**Приложение A. Порядок выполнения лабораторных работ**

Практические занятия по курсу состоят из тематических работ, включающих одну или несколько лабораторных работ.

# **Практическая работа №1 лаб.1-3**

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 С# (Лаб 2.).

5.1. Создать проект – консольное приложение:

5.2. Используйте следующие фрагменты программ для реализации КА, распознающего заданный язык.

**Пример 1.** Создайте проекта простого консольного приложения.

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

namespace ConsoleApplication {

class Program {

static void Main(string[] args) {

ArrayList Q = new ArrayList();

string str;

Console.WriteLine("Hello");

str = Console.ReadLine();

Console.WriteLine("is= "+str);

Console.ReadLine();

}

}

}

**Пример 2.** Разбор входной строки по символам.

**Пример 3.** Проектирование и реализация правил.

Delta delta = null;

// 1. transition

delta = new Delta("S0", "0", new ArrayList() {"A","qf"});

DeltaList.Add(delta);

// 2. transition

delta = new Delta("A", "1", new ArrayList() { "B"});

DeltaList.Add(delta);

// 3. transition

delta = new Delta("B", "0", new ArrayList() { "A","qf"});

DeltaList.Add(delta);

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; } }

// модель правила

// delta( A, 1) = {qf}

// LeftNoTerm LeftTerm Right

public Delta(string LeftNoTerm, string LeftTerm, ArrayList Right) {

this.LeftNoTerm = LeftNoTerm;

this.LeftTerm = LeftTerm;

this.Right = Right;

}

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

**Пример 4.** Определение автомата как объекта и тестирование.

class Automate { // NDKA (Q,Sigma,deltaList,q0,F)

ArrayList Q = null; // множество вcех состояний

ArrayList Sigma = null; // конечный алфавит входных символов

ArrayList DeltaList = new ArrayList(); // множество вcех правил

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; } }

public Automate() {

this.Q = new ArrayList();

this.Sigma = new ArrayList();

this.DeltaList = new ArrayList();

this.F = new ArrayList();

}

public Automate(string aname) {

System.Console.WriteLine(aname);

// сделать диалог, инициализирующий NDKA

// альтернативные состояния переходов хранить в массиве см. Test

//init();

Test();

}

public void Test(){ // задание правил для тестирования

Q = new ArrayList() { "S0", "A", "B", "qf" }; // "C" для отладки

Sigma = new ArrayList() { "0", "1"};

q0 = "S0";

F = new ArrayList() { "qf" };

... задать правила

} // end test

**Пример 5.**  Считывание символа и выбор правила**:**

foreach (Delta d in this.DeltaList) {

if (d.leftNoTerm == q && d.leftTerm == chain.Substring(i,1) ) {

…

}

…

}

6.Определить свойства КА. Реализовать алгоритм преобразования НДКА в ДКА (Лаб 3.).

**Пример 6.** Построение побитового кода булеана для множества мощности n:

Для построения Булеана вначале строим все побитовые комбинации от 0 до 2n – 1, где n – мощность множества. Затем побитовая комбинация преобразуется в символы. Например, бинарная последовательность для множества для множества из символов {SAB}, n = 3 приведена в таблице. "1" означает, что символ в подмножестве, “0” – его отсутствие:

**Пример 7.** Построение по побитовому коду дельта правила:

**Реализация алгоритма преобразования на С#:**

// NDKA (Q,S,Delta,q0,F)

namespace NDKA2DKA {

class Program {

static void Main(string[] args) {

Automate NDKA = new Automate("NDKA");

NDKA.Debug();

Converter converter = new Converter();

Automate DKA = converter.convert(NDKA);

DKA.Debug();

DKA.recognize(DKA.q0, "01010", 0);

}

}

//

class Converter {

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 Converter() {}

public Automate convert(Automate NDKA) {

// инициализировать данные для каждого вызова convert

this.DKA = new Automate();

this.deltaListAll = new ArrayList();

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 yже 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

*7. Оформить работу согласно указанным шагам.*

**Практическая работа №2 (4-6 лаб.)**

Привести заданную КС-грамматику G = (T, V, P, S) к *приведенной* форме.

