Московский авиационный институт

(национальный исследовательский институт)

Факультет «Прикладная математика и физика»

**Лабораторные работы**

**по курсу**

**«Системное программное обеспечение»**

**IV семестр**

1. Спроектировать грамматику по паттерн-модели регулярного языка

2. Преобразовать спроектированной грамматики в конечный автомат , составить диаграмму переходов КА и реализовать

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

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

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

6 Определить форму КС-грамматики и сделать ее приведение

7. Спроектировать МП автомат для приведенной КС-грамматики

8. Реализовать МП автомат для приведенной КС-грамматики

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

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

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

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

13. Для LR(k) -грамматики 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:* Михеева К.О.

*Группа:*  М08-207Б-20

*Руководитель:* Семёнов А. С.

*Оценка:*

*Дата:*

**Москва. 2022**

**Лабораторная работа 1-3**

1. Спроектировать грамматику по паттерн-модели регулярного языка

2. Преобразовать спроектированной грамматики в конечный автомат , составить диаграмму переходов КА и реализовать

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

**Пояснение данного языка**

обозначает замыкание Клини - множество любой длины, состоящее из элементов замыкаемого множества, а также включая пустое множество.

обозначает итерацию – множество цепочек, состоящих из циклического повторения цепочки 10.

// pattern\_4 = @"^(\W|^)[\w.\-]{0,25}@(yahoo|hotmail|gmail)\.com(\W|$)";

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

//T = {@, .com, yahoo, hotmail, gmail, A-Z, a-z, 0-9, \_ , - , . ,};

//P = {p1, p2, p3, p4, p5, ... , p30};

//V = {V\_1, V\_2, V\_3, V\_4, V\_5, ..., V\_26, S\_, W\_, X\_};

//p1: S0 -> AV\_1 | ... | ZV\_1 | aV\_1 | ... | zV\_1 | 0V\_1 ... | 9V\_1 | .V\_1 | \_V\_1 | -V\_1 | V\_1,

//p2: V\_1 -> AV\_2 | ... | ZV\_2 | aV\_2 | ... | zV\_2 | 0V\_2 ... | 9V\_2 | .V\_2 | \_V\_2 | -V\_2 | V\_2,

// ...

//p26: V\_25 -> AV\_26 | ... | ZV\_26 | aV\_26 | ... | zV\_26 | 0V\_26 ... | 9V\_26 | .V\_26 | \_V\_26 | -V\_26 | V\_26,

//p27: V\_26 -> @S\_,

//p28: S\_ -> yahooW\_ | hotmailW\_ | gmailW\_,

//p29: W\_ -> .comX\_,

//p30: X\_ -> ε

//Цепочка: S0 => nV\_1 => noV\_2 => nonV\_3 => nonaV\_4 => nonamV\_5 => nonameV\_6 => ... => nonameV\_26 => noname@S\_ => noname@yahooW\_ => noname@yahoo.comX\_ => noname@yahoo.com

//Цепочка: S0 => -V\_1 => -aV\_2 =. -abV\_3 => -abcV\_4 => -abcV\_5 => ... -abcV\_26 => -abc@S\_ => -abc@gmailW\_ => -abc@gmail.comX\_ => -abc@gmail.com

//Конечный автомат

//L(KA) == L(G)

//KA = (Q, sigma, delta, q0, F)

//KA = T = ({S0, V\_1, V\_2, V\_3, V\_4, V\_5, ..., V\_26, S\_, W\_, X\_, qf}, {@, .com, yahoo, hotmail, gmail, A-Z, a-z, 0-9, \_ , - , .}, delta, S0, {qf})

// Функция перехода(delta):

//delta(S0, A-Z) = {V\_1}; delta(S0,a-z) = {V\_1}; delta(S0,0-9) = {V\_1}; delta(S0,\_) = {V\_1}; delta(S0, -) = {V\_1}; delta(S0,.) = {V\_1};

//delta(V\_1, A-Z) = {V\_2}; delta(V\_1, a-z) = { V\_2}; delta(V\_1, 0-9) = {V\_2}; delta(V\_1, \_) = {V\_2}; delta(V\_1, -) = {V\_2}; delta(V\_1,.) = {V\_2};

// .....

//delta(V\_25, A-Z) = {V\_26}; delta(V\_25, a-z) = {V\_26}; delta(V\_25, 0-9) = {V\_26}; delta(V\_25, \_) = {V\_26}; delta(V\_25,-) = {V\_26}; delta(V\_25,.) = {V\_26};

//delta(V\_26, @) = {S\_};

//delta(S\_, yahoo) = {W\_}; delta(S\_, hotmail) = {W\_}; delta(S\_, gmail) = {W\_};

//delta(W\_, .com) = {X\_};

//delta(X\_, ε) = {qf}.

//Конфигурация КА:

//(S0, -abc@gmail.com) |- (V\_1, abc@gmail.com) |- (V\_2, bc@gmail.com) |- (V\_3, c@gmail.com) |- (V\_4, @gmail.com) |- ... |- (V\_26, @gmail.com) |- (S\_, gmail.com) |- (W\_, .com) |- (X\_, ε) |- (qf, ε)

//Недетермированный КА

//pattern\_6 = @"(\W|^)po[#\-]{0,1}\s{0,1}\d{2}[\s-]{0,1}\d{4}(\W|$)";

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

//T = {po, 0-9, -, #, " "};

//P = {p1, p2, p3, p4, p5, p6, p7, p8, p9, p10};

//V = {S0, A, B, C, D, E, F, G, H, I}.

//p1: S0 -> poA,

//P2: A -> #B | -B | " "B, B,

//p3: B -> 0C | 1C | 2C | 3C | 4C | 5C | 6C | 7C | 8C | 9C,

//p4: C -> 0D | 1D | 2D | 3D | 4D | 5D | 6D | 7D | 8D | 9D,

//p5: D -> -E | " "E | E,

//p6: E -> 0F | 1F | 2F | 3F | 4F | 5F | 6F | 7F | 8F | 9F,

//p7: F -> 0G | 1G | 2G | 3G | 4G | 5G | 6G | 7G | 8G | 9G,

//p8: G -> 0H | 1H | 2H | 3H | 4H | 5H | 6H | 7H | 8H | 9H,

//p9: H -> 0I | 1I | 2I | 3I | 4I | 5I | 6I | 7I | 8I | 9I,

//p10: I -> ε

//Цепочка: S0 => poA => po#B => po#2C => po#21D => po#21 E => po#21 0F => po#21 05G => po#21 051H =>po#21 0513I => po#21 0513

//Цепочка: S0 => poA => poB => po0C => po01D => po01-E => po01-4F => po01-42G => po01-420H => po01-4205I => po01-4205

//Конечный автомат

//L(KA) == L(G)

//KA = (Q, sigma, delta, q0, F)

//KA = T = ({S0, A, B, C, D, E, F, G, H, I, qf}, {po, 0-9, -, #, " "}, delta, S0, {qf})

// Функция перехода(delta):

//delta(S0, po) = {A};

//delta(A, #) = {B}; delta(A, -) = {B}; delta(A, " ") = {B};

//delta(B, 0-9) = {C};

//delta(C, 0-9) = {D};

//delta(D, -) = {E}; delta(D," ") = {E};

//delta(E, 0-9) = {F};

//delta(F, 0-9) = {G};

//delta(G, 0-9) = {H};

//delta(H, 0-9) = {I};

//delta(I, ε) = {qf}.

//Конфигурация КА:

//(S0, po#21 0513) |- (A, #21 0513) |- (B, 21 0513) |- (C, 1 0513) |- (D, " "0513), |- (E, 0513) |- (F, 513) |- (G, 13) |- (H, 1) |- (I, ε) |- (qf, ε)

//Недетермированный КА

//pattern\_7 = @"^\+?\d{0,2}\-?\d{4,5}\-?\d{5,6}";

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

//T = {+, ' ', -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

//P = {p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14, p15, p16, p17};

//V = {S0, A, C, D, E, F, G, H, I, J, K, L, M, N, O, Q, R}.

