Московский авиационный институт
(государственный технический университет)

Факультет «Прикладная математика и физика»

Лабораторные работы
по курсу
«Системное программное обеспечение»
IV семестр

1. Спроектировать грамматику по заданному языку
2. Спроектировать 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 КС-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 LR(k) анализатора построить управляющую таблицу M
10. Аналитически написать правила вывода для цепочки LR(k) анализатора.
11. Реализовать управляющую таблицу M Для LR(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: Шевчук П.В,
Группа: 08-204

Руководитель: Семёнов А. С.

Оценка:
Дата:

Москва, 2018

Московский авиационный институт

(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторные работы 1 — 3 по курсу СП:

Спроектировать автоматную грамматику по заданному языку L, построить конечный автомат.

1. Изучить классификацию Хомского (см. раздел 1.1., 1.2., 1.3.). Ответьте на вопрос, какие грамматики называются автоматными. Какие есть виды автоматных грамматик.

2. Спроектировать по заданному языку L автоматную грамматику и конечный автомат. Используйте пример, и последовательность выполнения работы из раздела 2.3. “Практическая работа 1”.

1. Постановка задачи.

2. Входные и выходные данные.

3.Спроектировать грамматику (Лаб 1).

4.Определить свойства грамматики.

5.Спроектировать конечный автомат , составить диаграмму переходов КА и реализовать на C# (Лаб 2.).

5.1. Создать проект – консольное приложение:

5.2. Исапользуйте следующие франменты программ для реализации КА, распознающего заданный язык.

Работу выполнил:

08-204 Шевчук П.В. _____ _____

Группа **ФИО:** **Подпись** **Вариант 13**

Руководитель: _____/Семенов А.С./

Подпись:

Дата: __ март 2018

Вариант №15

13. $L=\{0(01)^{*}+1\omega_1 0 \mid \omega_1\{0,1\}^{+}\}$

Пояснение данного языка

$\omega_1\{1,0\}^{+}$ - обозначает множество любой длины, состоящее и/из нулей и/или единиц, включая пустое множество.

Теория

Алфавит – конечное непустое множество символов.

Цепочка – конечная последовательность символов некоторого алфавита.

Язык – множество цепочек в одном и том же алфавите.

Формальная грамматика – способ описания формального языка, т.е. выделение некоторого подмножества из множества всех слов некоторого конечного алфавита.

Более формально цепочка символов в алфавите V определяется следующим образом:
(1) ε - цепочка в алфавите V;
(2) если α - цепочка в алфавите V и a - символ этого алфавита, то αa - цепочка в алфавите V;
(3) β - цепочка в алфавите V тогда и только тогда, когда она является таковой в силу (1) и (2).

если α и β - цепочки, то цепочка αβ называется *конкатенацией (*или *сцеплением)* цепочек α и β.
Например, если α = ab и β = cd, то αβ = abcd.
Для любой цепочки α всегда αε = εα = α.

Длина цепочки - это число составляющих ее символов.

Определение: обозначим через V* множество, содержащее все цепочки конечной длины в алфавите V, включая пустую цепочку ε.
Например, если V={0,1}, то V* = {ε, 0, 1, 00, 11, 01, 10, 000, 001, 011, ...}.

Определение: обозначим через V+ множество, содержащее все цепочки конечной длины в алфавите V, исключая пустую цепочку ε.
Следовательно, V* = V+ ∪ {ε}.
Ясно, что каждый язык в алфавите V является подмножеством множества V*.

VT - алфавит *терминальных символов (терминалов)*,
VN - алфавит *нетерминальных символов (нетерминалов)*, не пересекающийся с VT,
P - конечное подмножество множества (VT ∪ VN)+ × (VT ∪ VN)*; элемент (α, β) множества P называется *правилом вывода* и записывается в виде α → β,

S - *начальный символ (цель)* грамматики, S ∈ VN.
Для записи правил вывода с одинаковыми левыми частями
α→β1 α→β2 ... α→βn
будем пользоваться сокращенной записью
α→β1 | β2 |...| βn.
Каждое βi , i = 1, 2, ... ,n , будем называть *альтернативой* правила вывода из цепочки α.

