Московский авиационный институт
(государственный технический университет)

Факультет «Информационные технологии и прикладная математика»

Лабораторные работы
по курсу
«Системное программное обеспечение»
IV семестр

1. Спроектировать грамматику
2. Спроектировать конечный автомат, составить и реализовать диаграмму переходов КА
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: Дюсекеев А. Е.
Группа: М8О-204Б-17
Руководитель: Семёнов А. С.

Оценка:
Дата:

Москва, 2019

Лабораторная работа №1

Вариант №4

Формулировка задания

Спроектировать грамматику по заданному языку L

$$L = \{\omega_1 + \omega_2 + 1 \mid \omega_1 \in \{0,1\}^+, \omega_2 \in \{0,1\}^+ \}$$

Пояснение данного языка

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

$$L = \{\omega_1 + \omega_2 + 1 \mid \omega_1 \in \{0,1\}^+, \omega_2 \in \{0,1\}^+ \}$$

Грамматика

Строим регулярную грамматику, соответствующую регулярному языку L. $G = (T, V, S_0, P)$, где T − множество терминальных символов, V − множество состояний, S_0 − начальное состояние, P − множество правил.

$$G = (\{0,1\}, \{S_0, A, B, C, D\}, P, S_0)$$

$$\omega_1 = \{0,1,00,01,10, \dots\}$$

$$\omega_2 = \{0,1,00,01,10, \dots\}$$

Примеры цепочек из языка L: 0+0+1, 01+01+1, 01+0010+1 и т.д.

Правила переходов P:

S_0 -> 0A|1A
A -> 0A|1A|+B
B -> 0B|1B|+C
C -> 1

Пример вывода:

S_0 ⇒ 0A ⇒ 0+B ⇒ 0+0B ⇒ 0+0+C ⇒ 0+0+1

S_0 ⇒ 0A ⇒ 01A ⇒ 01+B ⇒ 01+0B ⇒ 01+01B ⇒ 01+01+C ⇒ 01+01+1

S_0 ⇒ 0A ⇒ 01A ⇒ 01+B ⇒ 01+0B ⇒ 01+00B ⇒ 01+001B ⇒ 01+0010B ⇒ 01+0010+C ⇒ 01+0010+1

Тогда язык, порождаемый грамматикой общего вида G, это язык

$$L(G) = \{\omega_1 + \omega_2 + 1 \mid \omega_1 \in \{0,1\}^+, \omega_2 \in \{0,1\}^+\}$$

Вывод:
Язык порождаемый грамматикой общего вида эквивалентен грамматике, так как одинаковое множество цепочек символов.

Лабораторная работа №2

Формулировка задания

1) Определить эквивалентен ли язык, порождаемый грамматикой языку, распознаваемого конечных автоматов
2) Спроектировать конечный автомат, составить и реализовать диаграмму переходов КА

Конечный автомат

$KA = (\Sigma, Q, \delta, q_0, F)$, где Σ – конечный алфавит входных символов, Q – конечное множество состояний, $Q \times \Sigma$ во множество $\mathcal{P}(Q)$ подмножеств Q: $\delta : Q \times \Sigma \to \mathcal{P}(Q)$, q_0 – начальное состояние, F – множество конечных состояний, δ - функция переходов, задаваемая отображением.

1) Спроектировать грамматику G = (T, V, P, S_0),

T = {0, 1, +} - множество терминальных символов
V = {S_0, A, B, C} - множество нетерминальных символов
$P = \{S_0 \to 0\text{A}|1A,\ A \to 0A|1A| + B, B \to 0B|1B| + C, C \to 1\}$ – множество правил

2) Определить свойства грамматики
Грамматика является право линейной, бесконечной.

Пример вывода: S_0 ⇒ 0A ⇒ 01A ⇒ 01+B ⇒ 01+0B ⇒ 01+01B ⇒ 01+01+C ⇒ 01+01+1

3) Используя грамматику G построить КА

КА = ({S_0, A, B, C, qf}, {0,1, +}, δ, S_0, qf)

δ(S_0, 0) = {A}; δ(S_0, 1) = {A};

δ(A, 0) = {A}; δ(A, 1) = {A}; δ(A, +) = {B};

δ(B, 0) = {B}; δ(B, 1) = {B}; δ(B, +) = {C};

δ(C, 1) = {qf};

Пример работы КА: $S_0$01+01+1 ⊢ A1+01+1 ⊢ A+01+1 ⊢ B01+1 ⊢ B1+1 B+1 ⊢ C1 ⊢ qf ξ

4) Определить свойства конечного автомата

Конечный автомат является детерминированным.

5) Построить диаграмму переходов ДК

Диаграммой переходов КА называется неупорядоченный граф, удовлетворяющий условиям:

- каждому состоянию q соответствует некоторая вершина, отмеченная его именем;
- диаграмма переходов содержит дугу из состояния q_k в состояние $q_{n,}$ отмеченную символом a, если q_k ($q_{n,}$,a) . Дуга может быть помечена множеством символов, переводящих автомат из состояния q_k в состояние q_n;
- вершины, соответствующие заключительным состояниям q_f принадлежащим F, отмечаются двойным кружком.

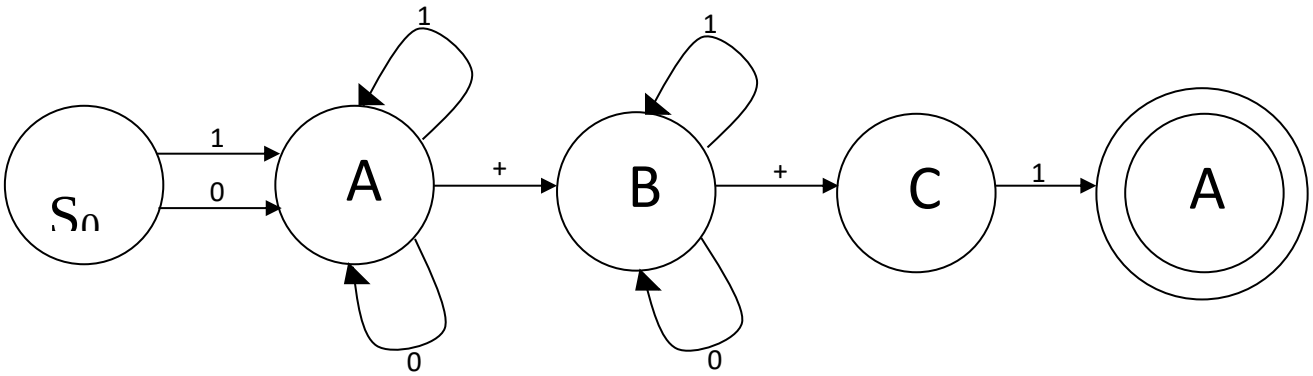


Рис 1. Диаграмма переходов ДКА

Лабораторная работа №3

Формулировка задания

Реализовать конечный автомат.

Код программы:

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

namespace MPTranslator
{
   class Program
   {

      static void Dialog()
      {
         Console.WriteLine("\n_____________________________________________________");
         Console.WriteLine("| DKA ( lab 2 ) ..... Enter 1   |");
         Console.WriteLine("| Convert NDKA to DKA                |");
         Console.WriteLine("| example ..... Enter 2.1 |");
         Console.WriteLine("| lab 3   ..... Enter 2   |");
         Console.WriteLine("| Grammar ( lab 4 - 6 )  ..... Enter 3   |");
         Console.WriteLine("| MP - auto ( lab 7,8 )  ..... Enter 4   |");
         Console.WriteLine("| LL - analizator ( lab 9 - 11 )  ..... Enter 5   |");
         Console.WriteLine("| LR - analizator                |");
         Console.WriteLine("| lab 12 - 16   ..... Enter 6   |");
         Console.WriteLine("| example ..... Enter 6.1 |");
         Console.WriteLine("| MP_Automate with delta rules  ..... Enter 7   |");
         Console.WriteLine("| MP_Translator   ..... Enter 8   |");
         Console.WriteLine("|_____________________________________________________|");
      }

      struct Tablekey
      {
         public int I;
         public char J;
         public Tablekey(int i, char j) { I = i; J = j; }
      }

      static ArrayList Grammar = new ArrayList();  //  правила грамматики
      static string Terminals;                  //  список терминалов
      static string NonTerminals;               //  список нетерминалов


      static void Execute()
```

```
    {
      Console.WriteLine("\nИсходная ");
      Info();
      RemoveEpsilonRules();
      Console.WriteLine("\nПосле удаления e-продукций");

      Grammar.Add("П S");    //дополнить грамматику правилом П -> S
      NonTerminals += "П";
      Terminals += "$";

      Console.WriteLine("\nПравила: \n ");
      for (IEnumerator rule = Grammar.GetEnumerator(); rule.MoveNext();)
        Console.WriteLine(rule.Current);

      Console.WriteLine("Терминалы : " + Terminals);
      Console.WriteLine("Нетерминалы: " + NonTerminals);
      Console.WriteLine("-----");
      Console.ReadLine();

      // генерация LR(1) таблицы
      //

      ComputeFirstSets();                  // вычислить множества FIRST

      Console.WriteLine("Вычислены множества FIRST для символов грамматики и строк \n ");

      string Symbols = NonTerminals;
      for (int i = 0; i < Symbols.Length; i++)
      {  //для каждого символа грамматики X
        char X = Symbols[i];
        Console.WriteLine("First( " + X + " ): " + First(X));
      }
      Console.WriteLine();
      for (IEnumerator rule = Grammar.GetEnumerator(); rule.MoveNext();)
      {
        string str = ((string)rule.Current).Substring(2);
        Console.WriteLine("First( " + str + " ): " + First(str));
      }

