Московский авиационный институт

(национальный исследовательский институт)

Институт «Компьютерные науки и прикладная математика»

**Лабораторные работы**

**по курсу**

**«Системы программирования»**

**IV семестр**

1. Спроектировать грамматику по паттерн-модели регулярного языка.

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

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

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

5. Устранить из 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 cпроектировать 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:* Орусский В.Р.

*Группа: М8О-206Б-21*

*Руководитель: Киндинова В. В.*

*Оценка:*

*Дата:*

Оглавление

[**Практическая работа №1 (1-3 лаб.)** 3](#_Toc146809900)

[*Лабораторные работы №1-3* 3](#_Toc146809901)

[**Практическая работа №2 (лабораторные 4-8)** 16](#_Toc146809902)

[*Лабораторная работа №4:* 16](#_Toc146809903)

[*Лабораторная работа №5: Устранить из KС-грамматики цепные правила и левую рекурсию* 21](#_Toc146809904)

[*Лабораторная работа №6* 23](#_Toc146809905)

[*Лабораторная работа №7* 28](#_Toc146809906)

[*Лабораторная работа №8* 31](#_Toc146809907)

[**Практическая работа №3 (лабораторные 9-11)** 34](#_Toc146809908)

[*Лабораторная работа №9:* 34](#_Toc146809909)

[*Лабораторная работа №10:* 37](#_Toc146809910)

[*Лабораторная работа №11:* 38](#_Toc146809911)

[**Практическая работа №4 (лабораторные 12-16)** 41](#_Toc146809912)

**Практическая работа №1 (1-3 лаб.)**

## *Лабораторные работы №1-3*

Формулировка задания:

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

Спроектируем грамматику для заданного языка:

1. Нахождение внутри текста обособленного слова baloney без чувствительности к регистру

**pattern = “(?i)(\W|^)(baloney)(\W|$)”**

**Автоматная грамматика:**

***Правила регулярной грамматики:***

***Пример цепочек:***

**Конечный автомат:**

, где , ,  
,

***Примеры конфигурации КА:***

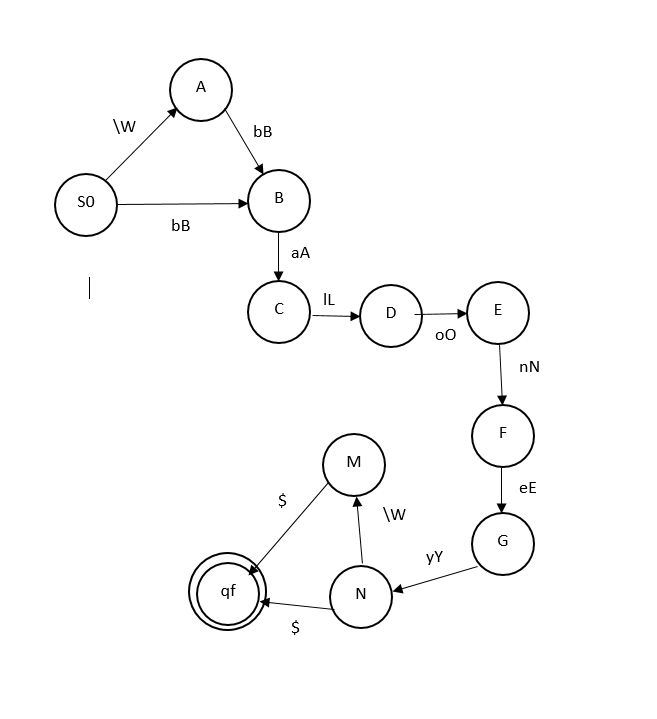


Рисунок 1 – Визуальное представление КА

**Лемма о накачке:**

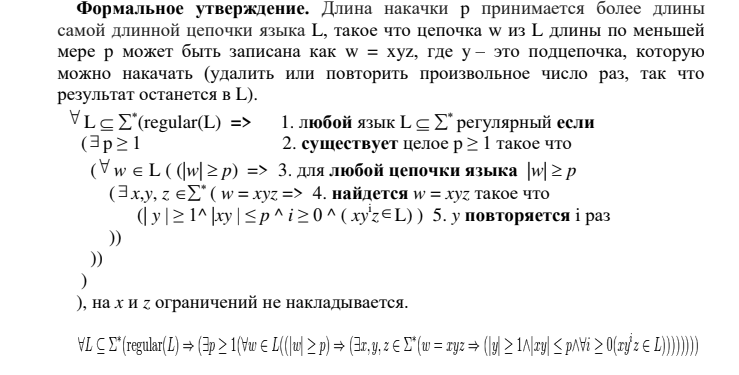


Рисунок 2 – Формулировка леммы о накачке

***Проверка:***

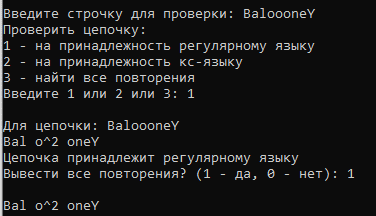


Рисунок 3 – Проверка цепочки на принадлежность регулярному языку

***Проектирование* *КА*:**

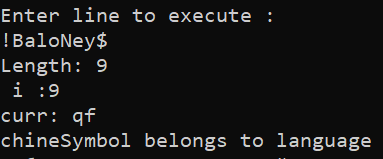


Рисунок 4 – проверка спроектированного КА

**Код*:***

string rule\_file = @"D:\Slavik\Coding\С#\SystemProg\RaGsys\RaGsys\Grammar1\rule\_file.txt";

string automate\_file=@"D:\Slavik\Coding\С#\SystemProg\RaGsys\RaGsys\Grammar1\automat\_file.txt";

FSAutomate ka;

using (var reader = new StreamReader(automate\_file)) {

string[] automate\_symbols = reader.ReadToEnd().Split(Environment.NewLine); // Получаем параметры конечного автомата

// Преобразуем каждую строку в Symbol

ka = new FSAutomate(new List<Symbol>(automate\_symbols[0].Split(", ").Select(x =>

new Symbol(x))),

new List<Symbol>(automate\_symbols[1].Split(", ").Select(x => new Symbol(x))),

new List<Symbol>(automate\_symbols[2].Split(", ").Select(x => new Symbol(x))),

automate\_symbols[3]);

}

Console.WriteLine(Environment.NewLine);

using (var reader = new StreamReader(rule\_file)) {

string line;

while ((line = reader.ReadLine()) != null) {

string[] rule = line.Split(", ");

ka.AddRule(rule[0], rule[1], rule[2]);

}

}

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

ka.Execute(Console.ReadLine());

Использованные в программе файлы представлены на рисунках 5 и 6.

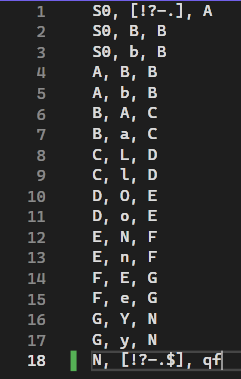


Рисунок 5 - rule\_file.txt

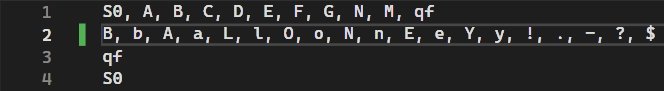


Рисунок 6 - automat\_file.txt

2. Нахождение внутри текста кусочка, похожего на локальный IP-адреса, в сети 192.168.0 (последний октет от 0 до 999).