Терминал (терминальный символ) — объект, непосредственно присутствующий в словах языка, соответствующего грамматике, и имеющий конкретное, неизменяемое значение (обобщение понятия «буквы»). В формальных языках, используемых на компьютере, в качестве терминалов обычно берут все или часть стандартных символов ASCII — латинские буквы, цифры и спец. символы.

Нетерминал (нетерминальный символ) — объект, обозначающий какую-либо сущность языка (например: формула, арифметическое выражение, команда) и не имеющий конкретного символьного значения.

Типы грамматик

По иерархии Хомского, грамматики делятся на 4 типа, каждый последующий является более ограниченным подмножеством предыдущего (но и легче поддающимся анализу):

тип 0. ***неограниченные грамматики*** — возможны любые правила

тип 1. ***контекстно-зависимые грамматики*** — левая часть может содержать один нетерминал, окруженный «контекстом» (последовательности символов, в том же виде присутствующие в правой части); сам нетерминал заменяется непустой последовательностью символов в правой части.

тип 2. ***контекстно-свободные грамматики*** — левая часть состоит из одного нетерминала.

тип 3***. регулярные грамматики*** — более простые, эквивалентны конечным автоматам.

Праволинейная грамматика — в теории конечных автоматов — специальный случай регулярной грамматики.

Грамматика называется ***праволинейной***, если она содержит только правила вида A→a, A→aB.

Грамматика G = (VT, VN, P, S) называется *леволинейной*, если каждое правило из P имеет вид A → Bt либо A → t, где A ∈ VN, B ∈ VN, t ∈ VT.

Конечный автомат — абстрактный автомат без выходного потока, число возможных состояний которого конечно. Результат работы автомата определяется по его конечному состоянию.

Существуют различные варианты задания конечного автомата. Например, конечный автомат может быть задан с помощью пяти параметров: $M = (Q, \Sigma, \delta, q_0, F)$

- Q — конечное множество состояний автомата;
- q0 — начальное состояние автомата ($q_0 \in Q$);
- F — множество заключительных (или допускающих) состояний, таких что $F \subseteq Q$
- Σ — допустимый входной алфавит (конечное множество допустимых входных символов), из которого формируются строки, считываемые автоматом;
- δ — заданное отображение множества $Q \times \Sigma$ во множество $\mathcal{P}(Q)$ подмножеств Q:

$$\delta : Q \times \Sigma \to \mathcal{P}(Q)$$

Автомат начинает работу в состоянии q0, считывая по одному символу входной строки. Считанный символ переводит автомат в новое состояние из Q в соответствии с функцией переходов. Если по завершении считывания входного слова (цепочки символов) автомат оказывается в одном из допускающих состояний, то слово «принимается» автоматом. В этом случае говорят, что оно принадлежит языку данного автомата. В противном случае слово «отвергается».

Конечные автоматы подразделяются на детерминированные и недетерминированные

Детерминированным конечным автоматом (ДКА) называется такой автомат, в котором для каждой последовательности входных символов существует лишь одно состояние, в которое автомат может перейти из текущего.

Недетерминированный конечный автомат (НДКА) является обобщением детерминированного. Недетерминированность автоматов достигается двумя способами:

1) Существуют переходы, помеченные пустой цепочкой ε
2) Из одного состояния выходит несколько переходов, помеченных одним и тем же символом

T={0,1}

$S_0 = 1$

V: 1) S_0 -> 1A01B

2) A -> 01A|E

3) B -> 0|1|10|01|E

P_1: 1A01B -> 101A01B -> 10101B -> 101010
P_2: 1A01B -> 101B -> 10101
P_3: 1A01B -> 101B -> 101

V(правильное):

