0Московский авиационный институт

(национальный исследовательский институт)

Институт «Компьютерные науки и прикладная математика»

**Лабораторные работы**

**по курсу**

**«Системы программирования»**

**IV семестр**

**Практическая работа №1**

**1лаб**. Спроектировать грамматику по паттерн-модели регулярного языка.

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

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

**Практическая работа №2**

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

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

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

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

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

**Практическая работа №3**

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

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

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

**Практическая работа №4**

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

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

**14лаб.** Определить функции перехода g(X).

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

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

*Студент:* Карнаков Н.Д.*Группа: М8О-206Б-21*

*Руководитель:* Киндинова В. В.

*Оценка:*

*Дата:*

**Москва. 2023**

**Практическая работа №1 (1-3 лаб.)**

*Лабораторные работы №1-3*

**Формулировка задания**:

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

Спроектируем грамматику для заданного языка:

1. Сопоставление слова с разными вариантами написания или специальными символами

**1. pattern = "g[i!1][r@]a[vi@a]"**

**Автоматная грамматика:**

L(pattern) = L("g[i!1][g@]ra[vi@a]") = {“girav”, … , “g1@aa”}

G(T, V, P, S0) = G({g, i, !, 1, r, v, a, @}, {**S0**, **A, B, C, D, E**}, {p1, p2, p3, p4, p5}, **S0**)

*Правила регулярной грамматики:*

**p1:** **S0** → gA

**p2:** **A** → iB | !B | 1B

**p3:** **B** → rC | @C

**p4:** **C** → aD

**p5:** **D** → vE | iE | @E | aE

**p6:** **E** → ε

*Пример цепочек:*

**S0** =>1 g**A** =>2 gi**B** =>3 gir**C** =>4 gira**E** =>5 girav**E** =>6 girav

**S0** =>1 g**A** =>2 g!**B** =>3 g!@**C** =>4 g!@a**E** =>5 g!@aa**E** =>6 g!@aa

**S0** =>1 g**A** =>2 g1**B** =>3 g1r**C** =>4 g1ra**E** =>5 g1rai**E** =>6 g1rai

**Конечный автомат:**

L(КА) = L(G)

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

**Q** = { **S0**, **A**, **B, C, D, E,** **qf** }, **Σ** = { g, i, !, 1, r, v, a, @ }, **S0** = **S0**, **F** = **qf**,

**δ** = { 1. δ(**S0**, g) = {**A**},

2. δ(**A**, i) = {**B**},

3. δ(**A**, !) = {**B**},

4. δ(**A**, 1) = {**B**},

5. δ(**B**, r) = {**C**},

6. δ(**B**, @) = {**C**},

7. δ(**C**, a) = {**D**},

8. δ(**D**, v) = {**qf**},

9. δ(**D**, i) = {**qf**},

10. δ(**D**, @) = {**qf**},

11. δ(**D**, a) = {**qf**},

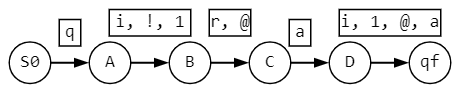
}

*Примеры конфигурации КА:*

**1.** (**S0**, g!rav) ⸠1 (**A**, !rav) ⸠3 (**B**, rav) ⸠5 (**C**, av) ⸠7 (**D**, v) ⸠8 (**qf**, ε)

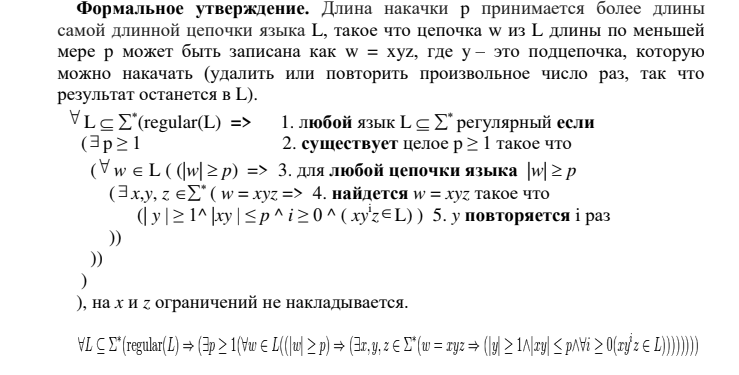
**2.** (**S0**, gi@a@) ⸠1 (**A**, i@a@) ⸠3 (**B**, @a@) ⸠6 (**C**, a@) ⸠7 (**D**, @) ⸠10 (**qf**, ε)

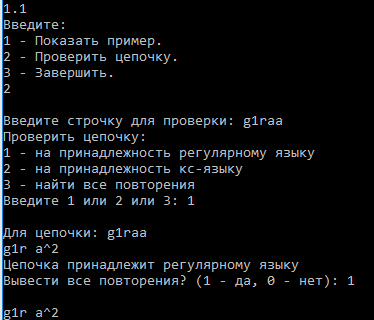
**3.** (**S0**, g!raa) ⸠1 (**A**, !raa) ⸠3 (**B**, raa) ⸠5 (**C**, aa) ⸠7 (**D**, a) ⸠11 (**qf**, ε)



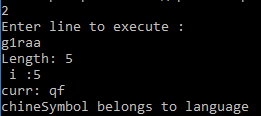
*Лемма о накачке:*

*Формулировка:*





**Проектирование КА:**



Код case 2:

var ka = new FSAutomate(new List<Symbol>() { "S0","A","B","C","D","qf" },

new List<Symbol>() { "i","1","!","g","a","@","v","r","" },

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

"S0");

ka.AddRule("S0","g","A");

ka.AddRule("A","i","B");

ka.AddRule("A","1","B");

ka.AddRule("A","!","B");

ka.AddRule("B","r","C");

ka.AddRule("B","@","C");

ka.AddRule("C","a","D");

ka.AddRule("D","v","qf");

ka.AddRule("D","i","qf");

ka.AddRule("D","a","qf");

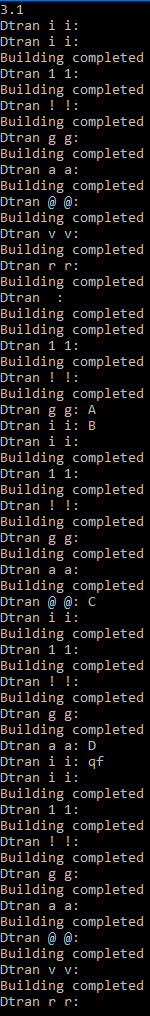
ka.AddRule("D","@","qf");

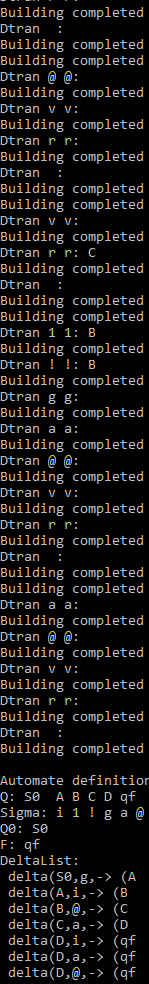
Console.WriteLine("Enter line to execute :");

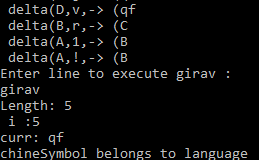
ka.Execute(Console.ReadLine());

break;

Перевод НДКА в ДКА:







Код case 3.1

var ndfsa = new FSAutomate(new List<Symbol>() { "S0", "A", "B", "C", "D", "qf" },

new List<Symbol>() { "i", "1", "!", "g", "a", "@", "v", "r", "" },

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

"S0");

ndfsa.AddRule("S0", "g", "A");

ndfsa.AddRule("A", "i", "B");

ndfsa.AddRule("A", "1", "B");

ndfsa.AddRule("A", "!", "B");

ndfsa.AddRule("B", "r", "C");

ndfsa.AddRule("B", "@", "C");

ndfsa.AddRule("C", "a", "D");

ndfsa.AddRule("D", "v", "qf");

ndfsa.AddRule("D", "i", "qf");

ndfsa.AddRule("D", "a", "qf");

ndfsa.AddRule("D", "@", "qf");

var dka = new FSAutomate();

dka.BuildDeltaDKAutomate(ndfsa,false);

dka.DebugAuto();

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

dka.Execute(Console.ReadLine());

break;

2. Совпадение с любым IP-адресом в диапазоне

**2. pattern = "192\.168\.0\.\d{1,3}""**

**Автоматная грамматика:**

**Автоматная грамматика:**

L(pattern) = L("192\.168\.0\.\d{1,3}") = {192.168.0.000, …, 192.168.0.999}

G(T, V, P, S0) = G({192.168.0., 0-9}, {S0, A, B, C, D}, {p1, p2}, **S0**)

*Правила регулярной грамматики:*

**p1:** **S0** → **1A**

**p2:** **A** → **9B**

**p3:** **B** → **2C**

**p4:** **C** → **.D**

**p5:** **D** → **1E**

**p6:** **E** → **6F**

**p7:** **F** → **8G**

**p8:** **G** → **.H**

**p9:** **H** → **0I**

**p10:** **I** → **.J**

**p11:** **J** → **0K | 1K | 2K | 3K | 4K | 5K | 6K | 7K | 8K | 9K**

**p12: K** → **0L | 1L | 2L | 3L | 4L | 5L | 6L | 7L | 8L | 9L**

**p13: L** → **0M | 1M | 2M | 3M | 4M | 5M | 6M | 7M | 8M | 9M**

**p14:** **M** → ε

*Пример цепочек:*

**S0** =>1 1A =>2 19B =>3 192C =>4 192.D =>5 192.1E =>6 192.16F =>7 192.168G =>8 192.168.H =>9 192.168.0I =>10 192.168.0.J =>11 192.168.0.3K =>12 192.168.0.32L =>13 192.168.0.321M =>14 192.168.0.321

**Конечный автомат:**

L(КА) = L(G)