*1. Изучить алгоритмы приведения* КС-грамматик к *приведенной* форме. (см. раздел 3.4.)*.* Ответьте на вопрос, какие КС-грамматики называются грамматиками в *приведенной* форме. Лаб. 4 А,В. Лаб. 5 С,Д., Лаб. 6 F,E.

*2. Использовать алгоритмы преобразования* КС-грамматик*.*.

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

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

C). Устранить из KС грамматики G цепные правила, применить алгоритм 3.6.

D).Устранить левую рекурсию в заданной КС-грамматике G1, порождающей скобочные арифметические выражения. Применить алгоритм 3.7. к грамматике G.

F). Определить в какой форме (Грейбах, Хомского) находится КС-грамматика G’.

E). G’– *приведенная* КС-грамматика.

*3. Оформить работу согласно шагам.*

**Практическая работа №3 (7-8 лаб.)**

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. 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.)*.* Ответьте на вопрос, чем отличается МП-автомат P от расширенного МП-автомата.

*2. Выполнить построение согласно алгоритмам*.Смотрите пример и последовательность выполнения работы из раздела 3.4. Практическая работа 4.

2.1. A). Построить МП-автомат по КС-грамматике G, используя алгоритм 3.8. Для моделирования магазина ипользуйте Stack из библиотеки C#.

Stack <string> stack = new Stack <string> ();

stack.Push("c");

stack.Push("d");

Console.WriteLine(" simbol: " + stack.Pop());

B). Определить последовательность тактов МП-автомата для выделенной цепочк.

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

B). Определить последовательность тактов расширенного МП-автомата при анализе входной выделенной цепочки P.

*3. Определить свойства построенный МП-автоматов.*

*4. Оформить работу согласно шагам.*

**Практическая работа №4 (9-10 лаб.)**

Разработать контекстно-свободную грамматику по заданной строке (см. раздел 3.3.). Алгоритм разбора реализуется в виде процедур, каждой, из которой соответствует диаграмма.

*1. Повторить классификацию Хомского* (см. раздел 1.1.)*.* Ответьте на вопрос, какие грамматики называются контекстно-свободными. Какие есть виды контекстно-свободных грамматик.

*2.Спроектировать по заданной строке* контекстно-свободную грамматику и автомат с магазинной памятью*.* Используйте пример и последовательность выполнения работы из раздела 3.3. Обратите внимание, что в качестве неявного стека могут выступать: рекурсивная процедура и древовидная структура объектов. Лаб. 9 1-4. Лаб. 10 5-7.

1. Спроектировать контекстно-свободную грамматику.

2. Записать вывод заданной строки по грамматики.

3. Определить свойства грамматики.

4.Устранить левую рекурсию и записать вывод заданной строки по грамматики.

5.Оптимизировать грамматику метасимволами {…} и записать вывод заданной строки по грамматики.

6.Составить синтаксический граф для грамматики.

7. Преобразовать граф в программу.

#include "stdafx.h"

#include <iostream>

using namespace std;

class State { // абстрактный класс с чисто виртуальной функцией

public:

virtual void parse(char c)=0;

};

//подкласс для объекта с состоянием правильного разбора

class OK:public State {

public:

OK (){}

virtual void parse(char c){cout << "OK" << endl;}

};

//подкласс для объекта с состоянием не правильного разбора

class ERROR:public State {

public:

ERROR(){}

virtual void parse(char c){cout << "ERROR string"<< endl;}

};

//класс для автомата агрегация по ссылке объектов классов OK, ERROR

class Automate {

public:

static State\* state; // полиморфная переменная

static ERROR\* error;

static OK\* ok;

Automate(string String){

i=0;

this->String = String;

}

char getNextChar(){// получить следующий символ из строки

if (i<(int)String.length()){char ch=String[i];i++;return ch;}

else {return ' ';}

//при окончании строки возвращается пробел

}

virtual void parse(){ // начать разбор

while ( (state !=error)&&(state !=ok) ){

state->parse(getNextChar());