      Console.ReadLine();
      ArrayList[] CArray = CreateCArray();          // создать последовательность C
      Console.WriteLine("Создана последовательность C: \n ");
      for (int i = 0; i < CArray.Length; i++) { Console.WriteLine("I" + i + DebugArrayList(CArray[i])); }

      Hashtable ACTION = CreateActionTable(CArray);   // создать ACTION таблицу
      if (ACTION == null) { Console.WriteLine("Грамматика не является LR(1)"); Console.ReadLine();
return; }
```

```
Hashtable GOTO = CreateGotoTable(CArray);      // создать GOTO таблицу
// распечатать содержимое ACTION и GOTO таблиц:
Console.WriteLine("\nСоздана ACTION таблица \n ");

for (IDictionaryEnumerator c = ACTION.GetEnumerator(); c.MoveNext();)
   Console.WriteLine("ACTION[" + ((Tablekey)c.Key).I + ", " + ((Tablekey)c.Key).J + "] = " + c.Value);
Console.WriteLine("\nСоздана GOTO таблица \n ");

for (IDictionaryEnumerator c = GOTO.GetEnumerator(); c.MoveNext();)
   Console.WriteLine("GOTO[" + ((Tablekey)c.Key).I + ", " + ((Tablekey)c.Key).J + "] = " + c.Value);

Console.ReadLine();

//синтакический анлиз
string answer = "y";
while (answer[0] == 'y')
{
   string input;
   Console.WriteLine("Введите строку: ");
   input = Console.In.ReadLine() + "$";            //считать входную строку
   Console.WriteLine("\nВведена строка: " + input + "\n");
   Console.WriteLine("\nПроцесс вывода: \n ");
   if (input.Equals("$"))
   {                    //случай пустой строки
      Console.WriteLine(AcceptEmptyString ?
          "Строка допущена" :
          "Строка отвергнута");
      Console.ReadLine();
      continue; //  return;
   }
   Stack stack = new Stack();     //Стек автомата
   stack.Push("0");            //поместить стартовое
   //нулевое состояние
   try
   {
      for (;;)
      {
         int s = Convert.ToInt32((string)stack.Peek());
         //вершина стека
         char a = input[0];     //входной симол
         string action = (string)ACTION[new Tablekey(s, a)];
         //элемент
         //ACTION -таблицы
         if (action[0] == 's')
         {        //shift
            stack.Push(a.ToString());   //поместить в стек a
            stack.Push(action.Substring(2));
```

```
            //поместить в стек s'
            input = input.Substring(1);
            //перейти к следующему символу строки
          }
          else if (action[0] == 'r')
          {    //reduce
            //rule[1] = A, rule[2] = alpha
            string[] rule = action.Split(' ');
            //удалить 2 * Length(alpha) элементов стека
            for (int i = 0; i < 2 * rule[2].Length; i++)
              stack.Pop();
            //вершина стека
            int state = Convert.ToInt32((string)stack.Peek());
            //поместить в стек A и GOTO[state, A]
            stack.Push(rule[1]);
            stack.Push((GOTO[new Tablekey(state, rule[1][0])]).ToString());

            //вывести правило
            Console.WriteLine(rule[1] + "->" + rule[2]); Console.ReadLine();
          }
          else if (action[0] == 'a')   //accept
            break;
        }
        Console.WriteLine("Строка допущена"); // Console.ReadLine();
      }
      catch (Exception) { Console.WriteLine("Строка отвергнута"); } // Console.ReadLine();
      Console.ReadLine();
      Console.WriteLine("\n Продолжить? (y or n) \n");
      answer = Console.ReadLine();
    }

}

static void ReadGrammar()
{
    Terminals = "";
    NonTerminals = "";
    Grammar.Clear();
    string s;
    Hashtable term = new Hashtable();      //  временная таблица терминалов
    Hashtable nonterm = new Hashtable();   //  и нетерминалов
    Console.WriteLine("\nВведите продукции: \n ");
    while ((s = Console.In.ReadLine()) != "")
    { //  считывание правил
      Grammar.Add(s);                  // добавитьть правило в грамматику
      for (int i = 0; i < s.Length; i++)
        //  анализ элементов правила
```

```
            if (s[i] != ' ')
            {
               //  если текущий символ - терминал, еще не добавленный в term
               if (s[i] == s.ToLower()[i] && !term.ContainsKey(s[i]))
                  term.Add(s[i], null);
               if (s[i] != s.ToLower()[i] && !nonterm.ContainsKey(s[i]))
                  nonterm.Add(s[i], null);
            }
      }
      //  переписываем терминалы и нетерминалы в строки Terminals и NonTerminals
      for (IDictionaryEnumerator c = term.GetEnumerator(); c.MoveNext();)
         Terminals += (char)c.Key;
      for (IDictionaryEnumerator c = nonterm.GetEnumerator(); c.MoveNext();)
         NonTerminals += (char)c.Key;
   }

   static string DebugArrayList(ArrayList arraylist)
   {
      string arraylist_str = " { ";
      for (int i = 0; i < arraylist.Count; i++)
      {
         if (i == 0)
            arraylist_str = arraylist_str + arraylist[i].ToString();
         else
            arraylist_str = arraylist_str + "; " + arraylist[i].ToString();
      }
      arraylist_str = arraylist_str + " } ";
      return arraylist_str;
   }

   static void Info()
   {
      Console.WriteLine("КС - грамматика : " +
                  " \nАлфавит нетерминальных символов: " + NonTerminals +
                  " \nАлфавит терминальных символов: : " + Terminals +
                  " \nПравила : \n" + DebugArrayList(Grammar));
      Console.ReadLine();
   }

   // список найденных комбинаций
   static ArrayList combinations = new ArrayList();
   static void GenerateCombinations(int depth, string s)
   {
      if (depth == 0)
         combinations.Add(s);
      else
      {
```

```
            GenerateCombinations(depth - 1, "0" + s);
            GenerateCombinations(depth - 1, "1" + s);
        }
    }

    //  создает список правил, в которых вычеркнут один или более символов A в правой части
    static ArrayList GenerateRulesWithout(char A)
    {
        ArrayList result = new ArrayList();  //  итоговый список
        //  цикл по правилам
        for (IEnumerator rule = Grammar.GetEnumerator(); rule.MoveNext();)
        {
            string current = (string)rule.Current;  //  текущее правило,
            string rhs = current.Substring(2);  //  его правая часть,
            string[] rhs_split = rhs.Split(A);  //  отдельные сегменты rhs, разделенные A
            int counter;
            if (rhs.IndexOf(A) != -1)
            { //  если правая часть содержит A
                counter = 0;  //  подсчитывает количество вхождений A
                for (int i = 0; i < rhs.Length; i++)
                    if (rhs[i] == A)
                        counter++;
                combinations.Clear();
                GenerateCombinations(counter, "");  //  генерация комбинаций
                for (IEnumerator element = combinations.GetEnumerator(); element.MoveNext();)
                    if (((string)element.Current).IndexOf('1') != -1)
                    {
                        //  если текущая комбинация содержит хоть один вычеркиваемый символ (т.е.
единицу)
                        string combination = (string)element.Current;
                        string this_rhs = rhs_split[0];
                        //  если текущий символ комвинации - единица,
                        //  то вычеркиваем A(просто соединяем сегменты правой части правила),
                        //  иначе вставляем дополнительный символ A)
                        //
                        for (int i = 0; i < combination.Length; i++)
                            this_rhs += (combination[i] == '0' ? A.ToString() : "") + rhs_split[i + 1];
                        result.Add(current[0] + " " + this_rhs);
                    }
            }  // end if
        }  // end for
        return result;
    }

    static bool AcceptEmptyString;     // допускать ли пустую строку
    static void RemoveEpsilonRules()
    {  // удаление e-правил
```

```
  AcceptEmptyString = false;     // флаг принадлежности пустой строки языку
  bool EpsilonRulesExist;
  do
  {
    EpsilonRulesExist = false;
    for (IEnumerator rule = Grammar.GetEnumerator(); rule.MoveNext();)
      if (((string)rule.Current)[2] == 'e')
      {   // нашли эпсилон-правило
        // принимаем пустую строку, если левая часть правила содержит стартовый символ
        char A = ((string)rule.Current)[0];
        if (A == 'S') { AcceptEmptyString = true; }
        Grammar.AddRange(GenerateRulesWithout(A));
        Grammar.Remove(rule.Current);      // удаляем e-правило
        EpsilonRulesExist = true;
        break;
      }
  }
  while (EpsilonRulesExist);     //  пока существуют эпсилон-правила
}

static Hashtable FirstSets = new Hashtable();      //Набор множеств First
public static void ComputeFirstSets()
{
  for (int i = 0; i < Terminals.Length; i++)
    FirstSets[Terminals[i]] = Terminals[i].ToString();   // FIRST[c] = {c}*/
  for (int i = 0; i < NonTerminals.Length; i++)
    FirstSets[NonTerminals[i]] = "";                  //First[x] = ""
  bool changes;
  do
  {
    changes = false;
    for (IEnumerator rule = Grammar.GetEnumerator(); rule.MoveNext();)
    {
      // Для каждого правила X-> Y0Y1…Yn
      char X = ((string)rule.Current)[0];
      string Y = ((string)rule.Current).Substring(2);
      for (int k = 0; k < Terminals.Length; k++)
      {
        char a = Terminals[k]; // для всех терминалов a
        // a принадлежит First[Y0]
        if (((string)FirstSets[Y[0]]).IndexOf(a) != -1)
          if (((string)FirstSets[X]).IndexOf(a) == -1)
          {
            //Добавить a в FirstSets[X]
            FirstSets[X] = (string)FirstSets[X] + a;
            changes = true;
          }
```