**pattern = “192\.168\.0\.\d{1,3}”**

**Автоматная грамматика:**

***Правила регулярной грамматики:***

***Пример цепочек:***

**Конечный автомат:**

, где , ,  
,

***Примеры конфигурации КА:***

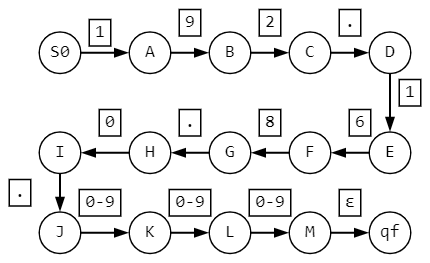


Рисунок 7 – Визуальное представление КА

**Лемма о накачке:**

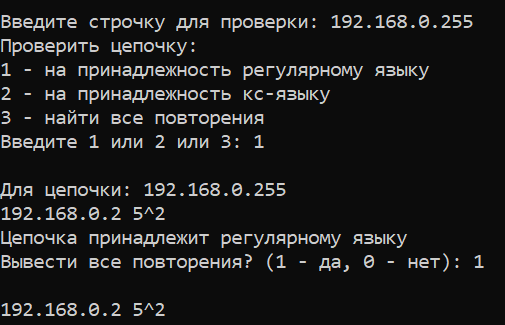


Рисунок 8 – Проверка цепочки на принадлежность регулярному языку

**Проектирование КА:**

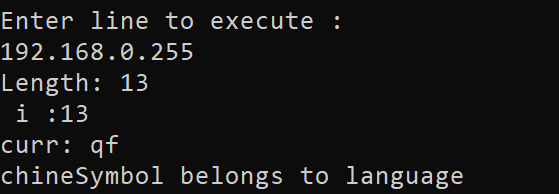


Рисунок 9 – проверка спроектированного КА

**Код:**

string rule\_file = @"D:\Slavik\Coding\С#\SystemProg\RaGsys\RaGsys\Grammar2\rule\_file.txt";

string automate\_file = @"D:\Slavik\Coding\С#\SystemProg\RaGsys\RaGsys\Grammar2\automat\_file.txt";

FSAutomate ka;

using (var reader = new StreamReader(automate\_file)) {

string[] automate\_symbols = reader.ReadToEnd().Split(Environment.NewLine); // Получаем параметры конечного автомата

// Преобразуем каждую строку в Symbol

ka = new FSAutomate(new List<Symbol>(automate\_symbols[0].Split(", ").Select(x => new Symbol(x))),

new List<Symbol>(automate\_symbols[1].Split(", ").Select(x => new Symbol(x))),

new List<Symbol>(automate\_symbols[2].Split(", ").Select(x => new Symbol(x))),

automate\_symbols[3]);

}

Console.WriteLine(Environment.NewLine);

using (var reader = new StreamReader(rule\_file)) {

string line;

while ((line = reader.ReadLine()) != null) {

string[] rule = line.Split(", ");

ka.AddRule(rule[0], rule[1], rule[2]);

}

}

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

ka.Execute(Console.ReadLine());

В программе использовались файлы, представленные на рисунках 10 и 11.

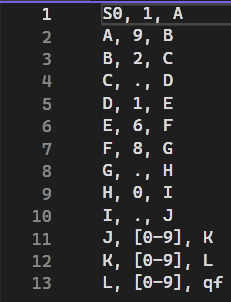


Рисунок 10 - rule\_file.txt

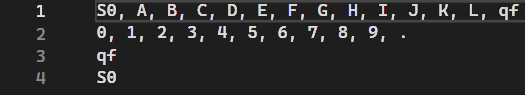


Рисунок 11 - automat\_file.txt

**Формулировка задания:**

Перевод НДКА в ДКА двух КА представленных выше.

**Код:**

static FSAutomate create\_automate(string path) {

FSAutomate ka;

using (var reader = new StreamReader(path)) {

string[] automate\_symbols = reader.ReadToEnd().Split(Environment.NewLine);

ka = new FSAutomate(new List<Symbol>(automate\_symbols[0].Split(", ").

Select(x => new Symbol(x))),

new List<Symbol>(automate\_symbols[1].Split(", ").Select(x => new Symbol(x))),

new List<Symbol>(automate\_symbols[2].Split(", ").Select(x => new Symbol(x))),

automate\_symbols[3]);

}

return ka;

}

static void read\_rules(ref FSAutomate ka, string path) {

using (var reader = new StreamReader(path)) {

string line;

while ((line = reader.ReadLine()) != null) {

string[] rule = line.Split(", ");

ka.AddRule(rule[0], rule[1], rule[2]);

}

}

}

rule\_file = @"D:\Slavik\Coding\С#\SystemProg\RaGsys\RaGsys\Grammar2\rule\_file.txt";

automate\_file = @"D:\Slavik\Coding\С#\SystemProg\RaGsys\RaGsys\Grammar2\automat\_file.txt";

var ndfsa = create\_automate(automate\_file);

read\_rules(ref ndfsa, rule\_file);

var dka = new FSAutomate();

dka.BuildDeltaDKAutomate(ndfsa, false);

dka.DebugAuto();

Console.WriteLine("Enter line to execute 192.168.0.255:");

dka.Execute(Console.ReadLine());

Код аналогичен для обеих грамматик, файлы, используемые в программе аналогичны тем, что использовались ранее.

***Пример работы программы:***

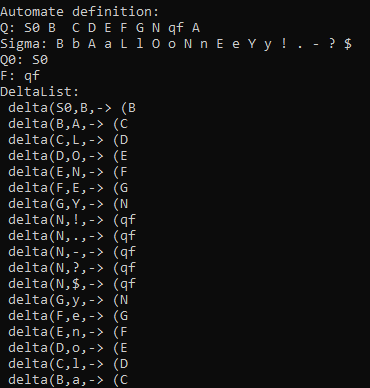
****

Рисунок 12.1 – Перевод НДКА в ДКА для первой грамматики

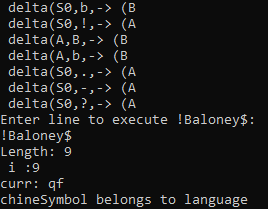
****

Рисунок 12.2 – Перевод НДКА в ДКА для первой грамматики

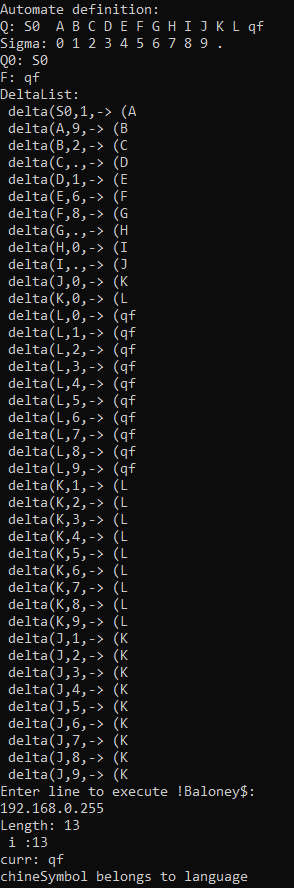
****

Рисунок 13 – Перевод НДКА в ДКА для второй грамматики

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

## *Лабораторная работа №4:*

**Формулировка задания:**

Устранить из КС-грамматики бесполезные символы и ε–правила

**Исходная грамматика (Вариант №16):**

**,** ***где*** , , .

P:

**Устранить из КС-грамматики бесполезные символы.**

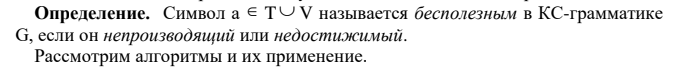


Рисунок 14 – Определение бесполезного символа в КС-грамматике

**1) Устранение непроизводящих символов.**

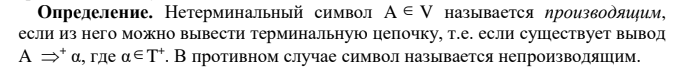


Рисунок 15 – Определение непроизводящего символа

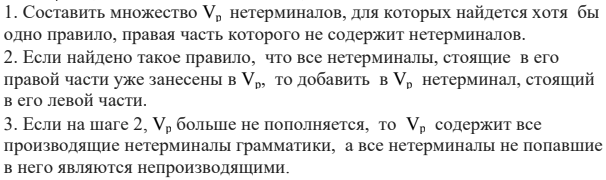
****

Рисунок 16 – Алгоритм нахождения непроизводящих символов

1. = – для этих нетерминалов нашлось 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. =

4. =

5. =

6. = содержит все производящие нетерминалы грамматики, а все нетерминалы, не попавшие в него, являются непроизводящими (V - Vp =).

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

=

**2) Устранение недостижимых символов (VTr – множество недостижимых символов):**

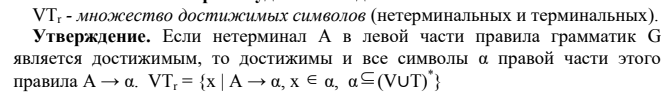
****

Рисунок 17 – Определение множества достижимых символов

Построим множество достижимых символов:

Так, не входящие в VTr символы {g, H} недостижимы.

Добавим втолько правила, состоящие из достижимых символов

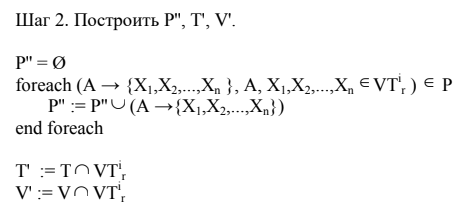
****

Рисунок 18 – алгоритм построения

Получаем

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

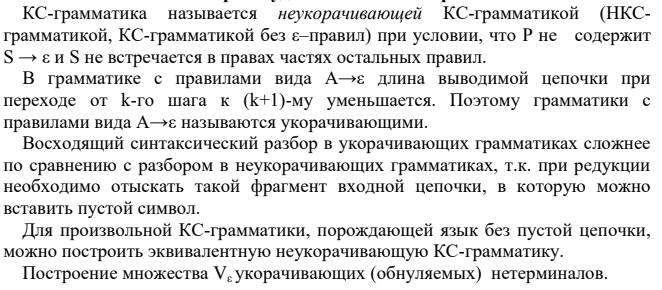
****

Рисунок 19 – Определение НКС-грамматики

*Построим множество укорачивающих нетерминалов согласно алгоритму:*



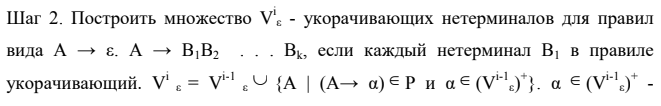




Рисунок 20 – Алгоритм построения НКС-грамматики

= (множество укорачивающих нетерминалов для правил вида A → ε)

Необходимо построить множествоукорачивающих нетерминалов для правилвида A → B1B2...Bk, если каждый нетерминал B1 в правилеукорачивающий. Однако в данном примере таких правил нет. Поэтому, множество укорачивающих терминалов: = .

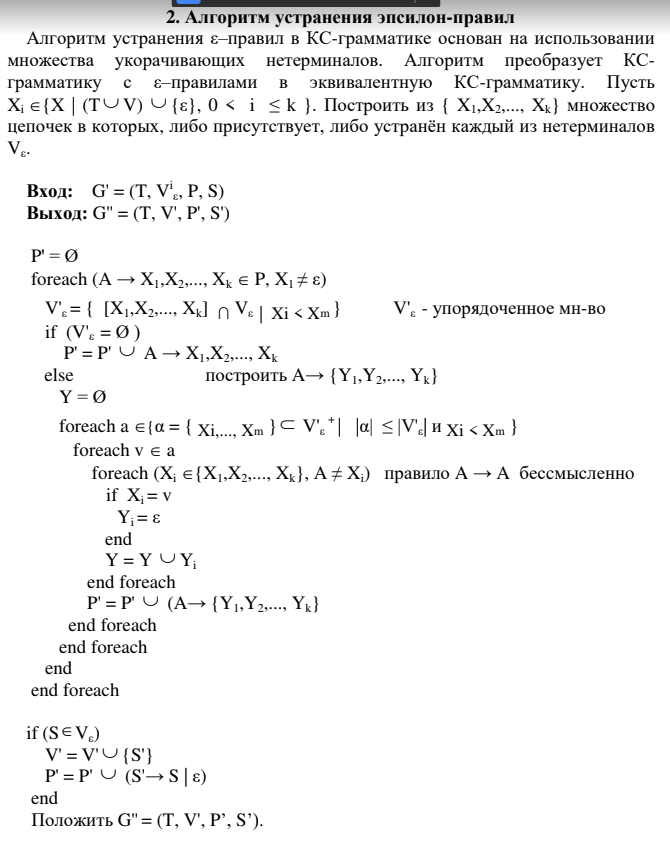


Рисунок 21 – Алгоритм удаления эпсилон правил

Составим на основе удаления эпсилон правил:

В грамматике отсутствуют бесполезные символы и -правила.