//p1: S0 -> +A | A,

//p2: A -> 0C | 1C | 2C | 3C | 4C | 5C | 6C | 7C | 8C | 9C | C,

//p3: C -> 0D | 1D | 2D | 3D | 4D | 5D | 6D | 7D | 8D | 9D | D,

//p4: D -> -E | E,

//p5: E -> 0F | 1F | 2F | 3F | 4F | 5F | 6F | 7F | 8F | 9F,

//p6: F -> 0G | 1G | 2G | 3G | 4G | 5G | 6G | 7G | 8G | 9G,

//p7: G -> 0H | 1H | 2H | 3H | 4H | 5H | 6H | 7H | 8H | 9H,

//p8: H -> 0I | 1I | 2I | 3I | 4I | 5I | 6I | 7I | 8I | 9I,

//p9: I -> 0J | 1J | 2J | 3J | 4J | 5J | 6J | 7J | 8J | 9J | J,

//p10:J -> -K | K,

//p11:K -> 0L | 1L | 2L | 3L | 4L | 5L | 6L | 7L | 8L | 9L,

//p12:L -> 0M | 1M | 2M | 3M | 4M | 5M | 6M | 7M | 8M | 9M,

//p13:M -> 0N | 1N | 2N | 3N | 4N | 5N | 6N | 7N | 8N | 9N,

//p14:N -> 0O | 1O | 2O | 3O | 4O | 5O | 6O | 7O | 8O | 9O,

//p15:O -> 0Q | 1Q | 2Q | 3Q | 4Q | 5Q | 6Q | 7Q | 8Q | 9Q,

//p16:Q -> 0R | 1R | 2R | 3R | 4R | 5R | 6R | 7R | 8R | 9R | R,

//p17:R -> ε

//Цепочка: S0 => +A => +9C => +91D => +91-E => +91-8F => +91-82G => +91-828H => +91-8281I => +91-8281J => +91-8281-K => +91-8281-9L => +91-8281-90M => +91-8281-900N => +91-8281-9001O => +91-8281-90013Q => +91-8281-900135R => +91-8281-900135

//Цепочка: S0 => A => C => D => E => 9F => 92G => 926H => 9265I => 92651J => 92651K => 926510L => 9265102M => 92651020N => 926510201O => 9265102019Q => 92651020199R => 92651020199

////Конечный автомат

//L(KA) == L(G)

//KA = (Q, sigma, delta, q0, F)

//KA = T = ({S0 ,A, C, D, E, F, G, H, I, J, K, L, M, N, O, Q, R, qf}, {+, ' ', -, 0-9}, delta, S0, {qf})

// Функция перехода(delta):

//delta(S0, +) = {A};

//delta(A, 0-9) = {B};

//delta(C, 0-9) = {D};

//delta(D, -) = {E};

//delta(E, 0-9) = {F};

//delta(F, 0-9) = {G};

//delta(G, 0-9) = {H};

//delta(H, 0-9) = {I};

//delta(I, 0-9) = {J};

//delta(J, -) = {K};

//delta(K, 0-9) = {L};

//delta(L, 0-9) = {M};

//delta(M, 0-9) = {N};

//delta(N, 0-9) = {O};

//delta(O, 0-9) = {Q};

//delta(Q, 0-9) = {R};

//delta(R, ε) = {qf}.

//Конфигурация КА:

//(S0,92651020199) |- (A,92651020199 ) |- (C, 2651020199) |- (D, 651020199) |- (E, 651020199), |- (F,51020199) |- (G, 1020199) |- (H, 020199) |- (I, 20199) |- (J, 20199) |- (K, 0199) |- (L, 199) |- (M, 99) |- (N, 9) |- (O, ) |- (R, ε) (qf, ε)

//Детермированный КА

*Лабораторная работа №3*

**Формулировка задания**

Реализовать конечный автомат.

**Код программы:**

Изображение выглядит как текст

Автоматически созданное описание

**Пример работы программы**

Изображение выглядит как текст

Автоматически созданное описание

**Практическая работа №3 (4-8 лаб.)**

**Задание:**

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

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

6 Определить форму КС-грамматики и сделать ее приведение

7. Спроектировать МП автомат для приведенной КС-грамматики

8. Реализовать МП автомат для приведенной КС-грамматики.

Исходная грамматика:

***G = {T, V, P, S0};***

***G = ({s, e, p, w, @, k, /, e, r, f},***

***{R, N, V, C, L, S, A, D, Y, I, B},***

***{R → S, , V → @V, V → aL, N → IV, L → /A, , C → kA, A → b, N → D, D → w, S → aY, Y → eps, N → r, I → IpR, S → fV, B → B@V},***

***R);***

***T = {s, e, p, w, @, k, /, b, r, f};***

***V = {R, N, V, C, L, S, A, D, Y, I, B};***

***P = {p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14, p15, p16, p17};***

***S0 = R;***

***p1: R → S***

***p2: V → @V***

***p3: V → aL***

***p4: L → /A***

***p5: N → IV***

***p6: C → kA***

***p7: A → b***

***p8: N → D***

***p9: D → w***

***p10:N → M***

***p11:M → e***

***p12:S → aY***

***p13:Y → eps***

***p14:N → r***

***p15:I → IpR***

***p16:S → fV***

***p17:B → B@V***

***1)*** Устранить из КС-грамматики бесполезные символы и ε–правила.

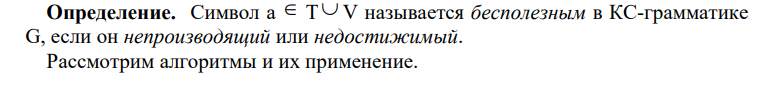
**1.1)** Удаление непроизводящих символов.

**1.2)** Удаление недостижимых символов.

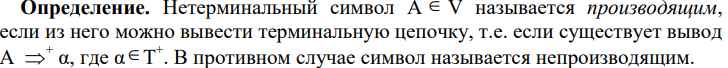
**2)** Устранить из ЛС-грамматики цепные правила и устранить левую рекурсию.

**3)** Определить форму КС-грамматики и сделать ее приведение.

/Устранить из КС-грамматики бесполезные символы.

******

**//Удаление непроизводящих символов.**

****

**1. Алгоритм удаление непроизводящих символов**

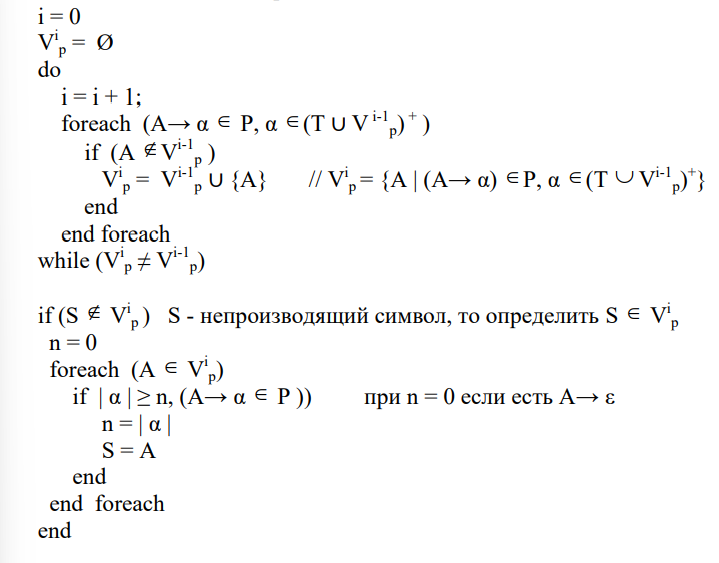
**Утверждение.** Если все символы цепочки α из правой части правила вывода A + α являются производящими, то нетерминал A V в левой части правила вывода также должен быть производящим: Vp = {A | A + α, A V , α T + } **Вход**: КС G = (T, V, P, S)

**Выход**: Vp = {A | A + α, A V, α T + }, S' 1.

1. Составить множество Vp 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 Vp, то добавить в Vp нетерминал, стоящий в его левой части.

3. Если на шаге 2, Vp больше не пополняется, то Vp содержит все производящие нетерминалы грамматики, а все нетерминалы не попавшие в него являются непроизводящими.



