МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЁТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Асс. | |  |  | | | |  | | Д. А. Кочин |
| должность, уч. степень, звание | |  | подпись, дата | | | |  | | инициалы, фамилия |
| ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №5 | | | | | | | | | | |
| Синтез конечных автоматов | | | | | | | | | | |
| по дисциплине: ТЕОРИЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ | | | | | | | | | | |
|  | | | | | | | | | | |
| РАБОТУ ВЫПОЛНИЛА | | | | | | | | | | |
| СТУДЕНТКА ГР. | 4931 | | |  | 27.10.2022 |  | | Е.Ю. Ильченко | | |
|  |  | | |  | подпись, дата |  | | инициалы, фамилия | | |
|  |  | | |  |  |  | |  | | |

Санкт-Петербург 2022

# **Цель работы**

Целью работы является изучение теории конечных автоматов, создание программы реализации алгоритма синтеза конечного автомата.

# **Постановка задачи**

В данной лабораторной работе требуется:  
• Построить конечный автомат, который осуществляет проверку входного слова на  
допустимость в заданном регулярном выражении используя алгоритм синтеза  
конечных автоматов;  
• Привести в отчете процесс синтеза конечного автомата;  
• Создать программу на языке высокого уровня реализующую алгоритм синтеза  
конечного автомата на основе заданного регулярного выражения

|  |  |
| --- | --- |
| **Вариант 11** | <<a><b>d>kc(b|d)x |

# **Алгоритм синтеза КНА**

1. Разметка всех мест;
2. Каждому основному месту (слева символы) приписывается натуральное число (отмечены зелёным цветом);
3. Формирование множества неосновных индексов по правилам подчинения мест:

1 правило. Начальные места всех термов или символов многочлена, помещённого в обычные или итерационные скобки подчинены месту, расположенному непосредственно слева от соответствующей открывающей скобки.

2 правило. Место, расположенное непосредственно справа от закрывающей скобки подчинено конечным местам всех термов многочлена, заключённого в эти скобки. В случае же итерационных скобок ещё и месту, расположенному непосредственно слева от соответствующей открывающей скобки.

3 правило. Начальные места всех термов многочлена, заключённые в итерационные скобки, подчинены месту, расположенному непосредственно справа от соответствующей закрывающей скобки.

4 правило. Если некоторое место c подчинено месту b, а место b подчинено месту a, то место c подчинено месту a.

5 правило. Каждое место подчинено само себе (рефлексивность)

6 правило. Других правил подчинения мест в регулярных выражениях нет

1. Выделение предосновных мест.
2. Формируется матрица переходов.

# **Конечный автомат заданный тремя способами**

R = < < a > < b > d > k c ( b | d ) x

0 1 2 3 4 5 6 7 8

0 0 0 0 4 5 5 6

1 1 1 3 7

3 2 2

3 3

* Таблица переходов

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Q  X | 0 (q0) | 1 (q1) | 2 (q2) | 3 (q3) | 4 (q4) | 5 (q5) | 6 (q6) | 7 (q7) | 8 (q8) |
| a | 1 | 1 | - | 1 | - | - | - | - | - |
| b | 2 | 2 | 2 | 2 | - | 6 | - | - | - |
| d | 3 | 3 | 3 | 3 | - | 7 | - | - | - |
| k | 4 | - | - | 4 | - | - | - | - | - |
| c | - | - | - | - | 5 | - | - | - | - |
| x | - | - | - | - | - | - | 8 | 8 | - |

Сопоставление состояний Q (конечные места):

