МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной техники

ОТЧЕТ О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ №5

«Программа преобразования НКА в ДКА»

по дисциплине
«Теория формальных языков»

Выполнил студент группы ИКБО-04-21 Исаев В. В.

Принял ассистент Боронников А. С.

Практическая работа выполнена «__»_____2022 г.

«Зачтено» «__»_____2022 г.

Москва 2022

Задание

Задание: на любом языке программирования написать программу преобразования недетерминированного конечного автомата (НКА) в детерминированный (ДКА).

Реализация решения задания на языке C++

main-1.cpp

```
#include "Automaton.h"
using namespace std;


int main()
{
    setlocale(LC_ALL, "rus");
    string begin = "1", end = "3";
    string first_states = "(1,a,1) (1,a,2) (1,b,3) (2,a,2)
(2,b,1) (2,b,3) (3,a,3) (3,b,3)";
    Op op(begin, end);
    op.NFAlist(first_states);
    op.outprint();
     cout << endl;
     system("pause");
}
```

Automaton.cpp

```
#include "Automaton.h"


void Op::NFAlist(string str) {
    int i = 0;
    string temp;
    enum states CS = F;
    while (i < str.size()) {
        switch (CS)
```

```
        {
        case F:
            CS = FNUM;
            i++;
            temp = str[i];
            break;
        case L:
            CS = F;
            i += 2;
            temp = "";
            break;
        case FNUM:
            if (str[i + 1] != ',') {
                temp += str[i];
                i += 1;
                break;
            }
            if (!check_node_for_FNUM(temp))
createListNFA(temp);
            CS = LETTER;
            i += 2;
            current_pos = temp;
            break;
        case LETTER:
            if (str[i] == 'a') ab = A;
            else ab = B;
            CS = LNUM;
            i += 2;
            temp = str[i];
            break;
        case LNUM:
            if (str[i + 1] != ')') {
                temp += str[i];
```

```
                i++;
                break;
            }
            check_nodes_for_LNUM(temp);
            CS = L;
            i++;
            break;
        }
    }
    //Начало создания списка ДКА
    //В первую очередь помещаем в список запись начального
состояния
    Node* node = new Node;
    node->name = start_state;
    DKAlist(node);
}

bool Op::check_node_for_FNUM(string str) {
    if (NFA.empty()) return false;
    for (auto node : NFA) {
        if (node->name == str) return true;
    }
    return false;
}

void Op::check_nodes_for_LNUM(string str) {
    string temp;
    list <Node*> ::iterator it = NFA.begin();
    Node* node = *it;
    while (it != NFA.end()) {
        node = *it;
        if (node->name == current_pos) break;
        it++;
```

```
    }
    if (ab == A) {
        if (node->A.size() == 0) {
            node->A += str;
        }
        for (int i = 0; i < str.size(); i++)
            for (int j = 0; j < node->A.size(); j++) {
                if (str[i] == node->A[j]) break;
                else temp += str[i];
            }
        node->A += temp;
    }
    else {
        if (node->B.size() == 0)
            node->B += str;
        for (int i = 0; i < str.size(); i++) {
            for (int j = 0; j < node->B.size(); j++) {
                if (str[i] == node->B[j]) break;
                else temp += str[i];
            }
        }
        node->B += temp;
    }
}

void Op::createListNFA(string str) {
    Node* node = new Node;
    node->name = str;
    NFA.push_back(node);
}

void Op::DKAlist(Node* node) {
    //пока не будет найдено повторение
```

```
    if (!check_notes_DKA(node->name)) {
        if (DFA.empty()) {
            create_list_DFA(start_state);
            node = DFA.front();
        }
        string notes = create_A_node(node);
        if (!check_notes_DKA(node->A))
        {
            create_list_DFA(notes);
            DKAlist(DFA.back());
        }
        notes = create_B_node(node);
        if (!check_notes_DKA(node->B))
        {
            create_list_DFA(notes);
            DKAlist(DFA.back());
        }
    }
}

string Op::create_A_node(Node* node) {
    string temp;
    for (int k = 0; k < node->name.size(); k++) {
        for (auto tmp : NFA) {
            if (tmp->name[0] == node->name[k]) {
                if (temp != "")
                    for (int i = 0; i < tmp->A.size(); i++)
                        for (int j = 0; j < temp.size(); j++)
{
                            if (tmp->A[i] == temp[j]) break;
                            else if (j == temp.size() - 1)
temp += tmp->A[i];
                        }
                else temp = tmp->A;
```

```
            }
        }

    }
    temp = sort(temp);
    node->A = temp;
    return temp;
}

string Op::create_B_node(Node* node) {
    string temp;
    for (int k = 0; k < node->name.size(); k++) {
        for (auto tmp : NFA) {
            if (tmp->name[0] == node->name[k]) {
                if (temp != "")
                    for (int i = 0; i < tmp->B.size(); i++)
                        for (int j = 0; j < temp.size(); j++)
{
                            if (tmp->B[i] == temp[j]) break;
                            else if (j == temp.size() - 1)
temp += tmp->B[i];
                        }
                else temp = tmp->B;
            };
        }
    }
    temp = sort(temp);
    node->B = temp;
    return temp;
}

bool Op::check_notes_DKA(string str) {
    list <Node*> ::iterator it = DFA.begin();
    Node* node;
```

```
    for (auto node : DFA)
        if (node->name == str && (node->A != "" || node->B != 
"")) return true;
    return false;
}

void Op::create_list_DFA(string str) {
    Node* node = new Node;
    node->name = str;
    DFA.push_back(node);
}

string Op::sort(string str) {
    char tmp;
    for (int i = 0; i < str.size() - 1; ++i) // i - номер 
прохода
        for (int j = 0; j < str.size() - 1; ++j) // внутренний 
цикл прохода
            if (str[j + 1] < str[j])
                swap(str[j], str[j + 1]);
    return str;
}

void Op::outprint() {
    cout << "DFA:\nSet of states: ";
    for (auto node : DFA)
        cout << node->name << " ";
    cout << "\nInput alphabet: a, b\nState-transitions 
function:\n";
    for (auto node : DFA) {
        cout << "D(" << node->name << ", a) = " << node->A << 
endl;
        cout << "D(" << node->name << ", b) = " << node->B << 
endl;
    }
```

```
    cout << "\nInitial states: " << start_state << endl;
    cout << "Final states: " << find_the_end_states();
}

string Op::find_the_end_states() {
    string final_st;
    for (auto node : DFA) {
        for (int i = 0; i < node->name.size(); i++) {
            if (node->name[i] == end_state[0]) {
                final_st += node->name + " ";
                break;
            }
        }
    }
    return final_st;
}
```