***V\_1p = {A, D, M, Y, N};***

***V\_2p = {A, D, M, Y, N, S, R};***

***V\_3p = {A, D, M, Y, N, S, R, C, V, L};***

***V - V\_p = {B, I} -*** непроизводящие символы

***P1 = {p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14}***

***p1: R → S***

***p2: S → aY***

***p3: Y → eps***

***p4: S → fV***

***p5: V → @V***

***p6: V → aL***

***p7: L → /A***

***p8: A → b***

***p9: C → kA***

***p10: N → D***

***p11: D → w***

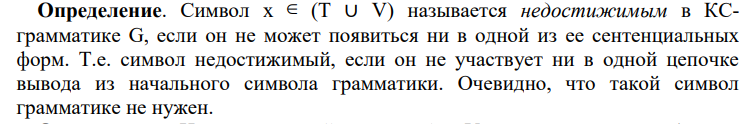
***p12: N → M***

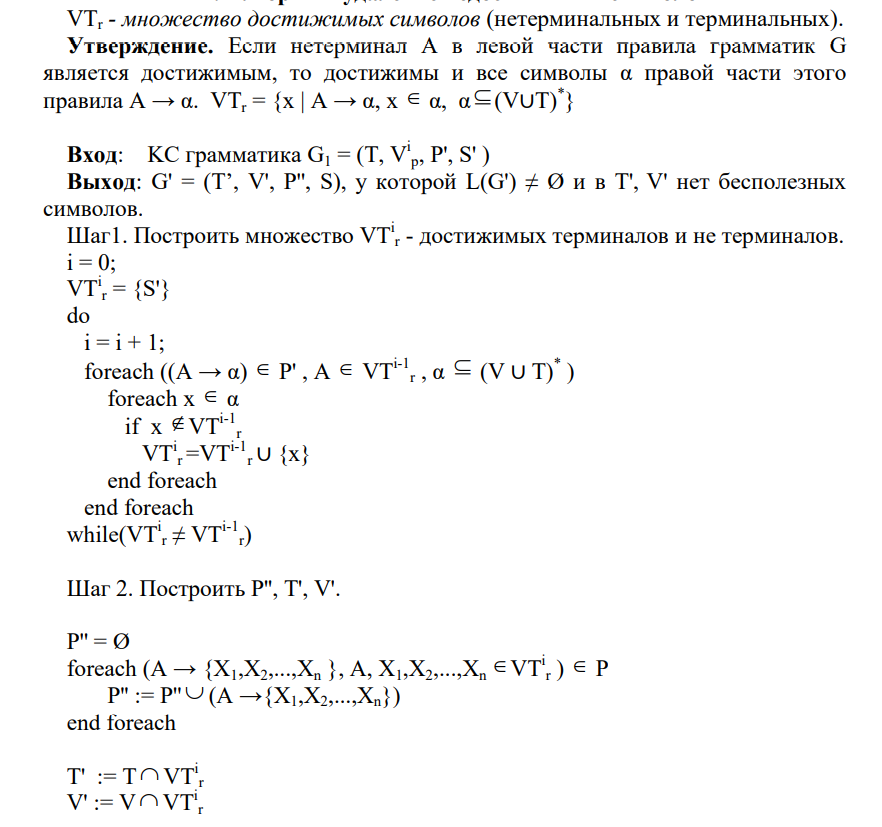
***p13: M → e***

***p14: N → r***

***G\_1 = ({a, r, w, k, e, @, /, b}, {A, D, M, Y, N, S, R, C, V, L}, P1, R)***

***//Удаление недостижимых символов.***

****

******

***VT1 = {R}***

***VT2r = {R, S}***

***VT3r = {R, S, Y, f, V}***

***VT4r = {R, S, Y, f, V, a, eps, @, L, /}***

***VT5r = {R, S, Y, f, V, a, eps, @, L, /, A}***

***VT6r = {R, S, Y, f, V, a, eps, @, L, /, A, b}, {D, M, N, C} - недостижимы.***

***P2:***

***p1: R → S***

***p2: S → aY***

***p3: Y → eps***

***p4: S → fV***

***p5: V → @V***

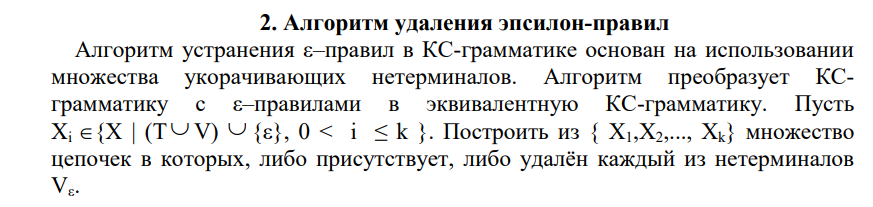
***p6: V → aL***

***p7: L → /A***

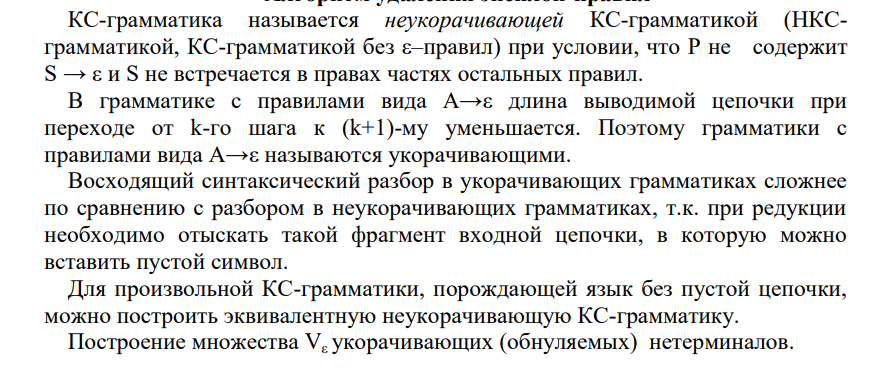
***p8: A → b***

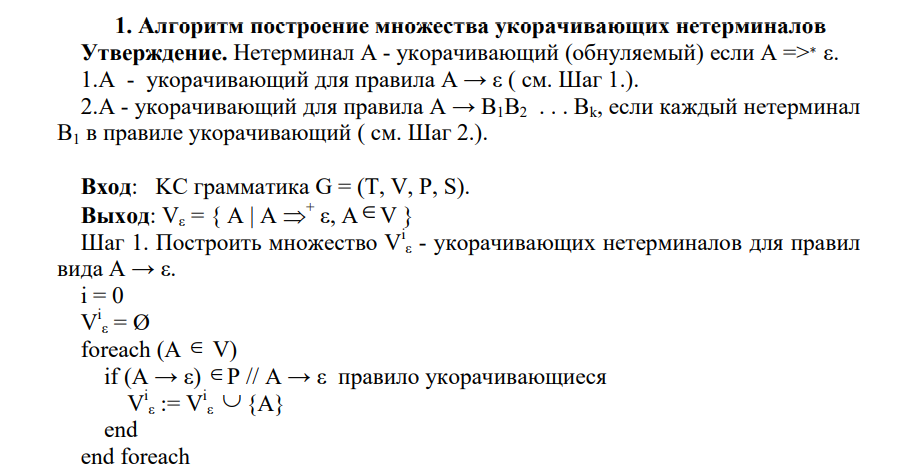
***G' = ({b, f, @, a, /}, {R, S, Y, V, L, A}, P2, R)***

***L(G') = {a}; L(G') = {f@a/b}***

***/Устранить из КС-грамматики ε–правила.***

***//Алгоритм построение множества укорачивающих нетерминалов.***

******

******

***1. V0e = {Y}***

***V1e = {Y}***

***2. P3:***

***p1: R → S***

***p2: S → fV***

***p3: V → @V***

***p4: V → aL***

***p5: L → /A***

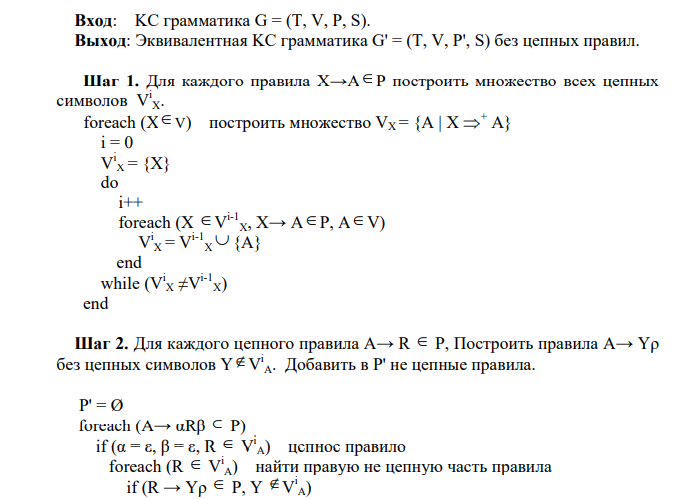
***p6: A → b***

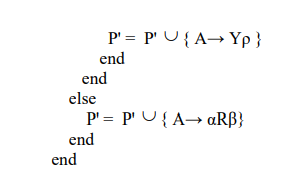
***G" = ({b, f, @, a, /}, {R, S, V, L, A}, P3, R)***

