Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Институт информационных технологий

Кафедра ИС

# ОТЧЁТ

по лабораторной работе №1

ИССЛЕДОВАНИЕ ДЕТЕРМИНИРОВАННОГО КОНЕЧНОГО АВТОМАТА

Выполнил:

ст. гр. ИС/б-21-2-о

Мовенко К. М.

Проверил:

Карлусов В. Ю.

Севастополь

2024

# ЦЕЛЬ РАБОТЫ

Научиться производить построение детерминированных конечных автоматов (ДКА), допускающих определённые цепочки символов языка. Освоить приёмы описания конечных автоматов (КА) в виде графов, таблиц переходов и регулярных выражений. Научиться выполнять построения ДКА по недетерминированным конечным автоматам (НКА). Научиться проводить построение минимальных детерминированных конечных автоматов (МДКА).

# Задание

* 1. Выполнить разметку регулярного выражения согласно варианту задания, по необходимости минимизировать. Если КА не является детерминированным, необходимо привести его к детерминированному виду;
  2. Построить таблицу переходов МДКА. Выполнить минимизацию по таблице переходов;
  3. Построить граф конечного автомата для визуального определения недостижимых состояний и, если таковые обнаружатся, окончательно минимизировать ДКА, построив окончательную таблицу МДКА;
  4. Написать программу на языке С/С++, реализующую КА. Довести её до успешной компиляции;
  5. Разработать тестовые последовательности литер, допускаемые и отвергаемые КА. Найти допускаемые последовательности минимальной длины. Предусмотреть соответствующие диагностические сообщения. Каждая цепочка литер должна быть снабжена соответствующей последовательностью вершин КА, которую он проходит в процессе анализа цепочки.

Вариант 43 – CA{B}B{BA˅BB˅BC} ˅ C{BB˅BA}

# ХОД РАБОТЫ

Была выполнена разметка регулярного выражения (рисунок 1).

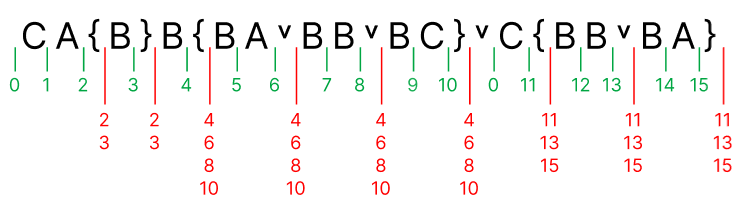


Рисунок 1 – Разметка регулярного выражения

Полученная разметка была пересчитана с учётом правил минимизации (рисунок 2).

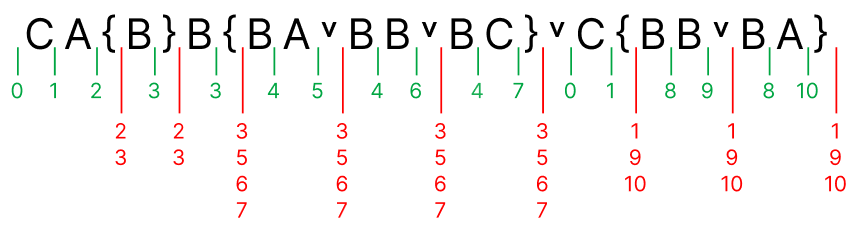


Рисунок 2 – Разметка регулярного выражения (минимизация)

КА, соответствующий регулярному выражению, был представлен в виде таблицы переходов (таблица 1).

Таблица 1 – Таблица переходов НКА

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1к** | **2** | **3к** | **4** | **5к** | **6к** | **7к** | **8** | **9к** | **10к** |
| **A** |  | 2 |  |  | 5 |  |  |  | 10 |  |  |
| **B** |  | 8 | 3 | 3,4 | 6 | 4 | 4 | 4 | 9 | 8 | 8 |
| **C** | 1 |  |  |  | 7 |  |  |  |  |  |  |

Полученный КА является недетерминированным (НКА). Образ функции перехода неоднозначен для столбца 3. Для этого было произведено последовательное объединение состояний, построено дерево приведения КА к однозначности функции переходов (рисунок 3).

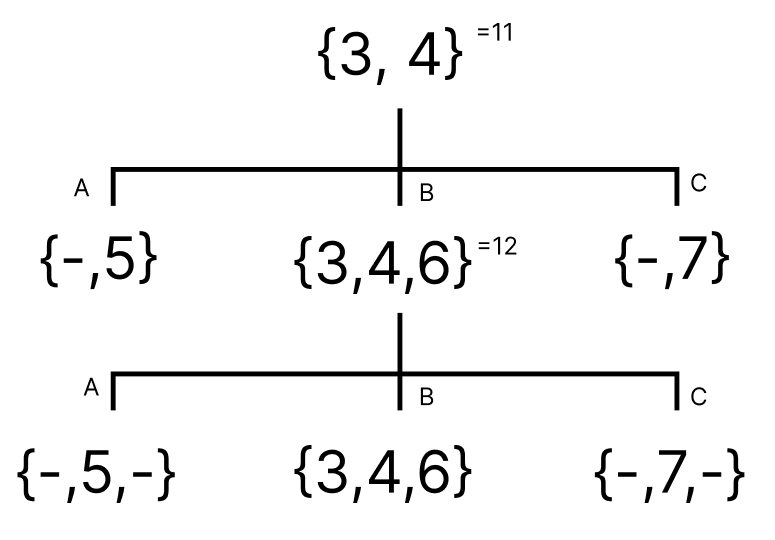


Рисунок 3 – Приведение НКА к ДКА

Таблица переходов была перестроена под ДКА (таблица 2).

