**Санкт-Петербургский национальный исследовательский университет**

**информационных технологий, механики и оптики**

**Кафедра вычислительной техники**

Формальные языки и грамматики

Домашние задание №2

Построение НКА и ДКА по регулярному выражению

Выполнил: Владимир Бакшенов

Группа P3218

Преподаватель: Лаздин А.В.

2017 г.

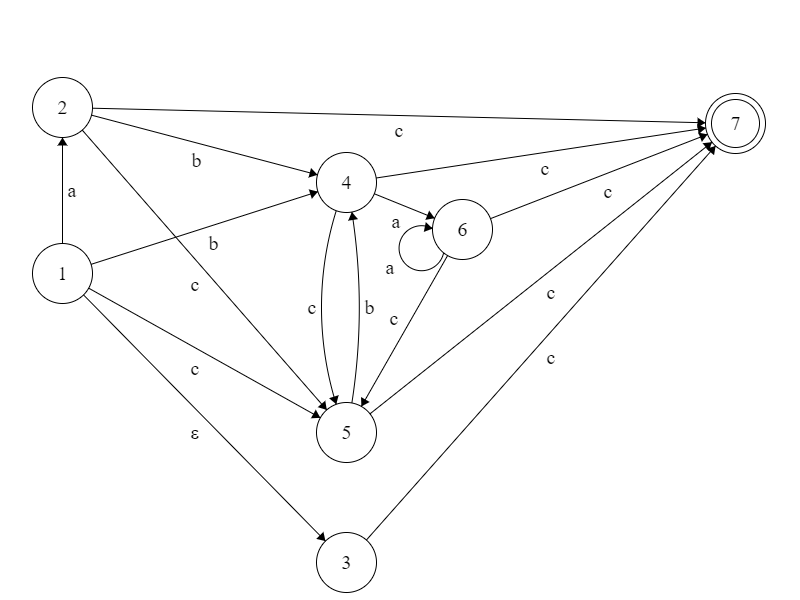
**1. Задание**

По заданному регулярному выражению a?(b(a\*)|c)\*c

* Построить недетерминированный КА
* По полученному НДА построить ДКА
* Для ДКА написать программу-распознаватель предложений языка, порождаемого регулярным выражением.

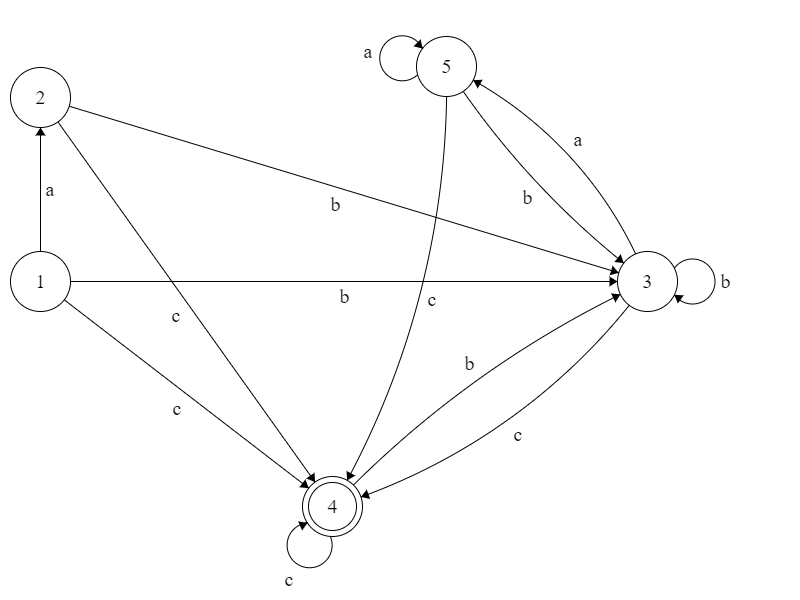
Продемонстрировать работу распознавателя на различных примерах (не менее трех правильных) предложений.

**2. Недетерминированный КА**



**3. Детерминированный КА**

Избавляемся от ε – переходов и недетерминированности



**4. Код программы**

***prog.h***

#pragma once

#include <vector>

struct State

{

int state;

std::vector<int> newState;

std::vector<char> transitionSymbols;

};

namespace constants

{

const int numberOfStates = 5;

const int validState = 4;

}

***prog.cpp***

#include <iostream>

#include <conio.h>

#include <string>

#include "main.h"

void preparingDFA(State states[])

{

states[0].state = 1;

states[0].newState = { 2, 3, 4 };

states[0].transitionSymbols = { 'a', 'b', 'c' };

states[1].state = 2;

states[1].newState = { 3, 4 };

states[1].transitionSymbols = { 'b', 'c' };

states[2].state = 3;

states[2].newState = { 5, 3, 4 };

states[2].transitionSymbols = { 'a', 'b', 'c' };

states[3].state = 4;

states[3].newState = { 3, 4 };

states[3].transitionSymbols = { 'b', 'c' };

states[4].state = 5;

states[4].newState = { 5, 3, 4 };

states[4].transitionSymbols = { 'a', 'b', 'c' };

}

std::string getInput()

{

std::string input = "";

std::cout << "Input: ";

std::getline(std::cin, input);

return input;

}

bool processingNewSymbol(State state, std::string input, int \*currentState, int \*currentSymbol)

{

for (int i = 0; i < state.transitionSymbols.size(); i++)

{

if (input[\*currentSymbol] == state.transitionSymbols[i])

{

\*currentState = state.newState[i] - 1;

(\*currentSymbol)++;

return true;

}

}

return false;

}

bool validateInput(std::string input, State states[])

{

bool inputIsValid = false;

int currentState = 0;

int currentSymbol = 0;

State state = states[currentState];

bool transitionExists = false;

while (currentSymbol < input.length())

{

transitionExists = processingNewSymbol(state, input, &currentState, &currentSymbol);

if (!transitionExists) return inputIsValid;

state = states[currentState];

}

state.state == constants::validState ? inputIsValid = true : inputIsValid = false;

return inputIsValid;

}

void outputResult(bool inputIsValid)

{

if (inputIsValid)

{

std::cout << "Result: Valid\n\n";

}

else

{

std::cout << "Result: Invalid\n\n";

}

}

int main()

{

std::string input = "";

bool inputIsValid = false;

State states[constants::numberOfStates];

preparingDFA(states);

while (true)

{

input = getInput();

inputIsValid = validateInput(input, states);

outputResult(inputIsValid);

}

return 0;

}

***Output***

**Input: ac**

**Result: Valid**

**Input: abcc**

**Result: Valid**

**Input: abaacc**

**Result: Valid**

**Input: abaacacc**

**Result: Invalid**

**Input: baaacbaac**

**Result: Valid**

**Input: bcccbabbadca**

**Result: Invalid**