МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Отчёт по лабораторным работам

по дисциплине «Системы программирования»

Выполнил:
Студент группы М8О-201Б-21
Голов Александр Юрьевич

Принял:
Доцент кафедры 806
Киндинова Виктория Валерьевна

Оценка:_____ **Дата:**_____

Москва, 2023

Практическая работа №1 (1-3 лаб.)

Лабораторные работы №1-2

Формулировка задания:

Спроектировать грамматику для трёх заданных паттернов. Составить наоснове 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, S_0) = G(

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

{S_0, A, B, C, D},

{p_1, p_2, p_3, p_4, p_5},S_0

)

Правила регулярной грамматики:p_1**: S_0 →**

192.168.1.**A**

P_2**: A** → 0**B** | 1**B** | **...** | 9**B** p_3**: B** → 0**C** | 1**C** |

... | 9**C** | **C**p_4**: C** → 0**D** | 1**D** | **...** | 9**D** | **D**p_5**: D**

→ ε

Пример цепочек:

S_0 =>^{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

S_0 =>^{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

S_0 =>^{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(KA) = L(G)

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

Q = {**S_0**, **A**, B, C **q_f**}, **Σ** = {0-9, 192.168.1.}, **S_0**= **S_0**, **F** = **q_f**,

δ = {

1. δ(**S_0**, 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) = {$\mathbf{q_f}$},

23. δ(**C**, 0) = {**D**},

…

32. δ(**C**, 9) = {**D**},

33. δ(**C**, ε) = {$\mathbf{q_f}$},

34. δ(**D**, ε) = {$\mathbf{q_f}$}

}

Примеры конфигурации КА:

1. (**S_0**, 192.168.1.1) $\vdash^1$ (**A**, 1) $\vdash^3$ (**B**, ε) $\vdash^{22}$ ($\mathbf{q_f}$, ε)

2. (**S_0**, 192.168.1.12) $\vdash^1$ (**A**, 12) $\vdash^3$ (**B**, 2) $\vdash^{14}$(**C**, ε) $\vdash^{33}$ ($\mathbf{q_f}$, ε)

3. (**S_0**, 192.168.1.123) $\vdash^1$ (**A**, 123) $\vdash^3$ (**B**, 23) $\vdash^{14}$(**C**, 3) $\vdash^{26}$ (**D**, ε) $\vdash^{34}$ ($\mathbf{q_f}$, ε)