```
            }
          }
        }
        while (changes); //  пока вносятся изменения
      }

      // функции доступа ко множествам FIRST
      public static string First(char X) { return (string)FirstSets[X]; }

      public static string First(string X) { return First(X[0]); }

      static ArrayList Closure(ArrayList I)
      {
        ArrayList result = new ArrayList();
        //      Console.WriteLine("Closure_множество ситуаций: " + DebugArrayList(I));
        //добавляем все элементы I в замыкание
        for (IEnumerator item = I.GetEnumerator(); item.MoveNext();)
          result.Add(item.Current);
        bool changes;
        do
        {
          changes = false;
          //для каждого элемента R
          for (IEnumerator item = result.GetEnumerator(); item.MoveNext();)
          {
            //A -> alpha.Bbeta,a
            string itvalue = (string)item.Current;
            int Bidx = itvalue.IndexOf('.') + 1;
            char B = itvalue[Bidx];                  //  B
            if (NonTerminals.IndexOf(B) == -1)         //  если после точки терминал, то ситуацию не
обрабатываем
              continue;
            string beta = itvalue.Substring(Bidx + 1);
            beta = beta.Substring(0, beta.Length - 2);   //  beta
            char a = itvalue[itvalue.Length - 1];     //  a
            //для каждого правила B -> gamma
            for (IEnumerator rule = Grammar.GetEnumerator(); rule.MoveNext();)
              if (((string)rule.Current)[0] == B)
              { //  B - >gramma
                string gamma = ((string)rule.Current).Substring(2);    //  gamma
                string first_betaa = First(beta + a);
                // для каждого b из FIRST(betaa)
                for (int i = 0; i < first_betaa.Length; i++)
                {
                  //          Console.WriteLine("i= " + i + "first_betaa[i]= " + first_betaa[i]);
                  char b = first_betaa[i];         //  b
                  string newitem = B + " ." + gamma + "," + b;
```

```
                //          Console.WriteLine("сгенерирована ситуация: " + newitem);
                // добавить элемент B -> .gamma,b
                if (!result.Contains(newitem))
                {
                   result.Add(newitem);
                   //              Console.WriteLine("добавлена новая ситуация: " + newitem);
                   changes = true;
                   goto breakloop;
                }
             }
          } // for по правилам B -> gamma
      } // for по  ситуациям R
      breakloop:;
   }
   while (changes);
   //     Console.WriteLine("Closure_замыкание_ result " + DebugArrayList(result));
   return result;
}

// Функция GoTo
static ArrayList GoTo(ArrayList I, char X)
{
   ArrayList J = new ArrayList();
   // для всех ситуаций из I
   for (IEnumerator item = I.GetEnumerator(); item.MoveNext();)
   {
      string itvalue = (string)item.Current;
      string[] parts = itvalue.Split('.');
      if ((parts[1])[0] != X)
         continue;
      //если ситуация имеет вид A alpha.Xbeta, a
      J.Add(parts[0] + X + "." + parts[1].Substring(1));
   }
   return Closure(J);
}

//Процедура получения последовательности C
static bool SetsEqual(ArrayList lhs, ArrayList rhs)
{
   string[] lhsArr = new string[lhs.Count];
   // преобразование списка
   lhs.CopyTo(lhsArr);           // в массив
   Array.Sort(lhsArr);           // и его сортировка
   string[] rhsArr = new string[rhs.Count];
   // то же для второго множества
   rhs.CopyTo(rhsArr);
   Array.Sort(rhsArr);
```

```
    if (lhsArr.Length != rhsArr.Length) // если размеры не равны множества точно не равны
        return false;
    for (int i = 0; i < rhsArr.Length; i++)
        if (!lhsArr[i].Equals(rhsArr[i])) // если же размеры равны, проверяем по элементам
            return false;
    return true;
}

// Функция SetsEqual() используется функцией Contatains,
// определяющей, является ли множество g элементом списка C
static bool Contains(ArrayList C, ArrayList g)
{
    for (IEnumerator item = C.GetEnumerator(); item.MoveNext();)
        if (SetsEqual((ArrayList)item.Current, g))
            return true;
    return false;
}

static ArrayList[] CreateCArray()
{
    string Symbols = Terminals + NonTerminals; // все символы грамматики
    ArrayList C = new ArrayList();
    Console.WriteLine("CreateCArray: ");
    // добавить элемент I0 = Closure ({"П .S,$"})
    C.Add(Closure(new ArrayList(new Object[] { "П .S,$" })));
    Console.WriteLine("I0 : " + DebugArrayList(Closure(new ArrayList(new Object[] { "П .S,$" }))));
    Console.ReadLine();
    int counter = 0;
    bool modified;
    do
    {
        modified = false;
        for (int i = 0; i < Symbols.Length; i++)
        {//для каждого символа грамматики X
            char X = Symbols[i];
            Console.WriteLine("Для символа " + X);
            // для каждого элемента последовательности C
            for (IEnumerator item = C.GetEnumerator(); item.MoveNext();)
            {
                ArrayList g = GoTo((ArrayList)item.Current, X);  // GoTo(Ii, X)
                Console.WriteLine("GoTo( " + DebugArrayList((ArrayList)item.Current) + "," + X + "): \n"
                                + DebugArrayList(g));
                Console.ReadLine();
                // если множество g непусто и еще не включено в C
                if (g.Count != 0 && !Contains(C, g))
                {
                    C.Add(g); counter++;
```

```
                    Console.WriteLine("добавлено I" + counter + " : " + DebugArrayList(g));
Console.ReadLine();
                    modified = true; break;
                  }
                }
              }
            }
            while (modified);      // пока вносятся изменения
            ArrayList[] CArray = new ArrayList[C.Count];
            // преобразование списка  в массив
            C.CopyTo(CArray);
            return CArray;
        }

        static bool WriteActionTableValue(Hashtable ACTION, int I, char J, string action)
        {
            Tablekey Key = new Tablekey(I, J);
            if (ACTION.Contains(Key) && !ACTION[Key].Equals(action))
            {
                Console.WriteLine("не LR(1) грамматика"); Console.ReadLine();
                return false;
            }                           // не LR(1) вид
            else
            {
                ACTION[Key] = action;
                return true;
            }
        }

        static Hashtable CreateActionTable(ArrayList[] CArray)
        {
            Hashtable ACTION = new Hashtable();
            for (int i = 0; i < CArray.Length; i++)
            { // цикл по элементам C
                // Для каждой ситуации из множества CArray[i]
                for (IEnumerator item = CArray[i].GetEnumerator(); item.MoveNext();)
                {
                    string itvalue = (string)item.Current;         // ситуация
                    char a = itvalue[itvalue.IndexOf('.') + 1];    // символ за точкой
                    // Если ситуация имеет вид "A alpha.abeta,b"
                    if (Terminals.IndexOf(a) != -1)              // если a - терминал
                        for (int j = 0; j < CArray.Length; j++)
                            if (SetsEqual(GoTo(CArray[i], a), CArray[j]))
                            {
                                // существует элемент CArray[j], такой,
                                // что GoTo(CArray[i], a) == CArray[j]
                                // запись ACTION[i, a] = shift j
```

```
                if (WriteActionTableValue(ACTION, i, a, "s " + j) == false)
                    return null;
                // грамматика не LR(1)
                break;
            }
        // Если ситуация имеет вид "A alpha., a"
        if (itvalue[itvalue.IndexOf('.') + 1] == ',')
        { // за точкой запятая
            a = itvalue[itvalue.Length - 1];  // определить значение a
            string alpha = itvalue.Split('.')[0].Substring(2);  // и alpha                5!
            if (itvalue[0] != 'П')
            {               // если левая часть не равна П
                // ACTION[i, a] = reduce A -> alpha
                if (WriteActionTableValue(ACTION, i, a, "r " + itvalue[0] + " " + alpha) == false)
                    return null;                // грамматика не LR(1)
            }
        }
        // Если ситуация имеет вид "П S.,$"
        if (itvalue.Equals("П S.,$"))
        {
            // ACTION[i, '$'] = accept
            if (WriteActionTableValue(ACTION, i, '$', "a") == false)
                return null; // грамматика не LR(1)
        }
    }
  }
  return ACTION;
}

static Hashtable CreateGotoTable(ArrayList[] CArray)
{
  Hashtable GOTO = new Hashtable();
  for (int c = 0; c < NonTerminals.Length; c++)
    // для каждого нетерминала A
    for (int i = 0; i < CArray.Length; i++)
    {            // для каждого элемента Ii из C
      ArrayList g = GoTo(CArray[i], NonTerminals[c]);
      // g=GoTo[Ii, A]
      for (int j = 0; j < CArray.Length; j++)
        // если в C есть Ij=g
        if (SetsEqual(g, CArray[j]))
          // GOTO[i, A] =j
          GOTO[new Tablekey(i, NonTerminals[c])] = j;
    }
  return GOTO;
}
```