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

**1) Устранить из KС-грамматики цепные правила.**

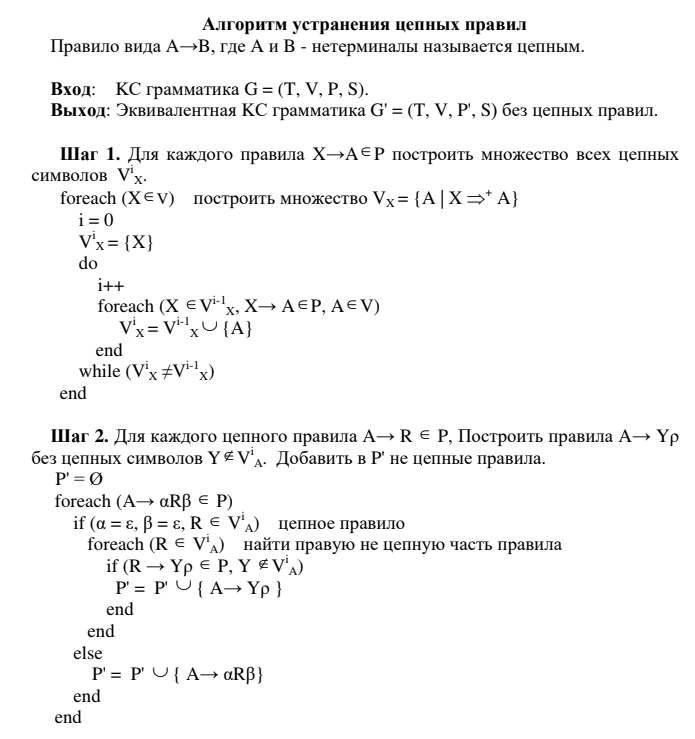


Рисунок 21 – Алгоритм устранения цепных правил

Цепные правила:

Правила , заменяем на

**2) Устранить из KС-грамматики левую рекурсию**

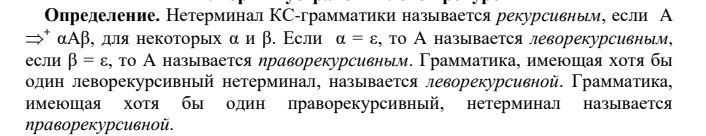


Рисунок 22 – Определение рекурсивных КС-грамматик

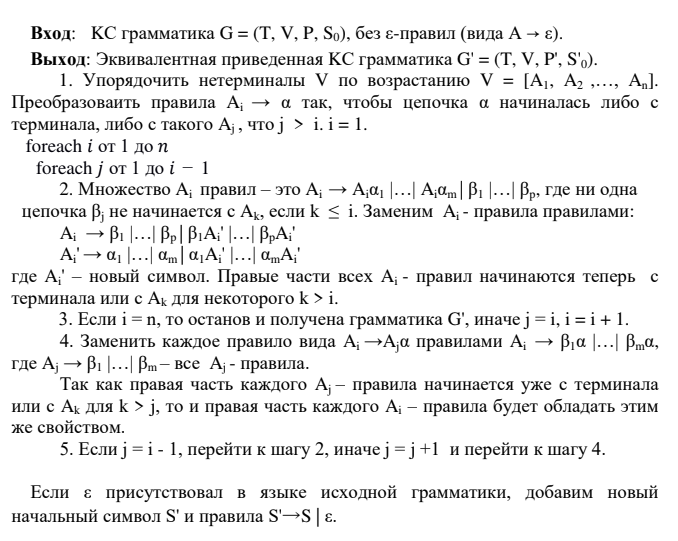


Рисунок 23 – Алгоритм Устранения непосредственной левой рекурсии

=

P =

ШАГ 1: Пусть ,  
тогда

ШАГ 2**:** Для i от 1 до 5 нет правил вида α. Для i = 6 преобразуем правило . Заменим его на правило , и добавим в грамматику правило для нового нетерминала .

ШАГ 3: i = n = 6. Алгоритм завершен.

В итоге получили новые правила грамматики:

## *Лабораторная работа №6*

**Формулировка задания:**

Определить форму КС-грамматики и сделать ее приведение

Грамматика, была получена из грамматики G применением следующих алгоритмов:

1. Устранение бесполезных символов: непроизводящих и недостижимых;

2. Преобразование в грамматику без ε–правил;

3. Устранение цепных правил;

4. Устранение левой рекурсии.

Таким образом, грамматика находится в приведённой форме.

,  
;  
;  
:

.

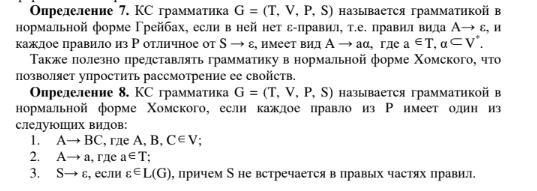


Рисунок 24 – Определение нормальных форм Хомского и Грейбах

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

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

Приведём грамматику к нормальной форме Хомского по алгоритму:

1. **Удаление длинных правил**

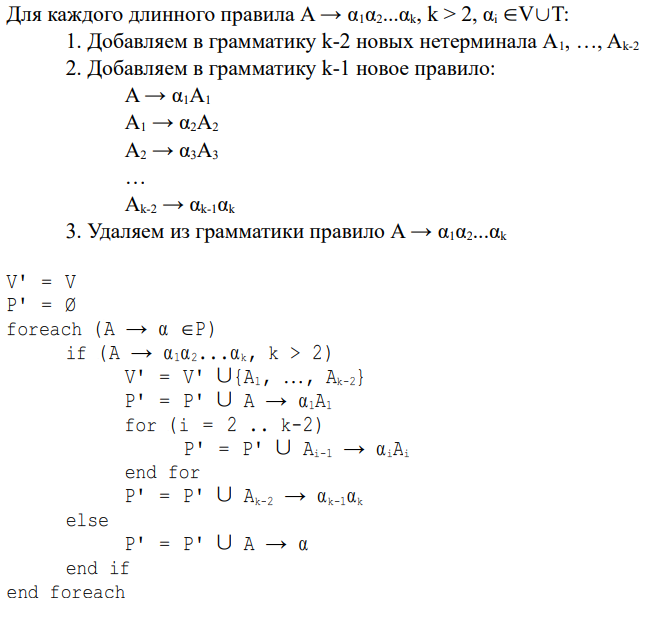


Рисунок 25 – Алгоритм удаления длинных правил

Правило:

1. Добавим новый нетерминал .

2. Добавим новые правила: и

3. Удалим из грамматики правило

Таким образом, получили новую грамматику

,  
;  
;  
:

.

1. Удаление бесполезных символов: В нет бесполезных символов
2. Удаление –правил: В нет -правил
3. Если начальный нетерминал встречается в правых частях правил, нужно убрать правила такого вида. Заведем новый стартовый символ и добавим правило , а также вместо , если оно было. В нет начального нетерминала в правых частях правил и правил вида .
4. Удаление цепных правил

В грамматике нет цепных-правил.