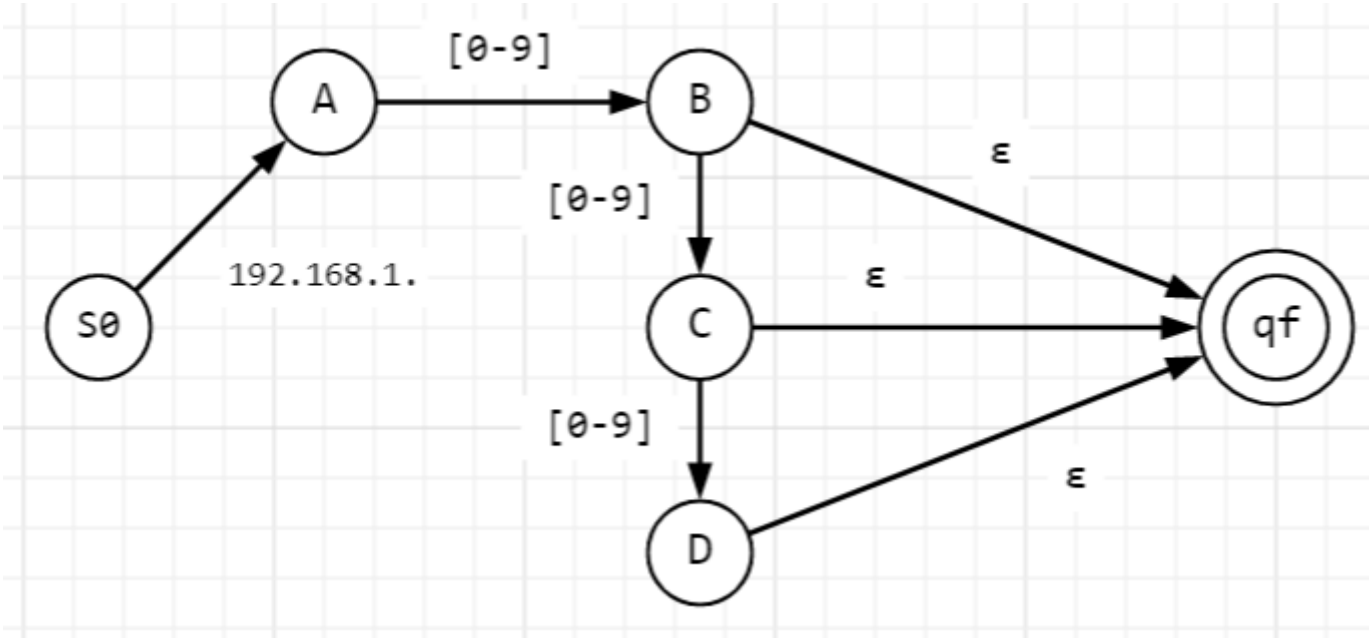

Рисунок 1 — Диаграмма автомата

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");
```

```
Enter line to execute 1:
192.168.1.25
Length: 12
 i :12
curr: qf
chineSymbol belongs to language
```

Рисунок 2 — Вывод конечного автомата

Лемма о накачке:

Формальное утверждение. Длина накачки p принимается более длины самой длинной цепочки языка L, такое что цепочка w из L длины по меньшей мере p может быть записана как w = xyz, где y – это подцепочка, которую можно накачать (удалить или повторить произвольное число раз, так что результат останется в L).

$\forall L \subseteq \Sigma^*(\text{regular}(L)) \Rightarrow$ 1. **любой** язык $L \subseteq \Sigma^*$ регулярный **если**

$(\exists p \geq 1$ 2. **существует** целое $p \geq 1$ такое что

$(\forall w \in L\ (|w| \geq p) \Rightarrow$ 3. для **любой цепочки языка** $|w| \geq p$

$(\exists x,y,z \in \Sigma^*\ (w = xyz \Rightarrow$ 4. **найдется** $w = xyz$ такое что

$(|y| \geq 1 \wedge |xy| \leq p \wedge i \geq 0 \wedge (xy^i z \in L))$ 5. y **повторяется** i раз

$))$

$))$

$)$

), на x и z ограничений не накладывается.

$$\forall L \subseteq \Sigma^*(\text{regular}(L) \Rightarrow (\exists p \geq 1(\forall w \in L((|w| \geq p) \Rightarrow (\exists x,y,z \in \Sigma^*(w = xyz \Rightarrow (|y| \geq 1 \wedge |xy| \leq p \wedge \forall i \geq 0(xy^i z \in L)))))))$$

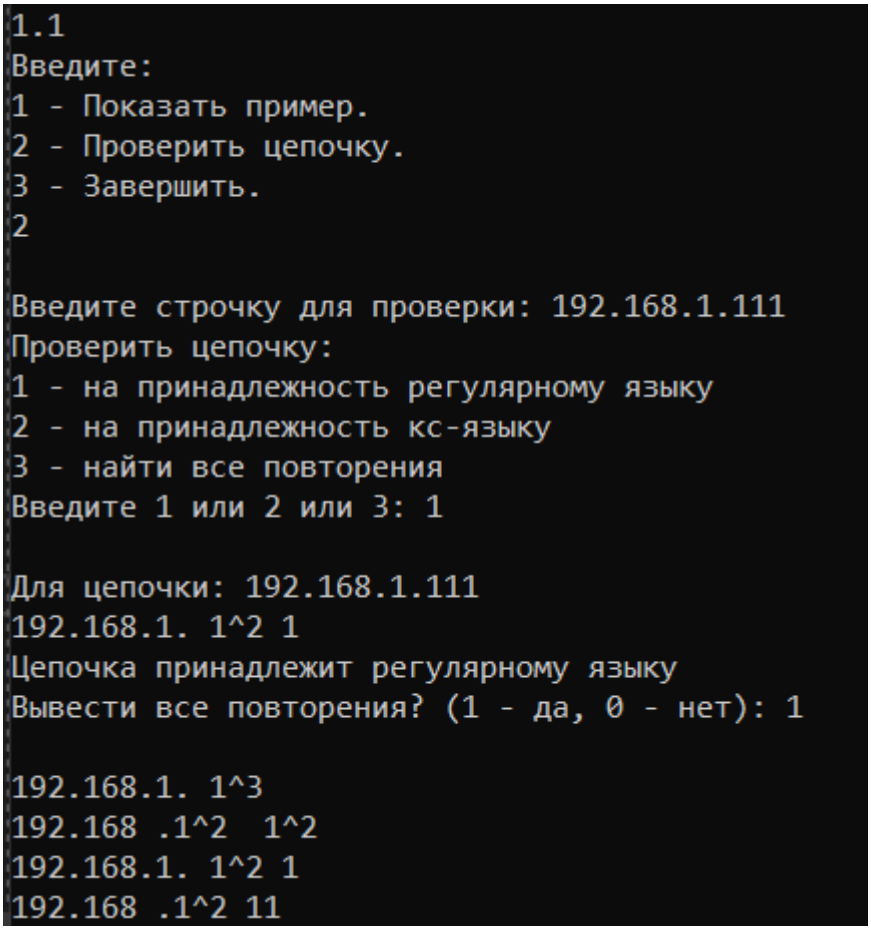

Рисунок 3 – Лемма о накачке

Проектирование ДКА:

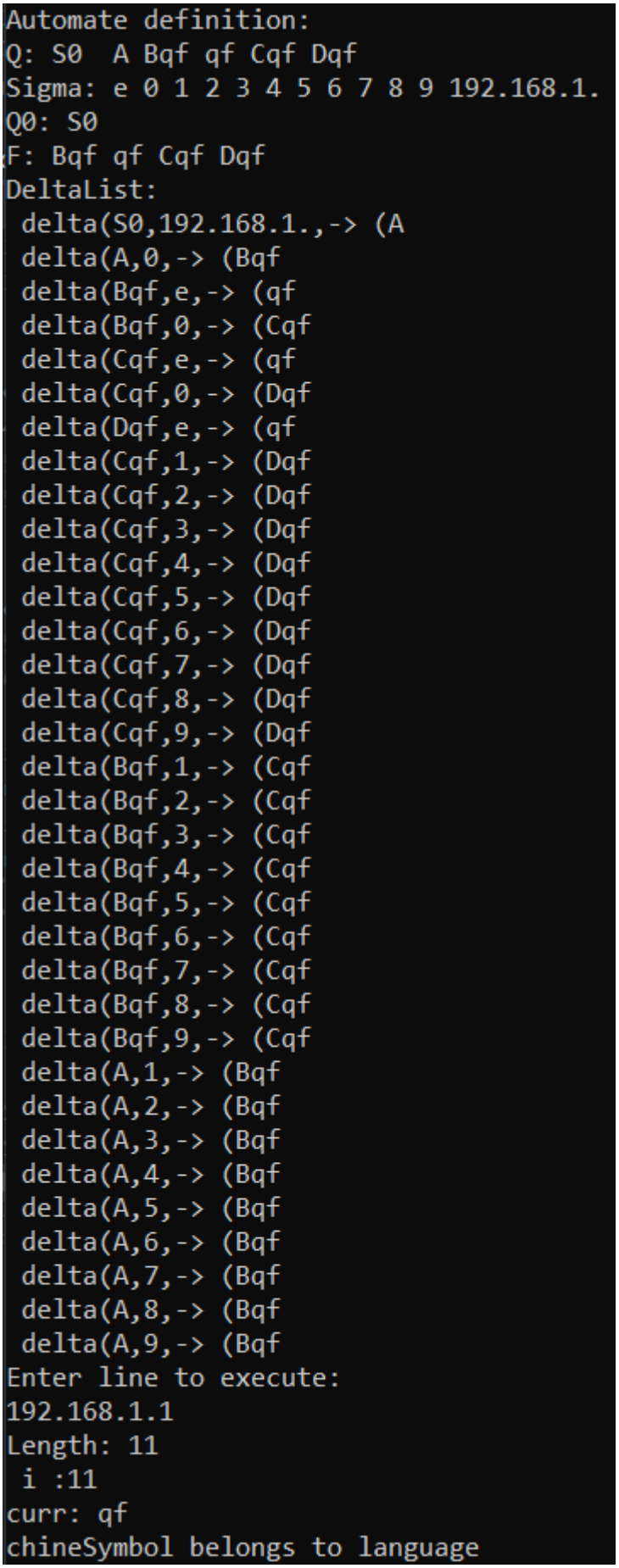

Рисунок 4 – Детерменированный конечный автомат

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}")

$G(T, V, P, S_0) = G(\{+, -, 0, \ldots, 9\}, \{S_0, A_0, A_1, A_2, B_0, \ldots, B_5, C_0, \ldots, C_6\}, \{p_1, \ldots, p_{17}\}, S_0)$

Правила регулярной грамматики:

p_1: $\mathbf{S_0} \rightarrow +\mathbf{A_0} \mid \mathbf{A_0}$

p_2: $\mathbf{A_0} \rightarrow 0\mathbf{A_1} \mid \ldots \mid 9\mathbf{A_1} \mid \mathbf{A_1}$

p_3: $\mathbf{A_1} \rightarrow 0\mathbf{A_2} \mid \ldots \mid 9\mathbf{A_2}$

p_4: $\mathbf{A_2} \rightarrow -\mathbf{B_0} \mid \mathbf{B_0}$

p_5: $\mathbf{B_0} \rightarrow 0\mathbf{B_1} \mid \ldots \mid 9\mathbf{B_1}$

…

p_9: $\mathbf{B_4} \rightarrow 0\mathbf{B_5} \mid \ldots \mid 9\mathbf{B_5} \mid \mathbf{B_5}$

p_{10}: $\mathbf{B_5} \rightarrow -\mathbf{C_0} \mid \mathbf{C_0}$

p_{11}: $\mathbf{C_0} \rightarrow 0\mathbf{C_1} \mid \ldots \mid 9\mathbf{C_1}$

…

p_{16}: $\mathbf{C_5} \rightarrow 0\mathbf{C_6} \mid \ldots \mid 9\mathbf{C_6} \mid \mathbf{C_6}$

p_{17}: $\mathbf{C_6} \rightarrow \varepsilon$

Пример цепочек:

$\mathbf{S_0} =>^1 +\mathbf{A_0} =>^2 +9\mathbf{A_1} =>^3 +91\mathbf{A_2} =>^4 +91-\mathbf{B_0} =>^5 +91-9\mathbf{B_1} =>^6 +91-96\mathbf{B_2} =>^7 +91-967\mathbf{B_3} =>^8 +91-9678\mathbf{B_4} =>^9 +91-96789\mathbf{B_5} =>^{10} +91-96789\mathbf{C_0} =>^{11} +91-967896\mathbf{C_1} =>^{12} +91-9678967\mathbf{C_2} =>^{13} +91-96789671\mathbf{C_3} =>^{14} +91-967896710\mathbf{C_4} =>^{15} +91-9678967101\mathbf{C_5} =>^{16} +91-9678967101\mathbf{C_6} =>^{17} +91-9678967101$

Конечный автомат:

L(КА) = L(G)

КА = ($\mathbf{Q}$, $\mathbf{\Sigma}$, $\boldsymbol{\delta}$, $\mathbf{S_0}$, $\mathbf{F}$), где

$\mathbf{Q} = \{ \mathbf{S_0}, \mathbf{A_0}, \mathbf{A_1}, \mathbf{A_2}, \mathbf{B_0}, \ldots, \mathbf{B_5}, \mathbf{C_0}, \ldots, \mathbf{C_6}, \mathbf{q_f}\}$,
$\mathbf{\Sigma} = \{+, -, 0\text{-}9\}$, $\mathbf{S_0} = \mathbf{S_0}$, $\mathbf{F} = \mathbf{q_f}$,

$\boldsymbol{\delta}$ = {

1. $\delta(\mathbf{S_0}, +) = \{\mathbf{A_0}\}$,

2. $\delta(\mathbf{S_0}, \varepsilon) = \{\mathbf{A_0}\}$,

3. $\delta(\mathbf{A_0}, 0) = \{\mathbf{A_1}\}$,

…

13. $\delta(\mathbf{A_0}, \varepsilon) = \{\mathbf{A_1}\}$,

14. $\delta(\mathbf{A_0}, \varepsilon) = \{\mathbf{A_2}\}$,

15. $\delta(\mathbf{A_1}, 0) = \{\mathbf{A_2}\}$,

…

25. $\delta(\mathbf{A_2}, -) = \{\mathbf{B_0}\}$,

26. $\delta(\mathbf{A_2}, \varepsilon) = \{\mathbf{B_0}\}$,

27. $\delta(\mathbf{B_0}, 0) = \{\mathbf{B_1}\}$,

…

77. $\delta(\mathbf{B_4}, \varepsilon) = \{\mathbf{B_5}\}$,

78. $\delta(\mathbf{B_5}, -) = \{\mathbf{C_0}\}$,

79. $\delta(\mathbf{B_5}, \varepsilon) = \{\mathbf{C_0}\}$,

80. $\delta(\mathbf{C_0}, 0) = \{\mathbf{C_1}\}$,

…

140. $\delta(\mathbf{C_5}, \varepsilon) = \{\mathbf{C_6}\}$,

141. $\delta(\mathbf{C_6}, \varepsilon) = \{\mathbf{q_f}\}$

}

Примеры конфигурации КА:

$(\mathbf{S_0}, +91\text{-}9678967101) \vdash^{1} (\mathbf{A_0}, 91\text{-}9678967101) \vdash^{12} (\mathbf{A_1}, 1\text{-}9678967101) \vdash^{14}$
$(\mathbf{A_2}, -9678967101) \vdash^{25} (\mathbf{B_0}, 9678967101) \vdash^{36} (\mathbf{B_1}, 678967101) \vdash^{43}$
$(\mathbf{B_2}, 78967101) \vdash^{54} (\mathbf{B_3}, 8967101) \vdash^{65} (\mathbf{B_4}, 967101) \vdash^{76} (\mathbf{B_5}, 967101) \vdash^{79}$
$(\mathbf{C_0}, 967101) \vdash^{89} (\mathbf{C_1}, 67101) \vdash^{96} (\mathbf{C_2}, 7101) \vdash^{107} (\mathbf{C_3}, 101) \vdash^{111} (\mathbf{C_4}, 01) \vdash^{120}$
$(\mathbf{C_5}, 1) \vdash^{131} (\mathbf{C_6}, \varepsilon) \vdash^{141} (\mathbf{q_f}, \varepsilon)$

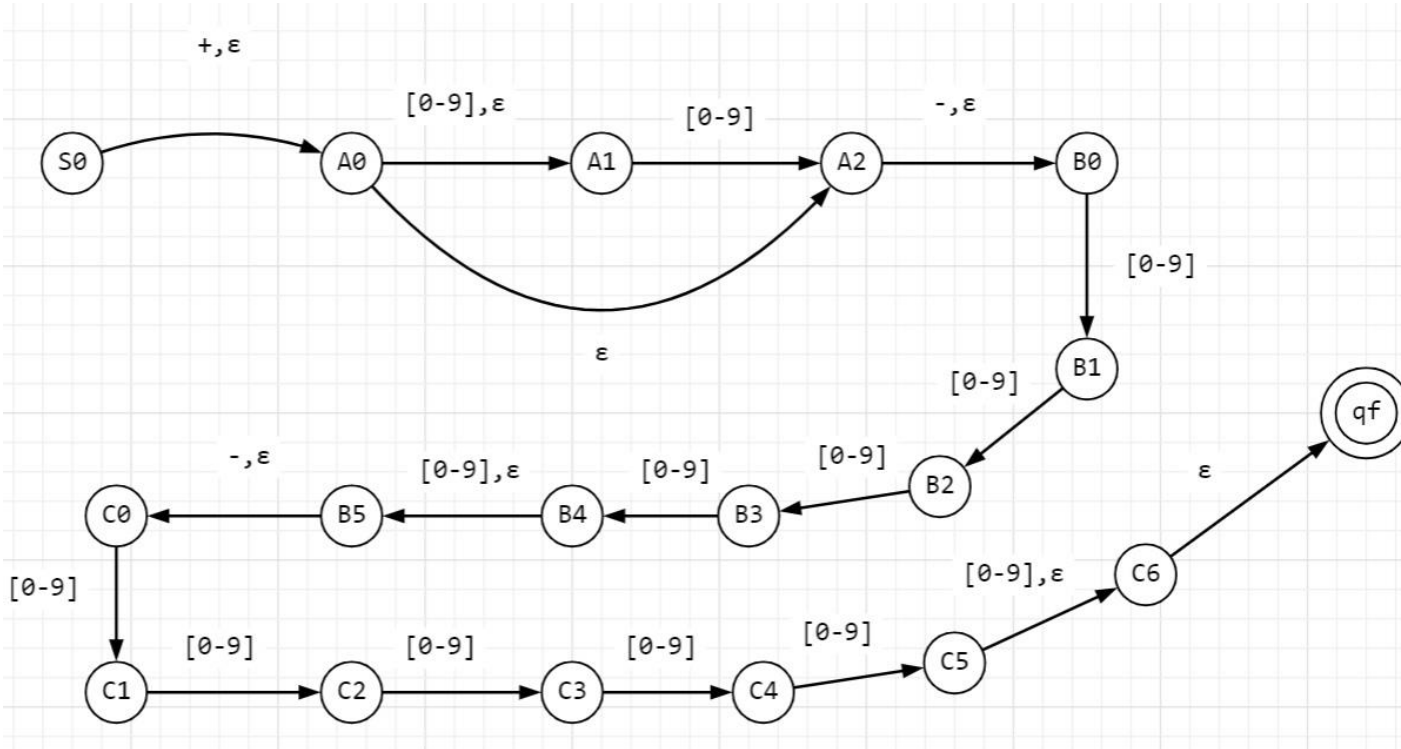

Рисунок 5 – Диаграмма преобразованного атвомата

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");
```

```
Enter line to execute 2:
+91-9678967101
Length: 14
 i :14
curr: qf
chineSymbol belongs to language
```