}

state->parse(getNextChar());// принцип подстановки

}

private:

string String;

int i;

}; // end class

// присвоение начальных значений переменным автомата

State\* Automate::state = NULL;

ERROR\* Automate::error = new ERROR;

OK\* Automate::ok = new OK;

// определение подкласса для объекта автомата, анализирующего

// контекстно-свободную грамматику

class AutomateCF: public Automate {

public:

AutomateCF(string String):Automate(String) {c =' ';}

virtual void parse(){// замещение функции parse в объекте

// класса Automate

c = getNextChar(); // вызов функции класса автомат

E(); // вызов метода

state->parse(getNextChar());

}

void T (){ // реализация функции по диаграмме

cout << "step1 T="<< c <<endl;

if ( (c == 'a'||c == 'b'||c == 'c')){

c = getNextChar();

cout << "step2 T="<< c <<endl;

}

if (c == ')') {c=getNextChar();}

else if (c == '(') {c=getNextChar(); E();}

}

void E (){// реализация функции по диаграмме

cout << "step0 E="<<c<<endl;

T(); // вызов функции

cout << "step1 E="<<c<<endl;

while ( (c == '+'||c == '-') ){

c = getNextChar();T(); // вызов функции

cout << "step2 E="<< c <<endl;

}

if (c == ' ') {state=ok;}

else {state=error;}

}

private:

char c;

};

// программа ввода строки с клавиатуры

string getString (){

//string String = "c+d-dkf-n";

string String = "a+(b–c)";

// string String = "a+b";

char c;

int N = (int)String.length();

cout << "Enter string "<<N<<" char "<< endl;

for (int i=0; i < N;i++){

cin>>c;

String[i] = c;

}

return String;

}

int main(){

//создание объекта автомат

Automate \* a = new AutomateCF(getString());

//инициализация начальных значений

a->state = Automate::ok;

a->parse(); //синтаксический анализ

return 0;

}

Обратите внимание, что в переменную “с”, записывается следующий символ, который доступен из функций T() и E(). В функциях указаны трассировочные шаги выполнения, которые необходимо сохранить в программе. Состояния OK и ERROR автомата используются для идентификации выполнения разбора.

*3. Оформить работу согласно шагам.*

**Практическая работа №5 (11-10 лаб.)**

Реализовать контекстно-свободную грамматику, полученную в работе 5 на основе, таблично-управляемой программы грамматического разбора. Изучить контекстно-зависимые грамматики (см. раздел 1.4.).

*1.Повторить классификацию Хомского* (см. раздел 1.1.)*.* Ответьте на вопрос, какие грамматики называются контекстно – зависимыми, неограниченными. Какие свойства контекстно – зависимых и неограниченных грамматик, в чем их отличие. Чем они отличаются от изученных раннее грамматик ?

*2.* Разработать объектно-ориентрованную реализацию для таблично-управляемой программы грамматического разбора.

1. Представить граф грамматики в виде структуры данных (см. раздел 3.3.).

2. Классифицировать типы вершин.

3. Выполнить подстановку графов и получить как можно меньшее число графов.

4. Последовательность вершин графа преобразовать в структуру узлов.

5. Реализовать программу разбора по структуре узлов.

В алгоритме разбора, результат каждого шага разбора выводится на экран, чтобы можно было видеть, как происходит разбор.

#include "stdafx.h"

#using <mscorlib.dll>

using namespace System;

using namespace std;

class State { // абстрактный класс с чисто виртуальной функцией

public:

virtual void parse(char c)=0;

};

// класс для объектов вершин, агрегация по указателю

class Node {

public:

Node (char c){ // задание символа вершинам

this->alt = NULL;

this->suc = NULL;

this->sym = NULL;

this->c = c;

}

// метод для соединения вершин

void link (Node \*alt, Node \*suc, Node \*sym){

this->alt = alt;

this->suc = suc;

this->sym = sym;