S_0 -> 1A

A -> 0B

B -> 1A

A -> 0C

C -> 1D

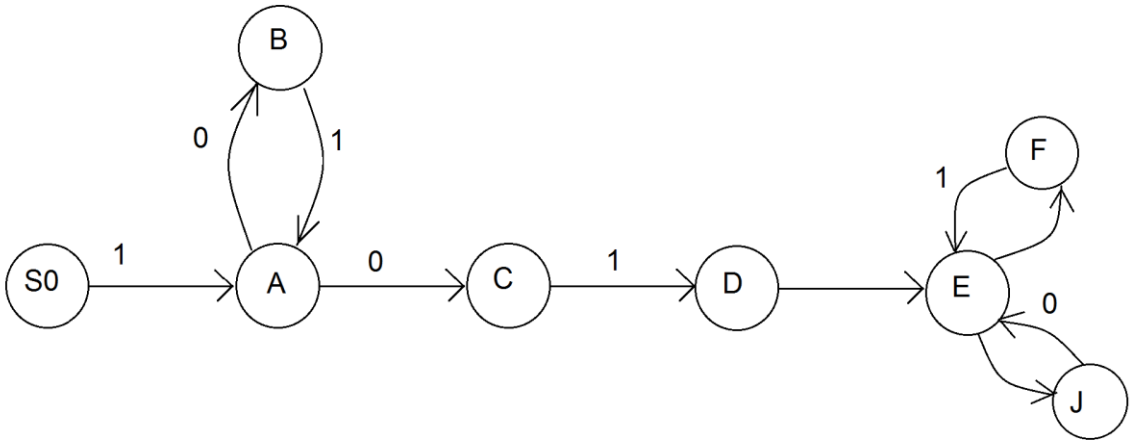
D -> E

E -> 0J

J -> E

E -> 1F

F -> E



1-ый вариант

Используя **CASE**

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;



namespace ConsoleApplication2

{
```

```
class Program
{
    static void Main(string[] args)
    {
        string str = "";
       byte q = 0; // СОСТОЯНИЕ Q
        int g = 1; // ПЕРЕМЕННАЯ КЛЮЧ для цикла
        int k = 1; //общий ключ для завершения
        Console.WriteLine("Enter string to check");
        str = Console.ReadLine();
        foreach (char c in str)
        {
            switch (q)
            {
                case 0:
                    if (c == '1')
                    {
                        q++;
                    }
                    else { q = -1; }
                    continue;

                case 1:
                    if (c == '(')
                    {
                        q++;
                    }
                    else { q = -1; }

                    continue;

                case 2:
```

```
                if (c == '0')
                {
                    g = 99; // Присваиваем 99 если встетили 0(далее проверяем клюи и 1 дальше или нет)
                    q++;
                }
                else if (c == ')') { q = q + 2; } // Если встретили вторую скобку прыгаем через состояние
                else { q = -1; }
                continue;

            case 3:
                if (c == '1' && g == 99)
                {
                    q--; //Если 1 возвращаеся назад тк может быть либо '01' либо ')'
                    continue;
                }
                else { q = -1; }
                continue;
            case 4:
                if (c == '0')
                {
                    q++;
                }
                else { q = -1; }
                continue;
            case 5:
                if (c == '1')
                {
                    q++;
                }
                else { q = -1; }
                continue;
```

```
                case 6:


                    if ((c == '1') || (c == '0'))
                    {
                    }
                    else { q = -1; }
                    continue;
                default:
                    Console.WriteLine("NO");
                    break;
            }
            break;
        } //end foreach
        if (k == 1 && q >= 3) { Console.WriteLine("YES"); }
        Console.ReadLine();