Рисунок 6 – Проверка цепочки на принадлежность к языку

Лемма о накачке:

```
1.1
Введите:
1 - Показать пример.
2 - Проверить цепочку.
3 - Завершить.
2

Введите строчку для проверки: +12-3344556677
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: +12-3344556677
+12- 3^2 44556677
Цепочка принадлежит регулярному языку
Вывести все повторения? (1 - да, 0 - нет): 1

+12-33445566 7^2
+12- 3^2 44556677
+12-33 4^2 556677
+12-3344 5^2 6677
+12-334455 6^2  7^2
+12- 3^2  4^2 556677
+12- 3^2 445566 7^2
+12-33 4^2  5^2 6677
+12-33 4^2 5566 7^2
+12-3344 5^2  6^2  7^2
+12-334455 6^2 77
+12- 3^2  4^2  5^2 6677
+12- 3^2  4^2 5566 7^2
+12- 3^2 4455 6^2  7^2
+12-33 4^2  5^2  6^2  7^2
+12-3344 5^2 66 7^2
+12-3344 5^2  6^2 77
```

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

Определение. Символ $a \in T \cup V$ называется *бесполезным* в КС-грамматике G, если он *непроизводящий* или *недостижимый*.

Рассмотрим алгоритмы и их применение.

1. *Устранение непроизводящих символов*

Определение. Нетерминальный символ $A \in V$ называется *производящим*, если из него можно вывести терминальную цепочку, т.е. если существует вывод $A \Rightarrow^{+} \alpha$, где $\alpha \in T^{+}$. В противном случае символ называется непроизводящим.

Построить каноническую форму множества ситуаций.

