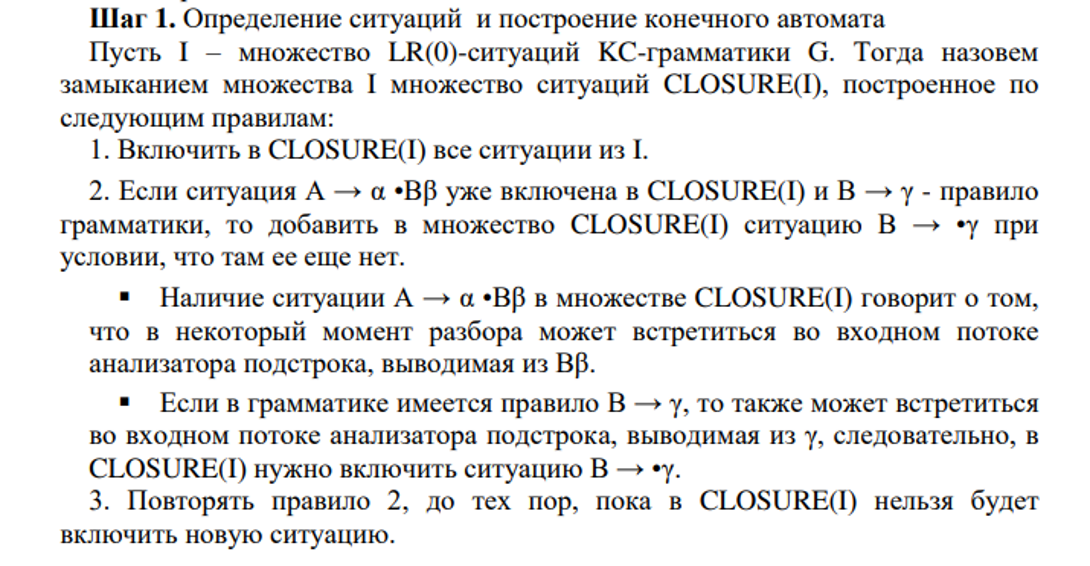
p8: V → t,

p9: V → tV’,

p10: V’ → t,

p11: V’ → tV’

}



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

G = (T, V, P, S, S’), где

T = {a, b, c, d, f, t, g}, V = {S, A, B, C, D, R, V, V’}, S0 = S,

P = {

p0: S → S’,

p1: A → bC,

p2: B → cA,

p3: C → dD,

p4: D → f,

p5: R → f,

p6: S → aAB,

p7: S → gV,

p8: V → t,

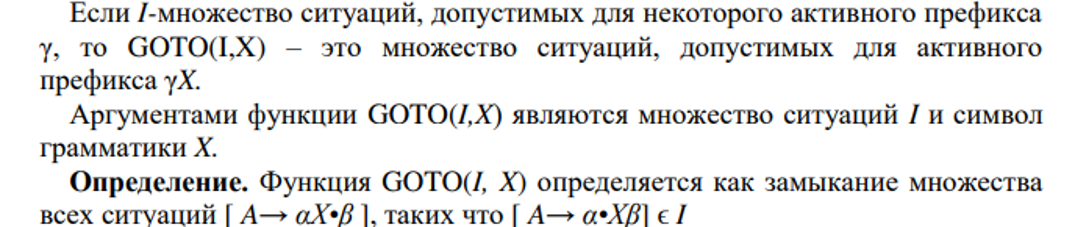
p9: V → tV’,

p10: V’ → t,

p11: V’ → tV’