1. Удаление ситуаций, в которых встречаются несколько терминалов

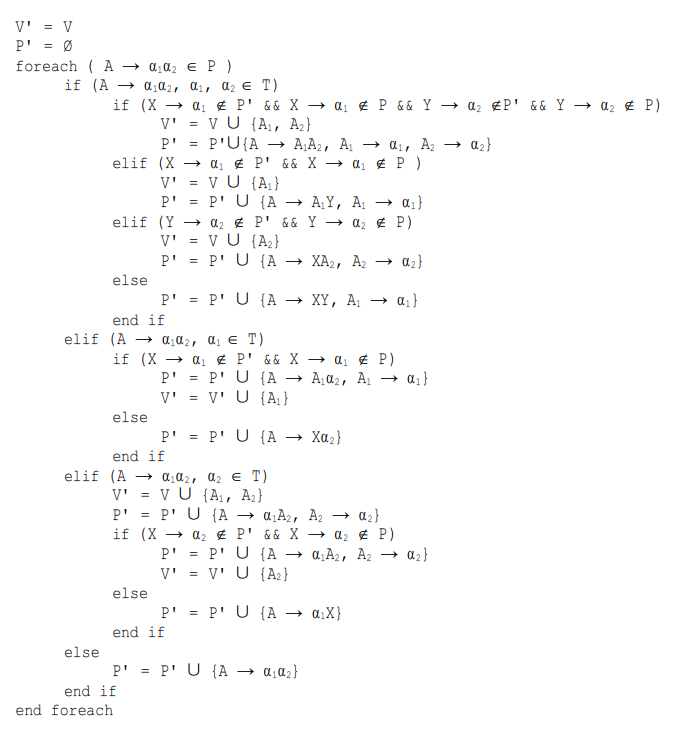


Рисунок 25 – Алгоритм удаления правил с несколькими терминалами

1. ;
2. ;
3. ;
4. ;
5. ;

Таким образом,

## *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.

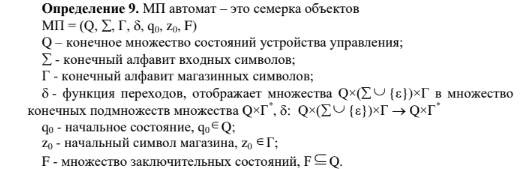


Рисунок 26 – Структура МП-автомата

Приведённая грамматика (в форме Хомского):

,  
,  
;  
:

***.***

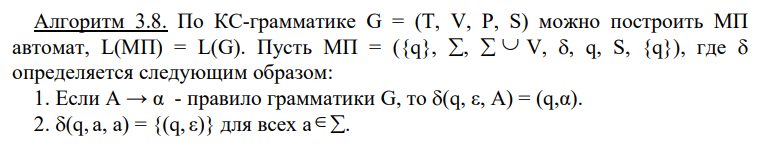


Рисунок 27 – Алгоритм построения МП-автомата

***,*** *где* , , , , *, ,*

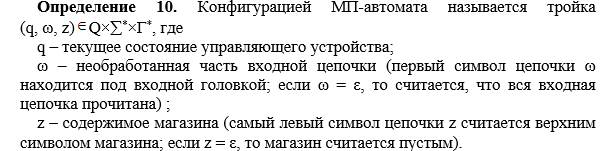


Рисунок 28 – Опредление конфигурации МП-автомата

Функция перехода **δ** определяется согласно алгоритму из рисунка 27:

Таблица 1 – Правила перехода **δ**

|  |  |
| --- | --- |
| ; | 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15: = для всех |

Последовательность тактов МП-автомата abdfcfdb:

МП-автоматы – детерминированные автоматы. МП-автомат относится к нисходящим синтаксическим анализаторам, так как строится сверху вниз.

## *Лабораторная работа №8*

**Формулировка задания:**

Реализовать спроектированный МП-автомат для приведённой КС-грамматики.

**Код:**

var CFGrammar = new Grammar(

new List<Symbol>() { "a", "b", "c", "d", "f", "p", "h", "i" },

new List<Symbol>() { "S", "S1", "S2", "A", "A1", "B", "B1", "C", "C1", "C2", "D", "F", "P", "K", "K1", "K2" },

"S");

CFGrammar.AddRule("S", new List<Symbol>() { "S2", "S1" });

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

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

CFGrammar.AddRule("A", new List<Symbol>() { "A1", "C" });

CFGrammar.AddRule("A1", new List<Symbol>() { "b" });

CFGrammar.AddRule("B", new List<Symbol>() { "B1", "A" });

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

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

CFGrammar.AddRule("C", new List<Symbol>() { "C1", "D" });

CFGrammar.AddRule("C1", new List<Symbol>() { "d" });

CFGrammar.AddRule("C", new List<Symbol>() { "C2", "K" });

CFGrammar.AddRule("C2", new List<Symbol>() { "i" });

CFGrammar.AddRule("K", new List<Symbol>() { "K2", "K1" });

CFGrammar.AddRule("K2", new List<Symbol>() { "h" });

CFGrammar.AddRule("K1", new List<Symbol>() { "K2", "K1" });

CFGrammar.AddRule("K1", new List<Symbol>() { "h" });

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());

***Пример работы программы:***

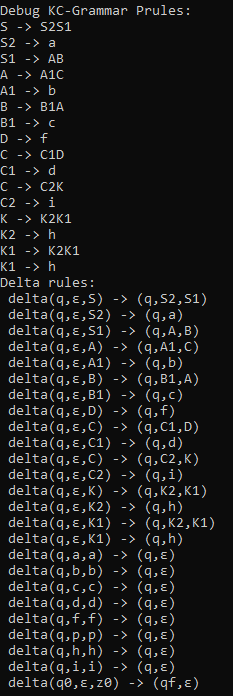
****

Рисунок 29.1 – Правила МП-автомата

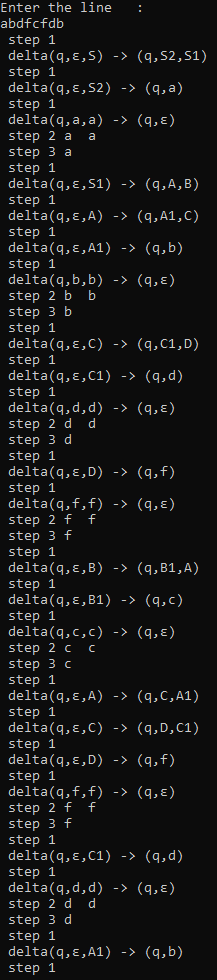
****

Рисунок 29.2 – Правила МП-автомата

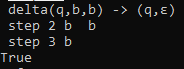
****

Рисунок 29.3 – Правила МП-автомата

**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.

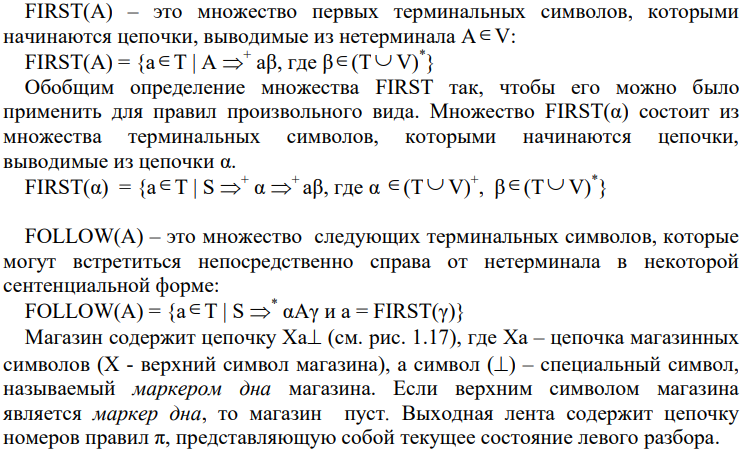
****

Рисунок 30 – Определение множеств FOLLOW и FIRST

Исходная грамматика:

,  
;  
;  
:

.