Построить управляющую таблицу для функции перехода g(x) 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. Составить множество V_p нетерминалов, для которых найдется хотя бы одно правило, правая часть которого не содержит нетерминалов.
2. 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. Если на шаге 2, V_p больше не пополняется, то V_p содержит все производящие нетерминалы грамматики, а все нетерминалы не попавшие в него являются непроизводящими.

```
i = 0
V^i_p = Ø
do
   i = i + 1;
   foreach (A→ α ∈ P, α ∈(T ∪ V^(i-1)_p)^+ )
      if (A ∉V^(i-1)_p )
        V^i_p = V^(i-1)_p ∪ {A}      // V^i_p = {A | (A→ α) ∈P, α ∈(T ∪ V^(i-1)_p)^+}
      end
   end foreach
while (V^i_p ≠ V^(i-1)_p)

if (S ∉ V^i_p)  S - непроизводящий символ, то определить S ∈ V^i_p
 n = 0
 foreach (A ∈ V^i_p)
    if | α | ≥ n, (A→ α ∈ P ))        при n = 0 если есть A→ ε
       n = | α |
       S = A
    end
 end foreach
end
```

1) V^1 = {**R, T, H, G, V**} – для этих нетерминалов нашлось хотя бы одно правило, правая часть которого не содержит нетерминалов
2) V^2 = {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) V^3 = {R, T, H, G, V, D, S, **C**}
4) V^4 = {R, T, H, G, V, D, S, C, **A**}
5) V^5 = {R, T, H, G, V, D, S, C, A, **B**} => все нетерминалы производящие

Тогда P’ = P = {p_1, …, p_{12}}

p_1: A → bC,

p_2: B → cA,

p_3: C → dD,

p_4: D → R,

p_5: R → f,

p_6: S → aABT,

p_7: T → ε,

p_8: H → z,

p_9: G → r,

p_{10}: S → gV,

p_{11}: V → Vt,

p_{12}: 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}

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

VT_r - *множество достижимых символов* (нетерминальных и терминальных).

Утверждение. Если нетерминал A в левой части правила грамматик G является достижимым, то достижимы и все символы α правой части этого правила $A \to \alpha$. $VT_r = \{x \mid A \to \alpha, x \in \alpha, \alpha \subseteq (V \cup T)^*\}$

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

Шаг 2. Построить P", T', V'.

$P'' = \emptyset$

foreach $(A \to \{X_1, X_2, ..., X_n\}, A, X_1, X_2, ..., X_n \in VT^i_r) \in P$

$P'' := P'' \cup (A \to \{X_1, X_2, ..., X_n\})$

end foreach

$T' := T \cap VT^i_r$

$V' := V \cap VT^i_r$

1) $VT^1 = \{S\}$
2) $VT^2 = \{S, \mathbf{a, A, B, T, g, V}\}$
3) $VT^3 = \{S, a, A, B, T, g, V, \mathbf{b, C, c, ε, t}\}$
4) $VT^4 = \{S, a, A, B, T, g, V, b, C, c, ε, t, \mathbf{d, D}\}$
5) $VT^5 = \{S, a, A, B, T, g, V, b, C, c, ε, t, d, D, \mathbf{R}\}$
6) $VT^6 = \{S, a, A, B, T, g, V, b, C, c, ε, t, d, D, R, \mathbf{f}\}$

Так, в VT не вошли символы {z, r, H, G}. Тогда P'' = $\{p_1, \ldots, p_{10}\}$

p_1: $A \to bC$,

p_2: $B \to cA$,

p_3: $C \to dD$,

p_4: $D \to R$,

p_5: $R \to f$,

p_6: $S \to aABT$,

p_7: $T \to ε$,

p_8: $S \to gV$,

p_9: $V \to Vt$,

p_{10}: $V \to t$

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

3. Устранить из КС-грамматики ε–правила

КС-грамматика называется *неукорачивающей* КС-грамматикой (НКС-грамматикой, КС-грамматикой без ε–правил) при условии, что P не содержит $S \to \varepsilon$ и S не встречается в правах частях остальных правил.

В грамматике с правилами вида $A \to \varepsilon$ длина выводимой цепочки при переходе от k-го шага к (k+1)-му уменьшается. Поэтому грамматики с правилами вида $A \to \varepsilon$ называются укорачивающими.

Восходящий синтаксический разбор в укорачивающих грамматиках сложнее по сравнению с разбором в неукорачивающих грамматиках, т.к. при редукции необходимо отыскать такой фрагмент входной цепочки, в которую можно вставить пустой символ.

Для произвольной КС-грамматики, порождающей язык без пустой цепочки, можно построить эквивалентную неукорачивающую КС-грамматику.

Построение множества V_ε укорачивающих (обнуляемых) нетерминалов.

1. Алгоритм построение множества укорачивающих нетерминалов

Утверждение. Нетерминал A - укорачивающий (обнуляемый) если $A \Rightarrow^* \varepsilon$.

1.A - укорачивающий для правила $A \to \varepsilon$ (см. Шаг 1.).

2.A - укорачивающий для правила $A \to B_1B_2 \ldots B_k$, если каждый нетерминал B_1 в правиле укорачивающий (см. Шаг 2.).

Вход: КС грамматика G = (T, V, P, S).

Выход: $V_\varepsilon = \{ A \mid A \Rightarrow^+ \varepsilon, A \in V \}$

Шаг 1. Построить множество V^i_ε - укорачивающих нетерминалов для правил вида $A \to \varepsilon$.

```
i = 0
V^i_ε = Ø
foreach (A ∈ V)
   if (A → ε) ∈P // A → ε правило укорачивающиеся
      V^i_ε := V^i_ε ∪ {A}
   end
end foreach
```

1) $V^0 = \{T\}$ - (множество укорачивающих нетерминалов для правил вида $A \to \varepsilon$)

Шаг 2. Построить множество V^i_ε - укорачивающих нетерминалов для правил вида $A \to \varepsilon$. $A \to B_1B_2 \ldots B_k$, если каждый нетерминал B_1 в правиле укорачивающий. $V^i{}_\varepsilon = V^{i-1}{}_\varepsilon \cup \{A \mid (A \to \alpha) \in P \text{ и } \alpha \in (V^{i-1}{}_\varepsilon)^+\}$. $\alpha \in (V^{i-1}{}_\varepsilon)^+$ - рассматриваются только правила, содержащие в правых частях только нетерминальные укорачивающие символы: $A \to \alpha$, например $A \to CDF$.

```
do
   i = i + 1
   V^{i-1}_ε = V^i_ε
   foreach ((A→ α)∈P и α ∈(V^{i-1}_ε)^+)
      V^i_ε := V^{i-1}_ε ∪ A
   end
while(V^{i-1}_ε ≠ V^i_ε)
```

Необходимо построить множество V^1 укорачивающих нетерминалов для правил вида $A \to$

$B_1B_2…B_k$, если каждый нетерминал B_i в правиле укорачивающий. Однако в данном примере таких правил нет.

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

2. Алгоритм устранения эпсилон-правил

Алгоритм устранения ε–правил в КС-грамматике основан на использовании множества укорачивающих нетерминалов. Алгоритм преобразует КС-грамматику с ε–правилами в эквивалентную КС-грамматику. Пусть $X_i \in \{X \mid (T \cup V) \cup \{\varepsilon\}, 0 < i \leq k\}$. Построить из $\{X_1, X_2, ..., X_k\}$ множество цепочек в которых, либо присутствует, либо устранён каждый из нетерминалов V_ε.

Вход: $G' = (T, V^i_\varepsilon, P, S)$
Выход: $G'' = (T, V', P', S')$

$P' = \emptyset$
foreach $(A \to X_1, X_2, ..., X_k \in P, X_1 \neq \varepsilon)$
  $V'_\varepsilon = \{ [X_1, X_2, ..., X_k] \cap V_\varepsilon \mid Xi < X_m \}$ V'_ε - упорядоченное мн-во
  if $(V'_\varepsilon = \emptyset)$
    $P' = P' \cup A \to X_1, X_2, ..., X_k$
  else построить $A \to \{Y_1, Y_2, ..., Y_k\}$
    $Y = \emptyset$
    foreach $a \in \{\alpha = \{ X_i, ..., X_m \} \subset V'^{+}_\varepsilon \mid |\alpha| \leq |V'_\varepsilon|$ и $Xi < Xm \}$
      foreach $v \in a$
        foreach $(X_i \in \{X_1, X_2, ..., X_k\}, A \neq X_i)$ правило $A \to A$ бессмысленно
          if $X_i = v$
            $Y_i = \varepsilon$
          end
          $Y = Y \cup Y_i$
        end foreach
        $P' = P' \cup (A \to \{Y_1, Y_2, ..., Y_k\}$
      end foreach
    end foreach
  end
end foreach

if $(S \in V_\varepsilon)$
  $V' = V' \cup \{S'\}$
  $P' = P' \cup (S' \to S \mid \varepsilon)$
end
Положить $G'' = (T, V', P', S')$.