Таблица 2 – Таблица переходов ДКА

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1к** | **2** | **3к** | **4** | **5к** | **6к** | **7к** | **8** | **9к** | **10к** | **11к** | **12к** |
| **A** |  | 2 |  |  | 5 |  |  |  | 10 |  |  | 5 | 5 |
| **B** |  | 8 | 3 | 11 | 6 | 4 | 4 | 4 | 9 | 8 | 8 | 12 | 12 |
| **C** | 1 |  |  |  | 7 |  |  |  |  |  |  | 7 | 7 |

Таблица переходов была минимизирована следующим образом:

* 0 → 0;
* 1 → 1;
* 2 → 2;
* 3 → 3;
* 4 → 4;
* {5, 6, 7} → 5;
* 8 → 6;
* {9, 10} → 7;
* {11, 12} → 8;

Таблица 3 – Таблица переходов МДКА

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1к** | **2** | **3к** | **4** | **5к** | **6** | **7к** | **8к** |
| **A** |  | 2 |  |  | 5 |  | 7 |  | 5 |
| **B** |  | 6 | 3 | 8 | 5 | 4 | 7 | 6 | 8 |
| **C** | 1 |  |  |  | 5 |  |  |  | 5 |

По таблице МДКА был построен граф конечного автомата (рисунок 4).

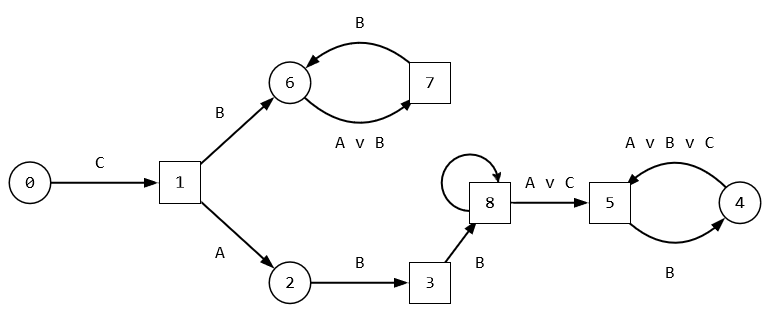


Рисунок 4 – Граф конечного автомата

Была написана программа на языке Python, которая считывает из файла тестовые последовательности символов и проверяет по таблице переходов допустимы ли они конечным автоматом.

Листинг 1 – Код проверки допустимости КА

*# таблица переходов*transitions = [  
 [None, 2, None, None, 5, None, 7, None, 5],  
 [None, 6, 3, 8, 5, 4, 7, 6, 8],  
 [1, None, None, None, 5, None, None, None, 5]  
]  
  
*# множество заключительных состояний КА*F = [1, 3, 5, 7, 8]  
  
  
def validate(string):  
 print(f'\nВходная строка - {string}')  
  
 q = 0 *# начальное состояние  
  
 # проходим по каждому символу строки, пока не достигнем конца или недопустимого состояния* for s in string:  
 if s == 'a':  
 q = transitions[0][q]  
 elif s == 'b':  
 q = transitions[1][q]  
 elif s == 'c':  
 q = transitions[2][q]  
 else:  
 q = None  
  
 *# print(s, q)* if q == None:  
 *# print(f'{s} - недопустимый символ')* break  
 *# else:  
 # print(f'{s} - допустимый символ')* if q == None:  
 print('СТРОКА НЕДОПУСТИМА')  
 elif q in F:  
 print('СТРОКА ДОПУСТИМА')  
 else:  
 print('СТРОКА НЕДОПУСТИМА (преждевременное окончание)')  
  
  
*# ОСНОВНОЙ КОД*print('Регулярное выражение - CA{B}B{BA˅BB˅BC} ˅ C{BB˅BA}')  
try:  
 with open("input.txt", "r") as file:  
 for line in file:  
 validate(line.strip())  
except FileNotFoundError:  
 print("Файл не найден")

На вход был подан ряд тестовых последовательностей. Программа верно определила допустимость каждой последовательности литер (рисунок 5). Минимальная длина допустимой последовательности – 1 литера.

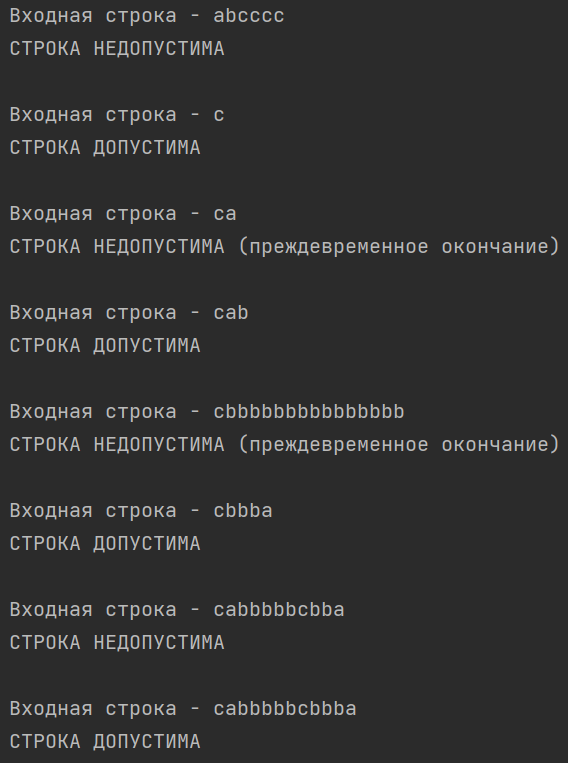


Рисунок 5 – Тестирование программы

# ВЫВОД

В ходе работы были получены знания по построению детерминированных конечных автоматов, допускающих определённые цепочки символов языка. Освоены приёмы описания конечных автоматов в виде графов, таблиц переходов и регулярных выражений.