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

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ № 1

Конечные автоматы
Тема

Преподаватель _____ Д. В. Личаргин
Подпись, дата — Инициалы, Фамилия

Студент КИ19-17/1Б, №031939174 _____ А. К. Никитин
Номер группы, зачетной книжки — Подпись, дата — Инициалы, Фамилия

Красноярск 2021

1 Цель

Целью данной работы является реализация и исследование детерминированных и недетерминированных конечных автоматов.

2 Задачи

Используя изученные механизмы, разработать в системе JFLAP детерминированный конечный автомат, а также предложить программную реализацию на любом языке программирования. В случае невозможности создания ДКА, это должно доказываться формально. В коде программы обязательно наличие сущностей и процедур, относящихся к табличному представлению автомата. Использование функций обработки строковых данных запрещено. Результат работы, выдаваемый программой на экран, внешне должен быть схож, а фактически эквивалентен результату, выдаваемому JFLAP на тех же тестовых цепочках.

Используя изученные механизмы, разработать в системе JFLAP недетерминированный конечный автомат, а также предложить программную реализацию на любом языке программирования. В случае невозможности создания НКА, это должно доказываться формально. В коде программы обязательно наличие сущностей и процедур, относящихся к табличному представлению автомата. Использование функций обработки строковых данных запрещено. Результат работы, выдаваемый программой на экран, внешне должен быть схож, а фактически эквивалентен результату, выдаваемому JFLAP на тех же тестовых цепочках.

3 Описание варианта

Вариант 2.

а) Построить ДКА, допускающий в алфавите {a, b} все строки, где количество символов a равно 2, и количество символов b больше 2.

б) Построить НКА, допускающий цепочки в алфавите Z = {1, 2, 3}, у которых последний символ цепочки не появлялся в ней раньше, например w = 2321.

4 Ход работы

4.1 Построенные конечные автоматы

На рисунке 1 и 2 представлены графы конечных автоматов.