G’ = {(a, b, c, d, f, ε, t, g),(S, A, B, C, D, R, V), P, S}, где P = {p_1, …, p_9}

p_1: A → bC,

p_2: B → cA,

p_3: C → dD,

p_4: D → R,

p_5: R → f,

p_6: S → aAB,

p_7: S → gV,

p_8: V → Vt,

p_9: V → t

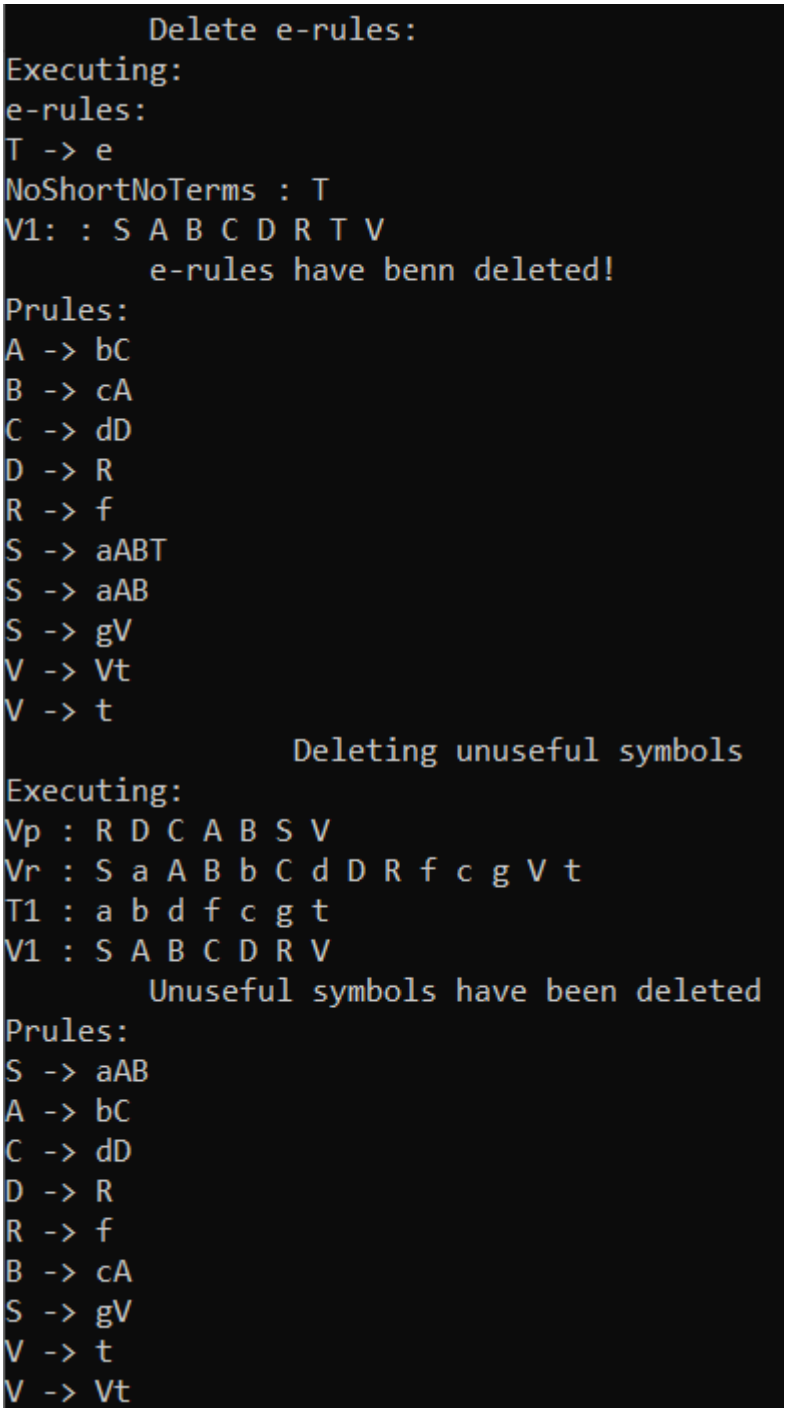

Рисунок 8 – Удаление эпсилон-правил

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

1. Устранить из КС-грамматики цепные правила.

Алгоритм устранения цепных правил

Правило вида A→B, где A и B - нетерминалы называется цепным.

Вход: КС грамматика G = (T, V, P, S).
Выход: Эквивалентная КС грамматика G' = (T, V, P', S) без цепных правил.

Шаг 1. Для каждого правила $X \to A \in P$ построить множество всех цепных символов V^i_X.

```
foreach (X ∈ V)   построить множество V_X = {A | X ⇒+ A}
  i = 0
  V^i_X = {X}
  do
    i++
    foreach (X ∈ V^(i-1)_X, X→ A ∈ P, A ∈ V)
      V^i_X = V^(i-1)_X ∪ {A}
    end
  while (V^i_X ≠ V^(i-1)_X)
end
```

Шаг 2. Для каждого цепного правила $A \to R \in P$, Построить правила $A \to Y\rho$ без цепных символов $Y \notin V^i_A$. Добавить в P' не цепные правила.

```
P' = Ø
foreach (A→ αRβ ∈ P)
  if (α = ε, β = ε, R ∈ V^i_A)   цепное правило
    foreach (R ∈ V^i_A)   найти правую не цепную часть правила
      if (R → Yρ ∈ P, Y ∉ V^i_A)
        P' = P' ∪ { A→ Yρ }
      end
    end
  else
    P' = P' ∪ { A→ αRβ}
  end
end
```

Цепные правила: D → R

$V^1 = \{R\}$

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

```
        ChainRule Deleting:
Executing:
ChainRules:
D -> R
Deleting...
        Chainrules have been deleted;
Prules:
S -> aAB
S -> gV
A -> bC
B -> cA
C -> dD
R -> f
V -> t
V -> Vt
D -> f
```

Рисунок 9 – Цепные правила

2. Устранить из КС-грамматики левую рекурсию.

Определение. Нетерминал КС-грамматики называется *рекурсивным*, если $A \Rightarrow^{+} \alpha A \beta$, для некоторых α и β. Если $\alpha = \varepsilon$, то A называется *леворекурсивным*, если $\beta = \varepsilon$, то A называется *праворекурсивным*. Грамматика, имеющая хотя бы один леворекурсивный нетерминал, называется *леворекурсивной*. Грамматика, имеющая хотя бы один праворекурсивный, нетерминал называется *праворекурсивной*.

Алгоритм Устранения непосредственной левой рекурсии

Вход: КС грамматика $G = (T, V, P, S_0)$, без ε-правил (вида $A \to \varepsilon$).

Выход: Эквивалентная приведенная КС грамматика $G' = (T, V, P', S'_0)$.

1. Упорядочить нетерминалы V по возрастанию $V = [A_1, A_2, \ldots, A_n]$. Преобразоваить правила $A_i \to \alpha$ так, чтобы цепочка α начиналась либо с терминала, либо с такого A_j, что $j > i$. $i = 1$.

foreach i от 1 до n

foreach j от 1 до $i - 1$

2. Множество A_i правил – это $A_i \to A_i\alpha_1 |\ldots| A_i\alpha_m | \beta_1 |\ldots| \beta_p$, где ни одна цепочка β_j не начинается с A_k, если $k \leq i$. Заменим A_i - правила правилами:

$A_i \to \beta_1 |\ldots| \beta_p | \beta_1 A_i' |\ldots| \beta_p A_i'$

$A_i' \to \alpha_1 |\ldots| \alpha_m | \alpha_1 A_i' |\ldots| \alpha_m A_i'$

где A_i' – новый символ. Правые части всех A_i - правил начинаются теперь с терминала или с A_k для некоторого $k > i$.