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

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

**δ** = { 1. δ(**S0**, 1) = {**A**},

2. δ(A, 9)= {**B**},

3. δ(B, 2) = {**C**},

4. δ(C, .) = {**D**},

5. δ(D, 1) = {**E**},

6. δ(E, 6) = {**F**},

7. δ(F, 8) = {**G**},

8. δ(G, .) = {**H**},

9. δ(H, 0) = {**I**},

10. δ(I, .) = {**J**},

11. δ(J, 0-9) = {**K**},

12. δ(K, 0-9) = {**L**},

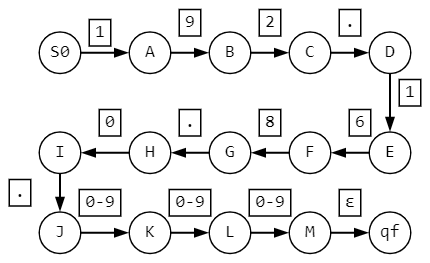
13. δ(L, 0-9) = {**M**},

14. δ(M, ε) = {**qf**}

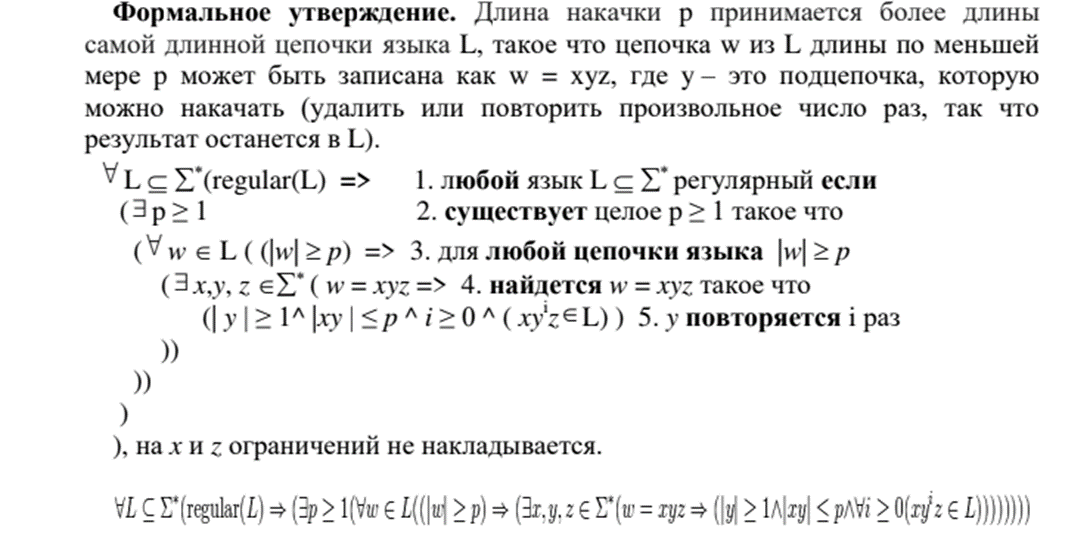
}

*Примеры конфигурации КА:*

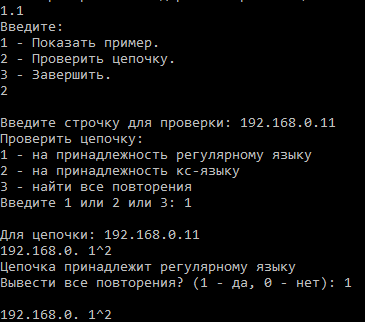
(**S0**, **192.168.0.321**) ⸠1 (**A**, **92.168.0.321**) ⸠2 (**B**, **2.168.0.321**) ⸠3 (**C**, **.168.0.321**) ⸠4 (**D**, **168.0.321**) ⸠5 (**E**, **68.0.321**) ⸠6 (**F**, **8.0.321**) ⸠7 (**G**, **.0.321**) ⸠8 (**H**, **0.321**) ⸠9 (**I**, **.321**) ⸠10 (**J**, **321**) ⸠11 (**K, 21**) ⸠12 (**L, 1**) ⸠13 **(M, ε)** ⸠14 (**qf**, ε)



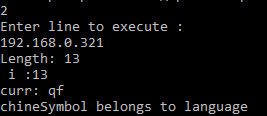
Формулировка:



*Лемма о накачке:*



*Проектирование КА:*



Код case 2

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

new List<Symbol>() { "0","1","2","3","4","5","6","7","8","9",".","" },

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

"S0");

ka.AddRule("S0","1","A");

ka.AddRule("A", "9", "B");

ka.AddRule("B", "2", "C");

ka.AddRule("C", ".", "D");

ka.AddRule("D", "1", "E");

ka.AddRule("E", "6", "F");

ka.AddRule("F", "8", "G");

ka.AddRule("G", ".", "H");

ka.AddRule("H", "0", "I");

ka.AddRule("I", ".", "J");

ka.AddRule("J", "0", "K");

ka.AddRule("J", "1", "K");

ka.AddRule("J", "2", "K");

ka.AddRule("J", "3", "K");

ka.AddRule("J", "4", "K");

ka.AddRule("J", "5", "K");

ka.AddRule("J", "6", "K");

ka.AddRule("J", "7", "K");

ka.AddRule("J", "8", "K");

ka.AddRule("J", "9", "K");

ka.AddRule("K", "0", "L");

ka.AddRule("K", "1", "L");

ka.AddRule("K", "2", "L");

ka.AddRule("K", "3", "L");

ka.AddRule("K", "4", "L");

ka.AddRule("K", "5", "L");

ka.AddRule("K", "6", "L");

ka.AddRule("K", "7", "L");

ka.AddRule("K", "8", "L");

ka.AddRule("K", "9", "L");

ka.AddRule("L", "0", "qf");

ka.AddRule("L", "1", "qf");

ka.AddRule("L", "2", "qf");

ka.AddRule("L", "3", "qf");

ka.AddRule("L", "4", "qf");

ka.AddRule("L", "5", "qf");

ka.AddRule("L", "6", "qf");

ka.AddRule("L", "7", "qf");

ka.AddRule("L", "8", "qf");

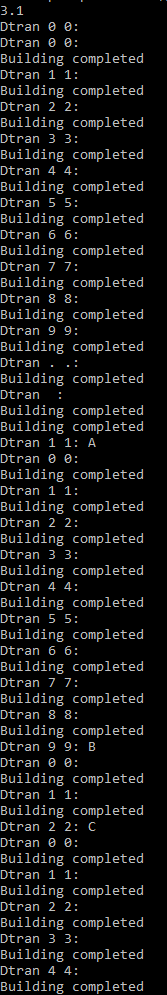
ka.AddRule("L", "9", "qf");

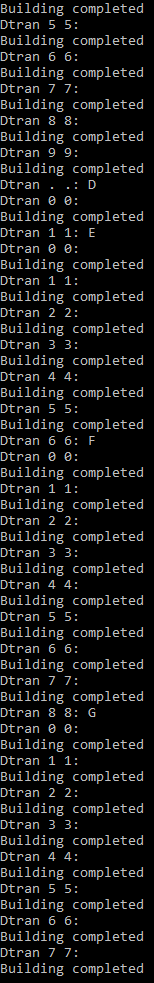
Console.WriteLine("Enter line to execute :");

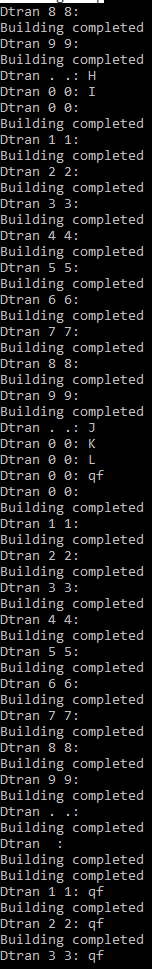
ka.Execute(Console.ReadLine());

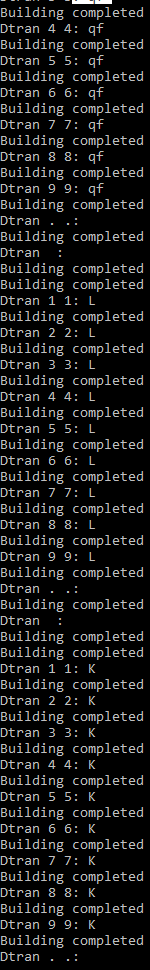
break;

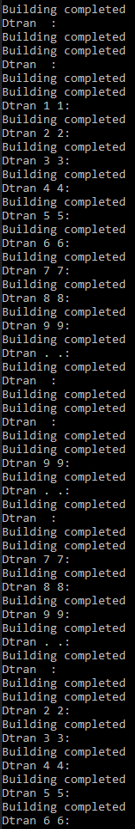
*Перевод НДКА в ДКА:*

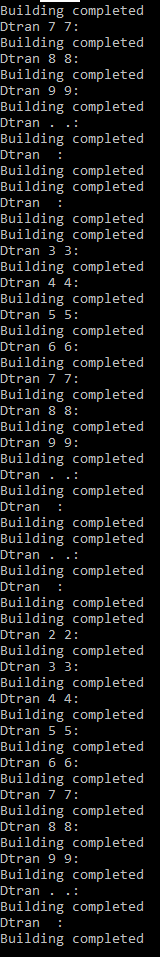


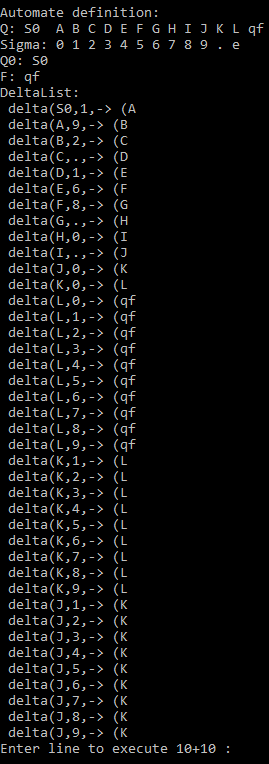


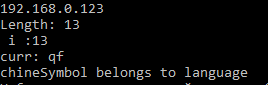












*Код case 3.1*

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

new List<Symbol>() { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ".", "" },