}

Node \* sym; // sym != NULL

Node \* suc; // terminal empty !=' ' && sym == NULL

Node \* alt;

char c; // empty=' ' && sym == NULL

};

//подкласс для объекта с состоянием правильного разбора

class OK:public State {

public:

OK (){}

virtual void parse(char c){cout << "OK" << endl;}

};

//подкласс для объекта с состоянием не правильного разбора

class ERROR:public State {

public:

ERROR(){}

virtual void parse(char c){cout << "ERROR string"<< endl;}

};

//класс для автомата агрегация по ссылке объектов классов OK,

// ERROR

class Automate {

public:

static State\* state; // полиморфная переменная

static ERROR\* error;

static OK\* ok;

Automate(string String){

i=0;

this->String = String;

}

char getNextChar(){// получить следующий символ из строки

if (i<(int)String.length()){char ch=String[i]; i++; return ch;}

else {return ' ';}

//при окончании строки возвращается пробел

}

virtual void parse(){// начать разбор

while ( (state !=error)&&(state !=ok) ){

state->parse(getNextChar()); // принцип подстановки

}

state->parse(getNextChar());

}

private:

string String;

int i;

}; // end class

// присвоение начальных значений переменным автомата

State\* Automate::state = NULL;

ERROR\* Automate::error = new ERROR;

OK\* Automate::ok = new OK;

// определение подкласса для объекта автомата, анализирующего

// контекстно-свободную грамматику

class AutomateCF: public Automate {

public:

AutomateCF(string String, Node &node):Automate(String) {

this ->node = &node; // передача начальной вершины

c =' ';

}

void parse(Node \* nd, bool b){

Node \*p = nd->alt; // Node head

do {

if (p->sym == NULL){ // терминал или пусто “empty”

if (p->c == c){

cout<< "step01 c=" << c<< ": p->c ="<<p->c<<endl;

b=true; c=getNextChar();

state=ok;

// конец строки

if (c == ' ') {cout<< "step012 c="<<endl; return;}

} else if (p->c ==' ') { // “empty”

cout<< "step02 if c=" << c<< ": p->c ="<<p->c<<endl;

b=true;

} else { b=false; state=error;

cout<< "step03 c=" << c<< ": p->c ="<<p->c<<endl;

}

}

else if (p->sym != NULL){ //- nil

cout<< "step04 c=" << c<< "p->sym ="<<p->sym<<endl;

parse(p->sym, b);

}

if (b) {

cout<< "step05 ok: next c=" << c<< endl;

p=p->suc;

}

else {p=p->alt;}

} while (p != NULL);

} // end parse

virtual void parse(){ // замещение функции parse в объекте

c = getNextChar();

parse(this->node, true);

state->parse(getNextChar());

}

private:

Node \* node;

char c;

};

// программа ввода строки с клавиатуры

string getString (){ // различные варианты для тестирования..

// string String = "c+d-dkf-n";

// string String = "a+(b-cba)-c";

// string String = "a+b";

// string String = "a+b-c-c+a";

// string String = "a+gfg+fgfg"; // error

string String = "a+(b-c)";

char c;

int N = (int)String.length();

cout << "Enter string "<<N<<" char "<< endl;

for (int i=0; i < N;i++){

cin>>c;

String[i] = c;

}

return String;

}

int main(){

// Построение графа

// 1. Объявление вершин

Node E = Node(' ');

Node braceL = Node('(');

Node nil = Node(' ');

Node braceR = Node(')');

Node c\_a = Node('a');

Node c\_b = Node('b');

Node c\_c = Node('c');

Node emptyR = Node(' ');

Node plus = Node('+');

Node minus = Node('-');

Node nil\_1 = Node(' ');

// 2. Соединение вершин в дерево методом link

// void link (Node \* alt, Node \* suc, Node \* sym)

E.link(&braceL,NULL,NULL);

braceL.link(&c\_a,&nil,NULL);

nil.link(NULL,&braceR,&E);

braceR.link(NULL,&plus,NULL);

c\_a.link(&c\_b,NULL,NULL);

c\_b.link(&c\_c,NULL,NULL);

c\_c.link(&emptyR,NULL,NULL);

emptyR.link(NULL,&plus,NULL);

plus.link(&minus,&nil\_1,NULL);

minus.link(NULL,&nil\_1,NULL);

nil\_1.link(NULL,NULL,&E);