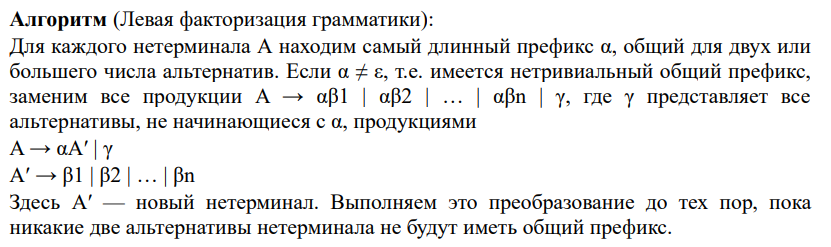


Рисунок 31 – Алгоритм левой факторизации

Для введём новый нетерминал и добавим правила ,

Таким образом, получили LL(1)-грамматику:

,  
;  
;  
:

.

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

*Вход:* LL(1) - грамматика

*Выход:* Управляющая таблица M для грамматики G.

Таблица M определяется на множестве по правилам:

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

Таблица 2 – Управляющая таблица M

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **a** | **b** | **c** | **d** | **f** | **p** | **h** | **i** | **ε** |
| **S** |  |  |  |  |  |  |  |  |  |
| **A** |  |  |  |  |  |  |  |  |  |
| **B** |  |  | ,3 |  |  |  |  |  |  |
| **C** |  |  |  |  |  |  |  |  |  |
| **D** |  |  |  |  |  |  |  |  |  |
| **F** |  |  |  |  |  |  |  |  |  |
| **P** |  |  |  |  |  |  |  |  |  |
| **K** |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **a** | ВЫБРОС |  |  |  |  |  |  |  |  |
| **b** |  | ВЫБРОС |  |  |  |  |  |  |  |
| **c** |  |  | ВЫБРОС |  |  |  |  |  |  |
| **d** |  |  |  | ВЫБРОС |  |  |  |  |  |
| **f** |  |  |  |  | ВЫБРОС |  |  |  |  |
| **p** |  |  |  |  |  | ВЫБРОС |  |  |  |
| **h** |  |  |  |  |  |  | ВЫБРОС |  |  |
| **i** |  |  |  |  |  |  |  | ВЫБРОС |  |
| **Ʇ** |  |  |  |  |  |  |  |  | ДОПУСК |

Пустые клетки в таблице означают ОШИБКУ.

Построим также аналитическое представление управляющей таблицы

Таблица 3 – Аналитическое представление таблицы M

|  |  |  |
| --- | --- | --- |
| Правило грамматики | Множество | Значение М |
| **p1**: | FIRST(S) = {a} | M(S, a) = , 1 |
| **p2**: | FIRST(A) = {b} | M(A, b) = , 2 |
| **p3**: | FIRST(B) = {c} | M(B, c) = , 3 |
| **p4**: | FIRST(D) = {f} | M(D, f) = , 4 |
| **p5**: | FIRST(C) = {d} | M(C, d) = , 5 |
| **p6**: | FIRST(C) = {i} | M(C, i) = , 6 |
| **p7**: | FIRST(K) = {h} | M(K, h) = , 7 |
| **p8**: | FIRST() = {h} | M(, h) = , 8 |
| **p9**: | FOLLOW() = { } | M(, ) = , 9 |
| **p10**: | FIRST() = {h} | M(, h) = , 10 |

## *Лабораторная работа №10:*

**Формулировка задания:**

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

Рассмотрим работу алгоритма для цепочки (abdfcdfb)

Алгоритм находится в начальной конфигурации ((abdfcdfb), , ε), где S0 = K

Значение управляющей таблицы M(K, f) = (A, 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.

Таблица 4 – Правила вывода для анализатора

|  |  |
| --- | --- |
| Текущая конфигурация | Значение М |
| (abdfcbdf, S⊥, ε) ⸠ | M(S, a) = , 1 |
| (abdfcbdf, ⊥, 1) ⸠ | M(a, a) = ВЫБРОС |
| (bdfcbdf, ⊥, 1) ⸠ | M(, b) = , 2 |
| (bdfcbdf, ⊥, 12) ⸠ | M(b, b) = ВЫБРОС |
| (dfcbdf, ⊥, 125) ⸠ | M(C, d) = , 5 |
| (dfcbdf, ⊥, 125) ⸠ | M(d, d) = ВЫБРОС |
| (fcbdf, ⊥, 1254) ⸠ | M(D, f) = f,4 |
| (fcbdf, ⊥, 1254) ⸠ | M(f, f) = ВЫБРОС |
| (cbdf, ⊥, 1254) ⸠ | M(B, c) = , 3 |
| (cbdf, ⊥, 12543) ⸠ | M(c, c) = ВЫБРОС |
| (bdf, ⊥, 12543) ⸠ | M(A, b) = 2 |
| (bdf, ⊥, 125432) ⸠ | M(b, b) = ВЫБРОС |
| (df, ⊥, 125432) ⸠ | M(C, d) = ,5 |
| (df, ⊥, 1254325) ⸠ | M(d, d) = ВЫБРОС |
| (f, ⊥, 1254325) ⸠ | M(D, f) = f,4 |
| (f, ⊥, 12543254) ⸠ | M(f, f) = ВЫБРОС |
| (ε, ⊥, 12543254) ⸠ | M(ε, ⊥) = ДОПУСК |

Так как = ДОПУСК, то цепочка abdfcbdf принадлежит языку и последовательность номеров правил 12543254 на выходной ленте – это её разбор.

## *Лабораторная работа №11:*

**Формулировка задания:**

Реализовать управляющую таблицу M для LL(k) анализатора.

**Код:**

var LL1 = new Grammar(

new List<Symbol>() { "a", "b", "c", "d", "f", "p", "h", "i", "" },

new List<Symbol>() { "S", "A", "B", "C", "D", "F", "P", "K", "K1", "K2" },

"S");

LL1.AddRule("S", new List<Symbol>() { "a", "A", "B" });

LL1.AddRule("A", new List<Symbol>() { "b", "C" });

LL1.AddRule("B", new List<Symbol>() { "c", "A" });

LL1.AddRule("D", new List<Symbol>() { "f" });

LL1.AddRule("C", new List<Symbol>() { "d", "D" });

LL1.AddRule("C", new List<Symbol>() { "i", "K" });

LL1.AddRule("K", new List<Symbol>() { "h", "K1" });

LL1.AddRule("K1", new List<Symbol>() { "h", "K2" });

LL1.AddRule("K2", new List<Symbol>() { "" });

LL1.AddRule("K2", new List<Symbol>() { "K1" });

var parser1 = new LLParser(LL1);

Console.WriteLine("Введите строку: ");

string stringChain = Console.ReadLine();

var chain = new List<Symbol> { };

foreach (var x in stringChain)