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

"S0");

ndfsa.AddRule("S0", "1", "A");

ndfsa.AddRule("A", "9", "B");

ndfsa.AddRule("B", "2", "C");

ndfsa.AddRule("C", ".", "D");

ndfsa.AddRule("D", "1", "E");

ndfsa.AddRule("E", "6", "F");

ndfsa.AddRule("F", "8", "G");

ndfsa.AddRule("G", ".", "H");

ndfsa.AddRule("H", "0", "I");

ndfsa.AddRule("I", ".", "J");

ndfsa.AddRule("J", "0", "K");

ndfsa.AddRule("J", "1", "K");

ndfsa.AddRule("J", "2", "K");

ndfsa.AddRule("J", "3", "K");

ndfsa.AddRule("J", "4", "K");

ndfsa.AddRule("J", "5", "K");

ndfsa.AddRule("J", "6", "K");

ndfsa.AddRule("J", "7", "K");

ndfsa.AddRule("J", "8", "K");

ndfsa.AddRule("J", "9", "K");

ndfsa.AddRule("K", "0", "L");

ndfsa.AddRule("K", "1", "L");

ndfsa.AddRule("K", "2", "L");

ndfsa.AddRule("K", "3", "L");

ndfsa.AddRule("K", "4", "L");

ndfsa.AddRule("K", "5", "L");

ndfsa.AddRule("K", "6", "L");

ndfsa.AddRule("K", "7", "L");

ndfsa.AddRule("K", "8", "L");

ndfsa.AddRule("K", "9", "L");

ndfsa.AddRule("L", "0", "qf");

ndfsa.AddRule("L", "1", "qf");

ndfsa.AddRule("L", "2", "qf");

ndfsa.AddRule("L", "3", "qf");

ndfsa.AddRule("L", "4", "qf");

ndfsa.AddRule("L", "5", "qf");

ndfsa.AddRule("L", "6", "qf");

ndfsa.AddRule("L", "7", "qf");

ndfsa.AddRule("L", "8", "qf");

ndfsa.AddRule("L", "9", "qf");

var dka = new FSAutomate();

dka.BuildDeltaDKAutomate(ndfsa,false);

dka.DebugAuto();

Console.WriteLine("Enter line to execute 10+10 :");

dka.Execute(Console.ReadLine());

break;

**Практическая работа №2 (лабораторные 4-8)**

*Лабораторная работа №4:*

**Формулировка задания:**

Устранить из КС-грамматики бесполезные символы и ε–правила

Исходная грамматика (Вариант №6):

**6 V G = {( a, b, c, d, f, g, epsilon, j, u, i ),( S, A, B, C, D, O, N, H, W, Z ),( A → BCD, B → cD, C → dfB, D → O, O → g, S → aAbN, N → epsilon, H → jS, W → j, S → iZ, Z → Zu, Z → u),S}**

p1: A → BCD

p2: B → cD

p3: C → dfB

p4: D → O

p5: O → g

p6: S → aAbN

p7: N → ε

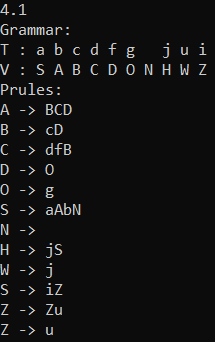
p8: H → jS

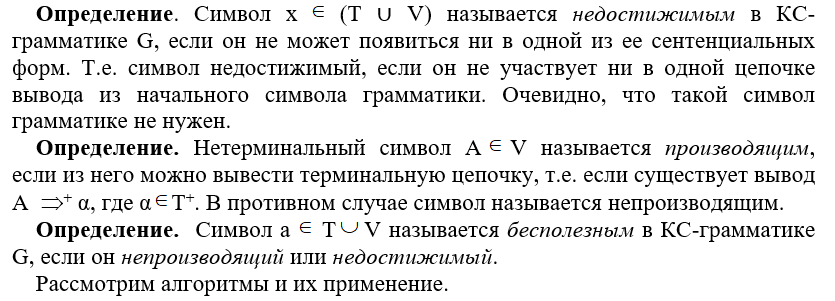
p9: W → j

p10: S → iZ

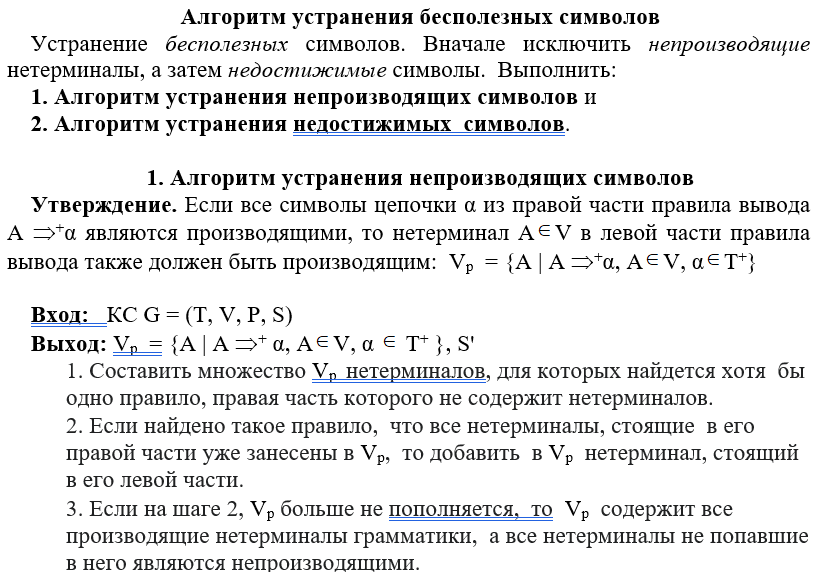
p11: Z → Zu

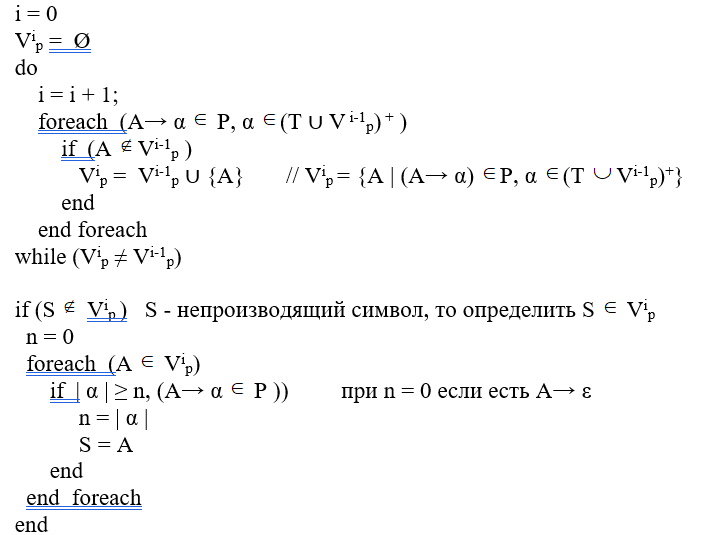
p12: Z → u



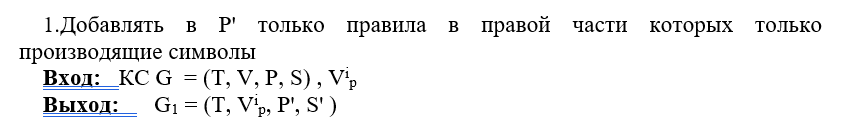


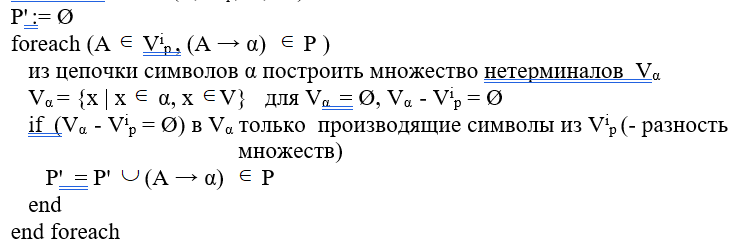
1 Этап – Устранение непроизводящих символов





1. V1p = {O, N, W, Z} – для этих нетерминалов нашелось хотя бы одно правило, правая часть которого не содержит нетерминалов;
2. V2p = {O, N, W, Z, D, S} – если найдено такое правило, что все нетерминалы, стоящие в его правой части уже занесены в V, то добавить в Vp, то добавить в Vp нетерминал, стоящий в его правой части;
3. V3p = {O, N, W, Z, D, S, B, H};
4. V4p = {O, N, W, Z, D, S, B, H, C};
5. V5p = {O, N, W, Z, D, S, B, H, C, A} – содержит все производящие нетерминалы грамматики.





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

P' = {p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12}

p1: A → BCD

p2: B → cD

p3: C → dfB

p4: D → O

p5: O → g

p6: S → aAbN

p7: N → ε

p8: H → jS

p9: W → j

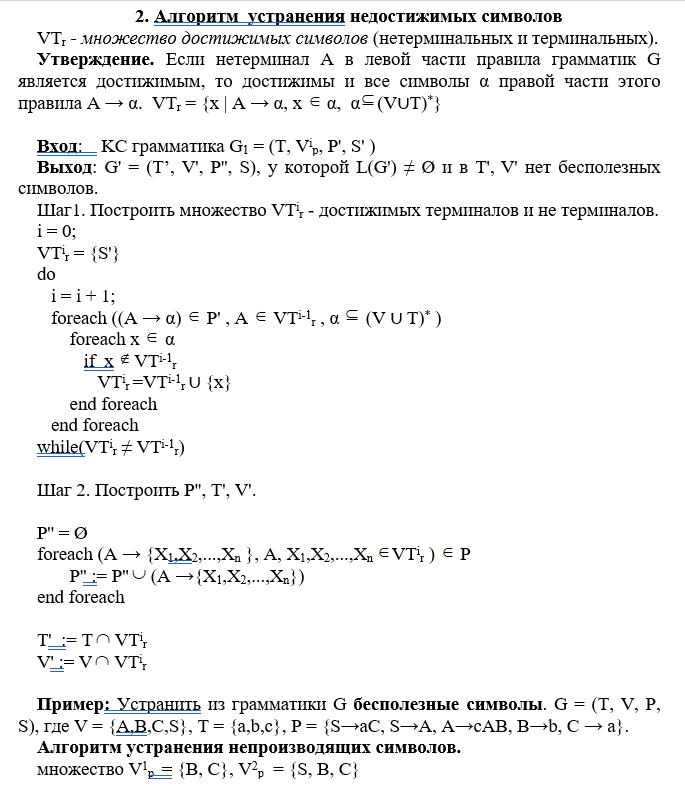
p10: S → iZ

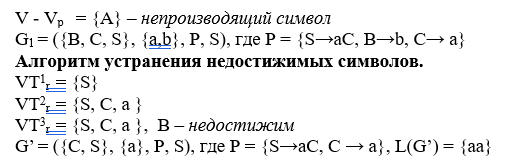
p11: Z → Zu

p12: Z → u

G1 = ({a, b, c, d, f, g, ε, j, i, u}, {A, B, C, D, O, N, H, W, S, Z}, P', S})