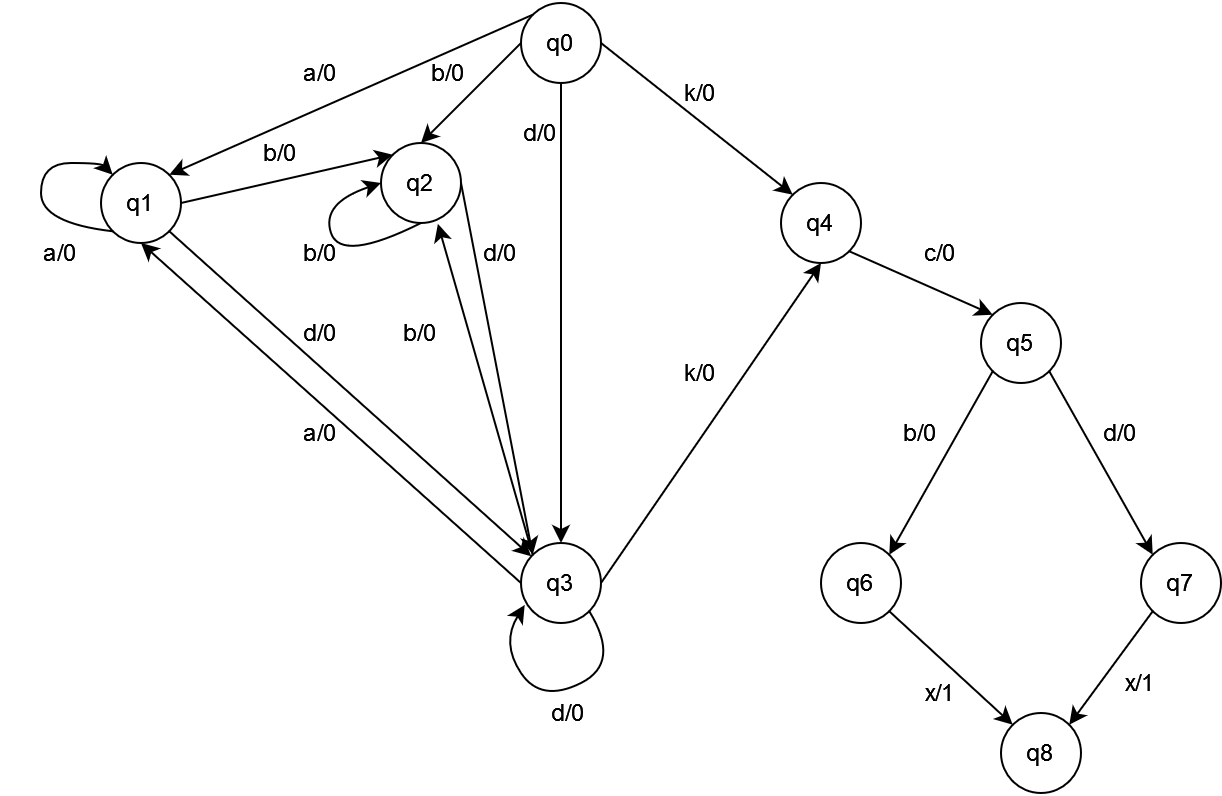
* Матрица переходов (Δ)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X  Q | a | b | d | k | c | x |
| q0 | q1 | q2 | q3 | q4 | - | - |
| q1 | q1 | q2 | q3 | - | - | - |
| q2 | - | q2 | q3 | - | - | - |
| q3 | q1 | q2 | q3 | q4 | - | - |
| q4 | - | - | - | - | q5 | - |
| q5 | - | q6 | q7 | - | - | - |
| q6 | - | - | - | - | - | q8 |
| q7 | - | - | - | - | - | q8 |
| q8 | - | - | - | - | - | - |

- Матрица выходов (Λ)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X  Q | a | b | d | k | c | x |
| q0 | 0 | 0 | 0 | 0 | - | - |
| q1 | 0 | 0 | 0 | - | - | - |
| q2 | - | 0 | 0 | - | - | - |
| q3 | 0 | 0 | 0 | 0 | - | - |
| q4 | - | - | - | - | 0 | - |
| q5 | - | 0 | 0 | - | - | - |
| q6 | - | - | - | - | - | 1 |
| q7 | - | - | - | - | - | 1 |
| q8 | - | - | - | - | - | - |

* Диаграмма переходов



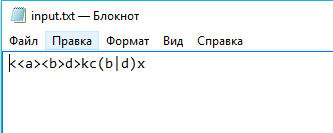
* Проверка

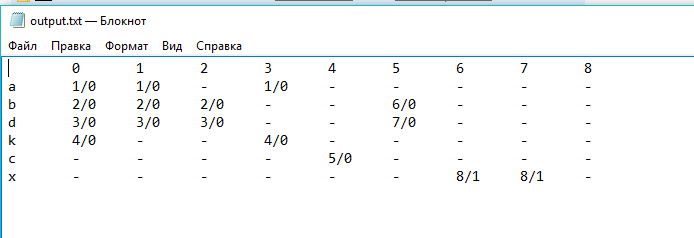
1. Входное слово α = abdkcbx → R ⇒ слово допустимо.
2. Входное слово α = cbx → ( ) ⇒ слово НЕ допустимо.
3. Входное слово α = dkcdx → R ⇒ слово допустимо.

