Нижегородский государственный технический университет им. Р.Е. Алексеева.

Институт радиоэлектроники и информационных технологий.

Кафедра «Прикладная информатика и математика».

Отчёт по лабораторной работе:

«Построение конечного автомата по регулярной грамматике»

Выполнил: студент 2 курса группы 23-ПМ-1 Гугин И.В.

Научный руководитель: Санников Н.А.

Нижний Новгород

2024

**Цель работы:** научиться строить конечные автоматы по заданной грамматике, построение конечного детерминированного автомата на основе недетерминированного.

**Задачи:**

1)Написать программу способную принять формальную грамматику, проверить ввод на ошибки и сохранить в память в удобной для работы форме.

2)Добавить программе способность считывания информации для работы из текстового файла.

2)Добавить к программе возможно создания конечного автомата.

3)Добавить программе возможность построения недетерминированного автомата.

4)Добавить программе возможность автоматически преобразовывать недетерминированный автомат в детерминированный.

5)Оформить вывод программы в качестве детерминированного конечного автомата.

6)Сделать проверку используя грамматику моего варианта(7):

G=({X,Y,Z,V,W},{0,1,x,y,z},P,E), P:

1)X->yY|xX, 2)Y->1V; 3)Z->0W|0Y; 4)V->xZ|xW|1; 5)W->1Y|0

**Теоретическая справка:**

**Определение:**

Детерминированным конечным автоматом (ДКА) называется пятерка объектов:

М = (Q,T, F, H, Z), где

Q- конечное множество состояний автомата;

T - конечное множество допустимых входных символов;

F - функция переходов, отображающая множество Q × Т во множество Q;

Н - конечное множество начальных состояний автомата;

Z - множество заключительных состояний автомата, Z **∈**Q

**Определение:**

Недетерминированным конечным автоматом (НКА) называется конечный автомат, в котором в качестве функции переходов используется отображение Q × Т во множество всех подмножеств множества состояний автомата P(Q), т.е. функция переходов неоднозначна, так как текущей паре (q,t) соответствует множество очередных состояний автомата q’**∈**P(Q).

**Способы представления функции переходов:**

**Командный способ.** Каждую команду КА записывают в форме F(q,t) = р, где q, р **∈**Q,t **∈**T.

**Табличный способ**. Строки таблицы переходов соответствуют входным символам автомата t **∈**T, а столбцы - состояниям Q. Ячейки таблицы заполняются новыми состояниями, соответствующими значению функции F(q, t). Неопределенным значениям функции переходов соответствуют пустые ячейки таблицы.

**Графический способ.** Строится диаграмма состояний автомата - неупорядоченный ориентированный помеченный граф. Вершины графа помечены именами состояний автомата. Дуга ведет из состояния q в состояние р и помечается списком всех символов t **∈**T, для которых F(q, t) = р. Вершина, соответствующая входному состоянию автомата, снабжается стрелкой. Заключительное состояние на графе обозначается двумя концентрическими окружностями.

**Построение КА по регулярной грамматике**

Вход: регулярная грамматика G = (Vt,Vn, P, S).

Выход: КА М= (Q,T, F, H, Z).

Шаг 1. Пополнить грамматику правилом A→aN, где A∈Vn. a∈Vt и N новый нетерминал, для каждого правила вида A → а, если в грамматике нет соответствующего ему правила A → аB, где В∈Vn.

Шаг 2. Начальный символ грамматики S принять за начальное состояние КА Н. Из нетерминалов образовать множество состояний автомата Q=Vn⋃{N}, а из терминалов множество символов входного алфавита Т=Ѵt.

Шаг 3. Каждое правило А→ aB преобразовать в функцию переходов F(A, а) = В, где A, B**∈**Vn,a∊Vt.

Шаг 4. Во множество заключительных состояний включить все вершины, помеченные символами В∈Vn из правил вида A → aВ, для которых имеются соответствующие правила A → а, где A, В∈Vn,a∈Vt.

Шаг 5. Если в грамматике имеется правило S→ e, где S - начальный символ грамматики, то поместить S во множество заключительных состояний.

Шаг 6. Если получен НКА, то преобразовать его в ДКА.

**Преобразование НКА в ДКА**

Вход: НКА М=(Q,T, F, H, Z).

Выход: ДКА M' = (Q', T, F', H, Z').

Шаг 1. Пометить первый столбец таблицы переходов М' ДКА начальным состоянием (множеством начальных состояний) НКА М.