```
static string[,] Mtable(myGrammar G)
{ // построение заданной таблицы
   string[,] t = new string[G.T.Count + G.V.Count + 2, G.T.Count + 1];
   t[0, 3] = "(F+L),1";
   t[1, 3] = "(L*),2";
   t[1, 0] = "i,3";
   t[2, 3] = "F,4";
   t[2, 0] = "F,4";
   t[3, 0] = "выброс";
   t[4, 1] = "выброс";
   t[5, 2] = "выброс";
   t[6, 3] = "выброс";
   t[7, 4] = "выброс";
   t[8, 5] = "допуск";
   return t;
}

static int parsing(string s, int c, myGrammar G)
{ // алгоритм разбора
   Stack z = new Stack();  // магазин
   string eline = null;
   string zline = null;
   string qline = null;

   z.Push("$");
   z.Push(s);
   string[,] table = Mtable(G);  // таблица
   ArrayList outstr = new ArrayList();  // выходная лента
   int i, j;
   bool d = false;  // допуск

   for (int k = 0; k < s.Length + 1; k++)
   {
      if (c == 0) return 3;  // разбор по шагам
      if (k == s.Length) j = G.T.Count;   // символ пустой строки
      else
      {
         if (!G.T.Contains(s[k].ToString())) { d = false; break; }  // не символ алфавита
         j = G.T.IndexOf(s[k].ToString());  // выбор столбца
      }
      if (G.V.Contains(z.Peek())) i = G.V.IndexOf(z.Peek());  // выбор строки
      else if (z.Peek().ToString() == "$") i = G.T.Count + G.V.Count;
      else i = G.T.IndexOf(z.Peek()) + G.V.Count;
      if (k != s.Length + 1)
      {   // вывод на экран
         eline = "";
         zline = "";
```

```
            qline = "";
            for (int m = k; m < s.Length; m++) eline = eline + (s[m].ToString());
            Stack z1 = new Stack();
            while (z.Count != 0)
            {
               zline = zline + (z.Peek().ToString());
               z1.Push(z.Pop());
            }
            while (z1.Count != 0)
               z.Push(z1.Pop());
            foreach (string o in outstr)
               qline = qline + o;
         }
         if (table[i, j] == "выброс") { z.Pop(); continue; }  // выброс символа
         else if (table[i, j] == "допуск") { d = true; break; }  // допуск строки
         else if (table[i, j] == null) { d = false; break; } // ошибка
         else
         {  // запись в магазин и на выходную ленту
            int zp = table[i, j].IndexOf(',');   // разбор ячейки таблицы до запятой
            z.Pop();
            for (int l = zp - 1; l >= 0; l--)
            {
               z.Push(table[i, j][l].ToString());  // в магазин
            }
            outstr.Add(table[i, j][zp + 1].ToString()); // на ленту
            k--;
         }
         c--;
      }
      if (d) return 1;
      else return 2;
   }


   static void Main()
   {
      while (true)
      {
         Dialog();
         switch (Console.ReadLine())
         {

            case "1":
               myAutomate ka = new myAutomate(new ArrayList() { "S0", "A", "B", "qf" },
                        new ArrayList() { "0", "1", "-", "+", "", "E" },
                        new ArrayList() { "qf" }, "S0");
              ka.AddRule("S0", "0", "A");
```

```
                ka.AddRule("S0", "1", "A");
                ka.AddRule("A", "0", "A");
                ka.AddRule("A", "1", "A");
                ka.AddRule("A", "+", "B");
                ka.AddRule("B", "0", "B");
                ka.AddRule("B", "1", "B");
                ka.AddRule("B", "+", "C");
                ka.AddRule("C", "1", "qf");
                Console.WriteLine("Enter line to execute :");
                ka.Execute(Console.ReadLine());
                break;

            case "2.1":
                myAutomate example = new myAutomate(new ArrayList() { "S0", "1", "2", "3", "4", "5",
"6", "7", "8", "9", "qf" },
                            new ArrayList() { "a", "b" },
                            new ArrayList() { "qf" }, "S0");
                example.AddRule("S0", "", "1");
                example.AddRule("S0", "", "7");
                example.AddRule("1", "", "2");
                example.AddRule("1", "", "4");
                example.AddRule("2", "a", "3");
                example.AddRule("4", "b", "5");
                example.AddRule("3", "", "6");
                example.AddRule("5", "", "6");
                example.AddRule("6", "", "1");
                example.AddRule("6", "", "7");
                example.AddRule("7", "a", "8");
                example.AddRule("8", "b", "9");
                example.AddRule("9", "b", "qf");

                myAutomate dkaEX = new myAutomate();
                dkaEX.BuildDeltaDKAutomate(example);
                dkaEX.DebugAuto();
                Console.WriteLine("Enter line to execute :");
                dkaEX.Execute(Console.ReadLine());

                break;

            case "2":
                myAutomate ndka = new myAutomate(new ArrayList() { "S0","1","2",  "3",  "4",  "5",  "6",
"7",  "8",  "9", "10",
                                                    "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "qf"
},
                            new ArrayList() { "1", "0", "+", "2" },
                            new ArrayList() { "qf" }, "S0");
                ndka.AddRule("S0", "1", "1");        //W1
                ndka.AddRule("1", "0", "2");
                ndka.AddRule("2", "+", "3");
```

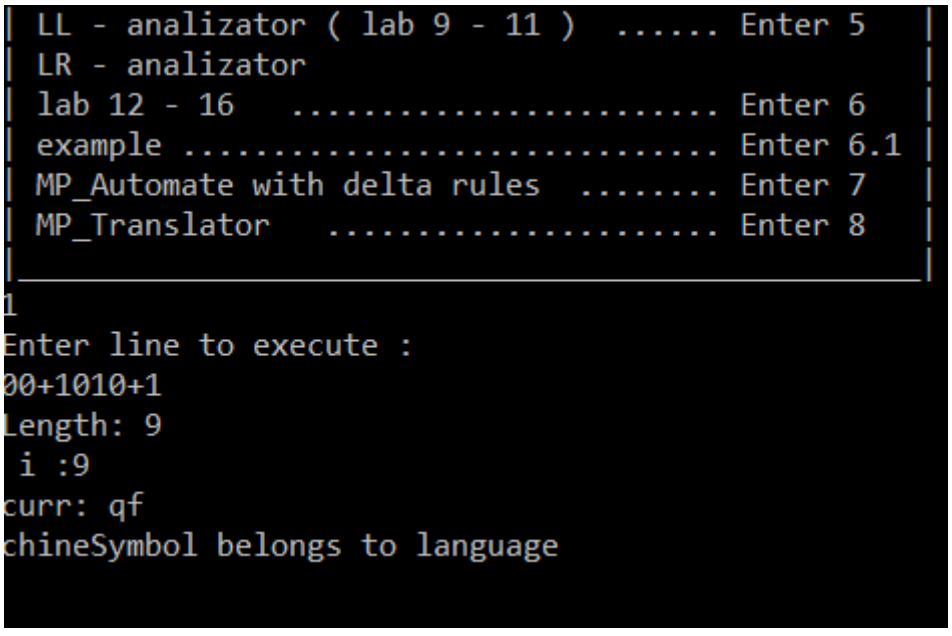
```
                ndka.AddRule("3", "", "4");          //W2
                ndka.AddRule("4", "", "5");
                ndka.AddRule("4", "", "7");
                ndka.AddRule("4", "", "9");
                ndka.AddRule("5", "1", "6");
                ndka.AddRule("7", "2", "8");
                ndka.AddRule("6", "", "9");
                ndka.AddRule("8", "", "9");
                ndka.AddRule("9", "", "4");
                ndka.AddRule("9", "", "10");

                ndka.AddRule("10", "1", "11");        //W3
                ndka.AddRule("11", "0", "12");
                ndka.AddRule("12", "", "13");
                ndka.AddRule("13", "", "9");
                ndka.AddRule("13", "", "14");

                ndka.AddRule("14", "", "15");         //W4
                ndka.AddRule("14", "", "17");
                ndka.AddRule("15", "0", "16");
                ndka.AddRule("17", "1", "18");
                ndka.AddRule("16", "", "19");
                ndka.AddRule("18", "", "19");
                ndka.AddRule("19", "", "14");
                ndka.AddRule("19", "", "20");
                ndka.AddRule("20", "", "15");
                ndka.AddRule("14", "", "qf");
                ndka.AddRule("20", "", "qf");

                myAutomate dka = new myAutomate();
                dka.BuildDeltaDKAutomate(ndka);
                dka.DebugAuto();
                Console.WriteLine("Enter line to execute :");
                dka.Execute(Console.ReadLine());
                break;

            case "3":
                myGrammar G = new myGrammar(new ArrayList() { "a", "b", "c", "d" }, new ArrayList() {
"S", "A", "B", "C", "F" }, "S");
                G.AddRule("S", new ArrayList() { "b" });
                G.AddRule("S", new ArrayList() { "c", "A", "B" });
                G.AddRule("A", new ArrayList() { "A", "b" });
                G.AddRule("A", new ArrayList() { "c" });
                G.AddRule("B", new ArrayList() { "c", "B" });
                G.AddRule("C", new ArrayList() { "C", "a" });
                G.AddRule("F", new ArrayList() { "d" });
```

```
            G.DebugPrules();

            myGrammar G1 = G.unUsefulDelete();
            G1.DebugPrules();

            myGrammar G2 = G1.EpsDelete();
            G2.DebugPrules();

            myGrammar G3 = G2.ChainRuleDelete();
            G3.DebugPrules();

            myGrammar G4 = G3.LeftRecursDelete();
            G4.DebugPrules();
            // G4 - приведенная грамматика

            break;

        case "4": //МП - автоматы
            myGrammar kcGrammar = new myGrammar(new ArrayList() { "i", ":", "*", "(", ")" },
                                    new ArrayList() { "S", "F", "L" }, "S");
            kcGrammar.AddRule("S", new ArrayList() { "(", "F", ":", "L", ")" });
            kcGrammar.AddRule("F", new ArrayList() { "L", "*" });
            kcGrammar.AddRule("F", new ArrayList() { "i" });
            kcGrammar.AddRule("L", new ArrayList() { "F" });

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

            myMp MP = new myMp(kcGrammar);
            Console.Write("Debug Mp ");
            MP.debugDelta();

