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 Наплавков А.А.

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=({S,C,D},{0,1},P,E), P,S):

1)S->1C|0D, 2)C->0D|0D|1; 3)D->1C|1S|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++. Несколько векторов для хранения грамматик и переходов (откуда, куда, и с помощью чего), также используем несколько вспомогательных переменных, При запуске с пользователя просят ввести все необходимые для работы данные, после проверок ввода, программа начинает при помощи нескольких циклов, преобразовать НКА в ДКА, после чего следует вывод.

**Сам код:**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <utility>

using namespace std;

string gram1;

string gram2;

int counter;

vector <char> gramG;

vector <char> gramP;

pair <char, char> GS;

pair <pair<char, char>, char> P;

vector <pair<pair<char, char>, char>> GPS;

string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int main() {

cout << "Insert grammatic G like this: {K,L,M,N} \n";

cin >> gram1;

for (int i = 0; i <= gram1.size(); i++) {

if (gram1[i] != '{' and gram1[i] != '}' and gram1[i] != ',')

gramG.push\_back(gram1[i]);

}

cout << "Insert grammatic P like this: {a,b,+,-} \n";

cin >> gram2;

for (int i = 0; i <= gram2.size(); i++) {

if (gram2[i] != '{' and gram2[i] != '}' and gram2[i] != ',')

gramP.push\_back(gram2[i]);

}

cout << "Enter the number of clicks \n";

cin >> counter;

for (int i = 0; i <= counter - 1; i++) {

cout << "Insert " << i + 1 << " letter of grammar, by using -> \n";

cin >> gram1;

if (gram1[1] != '-' || gram1[2] != '>')

{

cout << "incorrect insert \n ";

i--;

continue;

}

if (gram1.size() == 4) {

gramG.push\_back('@');

GS = make\_pair(gram1[0], '@');

P = make\_pair(GS, gram1[3]);

GPS.push\_back(P);

}

GS = make\_pair(gram1[0], gram1[4]);

P = make\_pair(GS, gram1[3]);

GPS.push\_back(P);

if (gram1.size() == 7) {

if (gram1[5] == '|' && (!(!gram1[6])) && (!gram1[7])) {

gramG.push\_back('@');

GS = make\_pair(gram1[0], '@');

P = make\_pair(GS, gram1[6]);

GPS.push\_back(P);

}

}

else if (gram1.size() == 8) {

if (gram1[5] == '|' && (!(!gram1[6])) && (!(!gram1[7]))) {

GS = make\_pair(gram1[0], gram1[7]);

P = make\_pair(GS, gram1[6]);

GPS.push\_back(P);

}

}

if (gram1.size() == 10) {

if (gram1[8] == '|' && (!(!gram1[9])) && (!gram1[10])) {

gramG.push\_back('@');

GS = make\_pair(gram1[0], '@');

P = make\_pair(GS, gram1[9]);

GPS.push\_back(P);

}

}

else if (gram1.size() == 11) {

if (gram1[8] == '|' && (!(!gram1[9])) && (!(!gram1[10]))) {

GS = make\_pair(gram1[0], gram1[10]);

P = make\_pair(GS, gram1[9]);

GPS.push\_back(P);

}

}

}

for (int i = 0; i <= gramG.size() - 1; i++) {

for (int j = 0; j <= (alphabet.size() - gramG.size() + 1); j++) {

if (gramG[i] == alphabet[j]) {

alphabet.erase(j, 1);

}

}

}

for (int i = 0; i <= GPS.size() - 1; i++) {

char letter\_first = GPS[i].first.first;

char transfer = GPS[i].second;

char letter\_second = GPS[i].first.second;

for (int j = i + 1; j <= GPS.size() - 1; j++) { //

char previos1 = letter\_second;

char previos2 = GPS[j].first.second;

char alph = alphabet[0];

if (letter\_first == GPS[j].first.first && transfer == GPS[j].second) {

GPS[i].first.second = alphabet[0];

alphabet.erase(0, 1);

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

for (int l = 0; l <= GPS.size() - 1; l++) {

if (GPS[l].first.second == previos2 || GPS[l].first.second == previos1) {

cout << "123";

GPS[l].first.second = alph;

}

if (GPS[l].first.first == previos1 || GPS[l].first.first == previos2) {

GPS[l].first.first = alph;

}

}

}

}

}

cout << "\n";

for (int i = 0; i <= GPS.size()-1; i++) {

cout << GPS[i].first.first << " -> " << GPS[i].second << " -> " << GPS[i].first.second << " \n";

}

return 0;

}

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