2 этап – Устранение недостижимых символов (VTr – множество недостижмых символов):





VT1r = {S}

VT2r = {S, a, A, b, N}

VT3r = {S, a, A, b, N, i, Z}

VT4r = {S, a, A, b, N, i, Z, B, C, D}

VT5r = {S, a, A, b, N, i, Z, B, C, D, ε}

VT6r = {S, a, A, b, N, i, Z, B, C, D, ε, u}

VT7r = {S, a, A, b, N, i, Z, B, C, D, ε, u, c}

VT8r = {S, a, A, b, N, i, Z, B, C, D, ε, u, c, d, f}

VT9r = {S, a, A, b, N, i, Z, B, C, D, ε, u, c, d, f, O}

VT10r = {S, a, A, b, N, i, Z, B, C, D, ε, u, c, d, f, O, g}

Так, не входящие в VTr символы {j} недостижимы

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

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

p1: A → BCD

p2: B → cD

p3: C → dfB

p4: D → O

p5: O → g

p6: S → aAbN

p7: N → ε

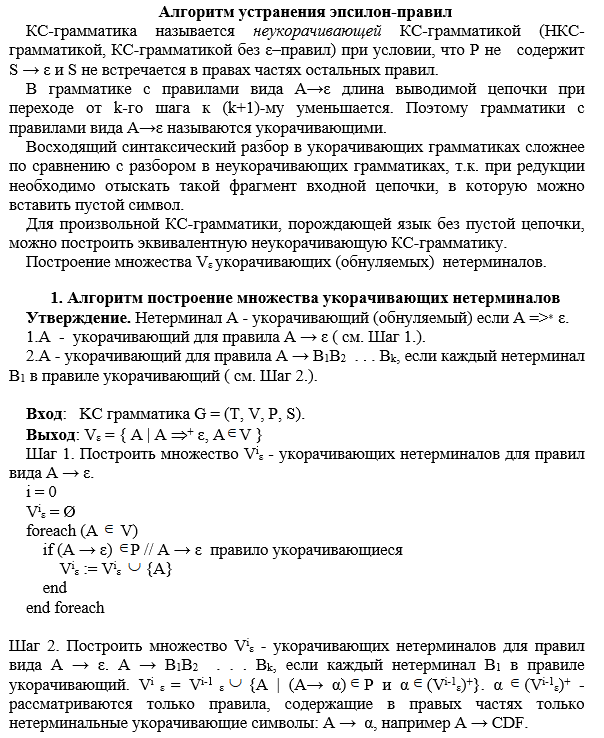
p8: S → iZ

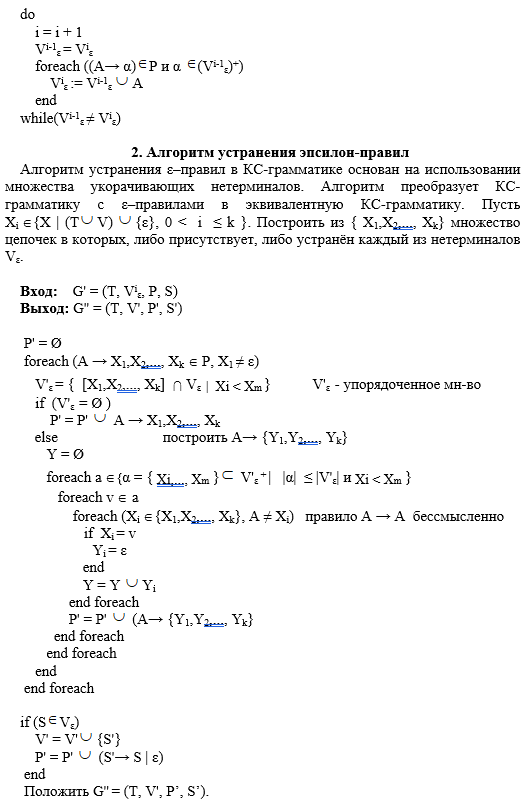
p9: Z → Zu

p10: Z → u

Получаем G' = ({a, b, c, d, f, g, ε, i, u }, {A, B, C, D, O, N, S, Z}, P'', S)

Этап 3 – устранить из КС-грамматики ε-правила.





Построим множество укорачивающих нетерминалов согласно алгоритму:

V0ε = {N} (множество укорачивающих нетерминалов для правил вида A → ε)

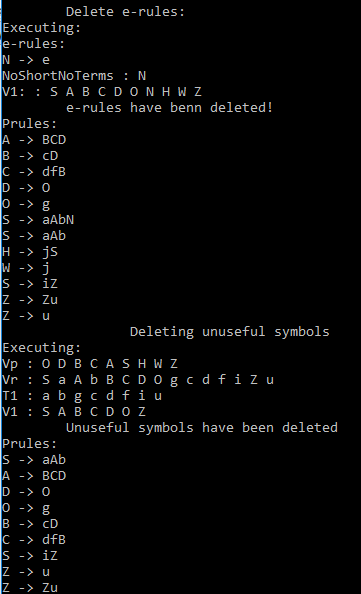
Теперь необходимо построить множество V1ε укорачивающих нетерминалов вида A → B1B2…Bk, если каждый нетерминал B1 в правиле укорачивающий. Однако в данном примере таких правил нет.

Итак, множество укорачивающих терминалов: Vε = {N}.

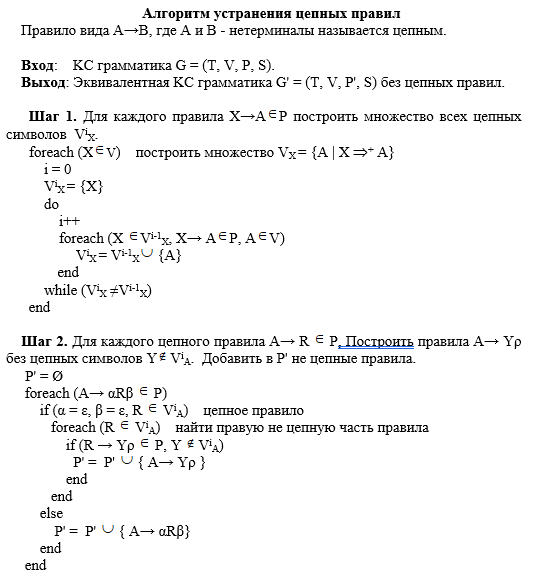
Составим G" на основе алгоритма удаления епсилон правил:

G' = ({a, b, c, d, f, g, i, u, ε}, {A, B, C, D, O, S, Z, N}, P'', S)

Получаем G'' = ({a, b, c, d, f, g, i, u}, {A, B, C, D, O, S, Z}, P'', S)



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

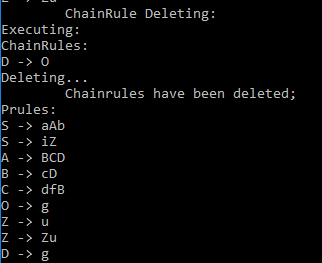


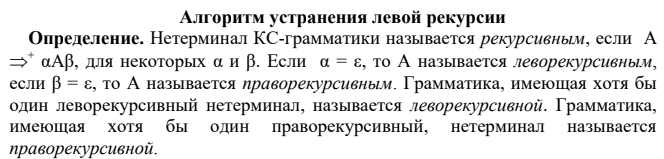
Цепные правила:

D → O, O → g

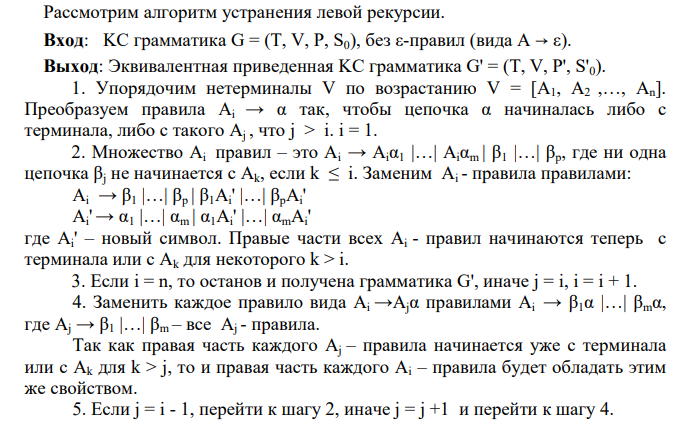
V1x = {O | D → O}

Правила D → O, O → g “заменим” на D → g

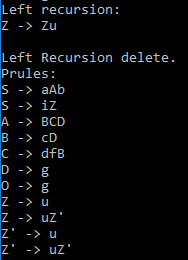




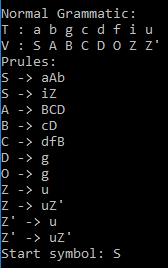
Устранение из КС-грамматики левой рекурсии



Левая рекурсия Z → Zu. Заменим правило Z → Zu на правила Z → uZ’, Z’ → u|uZ’



Приведенная грамматика:



Код:

case "4.1":

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

new List<Symbol>() { "S","A","B","C", "D", "O", "N", "H", "W", "Z" },

"S");

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

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

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

regGr.AddRule("D",new List<Symbol>() { "O" });

regGr.AddRule("O",new List<Symbol>() { "g" });

regGr.AddRule("S",new List<Symbol>() { "a", "A", "b", "N" });

regGr.AddRule("N",new List<Symbol>() { "" });

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

regGr.AddRule("W", new List<Symbol>() { "j" });

regGr.AddRule("S", new List<Symbol>() { "i", "Z" });

regGr.AddRule("Z", new List<Symbol>() { "Z", "u" });

regGr.AddRule("Z", new List<Symbol>() { "u" });