//создание объекта автомат

Automate \* a = new AutomateCF(getString(),E);

//инициализация начальных значений

a->state = Automate::ok;

a->parse(); //синтаксический анализ

return 0;

}

*6. Оформить работу согласно шагам.*

**Практическая работа №6 (12-13 лаб.).**

Построить управляющую таблицу M для LL(k)-грамматики, написать правило вывода, опредилить является ли G грамматика *сильно* LL(k)-грамматикой (см. раздел 3.4.).

1. Изучить раздел 3.4. Построение управляющей таблицы М для грамматики G =(T, V, P, S), работу алгоритма для определенной цепочки и определение *сильно* LL(k)-грамматики.

Пример построения управляющей таблицы.

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

namespace lab {

class Program {

static void Main(string[] args) {

MTable mTable = new MTable();

// 1 row

Row row = new Row();

row.addItem("TE',1");

row.addItem("TE',1");

row.addItem("");

row.addItem("");

row.addItem("");

row.addItem("");

mTable.setRow(row);

row = new Row();

row.addItem("");

row.addItem("");

row.addItem("E,3");

row.addItem("+TE',2");

row.addItem("");

row.addItem("E,3");

mTable.setRow(row);

// row.DebugRow();

mTable.DebugTable();

}

}

}//конец для namespace

class MTable {

ArrayList M = new ArrayList(); //

public void setRow(Row row) { M.Add(row); }

public void DebugTable() {

//foreach (row r in M) {

for(int i=0;i<M.Count;i++){

((Row) M[i]).DebugRow();

}

//Console.WriteLine(r);

}

}

class Row { // строка таблицы любого размера из string - это колонка

ArrayList row = new ArrayList(); //

public void addItem (string item) {

row.Add(item);

}

// public ArrayList getRow() { return row; }

public void DebugRow() {

string[] str = null;

foreach (string r in row) {

Console.WriteLine(r);

str = r.Split(',');

foreach (string l in str) {

Console.WriteLine(l);

}

}

}

}

2. Определить размеры управляющей таблица M. В соответствии с шагами алгоритм 3.10 построить управляющую таблицу M (см. раздел 3.3. “Практическая работа 7.”)

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

4. Для LL(k)-грамматики G определить является ли она *сильно* LL(k)-грамматикой.

*5. Оформить работу согласно шагам.*

**Практическая работа №7 (14-15 лаб.)**

Построить управляющую таблицу M для LR(k)-грамматики, написать правило вывода выделенной строки (см. раздел 3.4.). Описать работу алгоритма LR(k) анализатора.

1. Изучить раздел 3.4. Построение управляющей таблицы М для LR(k)-грамматики G =(T, V, P, S).

2. В соответствии с шагами алгоритм 3.12 построить управляющую таблицу M (см. раздел 1.4. “Практическая работа 8.”)

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

4. Для выделенной цепочки показать работу LR(k)-анализатора в соответствии с шагами алгоритм 3.11.

*5. Оформить работу согласно указанным пунктам и используемым шагам алгоритмов.*

**Практическая работа №8 (16 лаб.).**

Применить алгоритм типа “перенос-свертка” для заданной грамматики G= (T, V, P, S). Описать работу алгоритма.

1. Изучить алгоритм типа “перенос-свертка”

2. В соответствии с шагами алгоритм 3.13 по шагам рассмотреть работу алгоритма (см. раздел 3.4. “Практическая работа 8.”)

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

4. Для выделенной цепочки показать работу LR(k)-анализатора в соответствии с шагами алгоритм 3.13 .