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

if (parser1.Parse1(chain)) {

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

Console.WriteLine(parser1.OutputConfigure);

} else {

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

}

***Результат работы программы:***

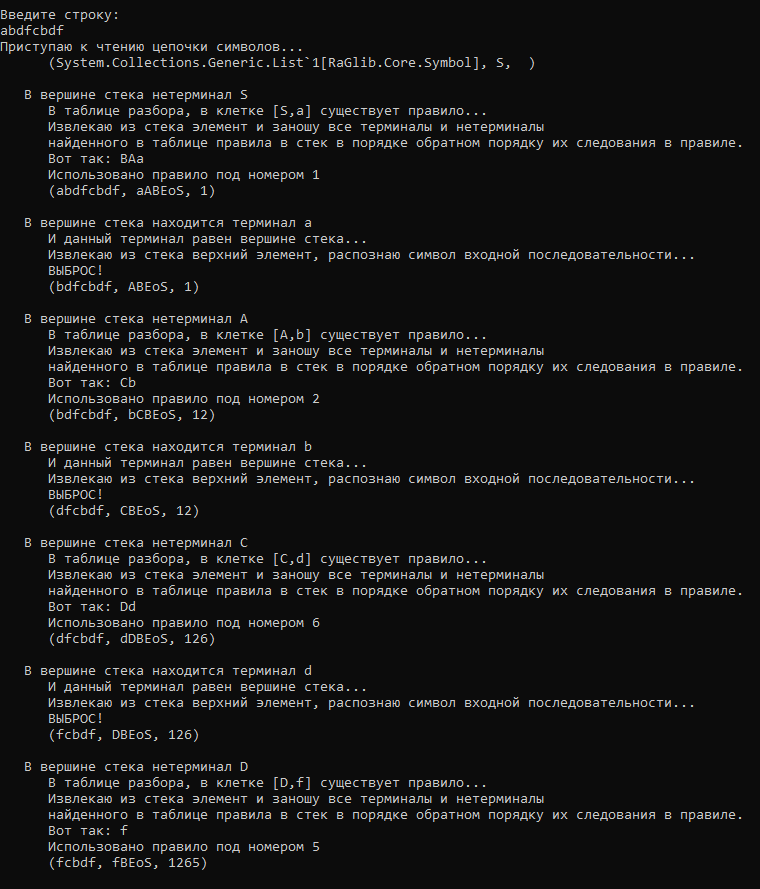


Рисунок 32.1 – Работа анализатора

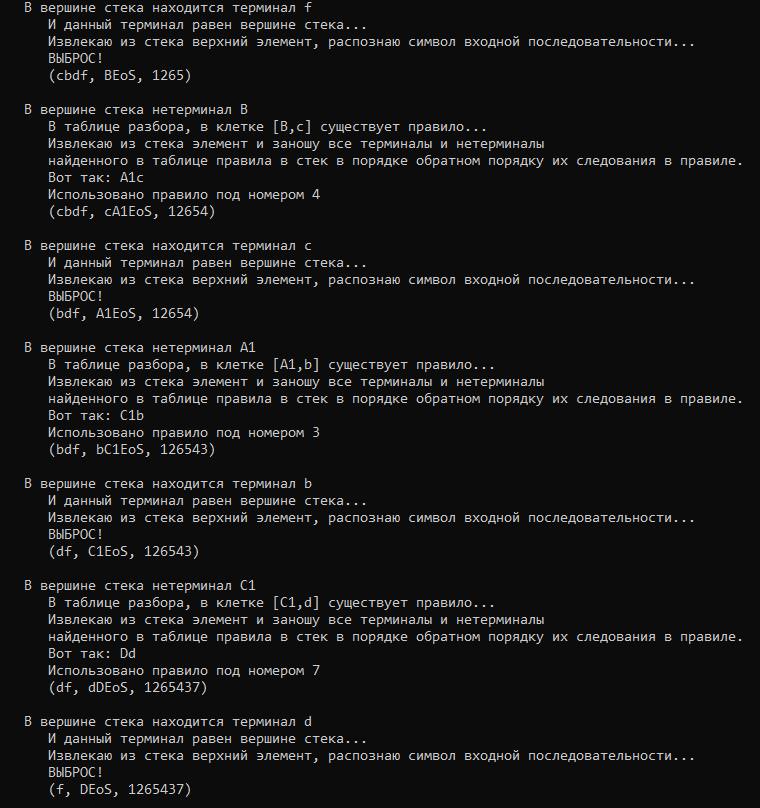


Рисунок 32.2 – Работа анализатора

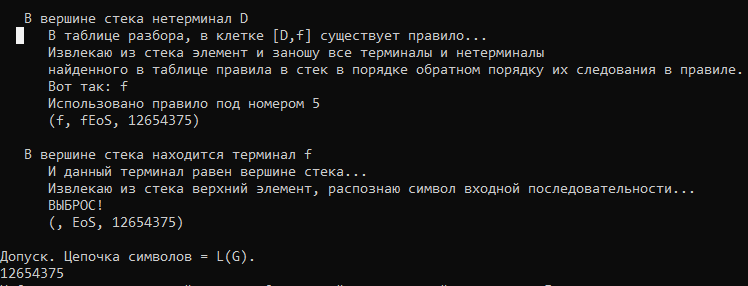


Рисунок 32.3 – Работа анализатора

Согласно результату программы, управляющая таблица M была построена верно. А цепочка abdfcbdf принадлежит языку. Изменение последовательности номеров связано с необъяснимым изменением поведения правил в случае их повторения, поэтому для корректной работы в программе были введены дополнительные состояния.

**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)-*грамматиками* (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).