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

***Алгоритм удаление цепных правил.***

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



******

***G" = ({b, f, @, a, /}, {R, S, V, L, A}, P3, R)***

***1. R → S***

***S → fV***

***2. P4:***

***p1: R → fV***

***p2: V → @V***

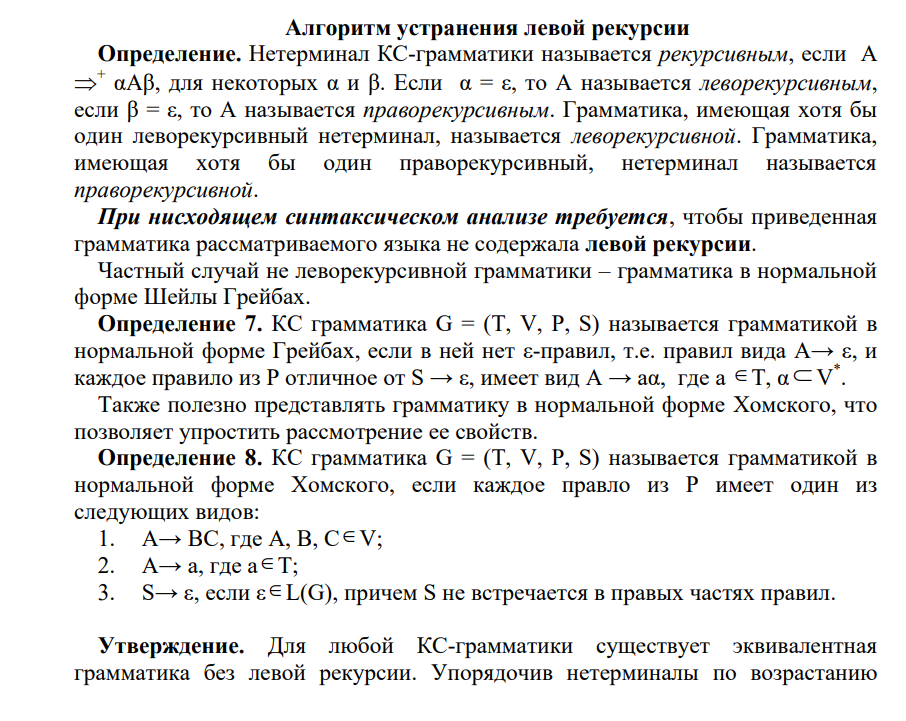
***p3: V → aL***

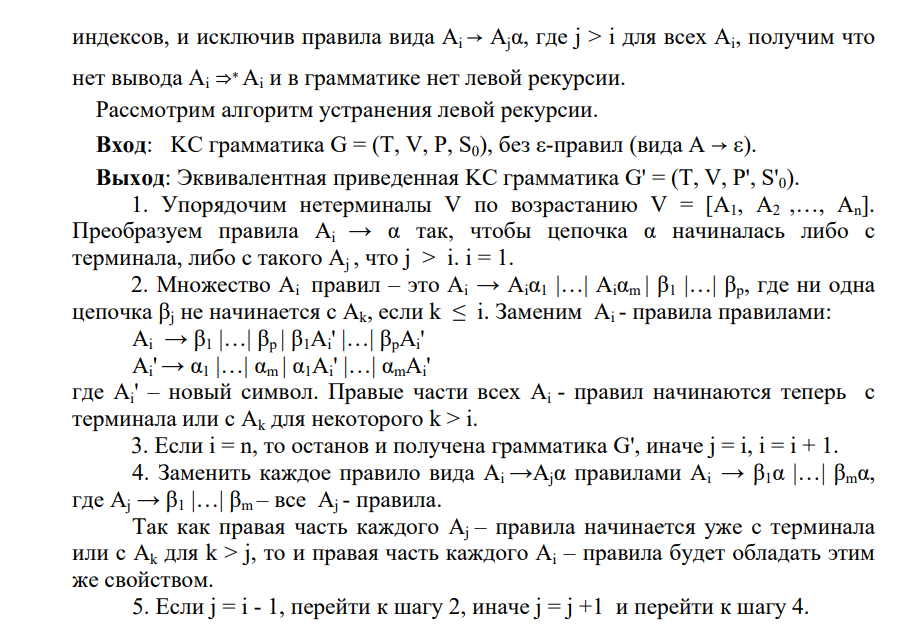
***p4: L → Aе***

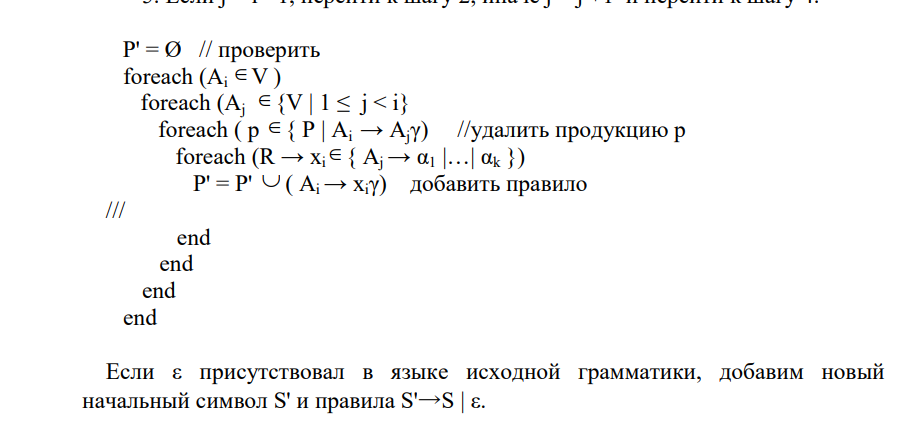
***p5: A → b***

***G"' = ({b, f, @, a, /}, {R, A, L, V}, P4, R)***

***/Устранить из KС-грамматики левой рекурсии.***



******

******

G = ({a,t, c, i}, {A, B, G, V },P, A)

P:

A → BG

B → Bc

G → a

G → tW

W → i

B → c

\* Устранение левой рекурсии

A1 = A, A2 = B, A3 = G, A4 = W

V = [A1, A2, A3, A4]

Для i = 2 преобразуем правила B → Bc | c, α = c, β = c

Заменим B правила правилами B → cB’ | c

P’:

A → BG

B → cB

G → a

G → tW

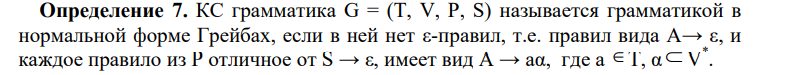
W → I

B → c

G' = ({a, i, c, t}, {A, B, G, W}, P, A)

***/Определить форму КС-грамматики и сделать ее приведение.***

КС-грамматика имеет нормальную форму Шейлы Грейбах.



***Приведенный вид грамматики:***

***G"" = ({b, f, @, a, /}, {R, A, L, V}, P5, R)***

**Лабораторная работа 7-8.**

**7) Спроектировать МП автомат для приведенной КС-грамматики**

**МП - автоматы** имеют вспомогательную память, называемую магазином В магазин можно поместить неограниченное количество символов. В каждый момент времени доступен только верхний символ магазина.

*Верхний символом магазина будем считать самый левый символ цепочки.*

***Определение 9*.** МП автомат – это семерка объектов **MП = (Q, ∑, Г, δ, q, z, F).**

**Q** – конечное множество состояний устройства управления;

**∑** - конечный алфавит входных символов;

**Г** - конечный алфавит магазинных символов;

**δ** - функция переходов, отображает множества **Q×(∑ᴗ {ε})×Г** в множество конечных подмножеств множества **Q×Г\*, δ : Q×(∑ᴗ {ε})×Г -> Q×Г\*** ;

**q0** - начальное состояние, **q0 ∈ Q;**

**z0** - начальный символ магазина, **z0 ∈ Г**;

**F** - множество заключительных состояний.

***Определение 10*.** **Конфигурацией МП-автомата** называется тройка

**(q, ω, z) ∈ Q×∑\*×Г\*:**

**q** – текущее состояние управляющего устройства;

**ω** – необработанная часть входной цепочки (первый символ цепочки **ω** находится под входной головкой; если **ω = ε**, то считается, что вся входная цепочка прочитана) ;

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

*Такт МП-автомата будем описывать бинарным отношением***|*-*** *, определенным на множестве конфигураций.*

**Если a ≠ ε**, то и входная цепочка прочитана не вся, то запись

**(q, aω, zɣ) |- (q’, ω, a)** означает, что МП-автомат в состоянии **q**, обозревая символ во входной цепочки и имея символ **z** в верхушке магазина, может перейти в новое состояние **q’**, сдвинуть входную головку на один символ вправо и заменить верхний символ магазина **z** символом **a** и цепочкой магазинных символов **ɣ**.

**Если z = a**, то верхний символ удаляется из магазина.

**Если a = ε**, то текущий входной символ в этом такте называется **ε-тактом**, не принимается во внимание и входная головка остается неподвижной. (**ε-такты** могут выполняться также в случае, когда вся входная цепочка прочитана, но если магазин пуст, то такт МП-автомата невозможен по определению.)

**Начальной** *конфигурацией МП-автомата* называется конфигурация вида **(q0, ω, z0)**, где устройство управления находится в начальном состоянии, на входной ленте записана цепочка **ω∈∑\*** , которую необходимо распознать, а магазин содержит только начальный символ **z0**.

**Заключительной** *конфигурацией МП-автомата* называется конфигурация вида **(q, ε, ɣ )**, где **q∈F** – одно из заключительных состояний устройства управления, входная цепочка прочитана до конца, а в магазине записана некоторая, заранее определенная цепочка **ɣ∈Г\*** .

**Язык распознаваемый МП-автоматом**, называется множество цепочек: **L(МП) = {ω|ω ∈∑\* и (q0,ω, z0)|-\* (q, ε, ɣ)** для некоторых **q∈F и ɣ∈Г\*}**

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

**Построение:**

G = ({b, f, @, a, /, w, e}, {R, V, L, A, N, M, B, W, D}, P, R)

P:

p1: R → fV

p2: V → @M

p3: M → aL

p4: L → /A

p5: A → bN

p6: N → eB

p7: B → wC

p8: C → @W

p9: W → aD

p10: D → e

**L(МП) = L(G)**

MП = (Q, ∑, Г, δ, q0, z, F):

∑ = T;

Г = ∑ ᴗV;

δ = P;

Q = {q};

F = {q};

q0;

z0 = S0;

МП = ({q}, {b, f, @, a, /, w, e}, {b, f, @, a, /, w, e,R, V, L, A, N, M, B, W, D}, δ, q0, R, {q})

**Функция перехода δ:**

δ: (q0, eps, R) = {q, fV};

δ: (q, eps, V) = {q, @M};

δ: (q, eps, M) = {q, aL};

δ: (q, eps, L) = {q, /A};

δ: (q, eps, A) = {q, bN};

δ: (q, eps, N) = {q, eB};

δ: (q, eps, B) = {q, wC};

δ: (q, eps, C) = {q, @W};

δ: (q, eps, W) = {q, aD};

δ: (q, eps, D) = {q, e;};

δ: (q, e, e) = {q, eps}; для любого e∈∑.