            Console.WriteLine("\nEnter the line :");
            Console.WriteLine(MP.Execute_(Console.ReadLine()).ToString());
            break;

        case "5": // LL Разбор
            myGrammar exemple = new myGrammar(new ArrayList() { "i", "(", ")", "+", "*", "" },
                                    new ArrayList() { "S", "E", "T", "T'", "P" }, "S");
            exemple.AddRule("S", new ArrayList() { "T", "E" });
            exemple.AddRule("E", new ArrayList() { "+", "T", "E" });
            exemple.AddRule("E", new ArrayList() { "" });
            exemple.AddRule("T", new ArrayList() { "P", "T'" });
            exemple.AddRule("T'", new ArrayList() { "*", "P", "T'" });
            exemple.AddRule("T'", new ArrayList() { "" });
            exemple.AddRule("P", new ArrayList() { "(", "S", ")" });
```

```
                exemple.AddRule("P", new ArrayList() { "i" });

                LLParser parser = new LLParser(exemple);
                Console.WriteLine("Введите строку: ");
                if (parser.Parse(Console.ReadLine()))
                {
                    Console.WriteLine("Успех. Строка соответствует грамматике.");
                    Console.WriteLine(parser.OutputConfigure);
                }
                else
                {
                    Console.WriteLine("Не успех. Строка не соответствует грамматике.");
                }
                break;
            case "6":
                ReadGrammar();
                Execute();
                break;
            case "6.1":
                Terminals = "+*i()";
                NonTerminals = "STP";
                Grammar.Add("S S+T");
                Grammar.Add("S T");
                Grammar.Add("T T*P");
                Grammar.Add("T P");
                Grammar.Add("P i");
                Grammar.Add("P (S)");
                Execute();
                break;

            case "7": //МП - автоматы
                myMp Mp = new myMp(new ArrayList() { "q0", "q1", "q2", "qf" }, new ArrayList() { "a", "b"
}, new ArrayList() { "z0", "a" }, "q0", new ArrayList() { "qf" });

                Mp.addDeltaRule("q0", "a", "z0", new ArrayList() { "q1" }, new ArrayList() { "a", "z0" });
                Mp.addDeltaRule("q1", "a", "a", new ArrayList() { "q1" }, new ArrayList() { "a", "a" });
                Mp.addDeltaRule("q1", "b", "a", new ArrayList() { "q2" }, new ArrayList() { "e" });
                Mp.addDeltaRule("q2", "b", "a", new ArrayList() { "q2" }, new ArrayList() { "e" });
                Mp.addDeltaRule("q2", "e", "z0", new ArrayList() { "qf" }, new ArrayList() { "e" });
                Console.Write("Debug Mp ");
                Mp.debugDelta();

                Console.WriteLine("\nEnter the line :");
                Console.WriteLine(Mp.Execute_(Console.ReadLine()).ToString());
                break;

            case "8": //МП-преобразователь
```

```
                TranslGrammar gramm = new TranslGrammar(new ArrayList() { "a", "b" }, new ArrayList() {
"S", "A" }, "S", new ArrayList() { "0", "1" });
                gramm.AddTrules("S", new ArrayList() { "a", "A", "b" }, new ArrayList() { "0", "1", "A" });
                gramm.AddTrules("A", new ArrayList() { "a", "A", "b" }, new ArrayList() { "0", "1", "A" });
                gramm.AddTrules("A", new ArrayList() { "e" }, new ArrayList() { "e" });
                Translator mytrans = new Translator(gramm);
                mytrans.debugDelta();
                Console.WriteLine("Напишите строку:");
                Console.WriteLine("Перевод: " + mytrans.Translation(Console.ReadLine()));
                break;

            default:
                Console.WriteLine("Выход из программы");
                return;