3. Если $i = n$, то останов и получена грамматика G', иначе $j = i$, $i = i + 1$.

4. Заменить каждое правило вида $A_i \to A_j\alpha$ правилами $A_i \to \beta_1\alpha |\ldots| \beta_m\alpha$, где $A_j \to \beta_1 |\ldots| \beta_m$ – все A_j - правила.

Так как правая часть каждого A_j – правила начинается уже с терминала или с A_k для $k > j$, то и правая часть каждого A_i – правила будет обладать этим же свойством.

5. Если $j = i - 1$, перейти к шагу 2, иначе $j = j + 1$ и перейти к шагу 4.

Если ε присутствовал в языке исходной грамматики, добавим новый начальный символ S' и правила $S' \to S \mid \varepsilon$.

Левая рекурсия $V \to Vt$

Заменим правило $V \to Vt$, на правила $V \to tV'$, $V' \to tV' | t$

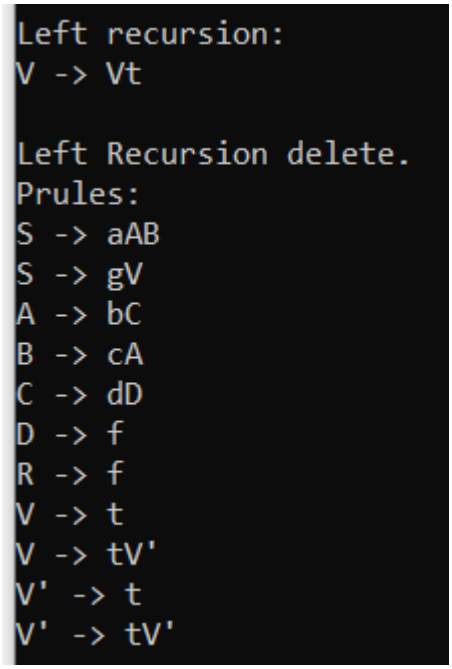

Рисунок 10 – Левая рекурсия

Приведенная грамматика:

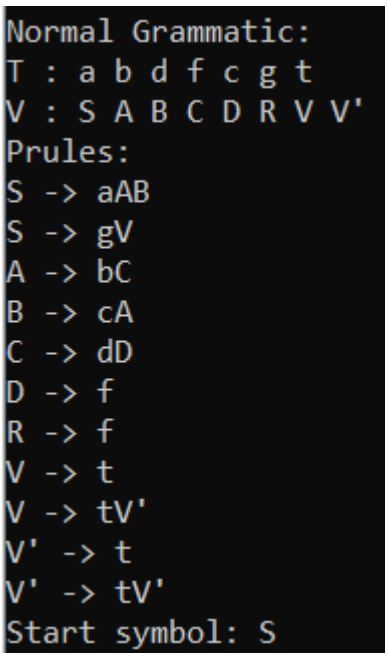

Рисунок 11 – Приведённая грамматика

```
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.

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 → 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'}, S_0 = S,

P = {

p_1: A → bC,

p_2: B → cA,

p_3: C → dD,

p_4: D → f,

p_5: R → f,

p_6: S → aAB,

p_7: S → gV,

p_8: V → t,

p_9: V → tV’,

p_{10}: V’ → t,

p_{11}: 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

Определение 9. МП автомат – это семерка объектов

МП = (Q, ∑, Γ, δ, q_0, z_0, F)

Q – конечное множество состояний устройства управления;

∑ - 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, $F \subseteq Q$.

Алгоритм 3.8. По КС-грамматике G = (T, V, P, S) можно построить МП автомат, L(МП) = L(G). Пусть МП = ({q}, ∑, ∑ ∪ V, δ, q, S, {q}), где δ определяется следующим образом:

1. Если A → α - правило грамматики G, то δ(q, ε, A) = (q,α).
2. δ(q, a, a) = {(q, ε)} для всех a ∈ ∑.

L(МП) = L(G)

МП = (Q, Σ, Γ, δ, q_0, z_0, F):

Q = {q}, Σ = T, Γ = T U V, δ = δ, q_0 = q_0, z_0 = S_0, 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, q_0, 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. $\delta(q, \varepsilon, R) = \{(q, f)\}$,

6. $\delta(q_0, \varepsilon, S) = \{(q, aAB)\}$,

7. $\delta(q_0, \varepsilon, S) = \{(q, gV)\}$,

8. $\delta(q, \varepsilon, V) = \{(q, t)\}$,

9. $\delta(q, \varepsilon, V) = \{(q, tV')\}$,

10. $\delta(q, \varepsilon, V') = \{(q, t)\}$,

11. $\delta(q, \varepsilon, V') = \{(q, tV')\}$,

12. $\delta(q, a, a) = \{(q, \varepsilon)\}, \forall\ a \in \Sigma$

}

Последовательность тактов МП-автомата для цепочки abdfcbdf:

$(q_0, abdfcbdf, S) \vdash^{6} (q, abdfcbdf, aAB) \vdash^{12} (q, bdfcbdf, AB) \vdash^{1} (q, bdfcbdf, bCB) \vdash^{12}$ (q, dfcbdf, CB) $\vdash^{3}$ (q, dfcbdf, dDB) $\vdash^{12}$ (q, fcbdf, DB) $\vdash^{4}$ (q, fcbdf, fB) $\vdash^{12}$ (q, cbdf, B) $\vdash^{2}$ (q, cbdf, cA) $\vdash^{12}$ (q, bdf, A) $\vdash^{1}$ (q, bdf, bC) $\vdash^{12}$ (q, df, C) $\vdash^{3}$ (q, df, dD) $\vdash^{12}$ (q, f, D) $\vdash^{4}$ (q, f, f) $\vdash^{12}$ (q, ε, ε)

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

```
Debug KC-Grammar Prules:
A -> bC
B -> cA
C -> dD
D -> f
R -> f
S -> aAB
S -> gV
V -> t
V -> tV'
V' -> t
V' -> tV'
Delta rules:
 delta(q,ε,A) -> (q,b,C)
 delta(q,ε,B) -> (q,c,A)
 delta(q,ε,C) -> (q,d,D)
 delta(q,ε,D) -> (q,f)
 delta(q,ε,R) -> (q,f)
 delta(q,ε,S) -> (q,a,A,B)
 delta(q,ε,S) -> (q,g,V)
 delta(q,ε,V) -> (q,t)
 delta(q,ε,V) -> (q,t,V')
 delta(q,ε,V') -> (q,t)
 delta(q,ε,V') -> (q,t,V')
 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,t,t) -> (q,ε)
 delta(q,g,g) -> (q,ε)
 delta(q0,ε,z0) -> (qf,ε)