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: ");

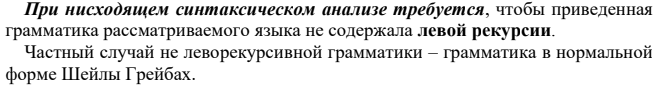
Console.WriteLine(G4.S0+"\n");

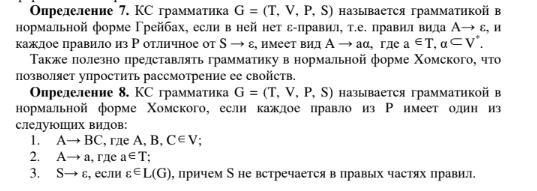
break;

*Лабораторная работа №6*

**Формулировка задания:**

Определить форму КС-грамматики и сделать ее приведение

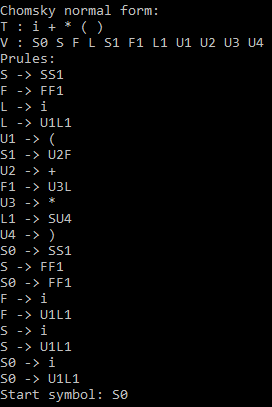




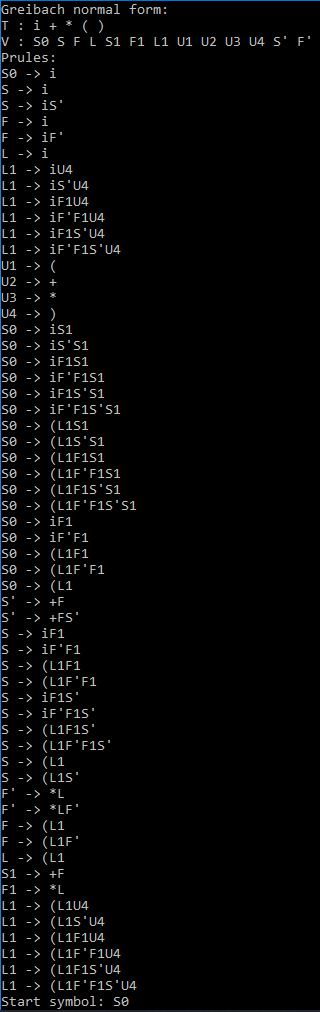
Приведенная грамматика не является грамматикой в нормальной форме Хомского, так как присутствует правило: S → aAbN.

Приведенная грамматика не является грамматикой в нормальной форме Грейбах, так как в ней есть правило вида A → BCD.

Приведем грамматику к нормальной форме Хомского:



Приведем грамматику к нормальной форме Грейбах:

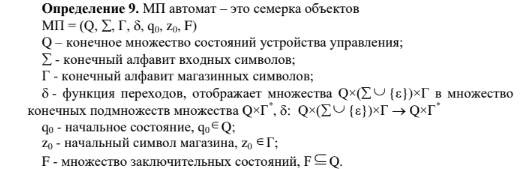


*Лабораторная работа №7*

**Формулировка задания:**

Спроектировать МП-автомат для приведённой КС-грамматики.

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



Приведенная грамматика (в форме Хомского):

G = (T, V, P, S), где

T = {a, b, g, c, d, f, u}, V = {S, S1, S2, S3, S4, S5, S6, S7, S8, A, B, C, D}, S0 = S

P:

p1: S → S4S1

p2: S4 → a

p3: S1 → AS5

p4: S5 → b

p5: A → BS2

p6: S2 → CD

p7: B → S6D

p8: S6 → c

p9: C → S7S3

p10: S7 → d

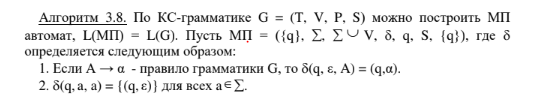
p11: S3 → S8B

p12: S8 → f

p13: D → g

Вывод цепочки:

S0 =>1 S4S1 =>2 aS1 =>3 aAS5 =>4 aAb =>5 aBS2b =>7 aS6DS2b =>8 acDS2b =>13 acgS2b =>6 acgCDb =>9 acgS7S3Db =>10 acgdS3Db =>11 acgdS8BDb =>12 acgdfBDb =>7 acgdfS6DDb =>8 acgdfcDDb =>13 acgdfcgDb =>13 acgdfcggb

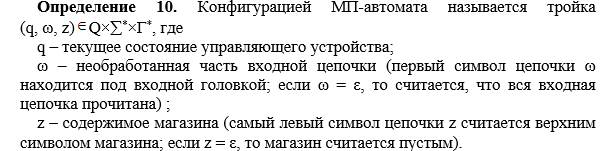


L(МП) = L(G)

МП = (Q, Σ, Г, δ, q0, z0, F):

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

МП = ({q}, { a, b, g, c, d, f, u }, {a, b, g, c, d, f, u, S, S1, S2, S3, S4, S5, S6, S7, S8, A, B, C, D}, δ, q0, S, {q})



Функция перехода **δ** определяется согласно алгоритму:



|  |  |
| --- | --- |
| p1: S → S4S1  p2: S4 → a  p3: S1 → AS5  p4: S5 → b  p5: A → BS2  p6: S2 → CD  p7: B → S6D  p8: S6 → c  p9: C → S7S3  p10: S7 → d  p11: S3 → S8B  p12: S8 → f  p13: D → g | 1. δ(q0, ε, S) = {(q, S4S1)}, 2. δ(q, ε, S4) = {(q, a)}, 3. δ(q, ε, S1) = {(q, AS5)}, 4. δ(q, ε, S5) = {(q, b)}, 5. δ(q, ε, A) = {(q, BS2)}, 6. δ(q, ε, S2) = {(q, CD)}, 7. δ(q, ε, B) = {(q, S6D)}, 8. δ(q, ε, S6) = {(q, c)}, 9. δ(q, ε, C) = {(q, S7S3)}, 10. δ(q, ε, S7) = {(q, d)}, 11. δ(q, ε, S3) = {(q, S8B)} 12. δ(q, ε, S8) = {(q, f)} 13. δ(q, ε, D) = {(q, g)} 14. δ(q, a, a) = {(q, ε)},  **Σ** |

Последовательность тактов МП-автомата для цепочки acgdfcgg:

(q0, acgdfcgg, S) ⸠1 (q, acgdfcgg, S4S1) ⸠2 (q, acgdfcgg, aS1) ⸠14 (q, cgdfcgg, S1) ⸠3 (q, cgdfcgg, AS5) ⸠5 (q, cgdfcgg, BS2) ⸠7 (q, cgdfcgg, S6DS2) ⸠8 (q, cgdfcgg, cDS2) ⸠14

(q, gdfcgg, DS2) ⸠13(q, gdfcgg, gS2) ⸠14(q, dfcgg, S2) ⸠6(q, dfcgg, CD) ⸠9 (q, dfcgg, S7S3D) ⸠10

(q, dfcgg, dS3D) ⸠14(q, fcgg, S3D) ⸠11 (q, fcgg, S8BD) ⸠12(q, fcgg, fBD) ⸠14 (q, cgg, BD) ⸠7

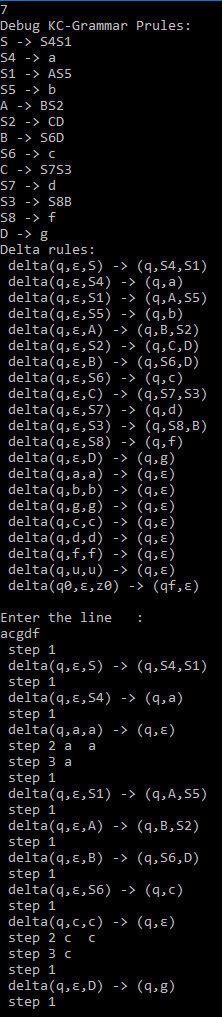
(q, cgg, S6DD) ⸠8(q, cgg, cDD) ⸠14(q, gg, DD) ⸠13 (q, gg, gD) ⸠14 (q, g, D) ⸠13(q, g, g) ⸠14

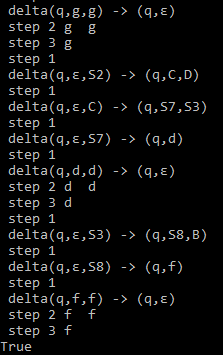
(q, ε, ε)

*Лабораторная работа №8*

**Формулировка задания:**

Реализовать спроектированный МП-автомат для приведённой КС-грамматики.





Код программы:

case "7": { // Algorithm Grammar to PDA не детерменированный

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

new List<Symbol>() { "S", "S1", "S2", "S3", "S4", "S5", "S6", "S7", "S8", "A", "B", "C", "D" },

"S");

CFGrammar.AddRule("S",new List<Symbol>() { "S4", "S1"});

CFGrammar.AddRule("S4",new List<Symbol>() { "a" });

CFGrammar.AddRule("S1",new List<Symbol>() { "A","S5"});

CFGrammar.AddRule("S5",new List<Symbol>() { "b" });

CFGrammar.AddRule("A",new List<Symbol>() { "B", "S2" });

CFGrammar.AddRule("S2",new List<Symbol>() { "C", "D" });

CFGrammar.AddRule("B",new List<Symbol>() { "S6", "D" });

CFGrammar.AddRule("S6",new List<Symbol>() { "c" });

CFGrammar.AddRule("C",new List<Symbol>() { "S7","S3" });

CFGrammar.AddRule("S7",new List<Symbol>() { "d" });

CFGrammar.AddRule("S3",new List<Symbol>() { "S8","B" });

CFGrammar.AddRule("S8", new List<Symbol>() { "f"});

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

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

CFGrammar.DebugPrules();

var pda = new PDA(CFGrammar);

pda.Debug();

Console.WriteLine("\nEnter the line :");

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

break;

}

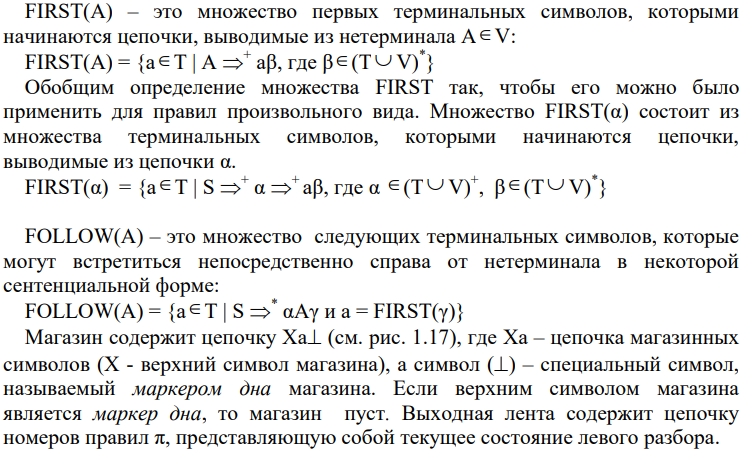
**Практическая работа №3 (лабораторные 9-11)**

*Лабораторная работа №9:*

**Формулировка задания:**

Для LL(k) анализатора построить управляющую таблицу M.