}

1. I0 = {S’→∙S}
2. I0 = {S’→∙S, S→∙aAB, S→∙gV}



I0 = {S’→∙S, S→∙aAB, S→∙gV}

GOTO(I0, a) = {S→a∙AB, A→∙bC} = I1

GOTO(I0, g) = {S→g∙V, V→∙t, V→∙tV’} = I2

GOTO(I0, S) = {S’→S∙} = I3

GOTO(I1, b) = {A→b∙C, C→∙dD} = I4

GOTO(I1, A) = {S→aA∙B, B→∙cA} = I5

GOTO(I2, t) = {V→t∙, V→t∙V’, V’→∙t, V’→∙tV’} = I6

GOTO(I2, V) = {S→gV∙} = I7

GOTO(I4, d) = {C→d∙D, D→∙f} = I8

GOTO(I4, C) = {A→bC∙} = I9

GOTO(I5, c) = {B→c∙A, A→∙bC} = I10

GOTO(I5, B) = {S→aAB∙} = I11

GOTO(I6, t) = {V’→t∙, V’→t∙V’, V’→∙t, V’→∙tV’} = I12

GOTO(I6, V’) = {V→tV’∙} = I13

GOTO(I8, f) = {D→f∙} = I14

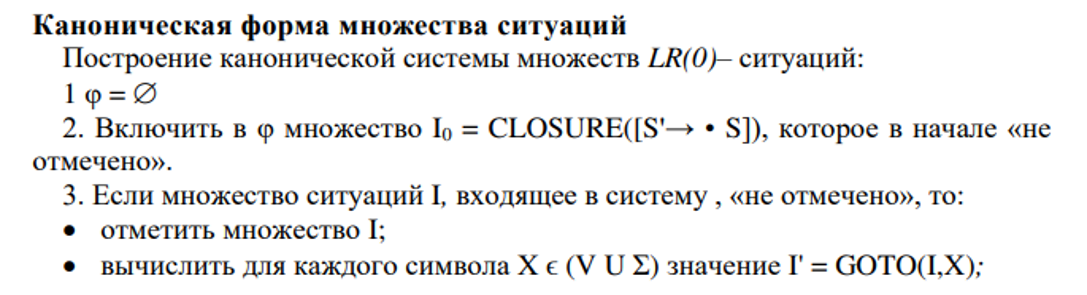
GOTO(I8, D) = {C→dD∙} = I15

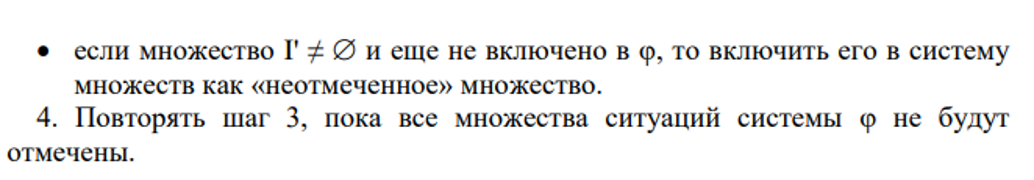
GOTO(I10, b) = {A→b∙C, C→∙dD} = I4 – уже отмечено

GOTO(I10, A) = {B→cA∙} = I16

GOTO(I12, t) = {V’→t∙, V’→t∙V’, V’→∙t, V’→∙tV’} = I12 – уже отмечено

GOTO(I12, V’) = {V’→tV’∙} = I17





φ = {

I0 = {S’→∙S, S→∙aAB, S→∙gV},

I1 = {S→a∙AB, A→∙bC},

I2 = {S→g∙V, V→∙t, V→∙tV’},

I3 = {S’→S∙},

I4 = {A→b∙C, C→∙dD},

I5 = {S→aA∙B, B→∙cA},

I6 = {V→t∙, V→t∙V’, V’→∙t, V’→∙tV’},

I7 = {S→gV∙},

I8 = {C→d∙D, D→∙f},

I9 = {A→bC∙},

I10 = {B→c∙A, A→∙bC},

I11 = {S→aAB∙},

I12 = {V’→t∙, V’→t∙V’, V’→∙t, V’→∙tV’},

I13 = {V→tV’∙},

I14 = {D→f∙},

I15 = {C→dD∙},

I16 = {B→cA∙},

I17 = {V’→tV’∙} }

Используя каноническую систему LR(0)–множеств, можно представить функцию GOTO в виде диаграммы детерминированного конечного автомата. Диаграмма переходов ДКА для активных префиксов грамматики:

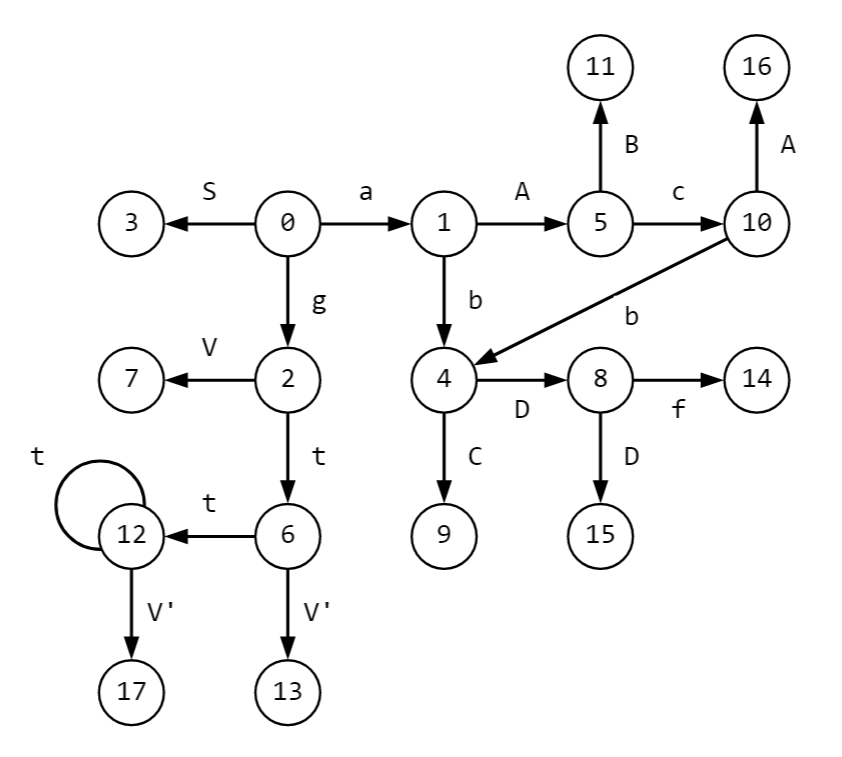
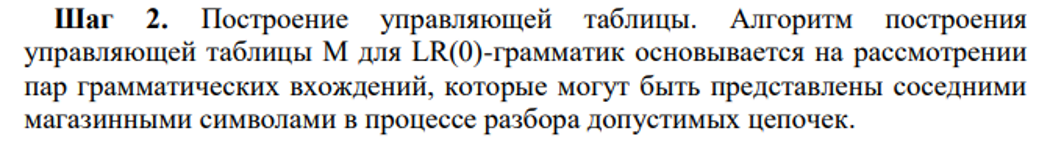


Рисунок 14 – Диаграмма переходов автомата



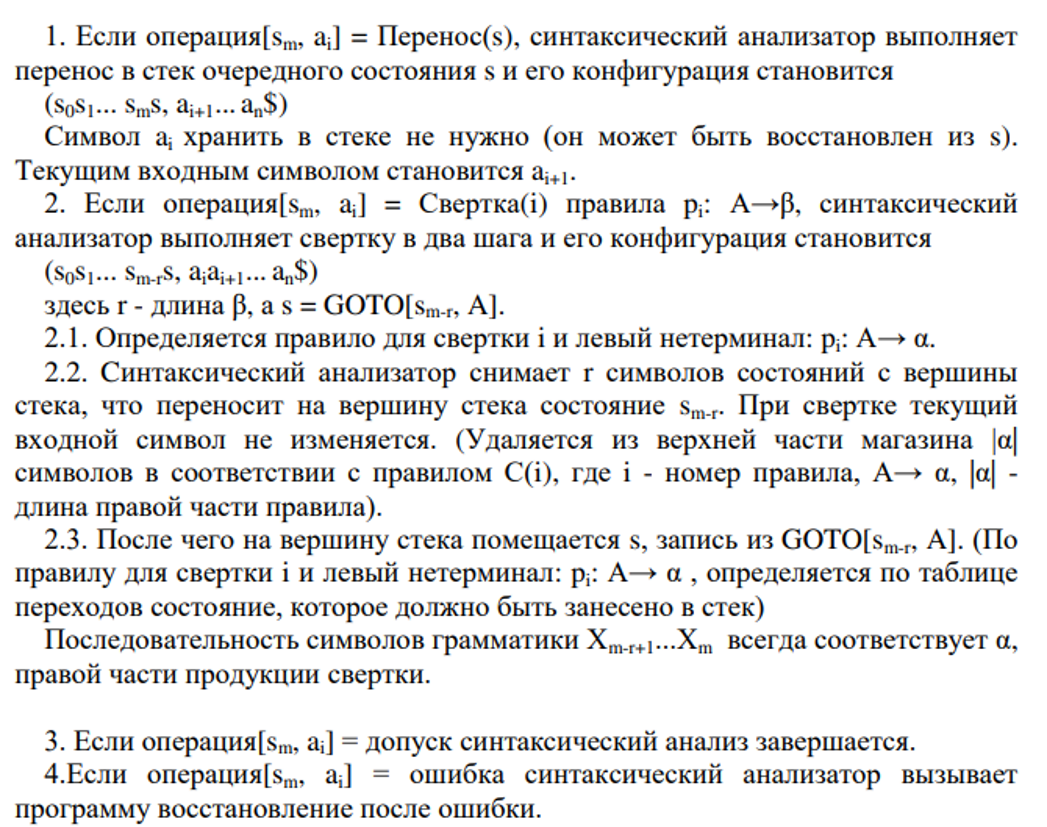
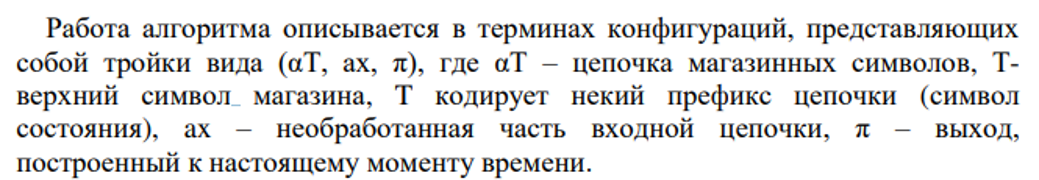