Automaton.h

```
#pragma once
#include <iostream>
#include <list>
#include <string>
#include <iterator>
using namespace std;

struct Node {
    string name;
    string A; //переход по a
    string B; //переход по b
};

enum states {
```

```
    F,
    L,
    FNUM,
    LETTER,
    LNUM
};

enum AorB {
    A, B
};

class Op {
    list <Node*> NFA; //список для НКА
    list <Node*> DFA; //список для ДКА
    enum AorB ab = A;
    string current_pos;
    string start_state, end_state; //начальные и конечные
состояния

public:
    Op(string b, string e) :start_state(b), end_state(e) {}

    //Обработка строки с переходами
    void NFAlist(string);

    bool check_node_for_FNUM(string);
    void check_nodes_for_LNUM(string);
    void createListNFA(string);

    //Операции для ДКА
    void DKAlist(Node*);
    string create_A_node(Node*);
    string create_B_node(Node*);
```

```
    //проверка наличия состояния в списке ДКА по имени
    bool check_notes_DKA(string);
    //добавление в список ДКА нового состояния(узла)
    void create_list_DFA(string);
    //сортировка строки, представляющей собой набор индексов
состояний
    string sort(string);
    //вывод
    void outprint();

    string find_the_end_states();
};
```

Тестирование

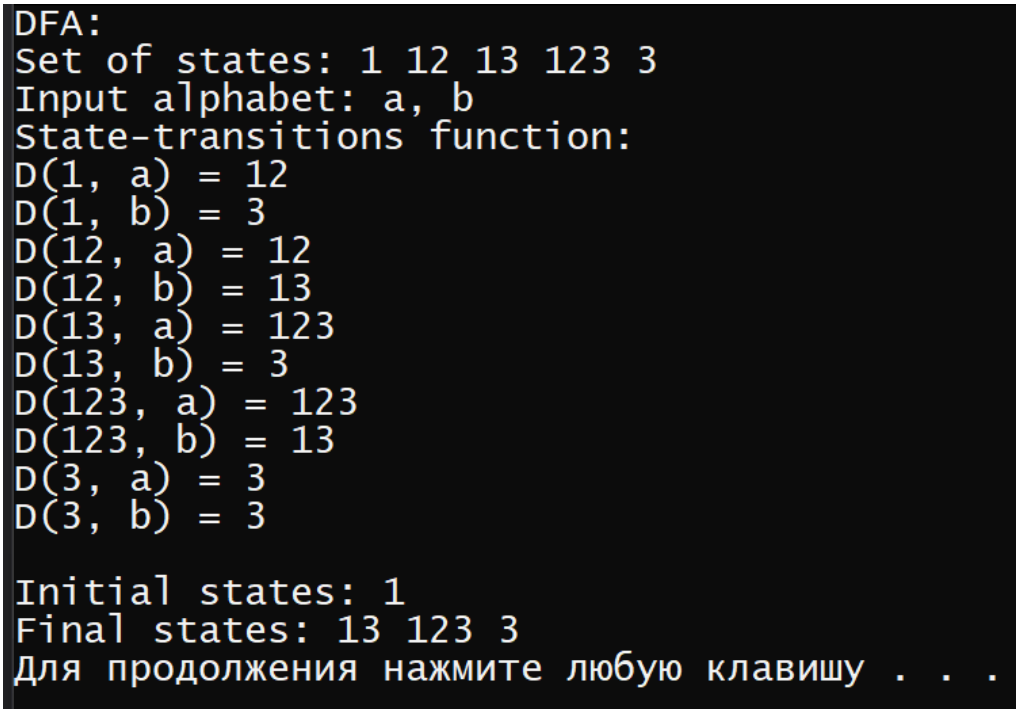

Рисунок 1. Результат программы

Вывод

В данной практической работе были получены навыки в работе с преобразователями автоматов и с процедурой преобразования автоматов.