    }
  }
}
```

2ой ВАРИАНТ

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;

namespace lab1
{
  class MainProgram
  {
```

```
public static void Main(string[] args)
{
   string str;
   string answer;
   Console.WriteLine("Здравствуйте!");
   Console.WriteLine("Хотите ввести строку и проверить, относится ли она к нашему языку?");
   Console.WriteLine("y - да, n - нет(NFA->DFA)");
   answer = Console.ReadLine();

   Automate automate = new Automate();                                    //создаём объект pro типа Atomate
   while (answer == "y")
   {
      Console.WriteLine("Введите строку:");
      str = Console.ReadLine();
                                                                          //записываем введённую строку в переменную str
    automate.parse(str);
                                                                          //запускаем процедуру program с входным параметром str
      Console.WriteLine("Хотите проверить очередную строку?");
      Console.WriteLine("y - да, n - нет");
      answer = Console.ReadLine();
   }//while

   if (answer == "n")
   {
      Automate NDKA = new Automate("NDKA");
      //NDKA.Debug();
      Conv conv = new Conv();
      Automate DKA = conv.convert(NDKA);
      //DKA.Debug();
      //DKA.recognize(DKA.q0, "01010", 0);
      Console.ReadLine();
      Console.WriteLine("Вы решили покинуть программу");
   }
```

```
    }//end Main
}//end class MainProgram




class Automate
{
    ArrayList Q = null;        // множество всех состояний
    ArrayList Sigma = null;       // конечный алфавит входных символов

    string Q0 = null;        // начальное состояние
    ArrayList F = null;       // заключительные состояния
    // атрибутное программирование на C#
    public ArrayList q { get { return Q; } set { Q = value; } }
    public ArrayList sigma { get { return Sigma; } set { Sigma = value; } }
    public ArrayList deltaList
    {
        get { return DeltaList; }
        set { DeltaList = value; }
    }
    public string q0 { get { return Q0; } set { Q0 = value; } }
    public ArrayList f { get { return F; } set { F = value; } }
    ArrayList DeltaList = new ArrayList();   // множество всех правил
    public Automate()
    {
        Q = new ArrayList() { "S0", "A", "B", "C", "D", "E", "F", "G", "S" };  // "C"  для отладки
        Sigma = new ArrayList() { "0", "1" };
        q0 = "S0";
        F = new ArrayList() { "S" };
        CreateDeltaList();
                                                                    //конструктор класса запускает процедуру добавления Delta-правил
    }//конструктор класса Automate
```

```
    public Automate(string aname)
    {
        System.Console.WriteLine(aname);
        Q = new ArrayList() { "S0", "A", "B", "C", "D", "E", "F", "G", "S" };  // "C"  для отладки
        Sigma = new ArrayList() { "0", "1" };
        q0 = "S0";
        F = new ArrayList() { "S" };

        CreateDeltaList();                                                    //конструктор класса запускает процедуру
добавления Delta-правил
    }//конструктор класса Automate

    public void CreateDeltaList()                                             //процедура добавления Delta-правил
    {
        Delta delta = null;
        delta = new Delta("S0", "1", new ArrayList() { "A" });
        DeltaList.Add(delta);
        delta = new Delta("A", "(", new ArrayList() { "B" });
        DeltaList.Add(delta);
        delta = new Delta("B", "0", new ArrayList() { "C" });
        DeltaList.Add(delta);
        delta = new Delta("C", "1", new ArrayList() { "D" });
        DeltaList.Add(delta);
        delta = new Delta("D", "0", new ArrayList() { "C" });
        DeltaList.Add(delta);
        delta = new Delta("D", ")", new ArrayList() { "E" });
        DeltaList.Add(delta);
        delta = new Delta("E", "0", new ArrayList() { "F" });
        DeltaList.Add(delta);
        delta = new Delta("F", "1", new ArrayList() { "G" });
```

```
        DeltaList.Add(delta);
        delta = new Delta("G", "0", new ArrayList() { "G" });
        DeltaList.Add(delta);
        delta = new Delta("G", "1", new ArrayList() { "G" });
        DeltaList.Add(delta);
        delta = new Delta("G", ".", new ArrayList() { "S" }); // ТОЧКА В КОНЦЕ
        DeltaList.Add(delta);

    }// end CreateDeltaList

    public void parse(string s)
    {
        int n = s.Length;
                                                                    //узнаём длину строки для
посимвольного считывания
        string State = "S0";
                                                                //начальное состояние автомата
        int count = 0;
                                                                    //переменная для подсчёта
общего кол-ва правил
        int count1 = 0;
                                                                    //переменная для подсчёта