* Автоматная матрица (M)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Q  Q | q0 | q1 | q2 | q3 | q4 | q5 | q6 | q7 | q8 |
| q0 | - | a/0 | b/0 | d/0 | k/0 | - | - | - | - |
| q1 | - | a/0 | b/0 | d/0 | - | - | - | - | - |
| q2 | - | - | b/0 | d/0 | - | - | - | - | - |
| q3 | - | a/0 | b/0 | d/0 | k/0 | - | - | - | - |
| q4 | - | - | - | - | - | c/0 | - | - | - |
| q5 | - | - | - | - | - | - | b/0 | d/0 | - |
| q6 | - | - | - | - | - | - | - | - | x/1 |
| q7 | - | - | - | - | - | - | - | - | x/1 |
| q8 | - | - | - | - | - | - | - | - | - |

# **4. Текстовые файлы входных-выходных данных программы**





**Вывод**

В ходе выполнения лабораторной работы были рассмотрены основные понятия теории КНА, создана программа на языке высокого уровня, реализующую алгоритм синтеза конечного автомата на основе заданного регулярного выражения.

**ПРИЛОЖЕНИЕ 1 (Текст программы на языке высокого уровня)**

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include <algorithm>

using namespace std;

struct table

{

string symbol;

vector <int> place;

};

// Из исходного файла считывается регулярное выражение

void read\_file(string& reg\_exp, string name)

{

ifstream file(name);

if (!file.is\_open())

cout << "File not found!\n";

else {

string str;

getline(file, str);

if (str.empty())

cout << "File is empty!" << endl;

else

reg\_exp = str;

}

file.close();

return;

}

// Запись матрицы в выходной файл

void write\_file(vector<vector<vector <int>>> cell, vector<table>row\_table, vector<vector <int>>headings\_table, vector<bool>outputs)

{

ofstream f("output.txt");

vector<vector<int>>::iterator it;

// Запись заголовков таблицы

for (int i = 0; i < headings\_table.size(); i++) {

// Запись tab и номер состояния q

f << '\t' << i;

}

for (int i = 0; i < row\_table.size(); i++) {

// Запись номера состояния q и tab

f << '\n' << row\_table[i].symbol << '\t';

for (int j = 0; j < headings\_table.size(); j++) {

// Если значение в ячейке пусто, то вывод '-'

if (cell[i][j].empty())

f << "-\t";

else {

it = find(headings\_table.begin(), headings\_table.end(), cell[i][j]);

// Запись переходов (q)

f << it - headings\_table.begin() << '/';

// Запись выходов

if (outputs[it - headings\_table.begin()] == true)

f << '1';

else

f << '0';

f << '\t';

}

}

}

f.close();

}

// Определение места перед скобкой ) или >

int placeBeforeStaple(string reg\_exp, int cur\_pos)

{

int count = 1;

string staples; // Скобки

// Если текущая позиция позиция в регулярном выражении (

if (reg\_exp[cur\_pos] == '(')

// то это обычные скобки

staples = "()";

else

// иначе итерационные скобки

staples = "<>";

while (1) {

cur\_pos++; // Увеличиваем позицию

if (reg\_exp[cur\_pos] == staples[0])

count++;

if (reg\_exp[cur\_pos] == staples[1]) {

count--;

if (count == 0)

return cur\_pos;

}

}

}

// Добавление на начальные места новые индексы неосновных мест

void add(vector <vector <int>>& separator, string reg\_exp, int begin, int end)

{

for (int i = begin; i < end; i++) {

if (i == begin) {//добавляем в начальное место, сразу за скобками

for (int j = 0; j < separator[begin - 1].size(); j++) {

if (find(separator[begin].begin(), separator[begin].end(), separator[begin - 1][j]) == separator[begin].end())

separator[begin].push\_back(separator[begin - 1][j]);

}

}

if (reg\_exp[i] != '>' && reg\_exp[i] != ')' && reg\_exp[i] != '|'