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

****

Исходная грамматика:

G = (**T**, **V**, **P**, **S**), где

**T** = {a, b, g, c, d, f, i, u}, **V** = {S, A, B, C, D, Z, Z’}, **S0** = S

**P**:

p1: S → aAb

p2: S → iZ

p3: A → BCD

p4: B → cD

p5: C → dfB

p6: D → g

p7: Z → u

p8: Z → uZ’

p9: Z’ → u

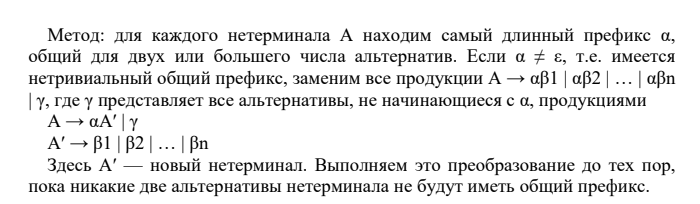
p10: Z’ → uZ’

**Преобразуем к LL(1) грамматике:**

Если грамматика не удовлетворяет требованиям, то применяют факторизацию - представляет собой преобразование грамматики в пригодную для предиктивного анализа. Основная идея левой факторизации заключается в том, что когда не ясно, какая из двух альтернативных продукций должна использоваться для нетерминала А, A-продукции можно переписать так, чтобы отложить принятие решения до тех пор, пока из входного потока не будет прочитано достаточно символов для правильного выбора.

**Изображение выглядит как текст

Автоматически созданное описание**



G = (**T**, **V**, **P**, **S**), где

**T** = {a, b, g, c, d, f, i, u}, **V** = {S, A, B, C, D, Z, Z’, T, T’}, **S0** = S

P:

p1: S → aAb

p2: S → iZ

p3: A → BCD

p4: B → cD

p5: C → dfB

p6: D → g

p7: Z → uT

p8: T → ε

p9: T → Z’

p10: Z’ → uT’

p11: T’ → ε

p12: T’ → Z’

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

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

*Выход:* Управляющая таблица M для грамматики G.

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

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

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **a** | **b** | **g** | **c** | **d** | **f** | **i** | **u** | **ε** |
| **S** | (aAb, 1) |  |  |  |  |  | (iZ, 2) |  |  |
| **A** |  |  |  | (BCD, 3) |  |  |  |  |  |
| **B** |  |  |  | (cD, 4) |  |  |  |  |  |
| **C** |  |  |  |  | (dfB, 5) |  |  |  |  |
| **D** |  |  | (g, 6) |  |  |  |  |  |  |
| **Z** |  |  |  |  |  |  |  | (uT,7) |  |
| **Z’** |  |  |  |  |  |  |  | (uT’,10) |  |
| **T** |  |  |  |  |  |  |  | (Z’,9) | (**ε, 8**) |
| **T’** |  |  |  |  |  |  |  | (Z’,12) | (**ε, 11**) |
| **a** | ВЫБРОС |  |  |  |  |  |  |  |  |
| **b** |  | ВЫБРОС |  |  |  |  |  |  |  |
| **g** |  |  | ВЫБРОС |  |  |  |  |  |  |
| **c** |  |  |  | ВЫБРОС |  |  |  |  |  |
| **d** |  |  |  |  | ВЫБРОС |  |  |  |  |
| **f** |  |  |  |  |  | ВЫБРОС |  |  |  |
| **i** |  |  |  |  |  |  | ВЫБРОС |  |  |
| **u** |  |  |  |  |  |  |  | ВЫБРОС |  |
| **Ʇ** |  |  |  |  |  |  |  |  | ДОПУСК |

Пустые клетки в таблице означают ОШИБКУ.

Ʇ - маркер дна

Аналитичекое представление для таблицы М:

|  |  |  |
| --- | --- | --- |
| Правило грамматики | Множество | Значение М |
| p1: S → aAb | FIRST(S) = {a} | M(S, a) = aAb, 1 |
| p2: S → iZ | FIRST(S) = {i} | M(S, i) = iZ, 2 |
| p3: A → BCD | FIRST(A) = {c} | M(A, c) = BCD, 3 |
| p4: B → cD | FIRST(B) = {c} | M(B, c) = cD, 4 |
| p5: C → dfB | FIRST(C) = {d} | M(C, d) = dfB, 5 |
| p6: D → g | FIRST(D) = {g} | M(D, g) = g, 6 |
| p7: Z → uT | FIRST(Z) = {u} | M(Z, u) = uT, 7 |
| p8: T → ε | FIRST(T) = {ε} | M(T, ε) = ε, 8 |
| p9: T → Z’ | FIRST(T) = {u} | M(T, u) = Z’, 9 |
| p10: Z’ → uT’ | FIRST(Z’) = {u} | M(Z’, u) = uT’, 10 |
| p11: T’ → ε | FIRST(T’) = {ε} | M(T’, ε) = ε, 11 |
| p12: T’ → Z’ | FIRST(T’) = {u} | M(T’, u) = Z’, 12 |

*Лабораторная работа №10:*

**Формулировка задания:**

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

Рассмотрим работу алгоритма для цепочки (acgdfcggb)

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

Значение управляющей таблицы M(K, f) = (A, B, 1), при этом выполняются следующие действия:

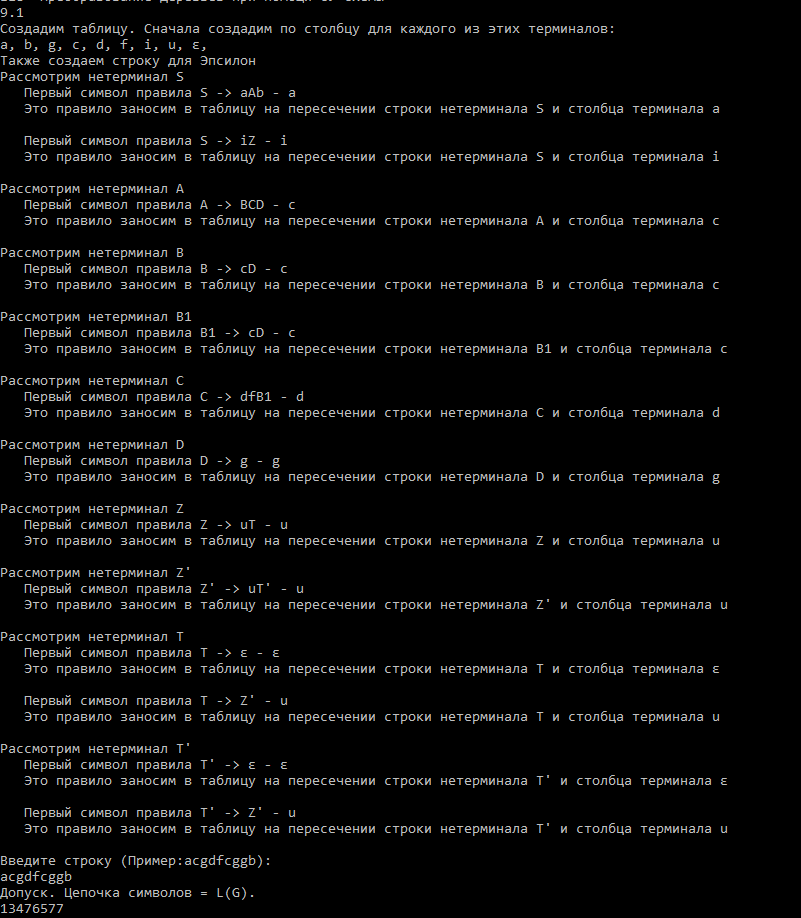
* Заменить верхний символ магазина R цепочкой **V**.
* Не сдвигать читающую головку.
* На выходную ленту поместить номер использованного правила 1.
* Шаг 2. Получаем следующие конфигурации:

|  |  |
| --- | --- |
| Текущая конфигурация | Значение М |
| (acgdfcggb, S⊥, ε) ⸠ | M(S, a) = aAb, 1 |
| (acgdfcggb, aAb⊥, 1) ⸠ | M(a, a) = ВЫБРОС |
| (cgdfcggb, Ab⊥, 1) ⸠ | M(A, c) = BCD, 3 |
| (cgdfcggb, BCDb⊥, 13) ⸠ | M(B, c) = cD, 4 |
| (cgdfcggb, cDCDb⊥, 134) ⸠ | M(c, c) = ВЫБРОС |
| (gdfcggb, DCDb⊥, 134) ⸠ | M(D, g) = g, 6 |
| (gdfcggb, gCDb⊥, 1346) ⸠ | M(g, g) = ВЫБРОС |
| (dfcggb, CDb⊥, 1346) ⸠ | M(C, d) = dfB, 5 |
| (dfcggb, dfBDb⊥, 13465) ⸠ | M(d, d) = ВЫБРОС |
| (fcggb, fBDb⊥, 13465) ⸠ | M(f, f) = ВЫБРОС |
| (cggb, BDb⊥, 13465) ⸠ | M(B, c) = cD, 4 |
| (cggb, cDDb⊥, 134654) ⸠ | M(c, c) = ВЫБРОС |
| (ggb, DDb⊥, 134654) ⸠ | M(D, g) = g, 6 |
| (ggb, gDb⊥, 1346546) ⸠ | M(g, g) = ВЫБРОС |
| (gb, Db⊥, 1346546) ⸠ | M(D, g) = g, 6 |
| (gb, gb⊥, 13465466) ⸠ | M(g, g) = ВЫБРОС |
| (b, b⊥, 13465466) ⸠ | M(b, b) = ВЫБРОС |
| (ε, ⊥, 13465466) | M(ε, ⊥) = ДОПУСК |

*Лабораторная работа №11:*

**Формулировка задания:**

Реализовать управляющую таблицу M Для LL(k) анализатора.