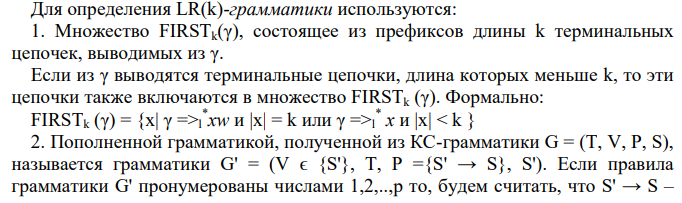




Рисунок 33 – Определения связанные с LR(k) -грамматикой

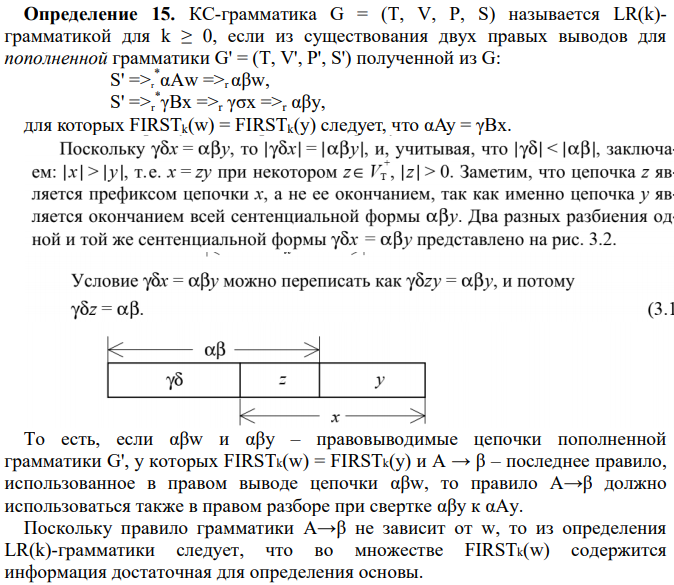


Рисунок 34 – Определение LR(k) -грамматики

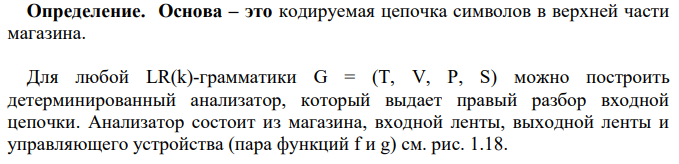


Рисунок 35 – Определение основы

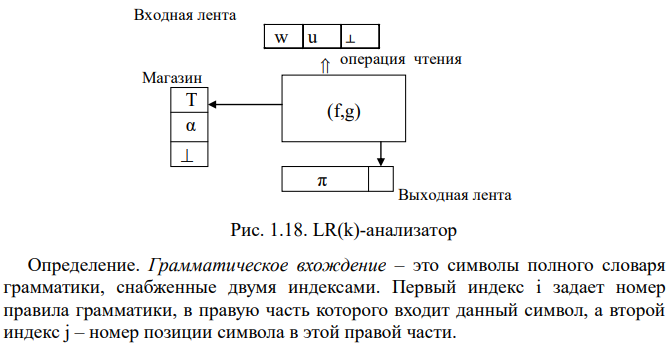


Рисунок 36 – Определение грамматического вхождения

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.

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

,  
;  
;  
:

1. **Построение LR(k) анализатора на основе активных префиксов и отношения OBLOW:**

Непосредственно из правил вывода грамматики получим:

OBLOW , OBLOW , OBLOW , OBLOW , OBLOW ,  
 OBLOW , OBLOW , OBLOW

Из определения отношения OBLOW следует, что . Из S можно вывести цепочку .

Следовательно, OFIRST(S0 = S) = {a1, S0}, и ,

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

для всех

Из можно вывести цепочки:

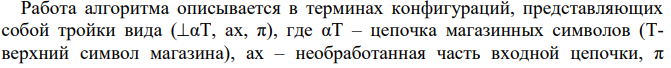
=>2 =>5 =>4

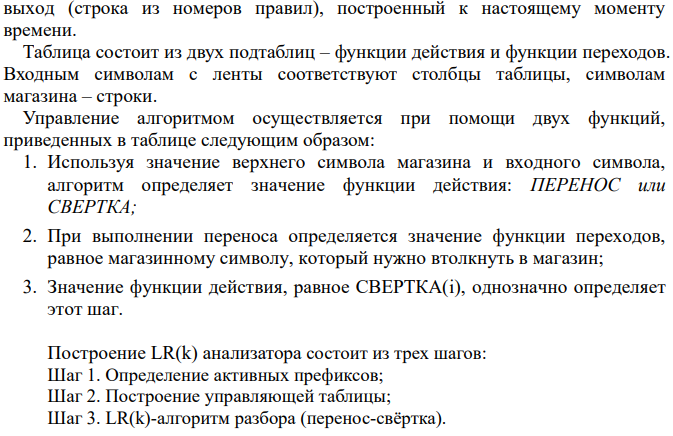
Следовательно, OFIRST(A­1) = {} и

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

Таблица 5 – Матрица отношения OBLOW

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OBLOW** | S0 = S |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **^** | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S0 = S |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 1 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | 1 |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | 1 |  |  |  | 1 |  | 1 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 1 |  |  | 1 |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | 1 |  | 1 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 |  | 1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ⊥ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |





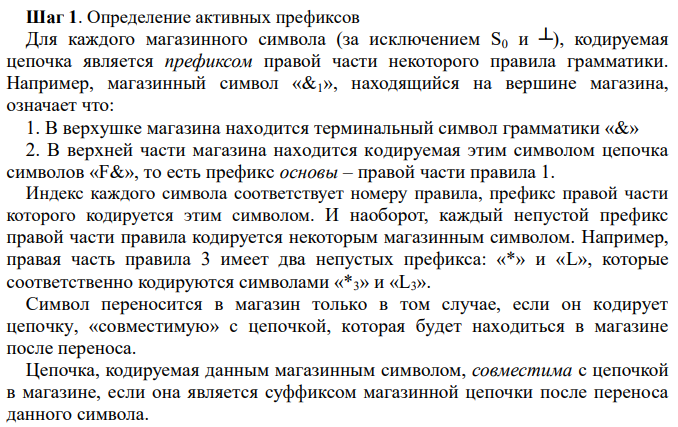


Рисунок 37 – Алгоритм построения LR(k) анализатора

Таблица 6 – Закодированные символы

|  |  |  |  |
| --- | --- | --- | --- |
| Символ грамматики | Магазинный символ | Кодируемая цепочка | Операции |
| S0 = S | S0 | ⊥S | Д |
| A | A­1 | aA | П |
| B | B1 |  | С1 |
| C | C2 | bC | П |
| D | D5 | dD | С5 |
| K | K6 | iK | С6 |
|  |  |  | С7  С8 |
| a | a1 | a | П |
| b | b2 | b | П |
| c | c3 | c | П |
| d | d5 | d | П |
| f | f4 | f | С4 |
| i | i6 | i | П |
| h | h789 | h | П  П  С9 |

Таблица 7 – Функция переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | функция переходов g(X) | | | | | | | | | | | | |
| a | b | c | d | f | i | h | A | B | C | D | K |  |
| S |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | , |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рассмотрим работу алгоритма на примере цепочки (abdfcbdf):

(⊥, acgdfcggb⊥, ɛ) ⸠ (⊥a1, bdfcbdf⊥, ɛ) ⸠ (⊥a1b2, dfcbdf⊥, ɛ) ⸠ (⊥a1b2d5, fcbdf⊥, ɛ) ⸠ (⊥a1b2d5f4, cbdf⊥, ɛ) ⸠ (⊥a1b2d5D5, cbdf⊥, 4) ⸠ (⊥a1b2C2, cbdf⊥, 45) ⸠   
(⊥a1A1, cbdf⊥, 452) ⸠ (⊥a1A1c3, bdf⊥, 452) ⸠ (⊥a1A1c3b2, df⊥, 452) ⸠   
(⊥a1A1c3b2d5, f⊥, 452) ⸠ (⊥a1A1c3b2d5f4, ɛ⊥, 452) ⸠ (⊥a1A1c3b2d5D5, ɛ⊥, 4524) ⸠

(⊥a1A1c3b2C2, ⊥, 45245) ⸠ (⊥a1A1c3A3, ⊥, 452452) ⸠ (⊥a1A1B1, ⊥, 4524523) ⸠

⸠ (⊥S0, ⊥, 45245231) ⸠ ДОПУСК.

Управляющая таблица функция действий f(x):