Enter the line   :
abdfcbdf
 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,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
True
```

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.

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), где

T = {a, b, c, d, f, t, g}, V = {S, A, B, C, D, R, V, V'}, S_0 = S,

P = {

p_1: A → bC,

p_2: B → cA,

p_3: C → dD,

p_4: D → f,

p_5: R → f,

p_6: S → aAB,

p_7: S → gV,

p_8: V → t,

p_9: V → tV',

p_{10}: V' → t,

p_{11}: V' → tV'

}

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

Метод: для каждого нетерминала A находим самый длинный префикс α, общий для двух или большего числа альтернатив. Если $\alpha \neq \varepsilon$, т.е. имеется нетривиальный общий префикс, заменим все продукции $A \to \alpha\beta1 \mid \alpha\beta2 \mid \ldots \mid \alpha\beta n \mid \gamma$, где γ представляет все альтернативы, не начинающиеся с α, продукциями

$A \to \alpha A' \mid \gamma$

$A' \to \beta1 \mid \beta2 \mid \ldots \mid \beta n$

Здесь A′ — новый нетерминал. Выполняем это преобразование до тех пор, пока никакие две альтернативы нетерминала не будут иметь общий префикс.

$G_ф$ = (T, V', P', S), где

T = {a, b, c, d, f, t, g}, V' = {S, A, B, C, D, R, V, V', T}, S_0 = S,

P' = {

p_1: A → bC,

p_2: B → cA,

p_3: C → dD,

p_4: D → f,

p_5: R → f,

p_6: S → aAB,

p_7: S → gV,

p_8: V → tT,

p_9: V' → tT,

p_{10}: T → V',

p_{11}: 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 **{⊥}**) × (**T** U **{ε}**) по правилам:

1. Если A → β – правило вывода грамматики с номером i, то M(A, a) = (β, i) для всех $a \neq \varepsilon$, 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}

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	
⊥								ДОПУС

Пустые клетки в таблице означают ОШИБКУ.

Аналитичекое представление для таблицы M:

Таблица 2 — Аналитическое представление управляющей таблицы

Правило грамматики	**Множество**	**Значение M**
p_1: A → bC	FIRST(A) = {b}	M(A, b) = (bC, 1)
p_2: B → cA	FIRST(B) = {c}	M(B, c) = (cA, 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: R → f	FIRST(R) = {f}	M(R, f) = (f, 5)
p_6: S → aAB	FIRST(S) = {a}	M(S, a) = (aAB, 6)
p_7: S → gV	FIRST(S) = {g}	M(S, g) = (gV, 7)
p_8: V → tT	FIRST(V) = {t}	M(V, t) = (tT, 8)
p_9: V’ → tT	FIRST(V’) = {t}	M(V’, t) = (tT, 9)
p_{10}: T → V’	FIRST(T) = {t}	M(T, t) = (V’, 10)
p_{11}: 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, S_0⊥, ε), где S_0 = 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 M
(abdfcbdf, S⊥, ε) ⊦	M(S, a) = (aAB, 6)
(abdfcbdf, aAB⊥, 6) ⊦	M(a, a) = ВЫБРОС
(bdfcbdf, AB⊥, 6) ⊦	M(A, b) = (bC, 1)
(bdfcbdf, bCB⊥, 61) ⊦	M(b, b) = ВЫБРОС
(dfcbdf, CB⊥, 61) ⊦	M(C, d) = (dD, 3)
(dfcbdf, dDB⊥, 613) ⊦	M(d, d) = ВЫБРОС
(fcbdf, DB⊥, 613) ⊦	M(D, f) = (f, 4)
(fcbdf, fB⊥, 6134) ⊦	M(f, f) = ВЫБРОС
(cbdf, B⊥, 6134) ⊦	M(B, c) = (cA, 2)
(cbdf, cA⊥, 61342) ⊦	M(c, c) = ВЫБРОС
(bdf, A⊥, 61342) ⊦	M(A, b) = (bC, 1)
(bdf, bC⊥, 613421) ⊦	M(b, b) = ВЫБРОС
(df, C⊥, 613421) ⊦	M(C, d) = (dD, 3)
(df, dD⊥, 6134213) ⊦	M(d, d) = ВЫБРОС
(f, D⊥, 6134213) ⊦	M(D, f) = (f, 4)
(f, f⊥, 61342134) ⊦	M(f, f) = ВЫБРОС
(ε, ⊥, 61342134) ⊦	M(⊥, ε) = ДОПУСК

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

```
9.1
Создадим таблицу. Сначала создадим по столбцу для каждого из этих терминалов:
a, b, c, d, f, t, g,  ,
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

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

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

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

Рассмотрим нетерминал V
   Первый символ правила V -> tT - t
   Это правило заносим в таблицу на пересечении строки нетерминала V и столбца терминала t

Рассмотрим нетерминал V'
   Первый символ правила V' -> tT - t
   Это правило заносим в таблицу на пересечении строки нетерминала V' и столбца терминала t

Рассмотрим нетерминал T
   Первый символ правила T -> V' - t
   Это правило заносим в таблицу на пересечении строки нетерминала T и столбца терминала t

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

Введите строку:
abdfcbdf
Допуск. Цепочка символов = L(G).
61342134
```

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.)

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

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

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

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

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

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

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

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.

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'}, S_0 = S,

P = {

p_1: A → bC,

p_2: B → cA,

p_3: C → dD,

p_4: D → f,

p_5: R → f,

p_6: S → aAB,

p_7: S → gV,

p_8: V → t,

p_9: V → tV',

p_{10}: V' → t,

p_{11}: V' → tV'

}

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, S’), где

T = {a, b, c, d, f, t, g}, V = {S, A, B, C, D, R, V, V’}, S_0 = S,

P = {

p_0: S → S’,

p_1: A → bC,

p_2: B → cA,

p_3: C → dD,

p_4: D → f,

p_5: R → f,

p_6: S → aAB,

p_7: S → gV,

p_8: V → t,

p_9: V → tV’,

p_{10}: V’ → t,

p_{11}: V’ → tV’

}

1. I_0 = {S’→·S}
2. I_0 = {S’→·S, S→·aAB, S→·gV}

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*→ *α*•*β*], таких что [*A*→ *α*•*X β*] ϵ *I*

I_0 = {S’→·S, S→·aAB, S→·gV}

GOTO(I_0, a) = {S→a·AB, A→·bC} = I_1

GOTO(I_0, g) = {S→g·V, V→·t, V→·tV’} = I_2

GOTO(I_0, S) = {S'→S·} = I_3

GOTO(I_1, b) = {A→b·C, C→·dD} = I_4

GOTO(I_1, A) = {S→aA·B, B→·cA} = I_5

GOTO(I_2, t) = {V→t·, V→t·V', V'→·t, V'→·tV'} = I_6

GOTO(I_2, V) = {S→gV·} = I_7

GOTO(I_4, d) = {C→d·D, D→·f} = I_8

GOTO(I_4, C) = {A→bC·} = I_9

GOTO(I_5, c) = {B→c·A, A→·bC} = I_{10}

GOTO(I_5, B) = {S→aAB·} = I_{11}

GOTO(I_6, t) = {V'→t·, V'→t·V', V'→·t, V'→·tV'} = I_{12}

GOTO(I_6, V') = {V→tV'·} = I_{13}

GOTO(I_8, f) = {D→f·} = I_{14}

GOTO(I_8, D) = {C→dD·} = I_{15}

GOTO(I_{10}, b) = {A→b·C, C→·dD} = I_4 – уже отмечено

GOTO(I_{10}, A) = {B→cA·} = I_{16}

GOTO(I_{12}, t) = {V'→t·, V'→t·V', V'→·t, V'→·tV'} = I_{12} – уже отмечено

GOTO(I_{12}, V') = {V'→tV'·} = 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 ϵ (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, S→·gV},

I_1 = {S→a·AB, A→·bC},

I_2 = {S→g·V, V→·t, V→·tV'},

I_3 = {S'→S·},

I_4 = {A→b·C, C→·dD},

I_5 = {S→aA·B, B→·cA},

I_6 = {V→t·, V→t·V', V'→·t, V'→·tV'},

I_7 = {S→gV·},

I_8 = {C→d·D, D→·f},

I_9 = {A→bC·},

I_{10} = {B→c·A, A→·bC},

I_{11} = {S→aAB·},

I_{12} = {V’→t·, V’→t·V’, V’→·t, V’→·tV’},

I_{13} = {V→tV’·},

I_{14} = {D→f·},

I_{15} = {C→dD·},

I_{16} = {B→cA·},

I_{17} = {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:

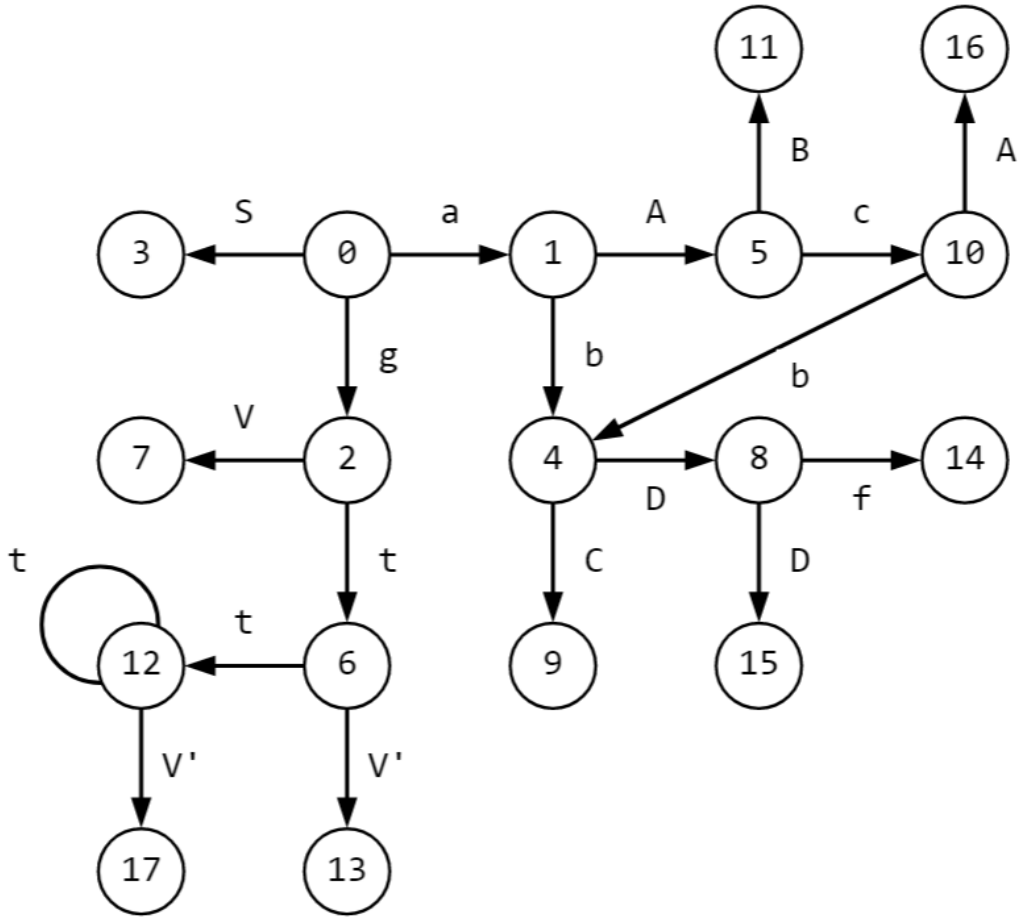

Рисунок 14 – Диаграмма переходов автомата

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.

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

Управляющая таблица																
I	f(u)								g(x)							
	a	b	c	d	f	t	g	⊥	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.

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 gt:

$(0, gt\perp, \varepsilon) \vdash^{П} (0\ 2, t\perp, \varepsilon) \vdash^{П} (0\ 2\ 6, \perp, \varepsilon) \vdash^{С} (0\ 2\ 7, \perp, 8) \vdash^{С} (0\ 3, \perp, 8\ 7) \vdash^{Д}$