**Последовательность тактов МП-автомата f@a/bew@ae:**

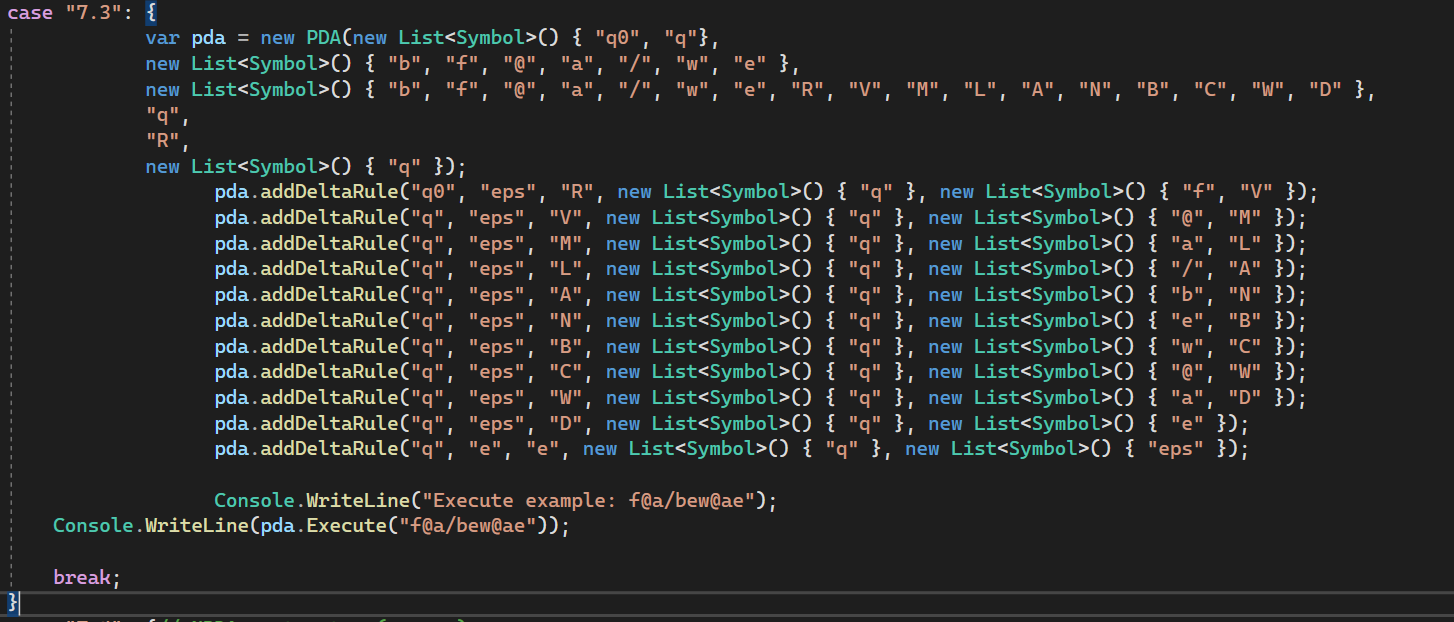
(q0, f@a/bew@ae, R) |-1 (q, @a/bew@ae, fV) |-2 (q, a/bew@ae, @M) |-3 ((q, /bew@ae, aL) |-4 (q, bew@ae, /A) |-5 (q, ew@ae, bN) |-6 (q, w@ae, eB) |-7 (q, @ae, wC) |-8 (q, ae, @W) |-9 (q, e, aD) |-10(q, e, e) |-11 (q, eps, eps).

МП автоматы – детерминированные автоматы.

МП автомат относится к нисходящим синтаксическим анализаторам, так как строится сверху вниз.

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

**Код:**

****

**Вывод:**

7.3

delta(q0,eps,R)-> (q,f,V) delta(q,eps,V) -> (q,@,M) delta(q,eps,M) -> (q,a,L) delta(q,eps,L) -> (q,/,A) delta(q,eps,A) -> (q,b,N) delta(q,eps,N) -> (q,e,B) delta(q,eps,B) -> (q,w,C) delta(q,eps,C) -> (q,@,W) delta(q,eps,W) -> (q,a,D) delta(q,eps,D) -> (q,e) delta(q,e,e) -> (q,eps)

Enter the line :

f@a/bew@ae

step 1

delta(q0,eps,R) -> (q,f,V)

step 2 f

delta(q,eps,V) -> (q,@,M)

step 3 @

delta(q,eps,M) -> (q,a,L)

step 4 a

delta(q,eps,L) -> (q,/,A )

step 5 /

delta(q,eps,A) -> (q,b,N)

step 6 b

delta(q,eps,N) -> (q,e,B)

step 7 e

delta(q,eps,B) -> (q,w,C)

step 8 w

delta(q,eps,C) -> (q,@,W)

step 9 @

delta(q,eps,W) -> (q,a,D)

step 10 a

delta(q,eps,D) -> (q,e)

step 11 e

delta(q,e,e) -> (q,ε)

step 12

True

**Практическая работа №4 (9-11 лаб.)**

***Задание:***

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.

**Синтаксический LL-анализатор** - анализирует цепочку символов входного алфавита на ленте слева **(L)** направо, и строит левый **(L)** вывод грамматики.

**Определение 13.** 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∈V**:

**FIRST(A) = {a∈T | A => + aβ, где β∈(TᴗV) \* }** 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(α) = {a∈T | S => + α => + aβ, где α∈ (TᴗV) + , β∈(TᴗV) \* }**

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

**FOLLOW(A) = {a∈T | S=> \* αAγ и a = FIRST(γ)}**

Магазин содержит цепочку **Xa┴**, 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.

Конфигурацию “1-предсказывающего” алгоритма разбора будем представлять в виде **(x, Xa┴, π)**, где **x** – неиспользуемая часть входной цепочки, **Xa** – цепочка в магазине, а **π** – цепочка на выходной ленте, отражающая состояние алгоритма.

Обозначим алфавит магазинных символов (без символа (┴) ) как **Vp.**

**M** – управляющая таблица управляет работой алгоритма. **М** задает отображение множества **(VpᴗTᴗ{┴})×(Tᴗ{ε})** в множество, состоящее из следующих элементов**:**

**1. (β, i),** где **βVp \*** правая часть правила вывода с номером **i.**

**2. ВЫБРОС.**

**3. ДОПУСК.**

**4. ОШИБКА.**

Работа алгоритма в зависимости от элемента управляющей таблицы

**M(X,α) = (β, i)** следующая:

**1.** **(x, Xα, π)|- i (x, βα, πi)**, если **M(X,α) = (β, i)**. В этом случае верхний символ магазина **X** заменяется на цепочку **βVp \*** , и в выходную цепочку дописывается номер правила **i**. Выходная головка при этом не сдвигается.

**2. (ax, aα, π) |- (x, α, π)**, если **M(a,a) = ВЫБРОС**. Это означает, что, если верхний символ магазина совпадает с текущим входным символом, он выбрасывается из магазина, и входная головка сдвигается на один символ вправо.

**3.** Если алгоритм достигает конфигурации **(ε,┴ , π)**, что соответствует элементу управляющей таблицы **M(┴, ε) = ДОПУСК**, то его работа прекращается, и выходная цепочка **π** является левым разбором входной цепочки.

**4.** Если алгоритм достигает конфигурации **(x, Xα, π)** и **M(X,a) = ОШИБКА**, то разбор прекращается и выдается сообщение об ошибке. Конфигурация **(ω, S, ε)**, где **S∈Vp** – начальный символ магазина (начальный символ грамматики), называется *начальной конфигурацией*.

Если **(ω, S, ε) + (ε, , π)**, то **π** называется **выходом алгоритма для входа** **ω**.

**Алгоритм 3.10.** Построение 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 **(VᴗTᴗ{┴})×(Tᴗ{ε})** по правилам:

**1.** Для каждого терминала **a ≠ ε** из **FIRST(α), A → α** – правило грамматики с номером **i**, добавляем в **M(А, a) = (α, i)**.

Если **ε∈FIRST(α)**, то для каждого терминала **b** из **FOLLOW(A)** добавляем **A → α** в **M(А, b) = (α, i)**. Если **ε∈FIRST(α)** и **ε = FOLLOW(A)**, то добавляем **A → α** в **M(А, ε)**.

**2. M(a, a) = ВЫБРОС** для всех **a∈T**.

**3.** **M(┴, ε) = ДОПУСК**.

**4.** В остальных случаях **M(X, a) = ОШИБКА** для **X(VᴗT ᴗ{┴})** и **a∈T{ε}**

**1.Tаблица M для G имеет следующий вид:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | b | f | @ | a | / | w | e | ε |
| R |  | V, 1 |  |  |  |  |  |  |
| V |  |  | M,2 |  |  |  |  |  |
| L |  |  |  |  | A,4 |  |  |  |
| A | N,5 |  |  |  |  |  |  |  |
| N |  |  |  |  |  |  | B,6 |  |
| M |  |  |  | L, 3 |  |  |  |  |
| B |  |  |  |  |  | C,7 |  |  |
| C |  |  | W,8 |  |  |  |  |  |
| W |  |  |  | D,10 |  |  |  |  |
| D |  |  |  |  |  |  | eps,10 |  |
| b | ВЫБРОС |  |  |  |  |  |  |  |
| f |  | ВЫБРОС |  |  |  |  |  |  |
| @ |  |  | ВЫБРОС |  |  |  |  |  |
| a |  |  |  | ВЫБРОС |  |  |  |  |
| / |  |  |  |  | ВЫБРОС |  |  |  |
| w |  |  |  |  |  | ВЫБРОС |  |  |
| e |  |  |  |  |  |  | ВЫБРОС |  |
| ┴ |  |  |  |  |  |  |  | ДОПУСК |

Пустые клетки в таблице означают **ОШИБКУ.**

**Аналитическое представление для таблицы М выглядит следующим образом:**

|  |  |  |
| --- | --- | --- |
| Правило грамматики | Множество | Значение М |
| p1: R → fV  p2: V → @M  p3: M → aL  p4: L → /A  p5: A → bN  p6: N → eB  p7: B → wC  p8: C → @W  p9: W → aD  p10: D → e | FIRST(R) = {f}  FIRST(V) = {@}  FIRST(M) = {a}  FIRST(L) = {/}  FIRST(A) = {b}  FIRST(N) = {e}  FIRST(B) = {w}  FIRST(C) = {@}  FIRST(W) = {a}  FIRST(D) = {e} | М(R, f ) = V, 1  M(V, @) =M, 2  M(M, a) =L, 3  М(L, /) = A, 4  M(A, b) = N, 5  M(N, e) = B, 6  M(B, w) = C, 7  M(C, @) = W, 8  M(W, a) = D, 9  M(D, e) = eps, 10  M(е, е) = ВЫБРОС, е∈T  M(ε, ┴) = ДОПУСК |

Строить множество **FOLLOW** не было необходимости, так как грамматика не содержит **ε-правил**.

