Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа №1. Конечные автоматы
Тема

Преподаватель _____ Д. В. Личаргин
подпись, дата | инициалы, фамилия

Студент КИ19-17/1б 031939175 _____ А. Д. Непомнящий
номер группы, зачетной книжки | подпись, дата | инициалы, фамилия

Красноярск 2021

1 Цель работы

Цель состоит в Реализация и исследование детерминированных и недетерминированных конечных автоматов.

2 Задачи

Выполнение работы сводится к следующим задачам.

1. Ознакомиться со сведениями по теории конечных автоматов.

2. Разработать в системе JFLAP согласно постановке задачи детерминированный конечный автомат, а также предложить программную реализацию на любом языке программирования. В коде программы обязательно наличие сущностей и процедур, относящихся к табличному представлению автомата. Результат работы, выдаваемый программой на экран, внешне должен быть схож, а фактически эквивалентен результату, выдаваемому JFLAP на тех же тестовых цепочках.

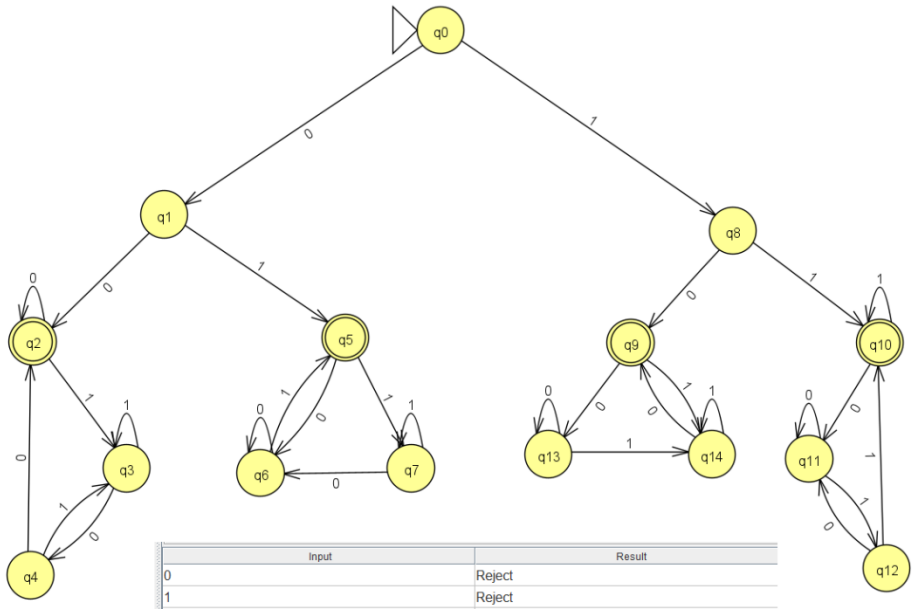
3. Используя изученные механизмы, разработать в системе JFLAP недетерминированный конечный автомат, а также предложить программную реализацию на любом языке программирования. В коде программы обязательно наличие сущностей и процедур, относящихся к табличному представлению автомата. Результат работы, выдаваемый программой на экран, внешне должен быть схож, а фактически эквивалентен результату, выдаваемому JFLAP на тех же тестовых цепочках.

4. Написать настоящий отчет и представить его к защите.

Вариант 10. Построить ДКА, допускающий в алфавите {0, 1} все цепочки нулей и единиц с одинаковыми парами символов на обоих краях цепочки. Построить НКА, допускающий язык из цепочек из 0 и 1, в которых хотя бы на одной из последних пяти позиций стоит 1.

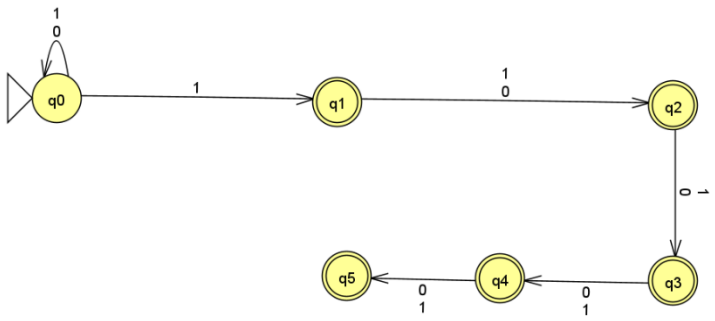
3 Графы переходов полученных конечных автоматов

На рисунках далее приведены графы переходов полученных детерминированного и недетерминированного автоматов соответственно.



Input	Result
0	Reject
1	Reject
10	Accept
11	Accept
100	Reject
101	Reject
110	Reject
111	Accept
1000	Reject
1001	Reject
1010	Accept
1011	Reject
1100	Reject
1101	Reject
1110	Reject
1111	Accept
10000	Reject
10001	Reject
10010	Accept
10011	Reject
10100	Reject
10101	Reject
10110	Accept
10111	Reject
11000	Reject
11001	Reject
11010	Reject
11011	Accept
11100	Reject
11101	Reject
11110	Reject
11111	Accept

Рисунок 1 – Граф переходов и результаты работы детерминированного КА



0	Reject
1	Accept
10000	Accept
100000	Reject
11000	Accept
110000	Accept
00001	Accept
000001	Accept
1100000	Reject
11000000	Reject
11000001	Accept

Рисунок 2 – Граф переходов и результаты работы недетерминированного КА

4 Реализация на языке программирования

Полученные автоматы были реализованы на языке программирования C#. Ниже приведен исходный код реализации.