```
I_10 = CLOSURE( B -> c.A )
B -> c.A
A -> .bC
I_11 = CLOSURE( S -> aAB. )
S -> aAB.
I_12 = CLOSURE( V' -> t. )
V' -> t.
V' -> t.V'
V' -> .t
V' -> .tV'
I_13 = CLOSURE( V -> tV'. )
V -> tV'.
I_14 = CLOSURE( D -> f. )
D -> f.
I_15 = CLOSURE( C -> dD. )
C -> dD.
I_16 = CLOSURE( B -> cA. )
B -> cA.
I_17 = CLOSURE( V' -> tV'. )
V' -> tV'.
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 t g $ S A B C D R V V' S'
Q0: 0
F: 3 6 7 9 11 12 13 14 15 16 17
DeltaList:
 delta(0,a,-> (1
 delta(0,g,-> (2
 delta(0,S,-> (3
 delta(1,b,-> (4
 delta(1,A,-> (5
 delta(2,t,-> (6
 delta(2,V,-> (7
 delta(4,d,-> (8
 delta(4,C,-> (9
 delta(5,c,-> (10
 delta(5,B,-> (11
 delta(6,t,-> (12
 delta(6,V',-> (13
 delta(8,f,-> (14
 delta(8,D,-> (15
 delta(10,b,-> (4
 delta(10,A,-> (16
 delta(12,t,-> (12
 delta(12,V',-> (17
```

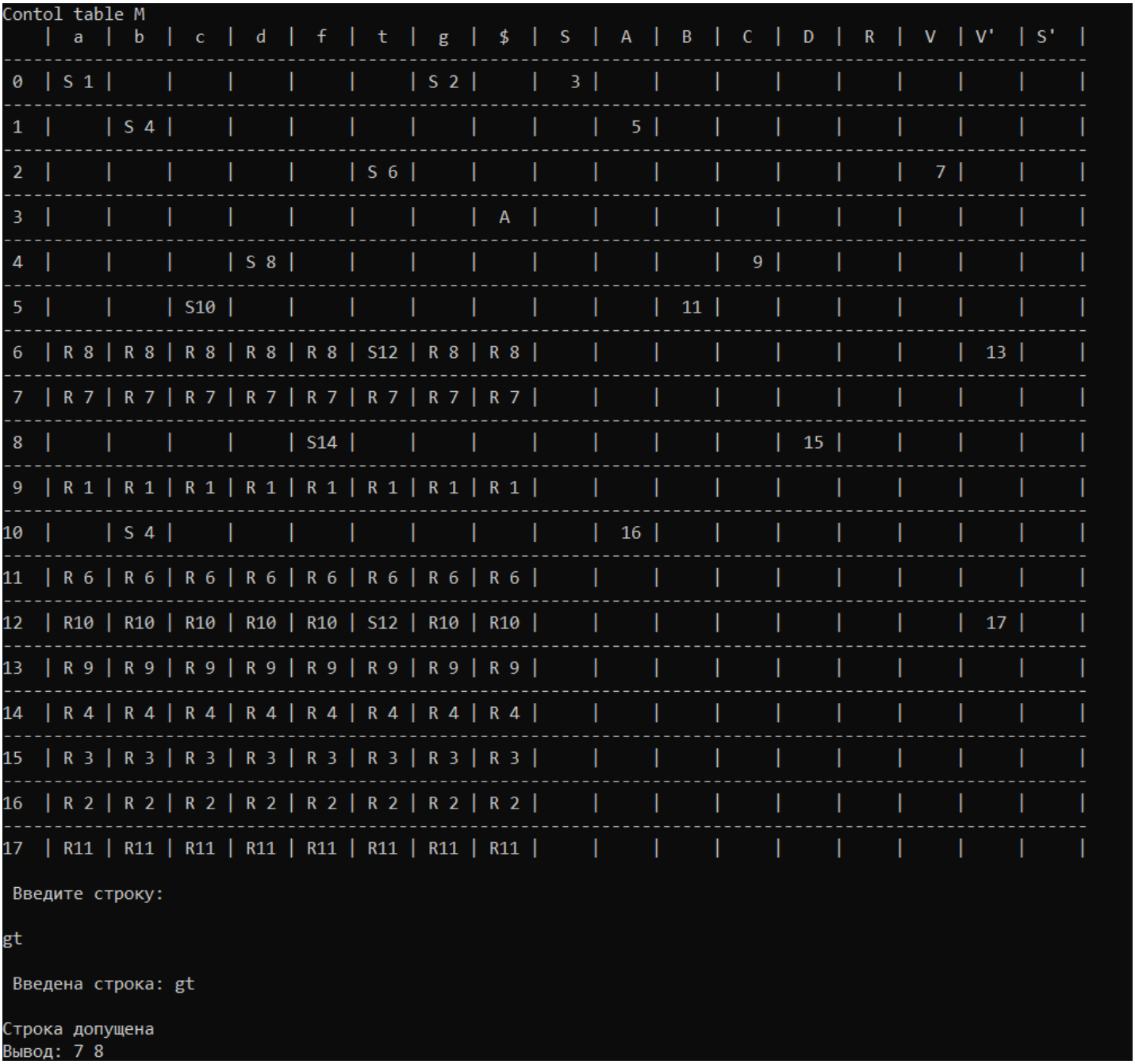

Рисунок 15 – Пример управляющей таблицы для LR(0)-анализатора