***2.* Рассмотрим работу алгоритма для цепочки (f@a/bew@ae)**

Шаг 1. Алгоритм находится в начальной конфигурации **(f@a/bew@ae), R┴, ε)**

Значение управляющей таблицы **М(R,f) = V, 1**; при этом выполняются следующие действия:

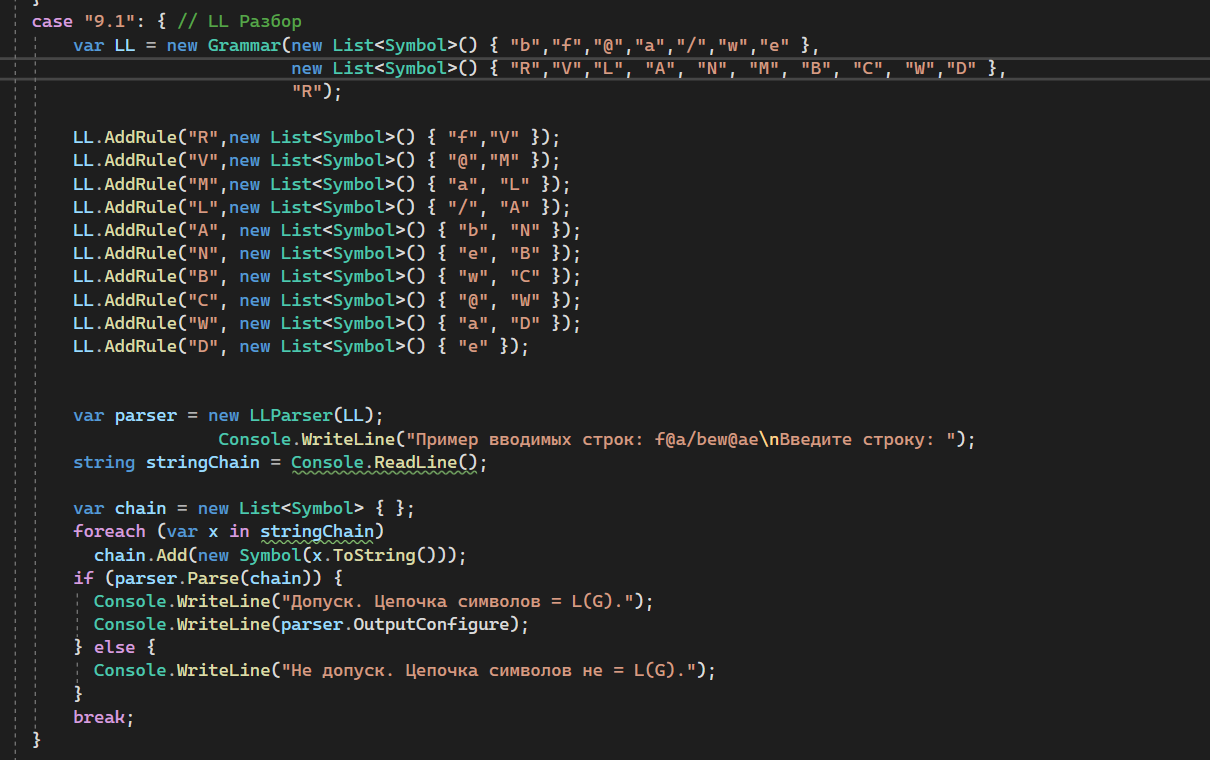
* Заменить верхний символ магазина **R** цепочкой **V**;
* Не сдвигать читающую головку;
* На выходную ленту поместить номер использованного правила 1.

Шаг 2. Получаем следующие конфигурации:

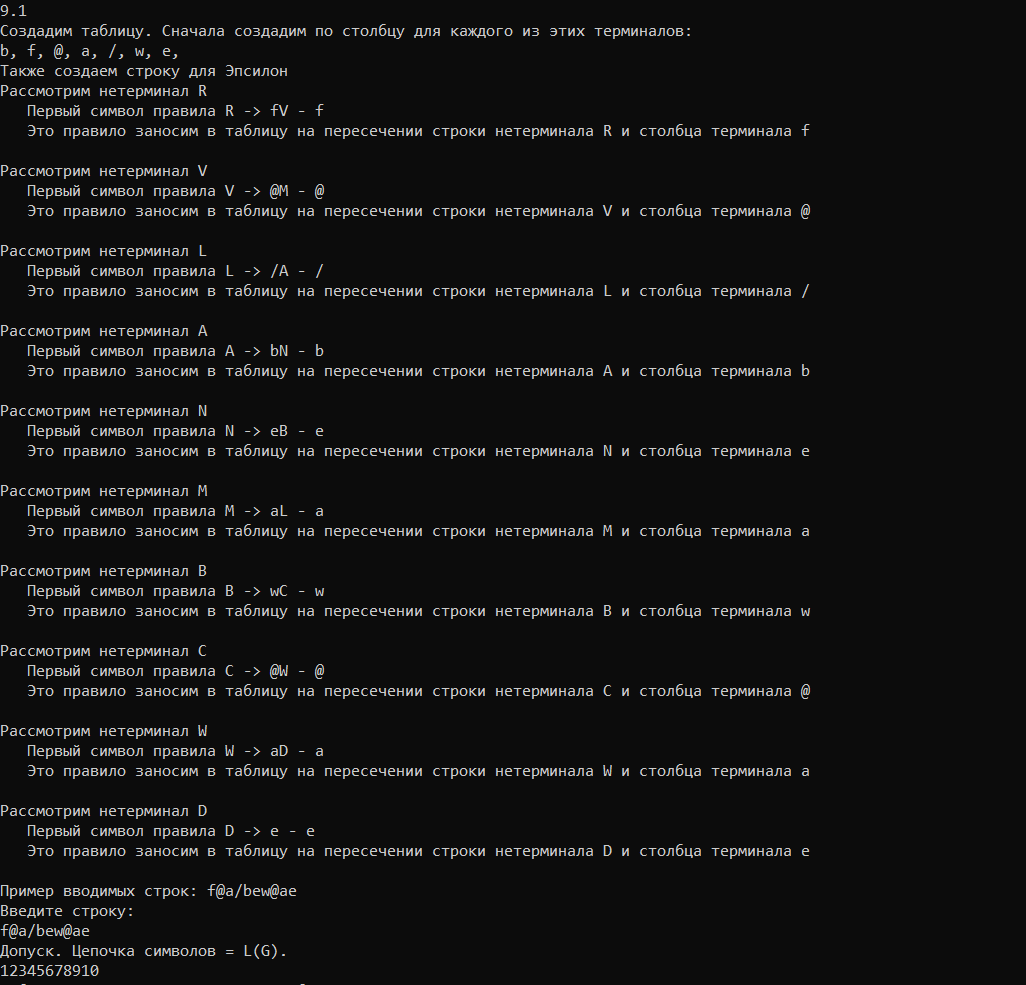
|  |  |
| --- | --- |
| Текущая конфигурация | Значение М |
| (f@a/bew@ae, V⊥, 1) ⎥⎯  (@a/bew@ae, M⊥, 12) ⎥⎯  (a/bew@ae, L)⊥, 123) ⎥⎯  (/bew@ae, A⊥, 1234) ⎥⎯  (bew@ae, N⊥, 12345) ⎥⎯  (ew@ae, B⊥, 123456⎥⎯  (w@ae, C)⊥, 1234567)⎥⎯  (@ae, W)⊥, 12345678)⎥⎯  (ae, D)⊥,123456789) ⎥⎯  (e, e,⊥, 12345678910) ⎥⎯  (e, eps,⊥, 12345678910) | М(R, f ) = V, 1  M(V, @) =M, 2  M(M, a) =L, 3  М(L, /) = A, 4  M(A, b) = N, 5  M(N, e) = B, 6  M(B, w) = C, 7  M(C, @) = W, 8  M(W, a) = D, 9  M(D, e) = eps, 10  M(e, e) = ВЫБРОС  M(ε, ⊥) = ДОПУСК |

**3. Реализация таблицы:**

**Код:**

****

**4.Вывод:**

**

**Лабораторная работа 12-16**

**Задача:**

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

13. Для LR(k) -грамматики спроектировать GOTO .

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.