        }
      }
    }
  }
}
```

Вывод программы:



Лабораторная работа №4

Привести заданную КС грамматику к приведенной форме

Грамматика G = (T, V, P, S0) называется контекстно-свободной (КС) (или бесконтекстной), если каждое правило из P имеет вид $A \rightarrow \alpha$, где $A \subset V$, $\alpha \subset (T \cup V)^*$.

A) Устранить из грамматики G бесполезные символы. Применить алгоритм 3.3 к грамматике G.

Алгоритм 3.1. Определение множества производящих нетерминальных символов V_p : если все символы цепочки из правой части правила вывода являются производящими, то нетерминал в левой части правила вывода также должен быть производящим.

Алгоритм 3.2. Определение множества достижимых символов V_r : если нетерминал в левой части правила грамматика является достижимым, то достижимы и все символы правой части этого правила.

Алгоритм 3.3. Устранение бесполезных символов. Вначале исключить непроизводящие (3.1) нетерминалы, а затем недостижимые (3.2) символы.

P = {S ⟶ cB, B ⟶ c, B ⟶ cA, A ⟶ Ab, C ⟶ Ca, F ⟶ d}

{C, F} – непроизводящие нетерминалы, {a, d} - недостижимые символы.

$V_p = \{A \mid (A \Rightarrow^{+} \alpha)\ A \in V, \alpha \in T^{+}\}$

Шаг 1. V_p = {S, B, F} – множество производящих.

{A, C} – множество непроизводящих нетерминалов.

{S, B, A} – множество достижимых терминалов.

{F, C} – множество недостижимых терминалов.

{a, d} – множество недостижимых символов.

Шаг 2. G_1 = ({S, B, A}, {a, b, c}, P, S), где

P = {S ⟶ cB, B ⟶ c, B ⟶ cA, A ⟶ Ab}.

Шаг 3. V_r = {S, B} – множество достижимых.

Шаг 4. G’ = ({S, B}, {c}, {S ⟶ cB, B ⟶ c}, S0}

Построим примеры строк L(G’) = {cc}.

Получается, что L(G) ≈ L(G`) – что и требовалось доказать.

B) Устранить ε–правила из грамматики G`, применить алгоритм 3.5.

P = {S ⟶ cB, B ⟶ cB, B ⟶ cA, A ⟶ ACb, A ⟶ ε, C ⟶ Ca, C ⟶ ε}

Шаг 1. Множество укорачивающих нетерминалов $V_ε$ = {A, C}.

Шаг 2. Положить P’ = Ø – правила не пустые.

Шаг 3. Для правила B → cA и A → ACb добавляем в P’: S → cB, B → cB и B → c.

Тогда P’ = {S ⟶ cB, B ⟶ cB, B ⟶ c}.

Шаг 4. G’ = ({a, c}, {S’, S, B}, {S ⟶ cB, B ⟶ cB, B ⟶ c}, S’)

C) Устранить из КС грамматики G цепны
D) е правила, применить алгоритм 3.6.

P = {S ⟶ cB, B ⟶ cB, B ⟶ cA, A ⟶ C, A ⟶ aB, C ⟶ Ca, C ⟶ cf}

Шаг 1. Находим все цепные правила в грамматике G.

Шаг 2. Для каждой пары добавить в грамматику G’ все правила вида

A → α, где α – не цепное правило из грамматики G.

Шаг 3. Удаляем все цепные правила

1) Из P = {S ⟶ cB, B ⟶ cB, B ⟶ cA, A ⟶ C, A ⟶ aB, C ⟶ Ca, C ⟶ cf} видим, что A → C порождает цепное правило.
2) Тогда смотрим, что порождает C: C → Ca, C → cf. Отсюда в P’ добавляем правило A → Ca, A → a.
3) Затем удаляем из P’ правило порождения A → C.

Получим новое конечное множество правил порождения:

P’ = {S ⟶ cB, B ⟶ cB, B ⟶ cA, A ⟶ aB, A → Ca, A → a, C ⟶ Ca, C ⟶ cf}

E) **Устранить левую рекурсию в заданной КС-грамматике G_1, порождающей скобочные арифметические выражения. Применить алгоритм 3.7. к грамматике G.**

P = {S → Ba, S → Ab, A → Sa, A → AAb, A → c, B → Sb, B → b }

Шаг 1. Находим правила в которых есть левая рекурсия: A → AAb, Ab = α.
Шаг 2. Находим правило порождающее α: S → Ab.

Шаг 3. Добавляем правила для P` A → S, и S → AbS.
Таким образом P` = {S → Ba, S → Ab, S → AbS, A → Sa, A → Ab, A → S, A → c, B → Sb, B → b}.

Практическая работа №3 (7-8 лаб.)

Задание:

Построить МП-автомат P по КС-грамматики G = (T, V, P, S), без левой рекурсии. Написать последовательность тактов автоматов для выделенной цепочки. Определить свойства автоматов. Лаб. 7. -2.1. Лаб. 8. -2.2.

1. Изучить алгоритмы построения МП-автомат P и расширенного МП-автомата по заданной КС-грамматике (см. раздел 3.4.).

2. Выполнить построение согласно алгоритмам. Смотрите пример и последовательность выполнения работы из раздела 3.4.

$$T = \{i, *, -, (,)\}, V = \{S, F, L\},$$

$$P = \{S \to (F) * L, F \to -L, F \to i, L \to F\}$$

Построить МП-автомат P и расширенный МП-автомат по КС-грамматики G = (T, V, P, S), без левой рекурсии. Написать последовательность тактов автоматов для выделенной цепочки. Определить свойства автоматов.

А). Построить МП-автомат по КС-грамматике G, используя алгоритм 3.8.

Вариант 4:

4. T = {i, *, -, (,)}, V = {S, F, L}, P = {S→ (F)* L, F→ - L, F→ i, L→F}

Решение:

Используя алгоритм, получим МП-автомат:

МП = ({q}, {i, *,-, (,)}, {i, *, -, (,), S, F, L}, δ, q_0, S, {q}),

в котором функция переходов δ определяется следующим образом:

1. δ(q_0, ε, S) = {(q, (F)* L)}
2. δ(q, ε, F) = {(q, - L), (q, i)}
3. δ(q, ε, L) = {(q, F)}
4. δ(q, a, a) = {(q, ε)} для всех a ϵ ∑ = {i, *, -, (,)}.

Определение 9. МП автомат – это семерка объектов МП = (Q, ∑, Г, δ, q, z, F) Q – конечное множество состояний устройства управления; ∑ - конечный алфавит входных символов; Г - конечный алфавит магазинных символов; δ - функция переходов, отображает множества Q × (∑∪ {ε}) × Г в множество конечных подмножеств множества Q×Г*; q - начальное состояние, q0 Q; z0 - начальный символ магазина, z ∈ Г; F - множество заключительных состояний.

Определение 10. Конфигурацией МП-автомата называется тройка (q, ω, z) ∈Q×∑*×Г*, где q – текущее состояние управляющего устройства; ω – необработанная часть входной цепочки (первый символ цепочки ω находится под входной головкой; если ω = ε, то считается, что вся входная цепочка прочитана); z – содержимое магазина (самый левый символ цепочки z считается верхним символом магазина; если z = ε, то магазин считается пустым).