&& reg\_exp[i - 1] == '|') {// добавляем всем остальным диз членам в скобках

for (int j = 0; j < separator[begin - 1].size(); j++) {

if (find(separator[i].begin(), separator[i].end(), separator[begin - 1][j]) == separator[i].end())

separator[i].push\_back(separator[begin - 1][j]);

}

}

if ((reg\_exp[i] == '<' || reg\_exp[i] == '(') && (reg\_exp[i - 1] == '|' ||

reg\_exp[i - 1] == '(' || reg\_exp[i - 1] == '<')) {

// Определение места перед скобкой ) или >

int paren = placeBeforeStaple(reg\_exp, i);

add(separator, reg\_exp, i + 1, paren);

i = paren;

}

}

}

// Определение основного места

int mainPlace(vector<table> r, char s[1])

{

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

// Если слева символ

if (s[0] == r[j].symbol[0])

return j;

}

return -1;

}

// Первый, второй и третий пункты алгоритма синтеза КНА - разметка и определение основных и неосновых мест

// begin = 0, count = 1

void markuping(vector <vector <int>>& separator, string reg\_exp, int begin, int end, int& cur\_pos,

vector<table>& row\_table)

{

// Начальные места дизъюнктивных членов (| или)

vector<int>disjunct;

vector<int>disjunct\_place;

int check;

// Для всего регулярного выражения

for (int i = begin; i < end; i++) {

// Если начало регулярного выражения

if (i == begin) {

if (i == 0) {

separator[0].push\_back(0);

if (reg\_exp[0] != '<' && reg\_exp[0] != '>' && reg\_exp[0] != '('

&& reg\_exp[0] != ')')

disjunct.push\_back(0);

}

else {

for (int j = 0; j < separator[i - 1].size(); j++)

separator[i].push\_back(separator[i - 1][j]);

disjunct.push\_back(i);

}

}

// Основные места

// Если текущая позиция регулярного выражения не равна итерационным или обычным скобкам или |

if (reg\_exp[i] != '(' && reg\_exp[i] != ')' && reg\_exp[i] != '<'

&& reg\_exp[i] != '>' && reg\_exp[i] != '|') {

check = mainPlace(row\_table, &reg\_exp[i]);

// Если слева символ

if (check != -1)

row\_table[check].place.push\_back(i);

// иначе (слева нет символа)

else

row\_table.push\_back(table{ { reg\_exp[i] },{ i } });

// Добавление разметки в указанную позицию

separator[i + 1].push\_back(cur\_pos);

cur\_pos++; // Увеличиваем позицию (продвигаемся по регулярному выражению)

}

if (i > 0) {

// 1 правило подчинения мест

// Индекс места перед любыми скобками распространяется на начальные места всех дизъюнктивных членов, записанных в этих скобках

if (reg\_exp[i] != '>' && reg\_exp[i] != ')' && reg\_exp[i] != '|'

&& reg\_exp[i - 1] == '|') {

if (begin == 0) {

for (int j = 0; j < separator[begin].size(); j++)

separator[i].push\_back(separator[begin][j]);

}

else {

for (int j = 0; j < separator[begin - 1].size(); j++)

separator[i].push\_back(separator[begin - 1][j]);

}

disjunct.push\_back(i);//начальные места дизъюнктивных членов

disjunct\_place.push\_back(i - 1);

}

}

if (i == end - 1) {

//Отмечаю конец последнего диз члена в скобках

if (reg\_exp[end - 1] == ')' || reg\_exp[end - 1] == '>')

disjunct\_place.push\_back(end - 1);

// 2 правило подчинения мест

//Индекс конечного места любого дизъюнктивного члена, заключенного в любые скобки,

//распространяется на место, непосредственно следующее за этими скобками.

for (int j = 0; j < disjunct\_place.size(); j++) {

for (int k = 0; k < separator[disjunct\_place[j]].size(); k++)

separator[end].push\_back(separator[disjunct\_place[j]][k]);

}

if (reg\_exp[end - 1] == '>') {

for (int j = 0; j < separator[begin - 1].size(); j++)

// Индекс места перед итерационными скобками распространяется на место, непосред - ственно следующее за этими скобками

if (find(separator[end].begin(), separator[end].end(), separator[begin - 1][j]) == separator[end].end())

separator[end].push\_back(separator[begin - 1][j]);