Код программы:

case "9.1":

{ // LL Разбор

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

new List<Symbol>() { "S", "A", "B", "B1", "C", "D", "Z", "Z'", "T", "T'" },

"S");

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

LL.AddRule("S", new List<Symbol>() { "i", "Z" });

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

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

LL.AddRule("B1", new List<Symbol>() { "c", "D" });

LL.AddRule("C", new List<Symbol>() { "d", "f", "B1" });

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

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

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

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

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

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

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

var parser = new LLParser(LL);

Console.WriteLine("Введите строку (Пример:acgdfcggb): ");

string stringChain = Console.ReadLine();

var chain = new List<Symbol> { };

foreach (var x in stringChain)

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

if (parser.Parse(chain))

{

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

Console.WriteLine(parser.OutputConfigure);

}

else

{

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

}

break;

}

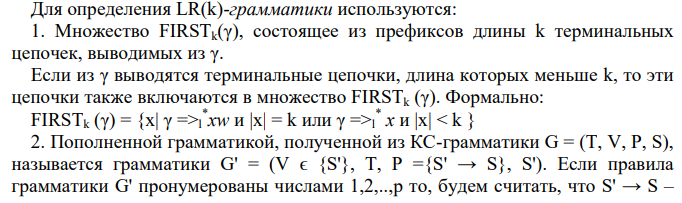
**Практическая работа №4 (лабораторные 12-16)**

**Формулировка задания:**

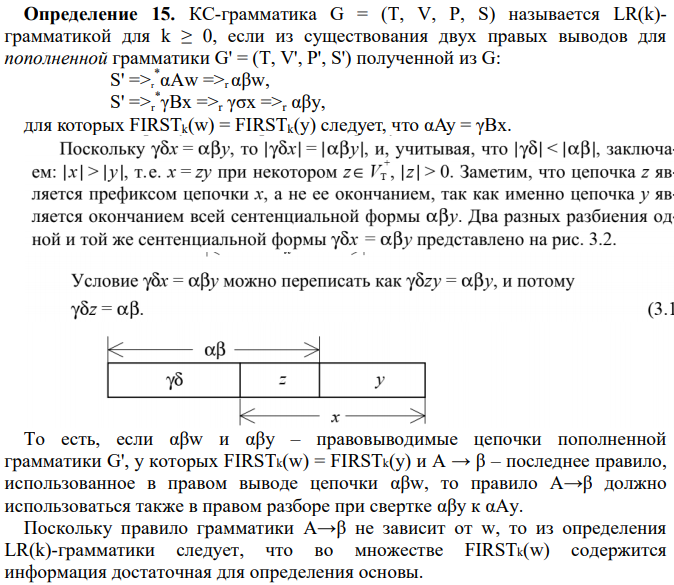
Построить управляющую таблицу M для LR(k)-грамматики, написать правило вывода выделенной строки. Описать работу алгоритма LR(k) анализатора. Построить LR(k) анализатор на основе грамматического вхождения.

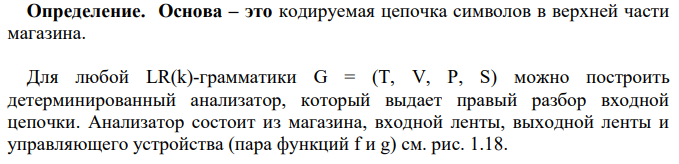
**Синтаксический LR-анализатор** анализирует входную цепочку слева направо (L), и строит правый (R) вывод грамматики.

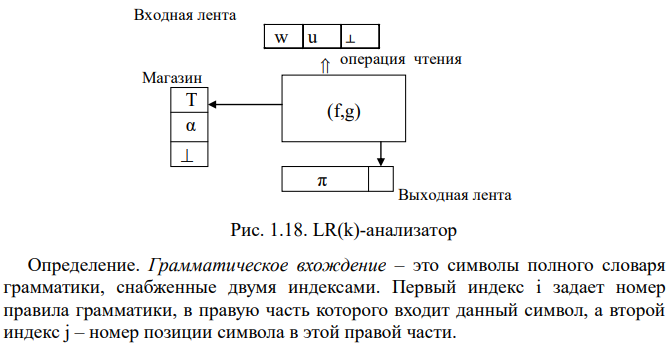
Грамматики, для которых можно построить детерминированный восходящий анализатор, называются LR(k)-*грамматикаaми* (входная цепочка читается слева (Left) направо, выходом анализатора является правый (Right) разбор, k – число символов входной цепочки, на которое можно “заглянуть” вперёд для выделения основы).











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

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

Пусть грамматические вхождения символов X и Z в правую часть i-го правила, а Yj - грамматическое вхождение символа Y в правую часть j-го правила. Определим множество OFIRST(Yi) (входит первым), в которое включим Yj и все грамматические вхождения, с которых могут начинаться цепочки, выводимые и Y:

OFIRST(Yi) = {Yj} U {Xi | Y =>\* Aβ => Xαβ и Xi самое левое грамматическое вхождение в правую часть правила A → X}

Если в грамматике есть правила с пустой правой частью, то на последующем шаге вывода они не применяются.

Отношение Xi OBLOW Yj определяет множество Q грамматических вхождений Xi, для которых представляющие их магазинные символы могут встретиться в магазине непосредственно под символом, представляющим Yj.

Отношение OBLOW будем представлять в виде матрицы, содержащей n столбцов и (n+1) рядов, где n - число грамматических вхождений пополненной грамматики G. Первые n рядов матрицы отмечены грамматическими вхождениями, а последний ряд - маркером дна стека. Если Xi OBLOW Yj, то элемент матрицы, расположенный в ряде Xi и столбце Yj равен 1.

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

G = (**T**, **V**, **P**, **S**), где

**T** = {a, b, g, c, d, f, i, u}, **V** = {S, A, B, C, D, Z, Z’}, **S0** = S

**P**:

p1: S → aAb

p2: S → iZ

p3: A → BCD

p4: B → cD

p5: C → dfB

p6: D → g

p7: Z → u

p8: Z → uZ’

p9: Z’ → u

p10: Z’ → uZ’

Непосредственно из правил вывода грамматики получим:

a1 OBLOW A1, A1 OBLOW b1, i2 OBLOW Z2, B3 OBLOW C3, C3 OBLOW D3, c4 OBLOW D4, d5 OBLOW f5, f5 OBLOW B5, u8 OBLOW Z’8, u10 OBLOW Z’10

Из определения отношения OBLOW следует, что ⊥ OBLOW Yj ⬄ Yj ∈ OFIRST(S0 = S). Из S можно вывести цепочки S => a1A1b1  и S => i2Z2  . Следовательно, OFIRST(S0 = S) = {a1, S0, i2}, и

⊥ OBLOW a1, ⊥ OBLOW S, ⊥ OBLOW i2

Рассмотрим правило грамматики с номером (1). Из определения отношения OBLOW следует, что:

a1 OBLOW Yj для всех Yj ∈ OFIRST(A1).

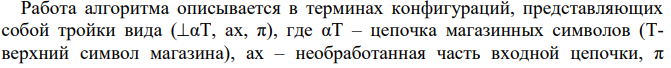
Из A1 можно вывести цепочки:

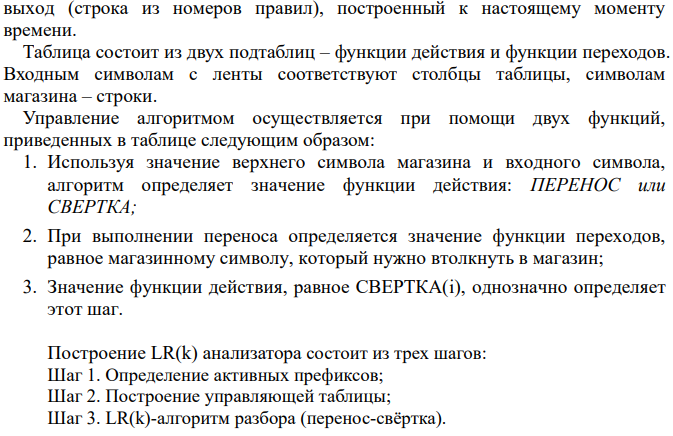
A1 =>2 B2C2D2 =>4 с4D4C2D2 =>6 с4g6C2D2 =>5 с4g6d5f5B5D2 =>4 с4g6d5f5с4D4D2 =>6 с4g6d5f5с4g6D2 =>6 с4g6d5f5с4g6g6

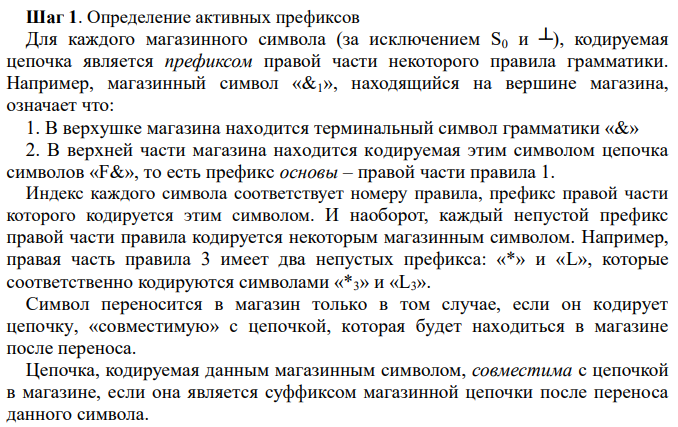
Следовательно, OFIRST(A1) = {c4} и a1 OBLOW c4