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

Грамматики, для которых можно построить детерминированный восходящий анализатор, получили название LR(k)-*грамматик* (входная цепочка читается слева (**Left**) направо, выходом анализатора является правый (**Right**) разбор, k-число символов входной цепочки, на которое можно ―заглянуть вперед для выделения основы).

Для определения LR(k)-грамматики используются:

**1.** Множество **FIRSTk(γ**), состоящее из префиксов длины **k** терминальных цепочек, выводимых из **γ**.

Если из **γ** выводятся терминальные цепочки, длина которых меньше **k**, то эти цепочки также включаются в множество **FIRSTk (γ)**. Формально:

**FIRSTk (γ) = {x| γ =>1 \* xw** и **|x| = k** или **γ =>1 \* x** и **|x| < k }.**

**2.** Пополненной грамматикой, полученной из КС-грамматики **G = (T, V, P, S)**, называется грамматики **G' = (V ϵ {S'}, T, P = {S' → S}, S').** Если правила грамматики **G'** пронумерованы числами **1,2,..,p** то, будем считать, **что S' → S** – нулевое правило грамматики **G'**, а нумерация остальных правил такая же, как в грамматики **G.**

**Определение 15.** КС-грамматика **G = (T, V, P, S)** называется **LR(k)- грамматикой** **для k ≥ 0**, если из существования двух правых выводов для *пополненной* грамматики **G' = (T, V', P', S')** полученной из **G**:

**S' =>r \* αAw =>r αβw,**

**S' =>r \* γBx =>r γσx =>r αβy,**

для которых **FIRSTk(w) = FIRSTk(y)** следует, что **αAy = γBx**.

**Алгоритм "Перенос-Свертка"**

При **LR(k)-анализе** применяется алгоритм перенос-свертка (англ. shift-reduce). Суть метода сводится к следующему.

**1.** Входная цепочка символов считывается до тех пор, пока не накопится в магазине цепочка, совпадающая с правой частью какого-нибудь из правил **P**. Операция **Перенос** переносит символ в магазин.

**2.** Далее все символы совпадающей цепочки извлекаются из стека и на их место помещается нетерминал, находящийся в левой части этого правила (операция **Свертка**).

**Aлгоритм “перенос-свертка”** λ описывается в терминах конфигураций вида **(**⊥**X1 … Xm, a1 … an, p1…pr)**:

⊥**X1 … Xm** – содержимое магазина, **Xm** – верхний символ магазина и **Xi ∈ V ∪ T**, символ **(**⊥**)** дна магазина; **a1 … an** – оставшаяся непрочитанная часть входной цепочки, **a1** – текущий входной символ; **p1…pr** – разбор правил, полученный к данному моменту времени на ленте **π**.

Один шаг алгоритма **λ** можно описать с помощью двух отношений: **⎥⎯S (shift, Перенос)** и **⎥⎯r (reduce, Cвертка)**, определенных на конфигурациях следующим образом:

**1.** Если **f(α,aw) = ПЕРЕНОС (П)**, то входной символ переносится в верхушку магазина и читающая головка сдвигается на один символ вправо. В терминах конфигураций этот процесс описывается так: **(α,aw, π) ⎥**⎯**s (αa, w, π)** для **α ∈ (V ∪ T ∪ {**⊥**}) \* , w ∈ (T ∪ {ε}) \* и π ∈ {1,…,p} \*** .

**2.** Если **f(αβ, w) = СВЕРТКА (C), g(αβ, w) = i и A → β** – правило грамматики с номером **i**, то цепочка **β** заменяется правой частью правила с номером **i**, а его номер помещается на выходную ленту, т.е. **(α,aw, π) ⎥⎯S (αA, w, πi)**.

**3.** Если **f(α, w) = ДОПУСК (Д)**, то **(α, w, π) ⎥**⎯ **ДОПУСК**. В остальных случаях **(α, w, π) ⎥**⎯ **ОШИБКА** (пустое значение в таблице). Отношение **⎥**⎯определяется как объединение отношений **⎥**⎯**S** и **⎥**⎯**r** , а транзитивное замыкание отношений ⎥⎯**+** и **⎥**⎯**\*** определяется как обычно.

Для практического применения алгоритм мало пригоден, так как для определения функции переноса он требует анализа всей цепочки в магазине и всей необработанной части входной цепочки. Аналогичный недостаток имеет алгоритм и при определении функции свертки. Для **w ∈ T \*** будем записывать **λ(w) = π**, **если (**⊥**,w, ε) ⎥**⎯**\* (**⊥**S, ε, π) ⎥**⎯ **ДОПУСК**, или **λ(w) = ОШИБКА**, в противном случае.

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

Для любой **LR(k)-грамматики G = (T, V, P, S)** можно построить детерминированный анализатор, который выдает правый разбор входной цепочки. Анализатор состоит из магазина, входной ленты, выходной ленты и управляющего устройства (пара функций **f** и **g**).

**Определение.** *Грамматическое вхождение* – это символы полного словаря грамматики, снабженные двумя индексами. Первый индекс **i** задает номер правила грамматики, в правую часть которого входит данный символ, а второй индекс **j** – номер позиции символа в этой правой части.

**Два способа построения LR(k) анализаторов.**

**II**. Построение анализатора *LR(0)-грамматик* на основе **LR(0)-ситуаций**: функции замыкания **CLOSURE** и перехода **GOTO**.

**Определение.** *LR(0) ситуация* - это правило грамматики с точкой в некоторой позиции правой части, например **[A → w1 • w2]**, если **A → w1w2** – правило КС-грамматики.

Для правила **S → (S)** можно получить 4 ситуации:

**[S → • (S)]**

**[S → (•S)]**

**[S → (S•)]**

**[S → (S)•]**

**Шаг 1**. Определение ситуаций и построение конечного автомата.

Пусть **I** – множество **LR(0)-ситуаций** KC-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 **А → α •Bβ** уже включена в **CLOSURE(I)** и **В → γ** - правило грамматики, то добавить в множество **CLOSURE(I)** ситуацию **В → •γ** при условии, что там ее еще нет.

▪ Наличие ситуации **А → 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 **В → γ**, то также может встретиться во входном потоке анализатора подстрока, выводимая из **γ**, следовательно, в **CLOSURE(I)** нужно включить ситуацию **В → •γ**.

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

**Замыкание множества ситуаций.**

G’ = ({b, f, @, a, /, w, e}, {R, V, L, A, N, M, B , S’}, P’, R)

P:

p0: S’ -> R.

p1: R → fV

p2: V → @M

p3: M → aL

p4: L → /A

p5: A → bN

p6: N → eB

p7: B → w

Пусть вначале множество ситуаций **CLOSURE(I0)** содержит одну ситуацию **S' → • R**, т.е. **CLOSURE(I0) = { S' → • R }.**

Ситуация **S'→ •R** содержит точку перед символом **R**, поэтому по второму правилу в **CLOSURE(I0)** необходимо включить **S-правила** с точкой слева. Окончательно получим:

I0 = CLOUSER({S’-> •R})

I0 = {S’-> •R};

I0 = {R-> •fV};

I0 = {S’-> •R, R->•fV}

**Определение.** Функция **GOTO(I, X)** определяется как замыкание множества всех ситуаций **[ A→ αX•β ]**, таких что **[ A→ α•Xβ] ϵ I**.

**Функция переходов GOTO**

|  |  |  |  |
| --- | --- | --- | --- |
| I0 |  | CLOSURE ({S’->•R}) | {S’->•R, R->•fV} |
| I1 | GOTO (I0, f) | CLOSURE ({R->f•V}) | {R->f•V, V->•@M} |
| I2 | GOTO (I0, R) | CLOSURE ({S’->R•}) | {S’->R•} |
| I3 | GOTO (I1, @) | CLOSURE ({V->@•M}) | {V-> @•M, M-> •aL} |
| I4 | GOTO (I1, V) | CLOSURE ({R->fV•}) | {R->fV•} |
| I5 | GOTO (I3, a) | CLOSURE ({M->a•L}) | {M->a•L, L->•/A} |
| I6 | GOTO (I3, M) | CLOSURE ({V->@M•}) | {V->@M•} |
| I7 | GOTO (I5, /) | CLOSURE ({L->/•A}) | {L->/•A, A->•bN} |
| I8 | GOTO (I5, L) | CLOSURE ({M->aL•}) | {M->aL•} |
| I9 | GOTO (I7, b) | CLOSURE ({A->b•N}) | {A->b•N, N->•eB} |
| I10 | GOTO (I7, A) | CLOSURE ({L->/A•}) | {L->/A•} |
| I11 | GOTO (I9, e) | CLOSURE ({N->e•B}) | {N->e•B, B-> •w } |
| I12 | GOTO (I9, N) | CLOSURE ({A->bN•}) | {A->bN•} |
| I13 | GOTO (I11, w) | CLOSURE ({B->w•}) | {B->w•} |
| I14 | GOTO (I11, B) | CLOSURE ({N->eB•}) | {N->eB•} |

**Каноническая форма множества ситуаций.**

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

**1.** **φ = ∅**

**2.** Включить в φ множество **I0 = CLOSURE([S'→ • R])**, 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.

Используя каноническую систему **LR(0)–множеств**, можно представить функцию **GOTO** в виде диаграммы детерминированного конечного автомата. Диаграмма переходов ДКА для активных префиксов грамматики **G0**:

**Диаграмма переходов ДКА для активных префиксов грамматики G0:**

**Изображение выглядит как вешалка

Автоматически созданное описание**

Переход в состояние, которому на диаграмме соответствует лист (вершина, не имеющая исходящих дуг) однозначно определяет операцию Свёртки с возвратом к предыдущему состоянию, 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**[sm, ai] = Перенос(s)**, синтаксический анализатор выполняет перенос в стек очередного состояния **s** и его конфигурация становится **(s0s1... sms, ai+1... an$)**.