Листинг 1 – Код класса, реализующего детерминированный КА

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace FSA
{
    public class Dfsa
    {
        private HashSet<string> _states;
        private readonly HashSet<string> _finalStates;
        private readonly HashSet<char> _alphabet;
        private readonly Dictionary<(string, char), string> _transitionTable;

        private string _currentState;
        private readonly string _initialState;

        public bool Iterate(string input, out string log)
        {
            log = "";
            _currentState = _initialState;
            if (input.Any(operation => !_alphabet.Contains(operation)))
                throw new ArgumentException("Input contains symbols besides
elements of alphabet");
            foreach (var operation in input)
            {
                log += _currentState + " --" + operation + "-> ";
                _currentState = _transitionTable[(_currentState, operation)];
            }
            log += _currentState + "\n " + input + " "+
(_finalStates.Contains(_currentState) ?  "Accept" : "Reject");
```

Окончание листинга 1

```
            return _finalStates.Contains(_currentState);
        }

        public Dfsa(HashSet<char> alphabet, HashSet<string> states, string
initialState,
            Dictionary<(string, char), string> transitionTable, HashSet<string>
finalStates)
        {
            if (alphabet.Count == 0) throw new ArgumentException("Alphabet must
not be empty");
            if (states.Count == 0) throw new ArgumentException("States set must
not be empty");
            if (finalStates.Count == 0) throw new ArgumentException("At least
one state must be final");
            if (!states.Contains(initialState))
                throw new ArgumentException("Current state must be an element of
states set");
            if (finalStates.Count == 0 || finalStates.Any(state =>
!states.Contains(state)))
                throw new ArgumentException("Final states must not be empty and
must be a subset of states set");
            if (transitionTable.Any(transition =>
!states.Contains(transition.Key.Item1) ||

!alphabet.Contains(transition.Key.Item2)
                                                    ||
!states.Contains(transition.Value)) || states.Any(i =>
                alphabet.Any(j => !transitionTable.ContainsKey((i, j)))))
                throw new ArgumentException(
                    "Table of transitions must completely cover cartesian
product of states and alphabet and only");

            _states = states;
            _finalStates = finalStates;
            _alphabet = alphabet;
            _transitionTable = transitionTable;
            _initialState = initialState;
            _currentState = initialState;
        }
    }
}
```

Листинг 2 – Код класса, реализующего недетерминированный КА

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace FSA
{
    public class Nfsa
    {
        private readonly HashSet<string> _states;
        private readonly HashSet<string> _finalStates;
        private readonly HashSet<char> _alphabet;
        private readonly Dictionary<(string, char), HashSet<string>>
_transitionTable;
```

Продолжение листинга 2

```
        private HashSet<string> _currentStates;
        private readonly string _initialState;

        public bool Iterate(string input, out string log)
        {
            log = "";
            _currentStates.Clear();
            _currentStates.Add(_initialState);
            if (input.Any(operation => !_alphabet.Contains(operation)))
                throw new ArgumentException("Input contains symbols besides
elements of alphabet");

            foreach (var operation in input)
            {
                var newStates = new HashSet<string>();

                foreach (var newState in _currentStates.Where(newState =>
                    _transitionTable.ContainsKey((newState, operation))))
                {
                    newStates.UnionWith(_transitionTable[(newState,
operation)]);
                }

                _currentStates = newStates;
            }

            //log += _finalStates.Contains(_currentState) ?  "Accept" :
"Reject");
            log += input + " " + (_currentStates.Any(state =>
_finalStates.Contains(state)) ? "Accept" : "Reject");
            return _currentStates.Any(state => _finalStates.Contains(state));
        }


        public Nfsa(HashSet<char> alphabet, HashSet<string> states, string
initialState,
            Dictionary<(string, char), HashSet<string>> transitionTable,
HashSet<string> finalStates)
        {
            if (alphabet.Count == 0) throw new ArgumentException("Alphabet must
not be empty");
            if (states.Count == 0) throw new ArgumentException("States set must
not be empty");
            if (finalStates.Count == 0) throw new ArgumentException("At least
one state must be final");
            if (!states.Contains(initialState))
                throw new ArgumentException("Current state must be an element of
states set");
            if (finalStates.Count == 0 || finalStates.Any(state =>
!states.Contains(state)))
                throw new ArgumentException("Final states must not be empty and
must be a subset of states set");
            if (transitionTable.Any(transition =>
!states.Contains(transition.Key.Item1) ||

!alphabet.Contains(transition.Key.Item2)
```

Окончание листинга 2

```
                                                || transition.Value.Any(newState =>
!states.Contains(newState))))
                throw new ArgumentException(
                    "Table of transitions must only refer states of automate");

            _states = states;
            _finalStates = finalStates;
            _alphabet = alphabet;
            _transitionTable = transitionTable;
            _initialState = initialState;
            _currentStates = new HashSet<string> { initialState };
        }
    }
}
```