Таблица 4 — Управляющая табллица для LR(0)-анализатора

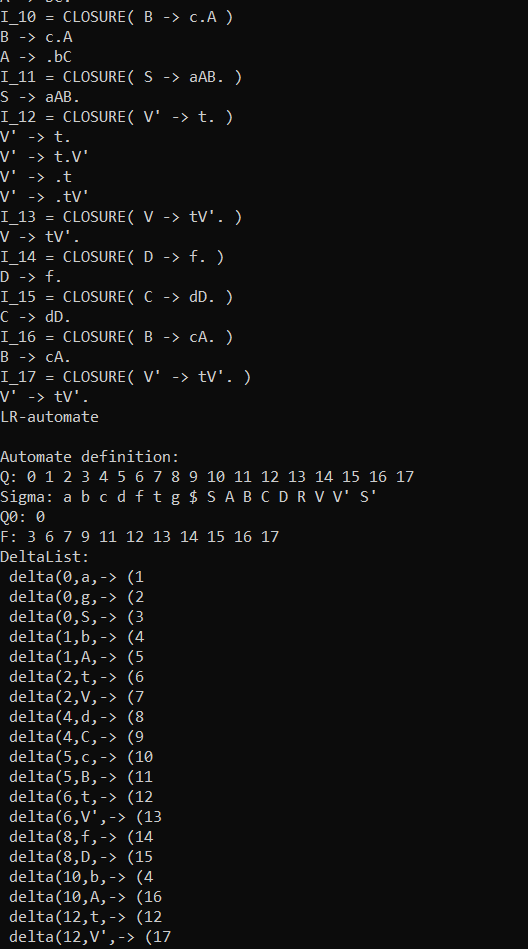
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Управляющая таблица | | | | | | | | | | | | | | | | |
| I | f(u) | | | | | | | | g(x) | | | | | | | |
| a | b | c | d | f | t | g | Ʇ | S | A | B | C | D | R | V | V’ |
| 0 | П1 |  |  |  |  |  | П2 |  | 3 |  |  |  |  |  |  |  |
| 1 |  | П4 |  |  |  |  |  |  |  | 5 |  |  |  |  |  |  |
| 2 |  |  |  |  |  | П6 |  |  |  |  |  |  |  |  | 7 |  |
| 3 |  |  |  |  |  |  |  | Д |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  | 9 | 8 |  |  |  |
| 5 |  |  | П10 |  |  |  |  |  |  |  | 11 |  |  |  |  |  |
| 6 | С8 | С8 | С8 | С8 | С8 | П12 | С8 | С8 |  |  |  |  |  |  |  | 13 |
| 7 | С7 | С7 | С7 | С7 | С7 | С7 | С7 | С7 |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  | П14 |  |  |  |  |  |  |  | 15 |  |  |  |
| 9 | С1 | С1 | С1 | С1 | С1 | С1 | С1 | С1 |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  | 16 |  |  |  |  |  |  |
| 11 | С6 | С6 | С6 | С6 | С6 | С6 | С6 | С6 |  |  |  |  |  |  |  |  |
| 12 | С10 | С10 | С10 | С10 | С10 | П12 | С10 | С10 |  |  |  |  |  |  |  | 17 |
| 13 | С9 | С9 | С9 | С9 | С9 | С9 | С9 | С9 |  |  |  |  |  |  |  |  |
| 14 | С4 | С4 | С4 | С4 | С4 | С4 | С4 | С4 |  |  |  |  |  |  |  |  |
| 15 | С3 | С3 | С3 | С3 | С3 | С3 | С3 | С3 |  |  |  |  |  |  |  |  |
| 16 | С2 | С2 | С2 | С2 | С2 | С2 | С2 | С2 |  |  |  |  |  |  |  |  |
| 17 | С11 | С11 | С11 | С11 | С11 | С11 | С11 | С11 |  |  |  |  |  |  |  |  |