Шаг 2. Заполняем очередной столбец таблицы переходов М', помеченный символами D, для этого определяем те состояния М, которые могут быть достигнуты из каждого символа строки при каждом входном символе x. Поместить каждое найденное множество R (в том числе пустое) в соответствующие позиции столбца D таблицы М’.

Шаг 3. Для каждого нового множества R (кроме пустого), полученного в столбце в таблицы D переходов М’, добавить новый столбец в таблицу, помеченный R.

Шаг 4. Если в таблице переходов КА М есть столбец с незаполненными позициями, то перейти к шагу 2.

Шаг 5. Во множество Z' ДКА М' включить каждое множество, помечающее столбец таблицы переходов М' и содержащее q ∈ Z НКА М.

Шаг 6. Составить таблицу новых обозначений множеств состояний и определить ДКА М' в этих обозначениях.

**Реализация:**

Программа написана на языке С++. Для хранения информации о переходах будем использовать 3 вектора. В первом хранится то, откуда был произведён переход, в 2 с помощью какого символа, в 3 куда. Например первый переход из Y в V с помощью 1, тогда в первый вектор запишем Y, в второй 1, в третий V. Если нам понадобиться доступ к этому переходу мы сможем обратиться по соответственному номеру. При запуске программы будет запрошен ввод ввода – ручной или из файла. Если выбран из файла, то будет произведён поиск необходимой информации, если её нет будет предложена создать файл. Если выбран ручной ввод то, будет запрошен список точек и символов перехода, для дальнейшего использования в проверках и выборе символов для новых точек при переходе к ДКА. Информация о точках и символах перехода так же храниться в двух соответствующих векторах. В процессе работы программы так же будет произведено множество проверок входных данных, при несоответствиях в зависимости от режима работы будет либо просто сообщение об ошибке, либо сообщение и завершение программы.

**Результат работы:**

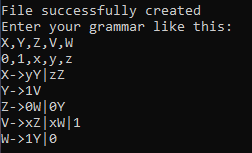
Выбор режима ввода:



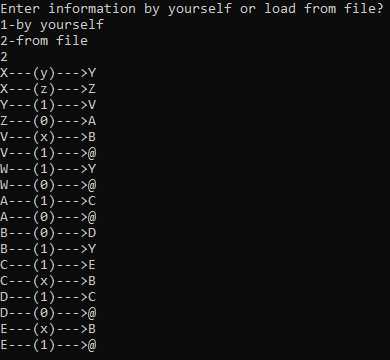
Необходимый файл не найден:



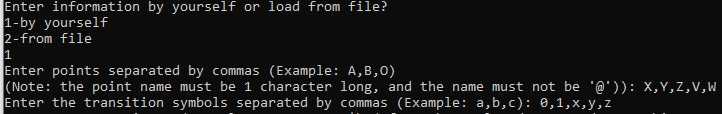
Файл создан:



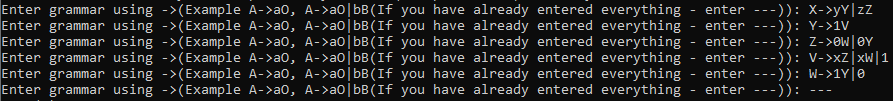
Вывод результата работы моего варианта при загрузке из файла:



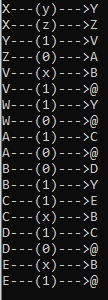
Ручной ввод точек и символов перехода:



Ручной ввод грамматики:



Вывод программы при ручном вводе:



**Сам код:**

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

using namespace std;

int **main**()

{

vector<char> from;

vector<char> to;

vector<char> with;

vector<char> points;

vector<char> step;

string g;

int ans;

string alph = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

char c;

char b;

char a;

char d;

int code;

int flags=0;

cout<<"Enter information by yourself or load from file?"<<endl;

cout<<"1-by yourself"<<endl;

cout<<"2-from file"<<endl;

cin>>code;

if(code==2){

ifstream in("gram.txt");

if(in.is\_open()==false){

cout<<"File not found. Would you like to create it?"<<endl;

cout<<"1-yes"<<endl;

cout<<"2-no"<<endl;

cin>>ans;

if(ans==1){

ofstream MyFile("gram.txt");

cout<<"File successfully created"<<endl;

cout<<"Enter your grammar like this:"<<endl;

cout<<"X,Y,Z,V,W"<<endl;

cout<<"0,1,x,y,z"<<endl;

cout<<"X->yY|zZ"<<endl;

cout<<"Y->1V"<<endl;

cout<<"Z->0W|0Y"<<endl;

cout<<"V->xZ|xW|1"<<endl;

cout<<"W->1Y|0"<<endl;