*5. Оформить работу согласно указанным пунктам и используемым шагам алгоритма.*

**Приложение B. Задания к практическим работам**

Для практической работы 1 (1-3) определены варианты автоматных языков:

1. L={0ω1+(01)\* ω1{0,1}\*}

2. L={01-(10)\*+ω101 ω1{0,1}\*}

3. L={0(00)\*+01ω1 ω1(0,1,2)\*}

4. L={ω1ω21ω1{1,0}+, ω2{1,0}+}

5. L={1ω11-(00)\* ω1{1,0}\*}

6. L={ω1-0ω2-0+ω3ω1{0,1}+, ω2{0,1}+, ω3{0,1}+}

7. L={(0+1)(01)\*+ω1 ω1{0,1}+}

8. L={(0+1)\*ω1ω2 ω1{0,1,2}+, ω2{0,1}+}

9. L={(01)\*-1-(01)\*+ω1 ω1{0,1}+}

10. L={1(01)\*0-1+ω1 ω1{0,1}+}

11. L={(0+1)+ω1+(01)\*0ω1{0,1}+}

12. L={0(000)\*(0+1)ω1 ω1{0,1}\*}

13. L={1(01)\*(01)ω1 ω1{0,1}\*}

14. L={00ω1+(1)\*ω1{0,1}\*}

15. L={0(01)\*+1ω10ω1{1,0}+}

16. L={1ω11ω21ω1{0,1}+, ω2{0,1}+}

17. L={011ω11(0)\* ω1 (0,1)+}

18. L={ω10-ω2 ω1{0,1}+, ω2{0,1}+}

19. L={ω1(1)\*0ω2 ω1{1,2}\*, ω2{0,1}\*}

20. L={10+ω1(10)\*ω2 ω1{1,2}\*, ω2{0,1}\*}

21. L={10ω10-1ω2 ω1{0,1}+, ω2{1,2}\*}

22. L={1-1ω101ω2 ω1{1,2}+, ω2{1,2}\*}

Для практической работы 3-8

1. P={S → b, S → F, S→ cFB, A→ Ab, A→ c, B → cB, F→ Ca, C→ d, C→ ε }

2. P={S →b, S→ cAB, A→ Ab, A→ c, B → cB, C→ Ca, F→ d, F→ ε }

3. P={S → b, S → BA, A→ Ab, A→ c, B → cB, F→ Ca, C→ d, C→ ε }

4. P={S → cB, B → cB, B → cA, A→ Ab, C→ Ca, F→ d, F→ ε }

5. P={S → c, F → A, S→ cAB, A→ Ab, A→ c, B → cB, C→ Ca, C→ d, B→ ε }

6. P={S → cFCB, A→ ACb, A→ cC, B → cB, C→ Ca, F→ d, B→ ε }

7. P={S → b, S → CF, S→ cCB, A→ Ab, A→ c, B → cB, C→ Ca, F→ d, B→ ε}

8. P={S →SAB, A → SA, A→ BB, A→ bB, A→ cA, B → c, B→ ε}

9. P={S → b, S→ cAB, A→ Ab, A→ cA, B → cB, B→ ε}

10 P={S → bA, S → bA, A→ Ab, A→ ε , B → cB}

11. P={S → cB, B → cB, B → cA, A→ ACb, A→ ε, C→ Ca, C→ ε }

12. P={S → Sc, S→ cAB, S→ ε, A→ c, B → cB}

13. P={S → cFCB, A→ ACb, A→ cC, B → cB, , C→ Ca, F→ ε }

14. P={S → b, S → C, S→ cCB, A→ Ab, A→ c, B → cB, C→ Ca, C→ ε}

15. P={S →SAB, A → S, A→ B, A→ bB, A→ c, B → c, B→ ε }

16. P={S → b, S→ cAB, A→ Ab, A→ B, B → cB, B→ b, A→ ε }

17. P={S → bA, S → bA, A→ Ab, A→ S, B → cB, B→ c, A→ ε }

18. P={S → cB, B → cB, B → cA, A→ C, A→ aB, C→ Ca, C→ cf, A→ ε }

19. P={S → Sc, S→ cAB, S→ a, A→ B, B → cB, B → f, A→ ε }

20. P={S →cFCB, A→ ACb, A→ C, B → cB, , B → b, C→ Ca, F→ c, A→ ε }

21. P={S →b, S → C, S→ cCB, A→ Ab, A→ C, B → cB, C→ Ca, C→ b, A→ ε }

22. P={S →Bc, S → Ad, A→ Sa, A→ AbB, A→ c, B → Sc, B → b}

23 P={S → SFA, S → c, A→ FS, A→ Sa, B → SB, B→ b, F→ f}

24. P={S → SBb, B → Sa, B → cB, B→ Ac, A→ cSB, A→ a}

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