Символ **ai** хранить в стеке не нужно (он может быть восстановлен из **s**). Текущим входным символом становится **ai+1**.

**2.** Если операция**[sm, ai] = Свертка(i)** правила **pi: A→β**, синтаксический анализатор выполняет свертку в два шага и его конфигурация становится **(s0s1... sm-rs, aiai+1... an$)** здесь **r** - длина **β**, а **s = GOTO[sm-r, A]**.

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

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

**2.3.** После чего на вершину стека помещается **s**, запись из **GOTO[sm-r, A].** (По правилу для свертки **i** и левый нетерминал: **pi: A→ α** , определяется по таблице переходов состояние, которое должно быть занесено в стек). Последовательность символов грамматики **Хm-r+1...Хm** всегда соответствует **α**, правой части продукции свертки.

**3.** Если операция**[sm, ai] = допуск** синтаксический анализ завершается.

**4.** Если операция**[sm, ai] = ошибка** синтаксический анализатор вызывает программу восстановление после ошибки.

**Управляющая таблица.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I** | **f(u)** | | | | | | | | **g(x)** | | | | | | | |
| **f** | **@** | **a** | **/** | **b** | **e** | **w** | ┴ | **R** | **V** | **M** | **L** | **A** | **N** | **B** | **S’** |
| **0** | **S1** |  |  |  |  |  |  |  | **2** |  |  |  |  |  |  |  |
| **1** |  | **S3** |  |  |  |  |  |  |  | **4** |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  | **Д** |  |  |  |  |  |  |  |  |
| **3** |  |  | **S5** |  |  |  |  |  |  |  | **6** |  |  |  |  |  |
| **4** | **R1** | **R1** | **R1** | **R1** | **R1** | **R1** | **R1** | **R1** |  |  |  |  |  |  |  |  |
| **5** |  |  |  | **S7** |  |  |  |  |  |  |  | **8** |  |  |  |  |
| **6** | **R2** | **R2** | **R2** | **R2** | **R2** | **R2** | **R2** | **R2** |  |  |  |  |  |  |  |  |
| **7** |  |  |  |  | **S9** |  |  |  |  |  |  |  | **10** |  |  |  |
| **8** | **R3** | **R3** | **R3** | **R3** | **R3** | **R3** | **R3** | **R3** |  |  |  |  |  |  |  |  |
| **9** |  |  |  |  |  | **S11** |  |  |  |  |  |  |  | **12** |  |  |
| **10** | **R4** | **R4** | **R4** | **R4** | **R4** | **R4** | **R4** | **R4** |  |  |  |  |  |  |  |  |
| **11** |  |  |  |  |  |  | **S13** |  |  |  |  |  |  |  | **14** |  |
| **12** | **R5** | **R5** | **R5** | **R5** | **R5** | **R5** | **R5** | **R5** |  |  |  |  |  |  |  |  |
| **13** | **R7** | **R7** | **R7** | **R7** | **R7** | **R7** | **R7** | **R7** |  |  |  |  |  |  |  |  |
| **14** | **R6** | **R6** | **R6** | **R6** | **R6** | **R6** | **R6** | **R6** |  |  |  |  |  |  |  |  |

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

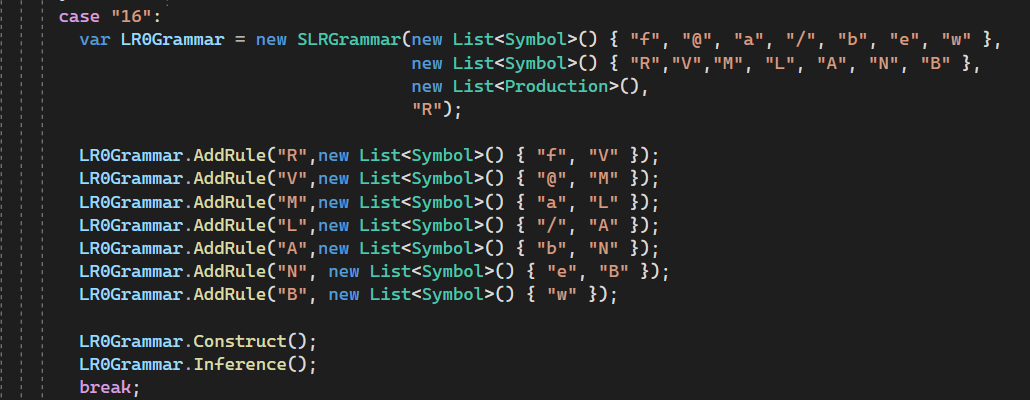
Работа алгоритма описывается в терминах конфигураций, представляющих собой тройки вида **(αT, ax, π)**, где **αT** – цепочка магазинных символов, **Т**-верхний символ магазина, **T** кодирует некий префикс цепочки (символ состояния), **ax** – необработанная часть входной цепочки, **π** – выход, построенный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

Рассмотрим последовательность тактов **LR(k)-алгоритма** при анализе входной цепочки **f@a/bew**. В магазине МП-автомата вместе с помещенным туда символами показаны и номера строк управляющей таблицы, в формате **«символ»«номер».**

**Распознавательная цепочка:**

(0, f@a/bew┴, ε) **⊢ П(1)** (0 1, @a/bew┴, ε) **⊢ П(3)** (0 1 3,a/bew┴, ε) **⊢ П(5**) (0 1 3 5, /bew┴, ε) **⊢ П(7)** (0 1 3 5 7, bew┴, ε) **⊢ П(9)** (0 1 3 5 7 9, ew┴, ε) **⊢ П(11)** (0 1 3 5 7 9 11, w┴, ε) **⊢ П(13)** (0 1 3 5 7 9 11 13, ┴, ε) **⊢ C(7)** (0 1 3 5 7 9 11 14, ┴, 7) **⊢ C(6)** (0 1 3 5 7 9 12, ┴, 6 7) **⊢ C(5)** (0 1 3 5 7 10, ┴, 5 6 7) **⊢ C(4)** (0 1 3 5 8, ┴, 4 5 6 7) **⊢ C(3)** (0 1 3 6, ┴, 3 4 5 6 7) **⊢ C(2**) (0 1 4, ┴, 2 3 4 5 6 7) **⊢ C(1)** (0 2, ┴, 1 2 3 4 5 6 7) ⊢ **Д**.

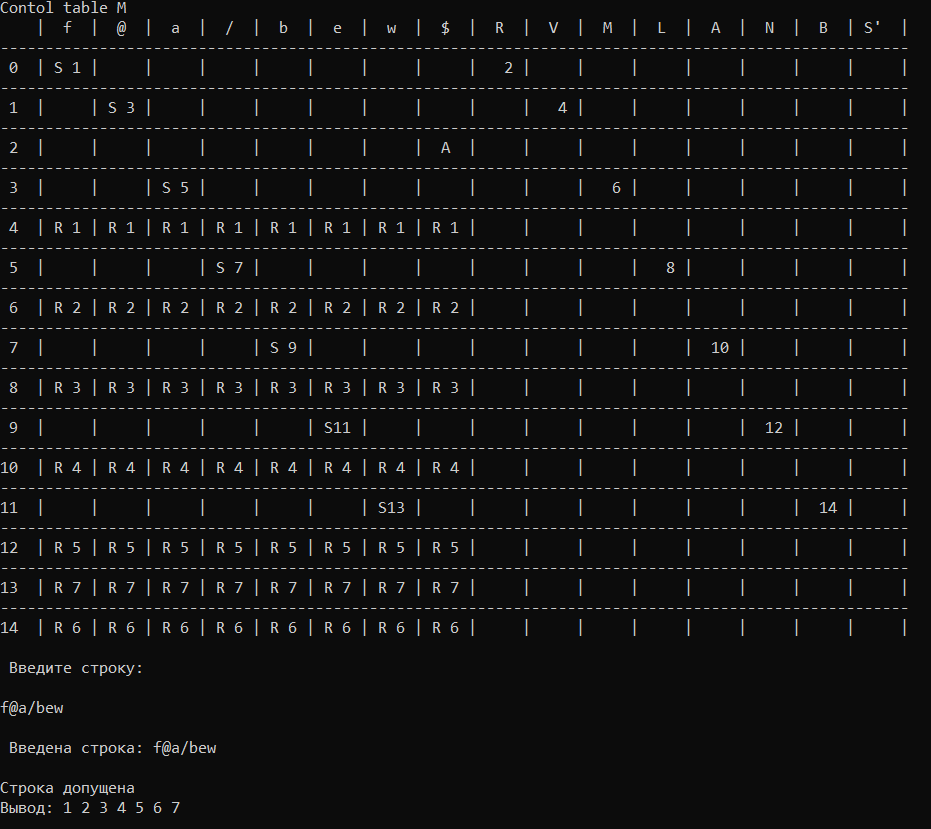
**Код программы:**



**Вывод программы:**

**Изображение выглядит как текст

Автоматически созданное описание**

****