UNIVERSITY OF AMSTERDAM

ASSIGNMENT 2

Lexicographic Analysis

February 21, 2021

Students:
René Kok
13671146

Aram Mutlu
13574116

Lecturer:
Dhr. dr. C.U. Grelck

Course:
Compiler Construction

Course code:
5062COMP6Y

1 Introduction

This report provides information regarding the second assignment "Lexicographic Analysis" of the course Compiler Construction lectured by Dhr. dr. C.U. Grelck. The goal of this assignment is to create a Direct-coded Scanner of the regex "(ac|ab)*". To accomplish this we will pay particular attention to Thompson's Construction, Subset Construction, Hopcroft's Algorithm and as last create a Direct-coded Scanner. In each section we will go deeper in the steps taken to accomplish the goal of the assignment.

2 Assignment

2.1 Thompson's Construction

The first step in creating a direct code scanner is to create a non-deterministic finite automaton (NFA) to recognize the regular expression "(ac|ab)*" using Thompson's Construction.
It starts with defining an automaton that recognizes "ac" (see Figure 1).

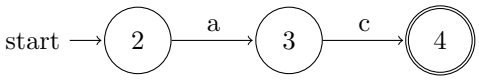

Figure 1: Step 1 creating NFA for "ac" → (ac|ab)*

Then the automaton for recognizing "ab" is defined (see Figure 2).

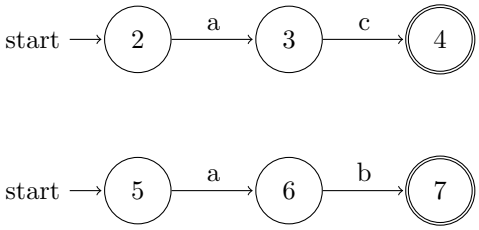

Figure 2: Step 2 creating NFA for (ab) → (ac|ab)*

Because the regex expression matches either "ab" or "ab", the two automatons are linked by an alternation (see Figure 3).

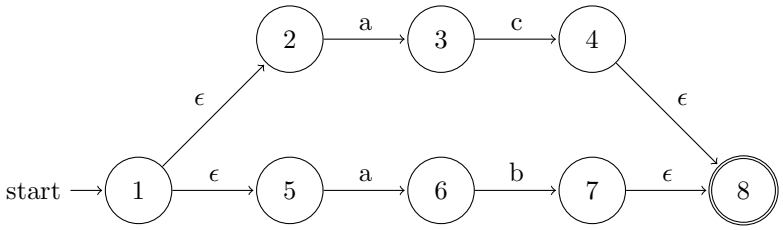

Figure 3: Step 3 creating NFA for (ac|ab) → (ac|ab)*

As the last step of Thompson's construction, the alternations are added to the automaton, creating the final automaton for the regex expression (see Figure 4).

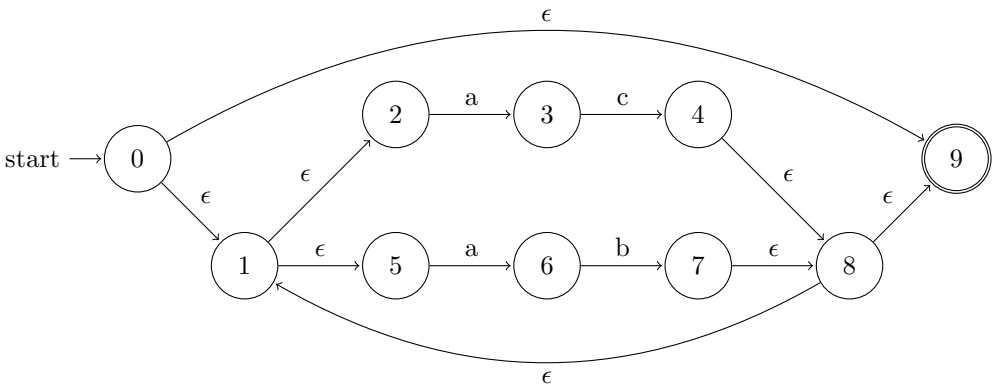

Figure 4: Final non-deterministic finite automaton for (ac|ab)*

2.2 Subset Construction

Based on the NFA diagram (see Figure 4), the following subset construction has been drawn up:

State	**ϵ-closures**
{0}	{0, 1, 2, 5, 9}
{1}	{1, 2, 5}
{2}	{2}
{3}	{3}
{4}	{4, 8, 9, 1, 2, 5}
{5}	{5}
{6}	{6}
{7}	{7, 8, 9, 1, 2, 5}
{8}	{8, 9, 1, 2, 5}
{9}	{9}

Table 1: Subset construction from NFA

With this subset construction the following DFA has been set up:

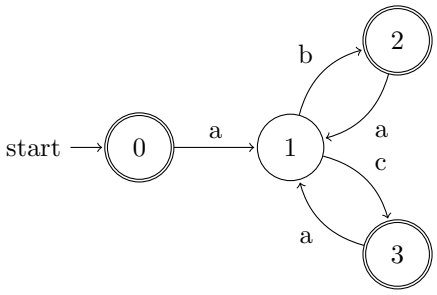

Figure 5: Deterministic finite automaton for (ac|ab)*

2.3 Hopcroft's Algorithm

The DFA diagram (see Figure 5) has been minimized using Hopcroft's algorithm and resulted in the following diagram:

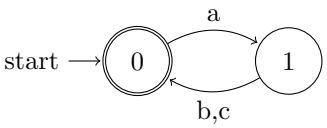

Figure 6: Minimized deterministic finite automaton for (ac|ab)*

2.4 Direct-coded Scanner

Following direct-coded scanner has been constructed based on on the minimized DFA diagram (see Figure 6):

```
/**
 * Authors: Rene Kok & Aram Mutlu
 * Scanner for the (ab|ac)* regex
 */
char *scanner(FILE *stream)
{
    static char buffer[max]; // fixed size buffer for token
    int pos = 0;
    char c;

// The first start/init state scans for the first character 'a'
state_init:
    switch (c = getc(stream))
    {
    case 'a':
        buffer[pos++] = c;
        goto state_0;
    default:
        goto state_err;
    }

// State to scan for the character 'b' OR 'c'
state_0:
    switch (c = getc(stream))
    {
    case 'b':
        buffer[pos++] = c;
        goto state_1;
    case 'c':
        buffer[pos++] = c;
        goto state_1;
    default:
        goto state_err;
    }

// State to scan for the character 'a' or end of file
state_1:
    switch (c = getc(stream))
    {
    case 'a':
        buffer[pos++] = c;
        goto state_0;
    default:
        goto state_succ;
    }

// State to return the buffer when reached
state_succ:
    return buffer;

// State to return NULL when error occures
state_err:
    return NULL;
}
```