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



Распознавание цепочки gt:

(0, gtꞱ, ε) ⸠П (0 2, tꞱ, ε) ⸠П (0 2 6, Ʇ, ε) ⸠С (0 2 7, Ʇ, 8) ⸠С (0 3, Ʇ, 8 7) ⸠Д



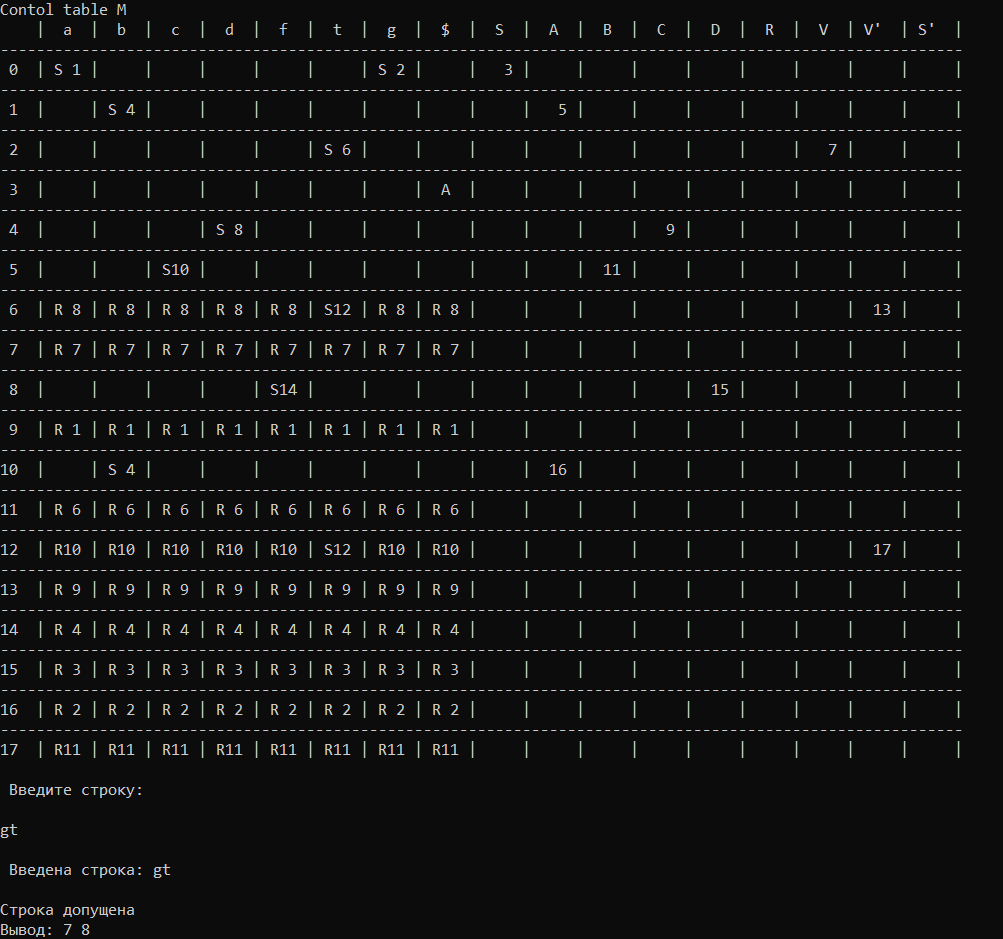


Рисунок 15 – Пример управляющей таблицы для LR(0)-анализатора