просмотренных правил

        foreach (Delta d in this.DeltaList)
            count++;
                                                                    //подсчитываем общее количество
правил

        for (int i = 0; i < n; i++)
        {
            foreach (Delta d in this.DeltaList)
                                                        //поочерёдно просматриваем каждое правило
            {
                if (d.leftNoTerm.Equals(State) && d.leftTerm.Equals(s.Substring(i, 1)))
             //просматриваем все правила и сравниваем с текущим состоянием, при этом сравниваем
                //текущий символ с символом правила
```

```
            {
               State = d.right[0].ToString();
                                                    //если условия соблюдены, то присваиваем новое состояние,и выходим
               break;
            }// проверка совпадения состояния и символа

            count1++;
            if (count1 == count) break;
                                                              //если ни одно из состояний не подошло, выходим для
дальнейшего исследования
         }// просмотр всех Delta-правил

         if (count1 == count)
            break;
         else
            count1 = 0;

         if (i == n - 1 && State == "S")
            Console.WriteLine("Данная строка принадлежит нашему языку!");
                  //если состояние - конечное, и перебраны все символы
      }// посимвольное считывание строки

      if (State != "S" || count1 == count)
         Console.WriteLine("Данная строка не принадлежит нашему языку!");
                  //если состояние - не конечное или ни одно из правил не подошло
   }// end program


   public class Delta
   {  // структура Delta правил переписывания

      private string LeftNoTerm = null;
      private string LeftTerm = null;
      private ArrayList Right = null;
```

```
        public string leftNoTerm { get { return LeftNoTerm; } set { LeftNoTerm = value; } }
        public string leftTerm { get { return LeftTerm; } set { LeftTerm = value; } }
        public ArrayList right { get { return Right; } set { Right = value; } }

        // delta(  A,                  1)            =      {qf}
        //          LeftNoTerm    LeftTerm          Right
        public Delta(string LeftNoTerm, string LeftTerm, ArrayList Right)
        {
            this.LeftNoTerm = LeftNoTerm;
            this.LeftTerm = LeftTerm;
            this.Right = Right;
        }

        // что то непонятное
        public void DebugDeltaRight()
        {
            int i = 0;
            for (; i < this.right.Count; i++)
            {
                if (i == 0)
                    System.Console.Write(" (" + this.right[i]);
                else
                    System.Console.Write("," + this.right[i]);
            }
            if (i == 0)
                System.Console.WriteLine();
            else
                System.Console.WriteLine(") ");
        }

    } // end class Delta
}// end class Automate
```

```
class Conv : Automate
{
   Automate DKA = null;
   //  множество всех правил deltaListAll
   ArrayList deltaListAll = null;  // all transitions
                        //  подмножества, которые содерджат  все заключительные
                        //состояния qf, то есть F'
   ArrayList FAll = null;

   public Conv() { }
   public Automate convert(Automate NDKA)
   {
      // инициализировать данные для каждого вызова convert
      this.DKA = new Automate();
      this.deltaListAll = new ArrayList();
      // вероятно не нужно
      Delta delta = null;
      delta = new Delta("S0", "1", new ArrayList() { "A" });
      deltaListAll.Add(delta);
      delta = new Delta("A", "(", new ArrayList() { "B" });
      deltaListAll.Add(delta);
      delta = new Delta("B", "0", new ArrayList() { "C" });
      deltaListAll.Add(delta);
      delta = new Delta("C", "1", new ArrayList() { "D" });
      deltaListAll.Add(delta);
      delta = new Delta("D", "0", new ArrayList() { "C" });
      deltaListAll.Add(delta);
      delta = new Delta("D", ")", new ArrayList() { "E" });
      deltaListAll.Add(delta);
      delta = new Delta("E", "0", new ArrayList() { "F" });
      deltaListAll.Add(delta);
```

```
    delta = new Delta("F", "1", new ArrayList() { "G" });
    deltaListAll.Add(delta);
    delta = new Delta("G", "0", new ArrayList() { "G" });
    deltaListAll.Add(delta);
    delta = new Delta("G", "1", new ArrayList() { "G" });
    deltaListAll.Add(delta);
    delta = new Delta("G", ".", new ArrayList() { "S" }); // ТОЧКА В КОНЦЕ
    deltaListAll.Add(delta);

    this.FAll = new ArrayList();

    // Шаг 1. Init q0 & sigma
    DKA.q0 = NDKA.q0;
    DKA.sigma = NDKA.sigma;

    //2. Создать множество всех подмножеств по Q (булеан) и
    //3. Создать множество всех правил DeltaList