26. P={S → cB, S→ cAB, S→ c, A→ AbB, B → b, B → aA}

27. P={S → cFCB, A→ ACb, B → cB, , B → b, C→ Ca, , C→ c, F→ f }

28. P={S → AB, S → SC, A→ BB, A→ Ab, A→ a, B → b, C→ Ca, C→ b}

Для практической работы 3-8 задана КС-грамматика G =(T, V, P, S), где

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

2. T = {i, &, \*,(,)}, V = {S, F, L}, P = {S→ F & L, S→ (S), F→ \* L, F→ i, L→F}

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

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

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

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

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

8. T = {i, & , ^,(,)}, V = {S, F, L}, P = {S→ F^L, S→ (F), F→ &L, F→ i, L→F}

9. T = {i, & , ^,(,)}, V = {S, F, L}, P = {S→ F^L, S→ (S), F→ &L, F→ i, L→F} 10. T = {i, & , ^,(,)},V= {S, F, L}, P = {S→ (F^L), S→ (S), F→ &L, F→ i, L→F}

11. T = {i, & , ^,(,)},V={S, F, L}, P = {S→ (F^L), F→ &L, F→ i, L→F}

12. T = {i, & , ^,(,)},V={S, F, L}, P = {S→ &F^, S→ (L), F→ L, F→ i, L→F}

13. T = {i, \* , :,(,)}, V={S, F, L}, P = {S→ F: L, S→ (L), F→ L\*, F→ i, L→F}

14. T = {i, \* , :,(,)}, V={S, F, L}, P = {S→(F: L), F→ L\*, F→ i, L→F}

15. T = {i, \* , :,(,)}, V={S, F, L}, P = {S→(F: L), S→ F, F→ L\*, F→ i, L→F}

16. T = {i, \* , +,(,)}, V={S, F, L}, P = {S→(F+ L), F→ L\*, F→ i, L→F}

17. T = {i, \* , +,(,)}, V={S, F, L}, P = {S→(F+ L), F→ (L\*), F→ i, L→F}

18. T = {i, \* , +,(,)}, V={S, F, L}, P = {S→F+ L, F→ (L\*), F→ i, L→F}

19. T = {i, \* , +,(,)}, V={S, F, L}, P = {S→F+ L, S→(S), F→ L\*, F→ i, L→F}

20. T = {i, @ , &,(,)},V={S, F, L}, P = {S→F@L, S→(F&L), F→ i, L→F}

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

Для практической работы 6, правила G грамматики рассмотреть как правила LL(k) грамматики, а для работы 7 как правила LR(k).

Пример. Пусть задан регулярный язык L={0(10)nn0}.Построить автоматную грамматику G= (T,V,P,S0) для заданного языка L и привести пример вывода строки. Используя грамматику G, построить КА= (Q, , , q0, F) и привести пример конфигурации КА.

1. Построение грамматики. L={0(10)n n0}, то 0, 010, 01010 и т.д. - этот язык порожден регулярной грамматикой G= (T,V, P,S0), где T={0, 1}, V={A,B} , P={S00, S00A, A1B, B0, B0A}. Пример вывода цепочки

S0  0A  01B 010A 0101B 01010.

2. Построение КА. Воспользуемся утверждением 1, тогда

КА= ({S0, A, B, qf}, {0, 1}, , S0, qf )

(S0, 0) = {A, qf };

(A, 1) = {B};

(B, 0) = {A, qf };

Пример конфигурации КА: S001010 A1010 B010 A10 B0 qf