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

// Индекс места за итерационными скобками распространяется на начальные места всех дизъюнктивных членов, заключенных в итерационные скобки

for (int k = 0; k < separator[end].size(); k++) {

if (find(separator[disjunct[j]].begin(), separator[disjunct[j]].end(), separator[end][k]) == separator[disjunct[j]].end())

separator[disjunct[j]].push\_back(separator[end][k]);

}

// Значит надо добавить на начальные места диз членов новые индексы

if (reg\_exp[disjunct[j]] == '<' || reg\_exp[disjunct[j]] == '(') {

add(separator, reg\_exp, disjunct[j] + 1, placeBeforeStaple(reg\_exp, disjunct[j]));

}

}

}

}

// Если текущая позиция регулярного выражения итерационные или обычные скобки

if (reg\_exp[i] == '<' || reg\_exp[i] == '(') {

// Определение места перед скобкой ) или >

int paren = placeBeforeStaple(reg\_exp, i);

markuping(separator, reg\_exp, i + 1, paren + 1, cur\_pos, row\_table);

i = paren;

}

}

}

// Четвертый пункт алгоритма синтеза КНА - определение предосновных мест

void beforeMainPlaces(vector<vector<vector <int>>>& cell, vector<table>row\_table, vector<vector <int>>& headings\_table,

vector <vector <int>> separ, vector<bool>& outputs)

{

int column = 0; // столбцы

while (column < headings\_table.size()) {

for (int i = 0; i < headings\_table[column].size(); i++) {

for (int j = 0; j < separ[separ.size() - 1].size(); j++) {

//Если состояние является конечным

if (headings\_table[column][i] == separ[separ.size() - 1][j]) {

outputs[column] = true;

break;

}

}

if (outputs[column] == true)

break;

}

// Для каждой строки

for (int i = 0; i < row\_table.size(); i++) {

cell[i].push\_back(vector<int>());

// Для каждого вхождения символа во входную строку

for (int j = 0; j < row\_table[i].place.size(); j++) {

//Для каждой позиции в заголовке столбца

for (int k = 0; k < headings\_table[column].size(); k++) {

// Определение предосновых мест

if (find(separ[row\_table[i].place[j]].begin(),

separ[row\_table[i].place[j]].end(), headings\_table[column][k]) != separ[row\_table[i].place[j]].end()

&& find(cell[i][column].begin(), cell[i][column].end(), separ[row\_table[i].place[j] + 1][0]) == cell[i][column].end())

// Добавление позиции основного места в ячейку

cell[i][column].push\_back(separ[row\_table[i].place[j] + 1][0]);

}

}

sort(cell[i][column].begin(), cell[i][column].end());

// Если такого заголовка нет

if (find(headings\_table.begin(), headings\_table.end(), cell[i][column]) == headings\_table.end() && !cell[i][column].empty()) {

//Добавляем

headings\_table.push\_back(cell[i][column]);

outputs.push\_back(false);

}

}

column++;

}

}

int main()

{

setlocale(LC\_ALL, "RUS");

// Переменная для регулярного выражения

string reg\_exp;

// Из исходного файла считывается регулярное выражение

read\_file(reg\_exp, "input.txt");

// Вектор для регулярного выражения с разделителями

vector <vector <int>> separator(reg\_exp.size() + 1, vector<int>());

// Строки таблицы (символы X)

vector<table>row\_table;

// Выходы КНА

vector<bool>outputs;

// Заголовки столбцов таблицы (состояния q)

vector<vector <int>>headings\_table;

headings\_table.push\_back(vector < int > {0});

outputs.push\_back(false);

// Позиция в регулярном выражении

int cur\_pos = 1;

// Первый, второй и третий пункты алгоритма синтеза КНА - разметка и определение основных и неосновых мест

markuping(separator, reg\_exp, 0, reg\_exp.size(), cur\_pos, row\_table);

// Первый вектор-строка, второй-столбец, третий - ячейка (cell)

vector<vector<vector <int>>> cell(row\_table.size(), vector<vector <int>>());

// Четвертый пункт алгоритма синтеза КНА - определение предосновных мест

beforeMainPlaces(cell, row\_table, headings\_table, separator, outputs);

// Запись матрицы в выходной файл

write\_file(cell, row\_table, headings\_table, outputs);

return 0;

}