    // 4 и 5,  подмножества, которые содерджат заключительные
    // состояния qf, то есть F'
    BuildDeltaList(NDKA);
    // Шаг 6. Опрелеоить достижимые состояния,
    // исключить недостижимые состояния из множества Q'
    // 1. Берем начальное состояние и определяем правило дельта,
    Reachability(DKA.q0);
    return DKA;
}

void BuildDeltaList(Automate NDKA)
{
    ArrayList right = null; // для нового правила
    int count = NDKA.q.Count;
    //    Time Complexity: O(n2^n),   Space Complexity: O(1)
```

```
// 1.  set size of power set of a set with set size  n is (2**n )
int sizeOfPowerSet = (int)Math.Pow(2, count);
string leftNoTerm = null; // is subset
string[] noTerm = null;     // для split
                    // 2. Run from counter 000..0 to 111..1
Console.WriteLine("Boolean_____ ");
for (int counter = 0; counter < sizeOfPowerSet; counter++)
{
   leftNoTerm = null;
   for (int j = 0; j < count; j++)
   {
      // Check if j-th bit in the counter is set If set then  build set, use comma
      // Console.WriteLine("!  0x{0:x8}", 1 << j);
      if ((counter & 1 << j) != 0)
      {
         // System.Console.WriteLine("NDKA.q[j] = " + NDKA.q[j]);
         if (leftNoTerm != null) leftNoTerm = leftNoTerm + ',' + NDKA.q[j];
         else leftNoTerm = "" + NDKA.q[j];
      }
   }
   if (leftNoTerm != null)
   { // 2**n -1 без пустых подмножеств
    // Найти delta'(S, a)
      noTerm = leftNoTerm.Split(',');
      // Шаг 4. построить subset  F
      BuildFAll(leftNoTerm, NDKA.f);
      foreach (string leftTerm in NDKA.sigma)
      {
         // по deltaListNDKA посмотреть имеющиеся правила для данного, одно
         foreach (string n in noTerm)
         { // ищем правило
            right = findTransition(n, leftTerm, NDKA.deltaList);
```

```
                if (right != null)
                {
                   deltaListAll.Add(new Delta(leftNoTerm, leftTerm, new ArrayList(right)));
                   break;
                }
             }
          }
          System.Console.WriteLine("       " + leftNoTerm); // булеан
       }
   } // end for
   DebugDeltaList(deltaListAll);
   DebugF(FAll);

} // BuildDeltaList

// найти переход
public ArrayList findTransition(string leftNoTerm, string leftTerm,
                                    ArrayList NDKAdeltaList)
{
   foreach (Delta d in NDKAdeltaList)
   { // найдено правило в ArrayList
      if (d.leftNoTerm == leftNoTerm && d.leftTerm == leftTerm)
         return d.right;
   }
   return null;
}

void BuildFAll(string leftNoTerm, ArrayList qf)
{
   // если в подмножестве noTerm есть заключительное состояние NDKA.f
   string[] noTerm = leftNoTerm.Split(',');
   foreach (string n in noTerm)
```

```
    { // ищем правило
        foreach (string f in qf)
        {
            if (n == f)
            {
                FAll.Add(leftNoTerm);
                // System.Console.WriteLine("  FAll.Add =      " + leftNoTerm);
                return;
            }
        }
    }
} // end BuildFAll



void Reachability(string q)
{
    // 1. Берем состояние по правилу дельта и определяем следующее дельта
    string right = null;
    foreach (Delta d in deltaListAll)
    {
        if (d.leftNoTerm == q)
        {
            // преобразовать в метку подмножество из right
            d.right = markSubset(d.right);
            DKA.deltaList.Add(d);
            DKA.q.Add(q);
            // всегда один элемент, так как  markSubset
            right = d.right[0].ToString();
            break;
        }
    }
    if (right == null) return; // нет достижимых состояний
```

```
    if (DKA.q.Contains(right))
    { // это состояние уже было,  останов
      //  заключительное состояние  должно быть F'
        if (FAll.Contains(right))
            // в F' оставить последнее достижимое состояние
            DKA.f.Add(right);
        else
        {
            System.Console.WriteLine(" Reachability error  " + 01);
            return;
        }
    }
    else Reachability(right);
} // end Reachability

ArrayList markSubset(ArrayList right)
{
    string r = null;
    foreach (string s in right)
    {
        if (r != null) r = r + ',' + s;
        else r = s;
    }
    return new ArrayList() { r };
}

void DebugF(ArrayList F)
{




        System.Console.WriteLine(" F all:_ ");
```

```
            for (int i = 0; i < F.Count; i++)
            {
               System.Console.WriteLine("        " + F[i]);
            }
        }
        public void DebugDeltaList(ArrayList deltaList)
        {
            System.Console.WriteLine("deltaList all:_ ");
            foreach (Delta d in deltaList)
            {
               Console.Write("          (" + d.leftNoTerm + "),"  + d.leftTerm + " = ");
               d.DebugDeltaRight();
            }
        }
    } // end class Convertor
}// end lab1
```