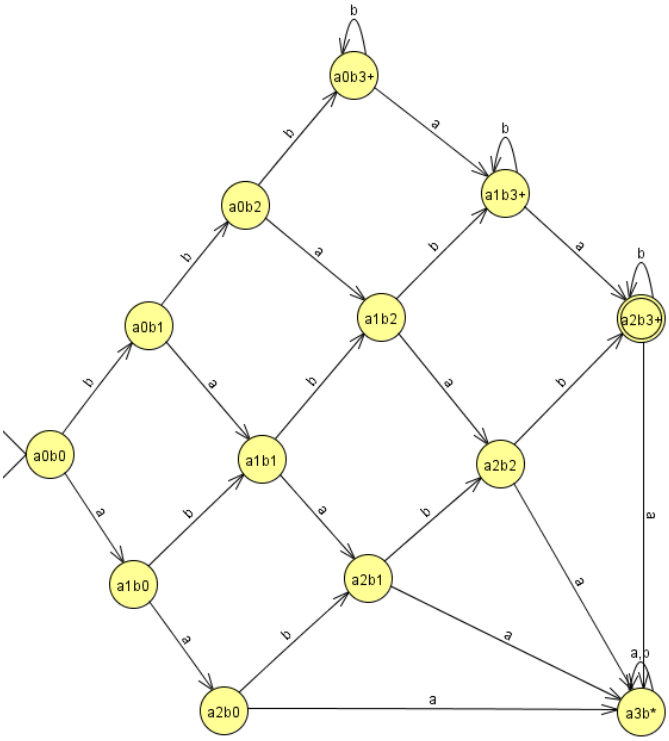

Рисунок 1 – Детерменированный конечный автомат

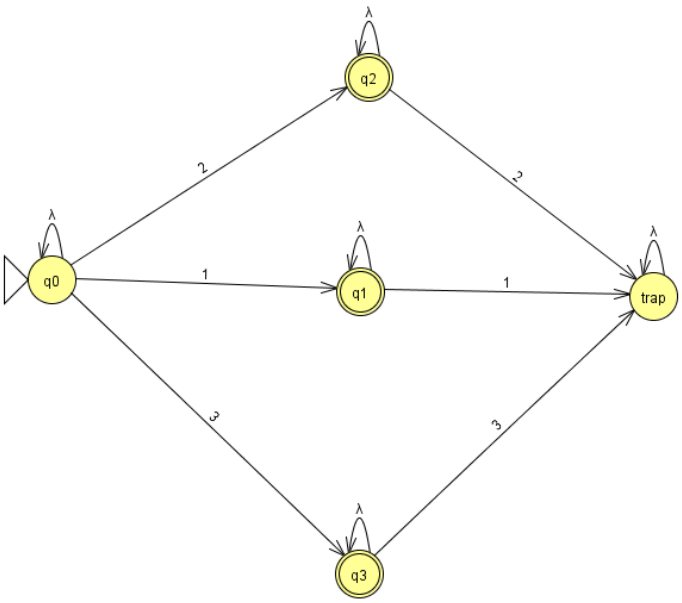

Рисунок 2 – Недетерменированный конечный автомат

4.2 Тестовые кейсы

На рисунках 3 и 4 представлены тестовые случаи для конечных автоматов

Input	Result
aabbb	Accept
aaaaaabbbbbbbbbbbba	Reject
abbbbbbbbbbbbba	Accept
abbbbaabbaa	Reject
ab	Reject
abba	Reject

Рисунок 3 – Тестовые случаи для ДКА

Input	Result
1231	Reject
12311	Reject
12123	Reject
32	Accept
1111	Reject
3232323231	Reject
	Reject

Рисунок 4 – Тестовые случаи НКА

4.3 Программная реализация

На листинге 1 представлен код реализации автоматов на языке Python.

Листинг 1 – ДКА и НКА

```
import pandas as pd
import graphviz
import os

GRAPHVIZ_PATH = 'C:/Program Files/Graphviz/'
os.environ["PATH"] += os.pathsep + GRAPHVIZ_PATH + 'bin/'

INITIAL_STATE = 'a0b0'
TRASH_STATES = ['a3b*']
FINAL_STATES = ['a2b3+']
TABLE = pd.DataFrame({'a0b0':  ['a1b0', 'a0b1'],
                          'a0b1':  ['a1b1', 'a0b2'],
                          'a0b2':  ['a1b2', 'a0b3+'],
                          'a0b3+': ['a1b3+','a0b3+'],
                          'a1b0':  ['a2b0', 'a1b1'],
                          'a1b1':  ['a2b1', 'a1b2'],
                          'a1b2':  ['a2b2', 'a1b3+'],
                          'a1b3+': ['a2b3+','a2b3+'],
                          'a2b0':  ['a3b*', 'a2b1'],
                          'a2b1':  ['a3b*', 'a2b2'],
                          'a2b2':  ['a3b*', 'a2b3+'],
                          'a2b3+': ['a3b*', 'a2b3+'],
                          'a3b*':  ['a3b*', 'a3b*'],
                          }, index=['a', 'b'])


class Table:
    def __init__(self, matrix, initial_state, final_states, trash_states=None):

        self.table = matrix
        self.alphabet = matrix.index.array
        self.initial_state = initial_state
        self.final_states = final_states

        if trash_states is None:
            self.trash_states = []
        else:
            self.trash_states = trash_states

        self.graph     =     graphviz.Digraph(filename="Graph",     format='png',
strict=True, graph_attr={
```

Продолжение листинга

```
            'concentrate': 'true',
            'rankdir': 'LR'
        })

    def _form_image(self):
        self.graph.node('a0b0',        style='filled',        colorscheme="SVG",
fillcolor='yellow')

        for final_state in self.final_states:
            self.graph.node(final_state,    style='filled',    colorscheme="SVG",
fillcolor='blue')

        for trash_state in self.trash_states:
            self.graph.node(trash_state,    style='filled',    colorscheme="SVG",
fillcolor='red')

        for column in self.table:
            if self.table[column]['a'] == self.table[column]['b']:
                self.graph.edge(column, self.table[column]['a'], label='a,b')
            else:
                self.graph.edge(column, self.table[column]['a'], label='a')
                self.graph.edge(column, self.table[column]['b'], label='b')

    def show_table(self):
        self._form_image()
        self.graph.view()

    def _validate_command(self, command: str):
        unique_chars = set(command)
        if any(map(lambda char: char not in self.alphabet, unique_chars)):
            return False
        return True

    def run(self, command: str):
        self._form_image()

        if not self._validate_command(command):
            print('Команда введена неверно!')
            return

        steps = {node: [] for node in self.table.columns}
```

Продолжение листинга

```
        current_state = self.initial_state
        for i, step in enumerate(command):
            steps[current_state].append(i + 1)
            current_state = self.table.at[step, current_state]

        for node in steps:
            if len(steps[node]) != 0:
                self.graph.node(node,        xlabel=f'{steps[node]}        шаг',
style='filled', colorscheme="SVG",
                                fillcolor='lightblue')


def main():
    table = Table(TABLE, INITIAL_STATE, FINAL_STATES, TRASH_STATES)
    table.run('abbbabbb')
    table.show_table()


if __name__ == '__main__':
    main()
```