2. Строим МП автомат по КС грамматике по алгоритму 3.8:

Алгоритм 3.8. По КС-грамматике G = (T, V, P, S) можно построить МП автомат, L(МП) = L(G). Пусть МП = ({q}, ∑, ∑∪V, δ, q, S, {q}), где δ определяется следующим образом: 1. Если A → α - правило вывода грамматики G, то δ (q, ε, A) = (q, α). 2. δ (q, a, a) = {(q, ε)} для всех a∈∑.

В). Определить последовательность тактов МП-автомата для выделенной цепочки.

Пусть будет цепочка: (-i) * i

Тогда последовательность тактов будет выглядеть следующим образом:

(q_0, (-i)* i, S) ⊢
(q, (-i)*i, (F)*L) ⊢ (q, -i)*i, F)*L) ⊢
(q, -i)* i, -L)*L) ⊢ (q, i)* i, L)*L) ⊢
(q, i)*i, F)*L) ⊢ (q, i)*i, i)*L) ⊢
(q,)*i,)*L) ⊢(q, *i, *L) ⊢ (q, i, L) ⊢
(q, i, F) ⊢
(q, i, i) ⊢
(q, ε, ε).

2.2. А). Построить расширенный МП-автомат, используя алгоритм 3.9.

Построение расширенного РМП-автомата (Q, ∑, Γ, δ, q_0, z_0, F), используем алгоритм 3.9, получим РМП = ({q, r}, {i, *, -, (,)}, {i, *, -, (,), S, F, L}, δ, q, δ, {r}), где функция переходов δ определена следующим образом:

1. δ(q, a, ε) = {q, a} для всех a∈∑ = {i, +, *, (,)}.
2. δ(q, ε, (F)*L) = {(q, S)}
3. δ(q, ε, -L) = {(q, F)}
4. δ(q, ε, i) = {(q, F)}
5. δ(q, ε, F) = {(q, L)}
6. δ(q, ε, ⊥S) = {(r, ε)}

В). Определить последовательность тактов расширенного МП-автомата при анализе входной выделенной цепочки Р.

При анализе входной цепочки (-i) *--i расширенный РМП-автомат выполнит следующую последовательность тактов:

(q, (-i)*--i, ⊥) ⊢ (q, -i)*--i, ⊥( )
⊢ (q, i)*--i, ⊥(–)
⊢ (q,)*--i, ⊥(–i)
⊢ (q, *--i, ⊥(-i))
⊢ (q, *--i, ⊥(-F))
⊢ (q, *--i, ⊥(-L))
⊢ (q, *--i, ⊥(F))
⊢ (q, --i, ⊥(F)*)
⊢ (q, -i, ⊥(F)*-)
⊢ (q, i, ⊥ (F)*--)
⊢ (q, ε, ⊥ (F)*--i)
⊢ (q, ε, ⊥ (F)*--F)
⊢ (q, ε, ⊥ (F)*--L)
⊢ (q, ε, ⊥ (F)*-F)
⊢ (q, ε, ⊥ (F)*-L)
⊢ (q, ε, ⊥ (F)*F)
⊢ (q, ε, ⊥ (F)*L)
⊢ (q, ε, ⊥S)
⊢ (q, ε, ε)

3. Определить свойства построенных МП-автоматов.

МП-автомат и расширенный РМП-автоматы – детерминированные автоматы.

Лабораторная работа №9-11

Построить управляющую таблицу M для LL(k)-грамматики, написать правило вывода, определить является ли G грамматика LL(k)-грамматикой.

1. **Изучить раздел 3.4. Построение управляющей таблицы M для грамматики G = (T, V, P, S), работу алгоритма для определенной цепочки и определение LL(k)-грамматики.**
2. **Определить размеры управляющей таблица M. В соответствии с шагами алгоритм 3.10 построить управляющую таблицу M (см. раздел 3.3. "Практическая работа 7.")**

4. T = {i, *, -, (,)}, V = {S, F, L}, P = {S→ (F)* L, F→ - L, F→ i, L→F}

Построить управляющую таблицу для грамматики:

G = ({I, *, -, (,)}, {S, F, L}, {p1, p2, p3, p4}, S), где p1: S→ (F)* L, p2: F→ - L, p3: F→ i, p4: L→F

FIRST(A) – это множество первых терминальных символов, которыми начинаются цепочки, выводимые из нетерминала A∈V: FIRST(A) = {a∈T | A ⇒+i aβ, где β∈(T ∪ V)*}

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

FOLLOW(A) = {a∈T | S ⇒*i aAγ и aAFIRST(γ)}

Таблица содержит 9 строк, помеченных символами из множества (VT{⊥}) и 6 столбцов, помеченных символами из (T{ε})

1. Нетерминалу S соответствует правило вывода грамматики p1: S→ (F)* L, так как FIRST((F)* L) = {(}, то M(S, () = (F)*L, 1
2. Для нетерминала F есть два правила вывода p2 и p3: p2 имеет FIRST(-L) = {-},

M(F,-) = -L, 2

p3 имеет простую правую часть, вычислим множество символов, следующих за нетерминалом F в сентенциальных формах. Построив левый вывод:

$S \Rightarrow (F)*L$

FOLLOW(F) = {), *}

Алгоритм 3.10. Построение управляющей таблицы M для LL(1) – грамматики.

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

Выход: Управляющая таблица M для грамматики G.

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

1. Если A → β – правило вывода грамматики с номером i, то M(A, a) = (β, i)

для всех a ≠ ε, принадлежащих множеству FIRST(β). Если εFIRST(β), то M(A, b) = (β, i) для всех bFOLLOW(A).

2. M(a, a) = ВЫБРОС для всех aT.

3. M(A, ε) = ДОПУСК.

4. В остальных случаях M(X, a) = ОШИБКА для X(V T {A}) и aT{ε}

	i	*	-	(	)	ε
S				(F)*L, 1		
F	i,3		-L, 2			
L	F, 4		F, 4			
I	ВЫБРОС					
*		ВЫБРОС				
-			ВЫБРОС			
(				ВЫБРОС		
)					ВЫБРОС	
⊥						ДОПУСК

Правило грамматики	Множество	Значение M
p1: S→ (F)* L	FIRST(S) = {(}	M(S, ( ) = (F)*L, 1
p2: F→ - L	FIRST(L) = {-, i}	M(F, -) = -L, 2
p3: F→ i	FIRST(F) = {-, i}	M(F, i) = i, 3
p4: L→F	FOLLOW(F) = {), *}	M(L, -) = F, 4

		M(L, i) = i, 3

3. Выделить цепочку, принадлежащую языку, порождаемому грамматикой G и написать правила вывода для цепочки.

Рассмотрим работу алгоритма для цепочки (i)*-i

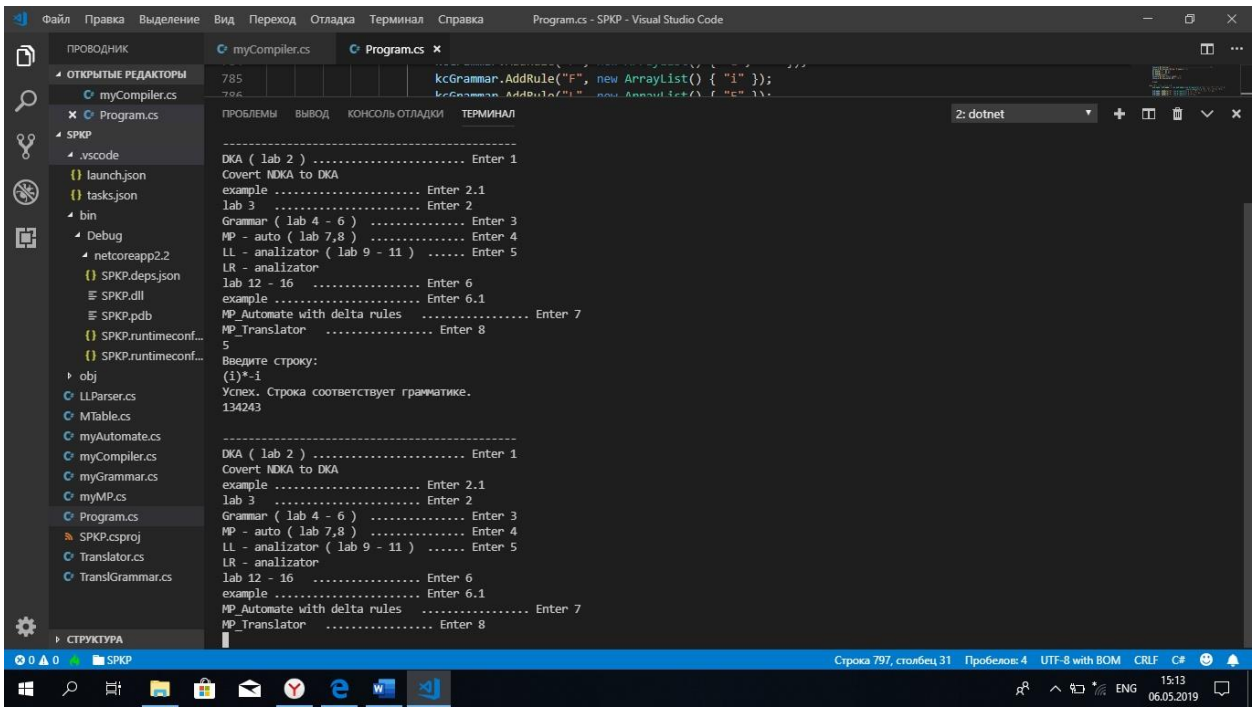
Шаг 1. Алгоритм находится в начальной конфигурации ((i)*-i, S$^{\perp}$, ε)

Значение управляющей таблицы M(S, () = (F)*L, 1, при этом выполняются следующие действия:

- Заменить верхний символ магазина S цепочкой (F)*L;
- Не сдвигать читающую головку
- На выходную ленту поместить номер использованного правила 1

Шаг 2. Получаем следующие конфигурации:

Текущая конфигурация	Значение M
((i)*-i, (F)*L⊥, 1) ⊢ (i)*-i, F)*L⊥, 1) ⊢ (i)*-i, i)*L⊥, 13) ⊢ ()*-i,)*L⊥, 13) ⊢ (*-i, *L⊥, 13) ⊢ (-i, L⊥, 13) ⊢ (-i, F⊥, 134) ⊢ (-i, -L⊥, 1342) ⊢ (i, L⊥, 1342) ⊢ (i, F⊥, 13424) ⊢ (i, i⊥, 134243) ⊢ (ε, ⊥, 134243)	M((,() = ВЫБРОС M(F, i) = i, 3 M(i, i) = ВЫБРОС M(),)) = ВЫБРОС M(*, *) = ВЫБРОС M(L, -) = F, 4 M(F, -) = -L, 2 M(-, -) = ВЫБРОС M(L, i) = F, 4 M(F, i) = (F, 3) M(i, i) = ВЫБРОС ДОПУСК



Вывод программы соответствует аналитическому выводу.

Листинг программы