Поступая подобным образом для остальных правил, получим матрицу отношения OBLOW:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OBLOW** | S0 = S | a1 | b1 | i2 | c4 | d5 | f5 | g6 | u7 | u8 | u9 | u10 | A1 | Z2 | B3 | C3 | D3 | D4 | B5 | Z'8 | Z'10 |
| S0 = S |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| a1 |  |  |  |  | 1 |  |  |  |  |  |  |  | 1 |  | 1 |  |  |  |  |  |  |
| b1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i2 |  |  |  |  |  |  |  |  | 1 |  |  |  |  | 1 |  |  |  |  |  |  |  |
| c4 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 1 |  |  |  |
| d5 |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| f5 |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |
| g6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| u7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| u8 |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  | 1 |  |
| u9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| u10 |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 1 |
| A1 |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Z2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B3 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |
| C3 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |
| D3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Z'8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Z'10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ⊥ | 1 | 1 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |







**P**:

p1: S → aAb p5: C → dfB p9: Z’ → u

p2: S → iZ p6: D → g p10: Z’ → uZ’

p3: A → BCD p7: Z → u

p4: B → cD p8: Z → uZ’

Таблица закодированных символов:

|  |  |  |  |
| --- | --- | --- | --- |
| Символ грамматики | Магазинный символ | Кодируемая цепочка | Операции |
| S0 = S | S0 | S | Д |
| A | A1 | aA | П |
| B | B3  B5 | B  dfB | П  С5 |
| C | C3 | BC | П |
| D | D3  D4 | BCD  cD | C3  C4 |
| Z | Z2 | iZ | C3 |
| Z’ | Z’8  Z’10 | uZ’  uZ’ | C8  C10 |
| a | a1 | a | П |
| b | b1 | aAb | C1 |
| c | c4 | c | П |
| d | d5 | d | П |
| f | f5 | df | П |
| g | g6 | g | C6 |
| i | i3 | i | П |
| u | u7  u8  u9  u10 | u  u  u  u | C7  П  C9  П |

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

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

Построим матрицу функции переходов g(x):

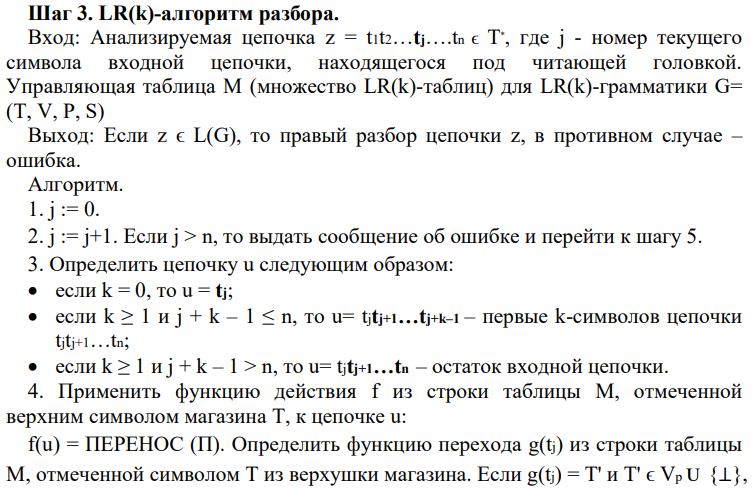
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | f | g | i | u | S | A | B | C | D | Z | Z’ |
| ⊥00 | a11 |  |  |  |  |  |  |  | S01 |  |  |  |  |  |  |
| a11 |  |  | c41 |  |  |  |  |  |  | A11 | B31 |  |  |  |  |
| S01 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| c41 |  |  |  |  |  | g61 |  |  |  |  |  |  | D42 |  |  |
| i21 |  |  |  |  |  |  |  | u71 |  |  |  |  |  | Z22 |  |
| A12 |  | b13 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B31 |  |  |  | d51 |  |  |  |  |  |  |  | C32 |  |  |  |
| b13 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| d51 |  |  |  |  | f52 |  |  |  |  |  |  |  |  |  |  |
| C32 |  |  |  |  |  | g61 |  |  |  |  |  |  | D33 |  |  |
| g61 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D33 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| u71 u91 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| u81  u10 1 |  |  |  |  |  |  |  | u91  u91 |  |  |  |  |  |  | Z’82  Z’10 2 |
| Z22 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D42 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| f52 |  |  |  |  |  | c41 |  |  |  |  | B53 |  |  |  |  |
| B53 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Z’82 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Z’10 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

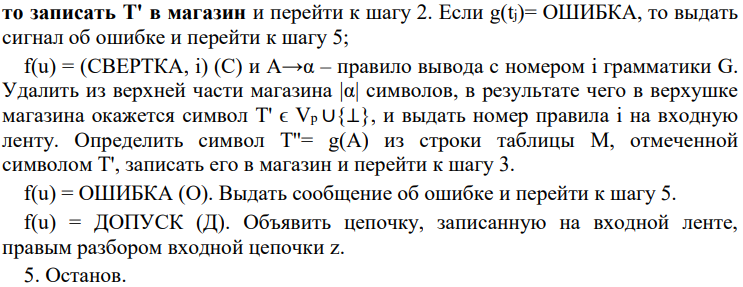
p1: S → aAb p5: C → dfB p9: Z’ → u

p2: S → iZ p6: D → g p10: Z’ → uZ’

p3: A → BCD p7: Z → u

p4: B → cD p8: Z → uZ’





Кодируемому магазинному символу соответствует единственное вхождение (детерменированный КА).

Определить функция действий F(u) на множестве (Vp{×(T{ε}) по правилам:

1. Если A → β – правило грамматики с номером i, то для конфигурации (αT, ax, π), где T кодирует цепочку β, F(u) = С(i) = (СВЕРТКА, i).

Если магазинному символу г соответствует только одно грамматическое вхождение Xi, являющееся самым правым вхождением в i-е правило вывода грамматики G, то все элементы ряда, помеченной г, имеют значение (СВЕРТКА, i).

1. Если A → β – правило грамматики с номером i, то для конфигурации (αT, ax, π), где г кодирует некий префикс цепочки β (но не саму основу), F(u) = Перенос.
2. Для конфигурации (S0, π), где S0 кодирует цепочку S, F(u) = ДОПУСК.
3. В противном случае, F(u) = ОШИБКА.

Рассмотрим работу алгоритма на примере цепочки (acgdfcggb):

(⊥,acgdfcggb⊥,ɛ) ⸠ (⊥a1, cgdfcggb⊥,ɛ) ⸠ (⊥a1с4,gdfcggb⊥,ɛ) ⸠ (⊥a1с4g6,dfcggb⊥,ɛ) ⸠ (⊥a1с4D4,dfcggb⊥,6) ⸠ (⊥a1B2d5,fcggb⊥,64) ⸠ (⊥a1B2d5d5f5,cggb⊥,64) ⸠ (⊥a1B2d5f5с4,ggb⊥,64) ⸠ (⊥a1B2d5f5с4g6,gb⊥,64) ⸠ (⊥a1B2d5f5с4D4,gb⊥,646) ⸠ (⊥a1B2d5f5B5,gb⊥,6464) ⸠ (⊥a1B2C2,gb⊥,64645) ⸠ (⊥a1B2C2g6,b⊥,64645) ⸠ (⊥a1B2C2D2,b⊥,646456) ⸠ (⊥a1A1,b⊥,6464562) ⸠ (⊥a1A1b1,⊥,6464562) ⸠ (⊥S0,⊥,64645621) ⸠ ДОПУСК.

Управляющая таблица (функция действий f(x) + функция переходов g(x)):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Функция действий f(x) | | | | | | | | | Функция переходов g(x) | | | | | | | | | | | | | | |
| a | b | c | d | f | g | i | u | ⊥ | a | b | c | d | f | g | i | u | S | A | B | C | D | Z | Z’ |
| ⊥00 | П |  |  |  |  |  | П |  |  | a11 |  |  |  |  |  |  |  | S01 |  |  |  |  |  |  |
| a11 |  |  | П |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | A11 |  |  |  |  |  |
| S01 |  |  |  |  |  |  |  |  | Д |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| c41 |  |  |  |  |  | П |  |  |  |  |  |  |  |  | g61 |  |  |  |  |  |  | D42 |  |  |
| i21 |  |  |  |  |  |  |  | П |  |  |  |  |  |  |  |  | u71 |  |  |  |  |  | Z22 |  |
| A12 |  | П |  |  |  |  |  |  |  |  | b13 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B31 |  |  |  | П |  |  |  |  |  |  |  |  | d51 |  |  |  |  |  |  |  | C32 |  |  |  |
| b13 |  |  |  |  |  |  |  |  | С1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| d51 |  |  |  |  | П |  |  |  |  |  |  |  |  | f52 |  |  |  |  |  |  |  |  |  |  |
| C32 |  |  |  |  |  | П |  |  |  |  |  |  |  |  | g61 |  |  |  |  |  |  | D33 |  |  |
| g61 |  | С6 |  | С6 |  | С6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D33 |  | C2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| u71 u91 |  | C7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| u81  u101 |  | C8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | u91  u91 |  |  |  |  |  |  | Z’82  Z’10 2 |
| Z22 |  | C2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D42 |  |  |  | C4 |  | C4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| f52 |  |  |  | П |  |  |  |  |  |  |  |  |  |  | c41 |  |  |  |  | B53 |  |  |  |  |
| B53 |  |  |  |  |  | С5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Z’82 |  | С8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Z’102 |  | С10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Исходный код:

public void Example\_LR1(){

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

Console.WriteLine("S aAb");

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

Console.WriteLine("S iZ");

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

Console.WriteLine("A BCD");

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

Console.WriteLine("B cD");

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

Console.WriteLine("C dfB");

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

Console.WriteLine("D g");

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

Console.WriteLine("Z u");

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

Console.WriteLine("Z uZ'");

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

Console.WriteLine("Z' u");

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

Console.WriteLine("Z' uZ'");

grammar.T.AddRange(new List<Symbol> { new Symbol("a"), new Symbol("b"), new Symbol("c"), new Symbol("d"), new Symbol("f"), new Symbol("g"), new Symbol("i"), new Symbol("u") });

grammar.V.AddRange(new List<Symbol> { new Symbol("S"), new Symbol("A"), new Symbol("B"), new Symbol("C"), new Symbol("D"), new Symbol("Z"), new Symbol("Z'") });

}