MyFile.close();

}

else{

cout<<"File error"<<endl;

return 0;

}

}

ifstream an("gram.txt");

if(an.is\_open()){

while (std::getline(in, *g*)){

if(flags==0){

for(int i=0;i<g.size();i++){

c=g[i];

if(c!=','){

points.push\_back(c);

}

flags=1;

}

continue;

}

if(flags==1){

for(int i=0;i<g.size();i++){

c=g[i];

if(c!=','){

step.push\_back(c);

}

}

flags=2;

continue;

}

int flag=0;

int flag2=0;

if(g[1]=='-' && g[2]=='>'){

flag2=1;

}

if(flag2==0){

cout<<"Reminder: use ->"<<endl;

return 0;

}

flag2=0;

for(int i =0;i<points.size();i++){

c=g[0];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

return 0;

}

flag2=0;

if(g.size()==4){

for(int i =0;i<points.size();i++){

c=g[4];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

return 0;

}

}

flag2=0;

if(g.size()==8){

for(int i =0;i<points.size();i++){

c=g[7];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

return 0;

}

}

flag2=0;

if(g.size()==11){

for(int i =0;i<points.size();i++){

c=g[10];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

return 0;

}

}

flag2=0;

for(int i =0;i<points.size();i++){

c=g[4];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

return 0;

}

flag2=0;

for(int i =0;i<step.size();i++){

c=g[3];

if(c==step[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Transition symbol error"<<endl;

return 0;

}

flag2=0;

if(g.size()>5){

for(int i =0;i<step.size();i++){

c=g[6];

if(c==step[i] && g[5]=='|') flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Transition symbol or '|' error"<<endl;

return 0;

}

}

flag2=0;

if(g.size()>8){

for(int i =0;i<step.size();i++){

c=g[9];

if(c==step[i] && g[5]=='|' && g[8]=='|') flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Transition symbol or '|' error"<<endl;

return 0;

}

}

flag2=0;

if(g.size()==5 || g.size()==4){

if(g[1]=='-' && g[2]=='>'){

c=g[0];;

from.push\_back(c);

c=g[3];

with.push\_back(c);

if(g.size()==4){

c='@';

to.push\_back(c);

}

else{

c=g[4];

to.push\_back(c);

}

}

flag=1;

}

if(g.size()==7 || g.size()==8){

if(g[1]=='-' && g[2]=='>' && g[5]=='|'){

c=g[0];

from.push\_back(c);

from.push\_back(c);

c=g[3];

with.push\_back(c);

c=g[6];

with.push\_back(c);

c=g[4];

to.push\_back(c);

if(g.size()==7){

c='@';

to.push\_back(c);

}

else{

c=g[7];

to.push\_back(c);

}

}

flag=1;

}

if(g.size()==10 || g.size()==11){

if(g[1]=='-' && g[2]=='>' && g[5]=='|' && g[8]=='|'){

c=g[0];

from.push\_back(c);

from.push\_back(c);

from.push\_back(c);

c=g[3];

with.push\_back(c);

c=g[6];

with.push\_back(c);

c=g[9];

with.push\_back(c);

c=g[4];

to.push\_back(c);

c=g[7];

to.push\_back(c);

if(g.size()==10){

c='@';

to.push\_back(c);

}

else{

c=g[10];

to.push\_back(c);

}

}

flag=1;

}

if(flag=0) cout << "Input error" << endl;

}

}

}

if(code==1){

cout << "Enter points separated by commas (Example: A,B,O)"<< endl << "(Note: the point name must be 1 character long, and the name must not be '@')): ";

cin>>g;

for(int i=0;i<g.size();i++){

c=g[i];

if(c!=','){

points.push\_back(c);

}

}

cout << "Enter the transition symbols separated by commas (Example: a,b,c): ";

cin>>g;

for(int i=0;i<g.size();i++){

c=g[i];

if(c!=','){

step.push\_back(c);