```
case "5": // LL Разбор
myGrammar exemple = new myGrammar(new ArrayList() { "i", "(", ")", "*", "-" },
new ArrayList() { "S", "F", "L"}, "S");
exemple.AddRule("S", new ArrayList() { "(", "F", ")", "*", "L" });
exemple.AddRule("F", new ArrayList() { "-", "L" });
exemple.AddRule("F", new ArrayList() { "i" });
exemple.AddRule("L", new ArrayList() { "F" });
LLParser parser = new LLParser(exemple);
Console.WriteLine("Введите строку: ");
if (parser.Parse(Console.ReadLine()))
{
Console.WriteLine("Успех. Строка соответствует грамматике.");
Console.WriteLine(parser.OutputConfigure);
}
else
{
Console.WriteLine("Не успех. Строка не соответствует грамматике.");
}
break;
```

Лабораторные работы 12-15

Лабораторная работа №12

(Для LR(k) -грамматики построить таблицу кодировки символов. Определить функции перехода g(X) и функцию переноса-свертки f(u))

G = {i, * , :,(,)}, V={S, F, L}, P = {S→(F: L), S→ F, F→ L*, F→ i, L→F}

Правила моей грамматики:

1)S→(F:L)
2)S→L*
3)S→i
4)L→L*
5)L→i
6)F→L*
7)F→i

Построение таблицы кодировки

Символ грамматики	*Кодировка*	*Значение*
S	S_1	$\perp S$
F	F_1	$(F$
L	L_1 L_2	$(F:L$ L
i	i_1	i
*	$*_1$	$L*$
:	$:_1$	$(F:$
(	$(_1$	$($
)	$)_1$	(F:L)

Определение функции перехода g(x)

	S	**F**	**L**	**i**	*****	**:**	**(**	**)**
S_1								
F_1						$:_1$		
L_1								$)_1$
L_2					$*_1$			
i_1								
$*_1$								
$:_1$			L_1	i_1				
$(_1$		F_1		i_1				

$)_1$								
⊥	S_1		L_2	i_1			$(_1$	

Определение функции переноса-свертки f(u)

	(	**i**	*****	**:**	**)**	⊥
S_1	П	П	П	П	П	ДОП
F_1	П	П	П	П	П	
L_1	П	П	П	П	П	
L_2	П	П	П	П	П	
i_1	П	П	С(5)	С(7)	С(5)	С(3)
$*_1$	П	П	П	С(6)	С(4)	С(2)
$:_1$	П	П	П	П	П	
$(_1$	П	П	П	П	П	
$)_1$	П	П	П	П	П	С(1)
⊥	П	П	П	П	П	

Лабораторная 13

Определить функцию действий f(U)

Функцию действий определим так :

Для всех закодированых символов , у которых нет суффиксов , используем операцию СВЕРТКА(i) где i - номер продукции , преффиксом правой части которой является кодируемый символ .

Для всех других - определим перенос в магазин . Переносимый символ определяем из функции переходов

	i	(	)	^	&	⊥
S2	П	П	П	П	П	
F1	П	П	П	П	П	
L1	C(1)	C(1)	C(1)	C(1)	C(1)	C(1)
L3	C(3)	C(3)	C(3)	C(3)	C(3)	C(3)
L5	C(5)	C(5)	C(5)	C(5)	C(5)	C(5)
i4	C(4)	C(4)	C(4)	C(4)	C(4)	C(4)
i6	C(6)	C(6)	C(6)	C(6)	C(6)	C(6)
(2	П	П	П	П	П	
)2	C(2)	C(2)	C(2)	C(2)	C(2)	C(2)
^1	П	П	П	П	П	
&3	П	П	П	П	П	
&5	П	П	П	П	П	
⊥	П	П	П	П	П	
S0	П	П	П	П	П	ДОПУСК

Лабораторная 14

построить управляющую таблицу

В итоге получим следующую управляющую таблицу .

	i	(	)	^	&	⊥	S	F	L	i	(	)	^	&
S2	П	П	П	П	П						(2	)2		
F1	П	П	П	П	П								^1	
L1	C(1)	C(1)	C(1)	C(1)	C(1)	C(1)								
L3	C(3)	C(3)	C(3)	C(3)	C(3)	C(3)								
L5	C(5)	C(5)	C(5)	C(5)	C(5)	C(5)								
i4	C(4)	C(4)	C(4)	C(4)	C(4)	C(4)								
i6	C(6)	C(6)	C(6)	C(6)	C(6)	C(6)								
(2	П	П	П	П	П		S2	F1		i4	(2			&3
)2	C(2)	C(2)	C(2)	C(2)	C(2)	C(2)								
^1	П	П	П	П	П				L1	i6				&5
&3	П	П	П	П	П				L3	i4				
&5	П	П	П	П	П				L5	i6				
⊥	П	П	П	П	П			F1	L3		(2			&3
S0	П	П	П	П	П	ДОПУСК								

Породим цепочку (&i^&i):

S⇒(S)⇒(F^L)⇒(&L^L)⇒(&i^L)⇒(&i^&L)⇒(&i^&i)

Проверим принадлежность цепочку языку с помощью данного анализатора :

(&i^&i)

(⌕,(&i^&i)⌕,e) □ (⌕(2,&i^&i)⌕,e) □

(⌕(2&3,i^&i)⌕,e) □ (⌕(2&3i6,^&i)⌕,e) □

(⌕(2&3L3,^&i)⌕,6) □ (⌕(2F1,^&i)⌕,63) □

(⌕(2F1^1,&i)⌕,63) □ (⌕(2F1^1&5,i)⌕,63) □

(⌕(2F1^1&5i6,)⌕,63) □ (⌕(2F1^1&5L5,)⌕,636) □

□(⌕(2F1^1L1,)v,6365) □ (⌕(2S2,)⌕,63651) □

(⌕(2S2)2,⌕,63651) □ (⌕S0,⌕,636512) □ ДОПУСК

Лабораторная 15

построить LR анализотор методом грамматических вхождений

Сначала требуется составить дополненную грамматику :

T = {i, & , ^,(,)},
V = {S',S, F, L},
P = {

1. S'→ S,
2. S→ F^L,
3. S→ (S),
4. F→ &L,
5. F→ i,
6. L→&L.
7. L→i

}

Начнем строить множество φ с первого правила :

A_0 = CLOSURE(S'→S) = {

S'→•S

S→•F^L

S→•(S)

F→•&L

F→•i

L→•&L

L→•i

}

Далее составляем множества из $\mathbf{A_0}$:

A_1=GOTO(A_0,S) = GOTO({S'→·S},S) = CLOSURE({S'→S·})={

S'→S· }

A_2=GOTO(A_0,F) = GOTO({S→·F^L},F) = CLOSURE({S→F·^L}) = {

S→F·^L }

A_3=GOTO(A_0,() = GOTO({S→·(S)},() = CLOSURE({S→(·S)}) = {

S→(·S)

S→·(S)

S→·F^L

F→·&L

F→·i

}

A_4=GOTO(A_0,&) = GOTO({F→·&L,L→·&L},&) = CLOSURE({F→·&L,L→·&L}) = {

F→&·L

L→&·L

L→·&L

L→·i

}

A_5=GOTO(A_0,i) = CLOSURE({L→i·,F→i·}) = {

L→i·,

F→i·

}

Далее идем по множеству $\mathbf{A_2}$, так как в множестве A_1 нет ситуаций , в которых точка стоит не в конце

A_6=GOTO(A_2,^) = CLOSURE({S→F^·L}) = {

S→F^·L,

L→·&L

L→·i

}

В множестве A_2 ситуации закончились , переходим в $\mathbf{A_3}$

A_7=GOTO(A_3,S) = CLOSURE({S→(S·)}) = {

S→(S·)

}

A_8=GOTO(A_3,&) = CLOSURE({F→&·L}) = {

F→&·L

L→·&L

L→·i

}

A_9=GOTO(A_3,i) = CLOSURE({F→·i}) = {

F→i·

}

GOTO(A_3,() = CLOSURE({S→(·S)}) = A_3

GOTO(A_3,F)=A_2

Переходим к множеству $\mathbf{A_4}$

A_{10}=GOTO(A_4,L) = CLOSURE({F→&L·,L→&L·)}) = {

F→&L·,

L→&L·

}

A_{11}=GOTO(A_4,i) = CLOSURE({L→i·)}) = {

L→i·

}

переходим к **A_6**

A_{12}=GOTO(A_6,L) = CLOSURE({S→F^L·}) = {

S→F^L·

}

GOTO(A_6,&) = A_8

GOTO(A_6,i) = A_{11}

Переходим к **A_7**

A_{13}=GOTO(A_7,)) = {

S→(S)·

}

Переходим к **A_8**

A_{14}=GOTO(A_8,L) = {

F→&L·

}

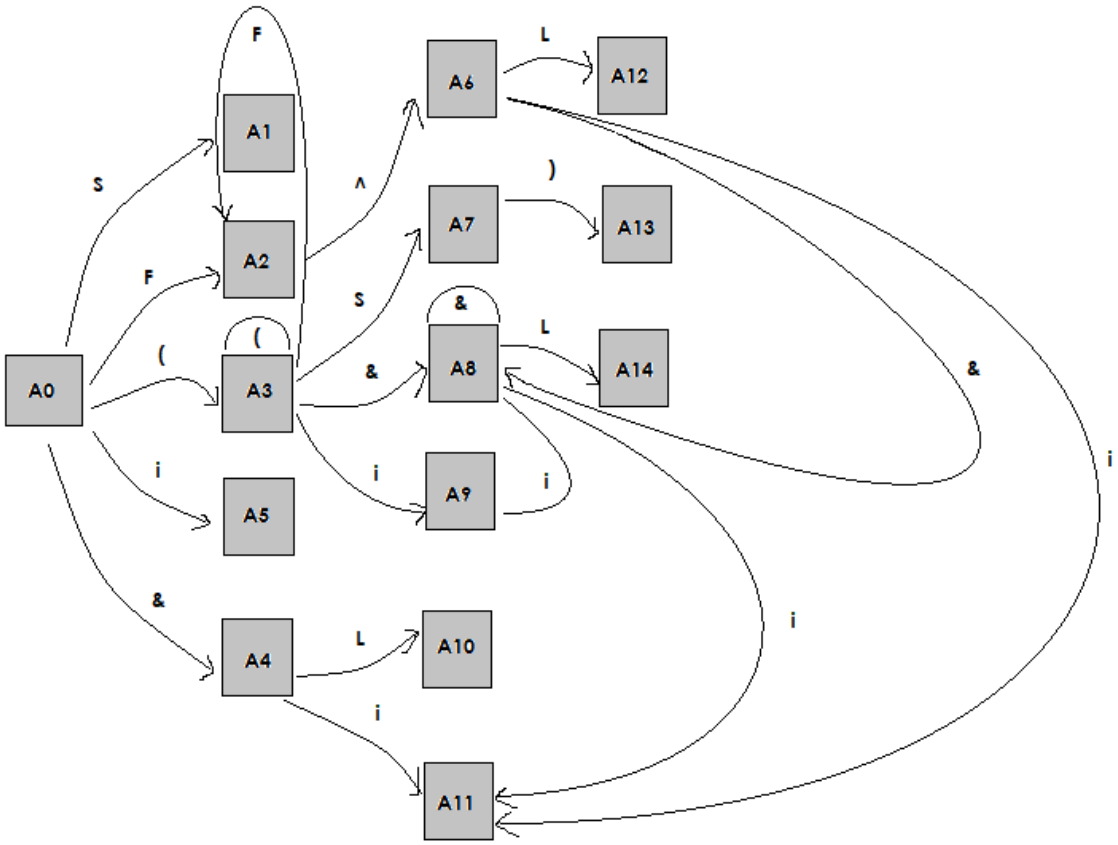
GOTO(A_8,&) = A_8

GOTO(A_8,i) = A_{11}

Для всех других множеств ничего нового не получим , поэтому построение множества φ заканчиваем .

В итоге получаем окончательную систему LR(0) - множеств φ：

Построим диаграмму переходов :



Построим управляющую таблицу :

	S	F	L	i	^	&	(	)	e
A0	A1	A2		A5		A4	A3		
A1									допуск
A2					A6				
A3	A7	A2		A9		A8	A3		
A4			A10	A11					
A5					r(5) A0				
A6			A12	A11		A8			
A7								A13	
A8			A14	A11		A8			
A9					r(5) A3				
A10					r(4) A0				
A11					r(7) A4|A8			r(7) A6|A8	
A12								r(2) A3	r(2) A0
A13									r(3) A0
A14					r(4) A3			r(6) A6	

Проанализируем строку : (i^&i)

(⌕,(i^&i),e)A0 (⌕(,i^&i) ,e)A3

(⌕(i ,^&i),e)A9 (⌕(F,^&i),5)A3

(⌕(F,^&i),5)A2 (⌕(F^,&i),5)A6

(⌕(F^&,i),5)A8 (⌕(F^&i,),5)A11

(⌕(F^&L,),57)A8 (⌕(F^&L,),57)A14

(⌕(F^L,),576)A6 (⌕(F^L,),576)A12

(⌕(S,),5762)A3 (⌕(S,),5762)A7

(⌕(S),e,5762)A13 (⌕S,e,57623)A0 (⌕S',e,57623)A1 допуск

Результат работы программы :

Создана последовательность C:

I0 { П .S,$; S .F^L,$; S .(S),$; F .&L,^; F .i,^ }

I1 { F &.L,^; L .&L,^; L .i,^ }

I2 { F i.,^ }

I3 { S (.S),$; S .F^L,); S .(S),); F .&L,^; F .i,^ }

I4 { П S.,$ }

I5 { S F.^L,$ }

I6 { F &L.,^ }

I7 { S F^.L,$; L .&L,$; L .i,$ }

I8 { L &.L,^; L .&L,^; L .i,^ }

I9 { L i.,^ }

I10 { S (.S),); S .F^L,); S .(S),); F .&L,^; F .i,^ }

I11 { S (S.),$ }

I12 { S F.^L,) }

I13 { S F^L.,$ }

I14 { S F^.L,); L .&L,); L .i,) }

I15 { L &.L,$; L .&L,$; L .i,$ }

I16 { L i.,$ }

I17 { S (S).,$ }

I18 { S (S.),) }

I19 { L &L.,^ }

I20 { L &.L,); L .&L,); L .i,) }

I21 { L i.,) }

I22 { S (S).,) }

I23 { S F^L.,) }

I24 { L &L.,$ }

I25 { L &L.,) }

Создана ACTION таблица

ACTION[14, i] = s 21

ACTION[7, &] = s 15

ACTION[3, i] = s 2

ACTION[2, ^] = r F i

ACTION[1, i] = s 9

ACTION[0, &] = s 1

ACTION[7, i] = s 16

ACTION[6, ^] = r F &L

ACTION[5, ^] = s 7

ACTION[4, $] = a

ACTION[11,)] = s 17

ACTION[10, &] = s 1

ACTION[9, ^] = r L i

ACTION[8, &] = s 8

ACTION[15, &] = s 15

ACTION[14, &] = s 20

ACTION[13, $] = r S F^L

ACTION[12, ^] = s 14

ACTION[19, ^] = r L &L

ACTION[1, &] = s 8

ACTION[17, $] = r S (S)

ACTION[16, $] = r L i

ACTION[23,)] = r S F^L

ACTION[22,)] = r S (S)

ACTION[21,)] = r L i

ACTION[20, &] = s 20

ACTION[25,)] = r L &L

ACTION[10, i] = s 2

ACTION[0, i] = s 2

ACTION[10, (] = s 10

ACTION[20, i] = s 21

ACTION[18,)] = s 22

ACTION[8, i] = s 9

ACTION[0, (] = s 3

ACTION[24, $] = r L &L

ACTION[3, &] = s 1

ACTION[15, i] = s 16

ACTION[3, (] = s 10

Создана GOTO таблица

GOTO[8, L] = 19

GOTO[0, S] = 4

GOTO[10, S] = 18

GOTO[10, F] = 12

GOTO[15, L] = 24

GOTO[14, L] = 23

GOTO[20, L] = 25

GOTO[3, S] = 11

GOTO[1, L] = 6

```
GOTO[0, F] = 5

GOTO[7, L] = 13

GOTO[3, F] = 12

Введите строку:

(i^&i)

Введена строка: (i^&i)$

Процесс вывода:

F->i

L->i

L->&L

S->F^L

S->(S)

Строка допущена
```