}

}

while(1){

cout << "Enter grammar using ->(Example A->aO, A->aO|bB(If you have already entered everything - enter ---)): ";

cin>>g;

int flag=0;

int flag2=0;

if(g=="---") break;

if(g[1]=='-' && g[2]=='>'){

flag2=1;

}

if(flag2==0){

cout<<"Reminder: use ->"<<endl;

continue;

}

flag2=0;

for(int i =0;i<points.size();i++){

c=g[0];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

continue;

}

flag2=0;

if(g.size()==4){

for(int i =0;i<points.size();i++){

c=g[4];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

continue;

}

}

flag2=0;

if(g.size()==8){

for(int i =0;i<points.size();i++){

c=g[7];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

continue;

}

}

flag2=0;

if(g.size()==11){

for(int i =0;i<points.size();i++){

c=g[10];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

continue;

}

}

flag2=0;

for(int i =0;i<points.size();i++){

c=g[4];

if(c==points[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Point error"<<endl;

continue;

}

flag2=0;

for(int i =0;i<step.size();i++){

c=g[3];

if(c==step[i]) flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Transition symbol error"<<endl;

continue;

}

flag2=0;

if(g.size()>5){

for(int i =0;i<step.size();i++){

c=g[6];

if(c==step[i] && g[5]=='|') flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Transition symbol or '|' error"<<endl;

continue;

}

}

flag2=0;

if(g.size()>8){

for(int i =0;i<step.size();i++){

c=g[9];

if(c==step[i] && g[5]=='|' && g[8]=='|') flag2=1;

if(flag2==1) break;

}

if(flag2==0){

cout<<"Transition symbol or '|' error"<<endl;

continue;

}

}

flag2=0;

if(g.size()==5 || g.size()==4){

if(g[1]=='-' && g[2]=='>'){

c=g[0];;

from.push\_back(c);

c=g[3];

with.push\_back(c);

if(g.size()==4){

c='@';

to.push\_back(c);

}

else{

c=g[4];

to.push\_back(c);

}

}

flag=1;

}

if(g.size()==7 || g.size()==8){

if(g[1]=='-' && g[2]=='>' && g[5]=='|'){

c=g[0];

from.push\_back(c);

from.push\_back(c);

c=g[3];

with.push\_back(c);

c=g[6];

with.push\_back(c);

c=g[4];

to.push\_back(c);

if(g.size()==7){

c='@';

to.push\_back(c);

}

else{

c=g[7];

to.push\_back(c);

}

}

flag=1;

}

if(g.size()==10 || g.size()==11){

if(g[1]=='-' && g[2]=='>' && g[5]=='|' && g[8]=='|'){

c=g[0];

from.push\_back(c);

from.push\_back(c);

from.push\_back(c);

c=g[3];

with.push\_back(c);

c=g[6];

with.push\_back(c);

c=g[9];

with.push\_back(c);

c=g[4];

to.push\_back(c);

c=g[7];

to.push\_back(c);

if(g.size()==10){

c='@';

to.push\_back(c);

}

else{

c=g[10];

to.push\_back(c);

}

}

flag=1;

}

if(flag=0) cout << "Input error" << endl;

}

}

for(int i=0;i<points.size();i++){

c=points[i];

for(int j=0;j<alph.size();j++){

b=alph[j];

if(b==c){

alph.erase(j,1);

}

}

}

for(int i=0;i<from.size();i++){

c=from[i];

for(int j=0;j<from.size();j++){

b=from[j];

if(c==b && with[i]==with[j] && to[i]!=to[j] && i!=j){

a=to[i];

d=to[j];

to[i]=alph[0];

for(int p = 0;p<to.size();p++){

if(from[p]==a){

from.push\_back(to[i]);

with.push\_back(with[p]);

to.push\_back(to[p]);

}

if(from[p]==d){

from.push\_back(to[i]);

with.push\_back(with[p]);

to.push\_back(to[p]);

}

}

to.erase(to.begin()+j);

with.erase(with.begin()+j);

from.erase(from.begin()+j);

alph.erase(0,1);

}

}

}

for(int i=0;i<from.size();i++){

cout<<from[i]<<"---("<<with[i]<<")--->"<<to[i]<<endl;

}

return 0;

}

**Вывод:** В данной лабораторной работе было реализована программа способная принять формальную грамматику как в ручном режиме, так и чтением из файла. Построение по этой грамматике конечного недетерминированного автомата и преобразование его в конечный детерминированный автомат. Проверка его работы используя заранее выбранную граматику.