Compiles construction

Assignment 2

Hamza ait messaoud

12916129

&

Michel Rummens

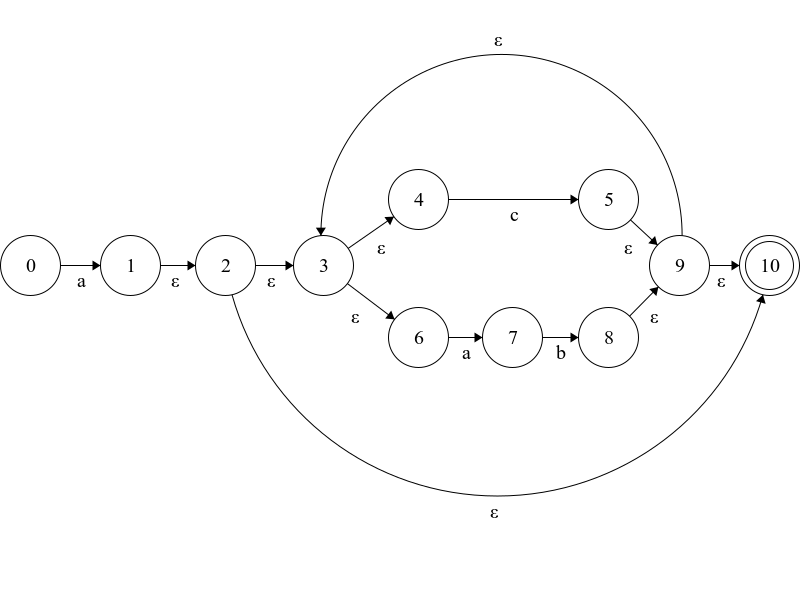
13108093

# Regex to NFA (using Thompson’s Construction)

We are going to generate a NFA based on the regex a(c|ab)\*, but first we will split the regex in smaller automatons.

|  |
| --- |
| 1. [a] -> a(c|ab)\* |
|  |
|  |
| 2. [c|ab] -> a(c|ab)\* |
|  |
|  |
| 1. [(c|ab)\*] -> a(c|ab)\* |
|  |

We can now complete the NFA from the parts we created.



NFA to DFA (using Subset Construction)

### E-closures from states given in the NFA:

For each state in Q’, find the possible set of states for each input symbol using transition function of NFA. If this set of states is not in Q’, add it to Q’.

|  |  |
| --- | --- |
| state | e-closures |
| 0 | {0} |
| 1 | {1, 2, 3, 4, 6, 10} |
| 2 | {2, 3, 4, 6, 10} |
| 3 | {3, 4, 6} |
| 4 | {4} |
| 5 | {5, 9, 3, 4, 6, 10} |
| 6 | {6} |
| 7 | {7} |
| 8 | {8, 9, 3, 4, 6, 10} |
| 9 | {9, 3, 4, 6, 10} |
| 10 | {10} |

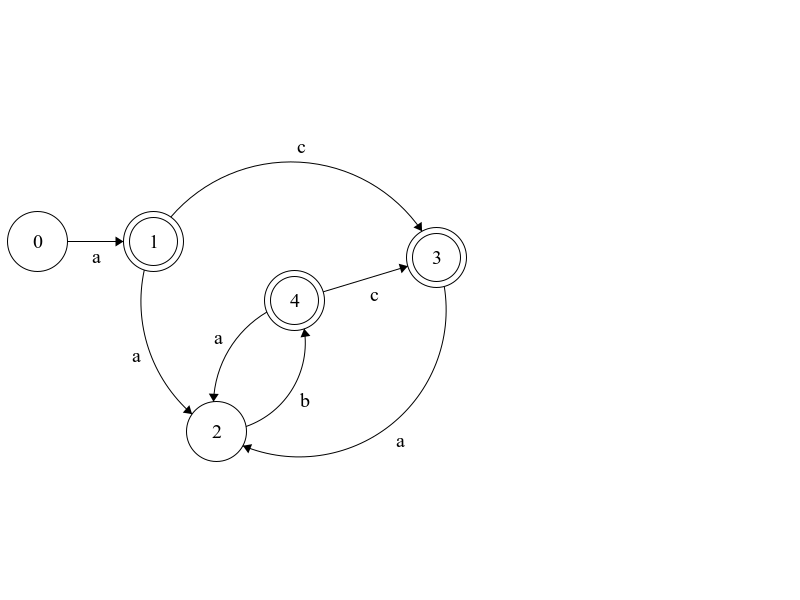
### Transition table

Currently, state in Q’ is q0, find moves from q0 on the input symbols using transition function of NFA and update the transition table of DFA.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | N | states | a | a | b | c |
| -> | 0 | [0] | [1, 2, 3, 4, 6, 10] | - | - | - |
| \* | 1 | [1, 2, 3, 4, 6, 10] | - | 7 | - | [5, 9, 3, 4, 6, 10] |
|  | 2 | [7] | - | - | [8, 9, 3, 4, 6, 10] | - |
| \* | 3 | [5, 9, 3, 4, 6, 10] | - | 7 | - | [5, 9, 3, 4, 6, 10] |
| \* | 4 | [8, 9, 3, 4, 6, 10] | - | 7 | - | [5, 9, 3, 4, 6, 10] |

### DFA-diagram:

The DFA based on the transition table

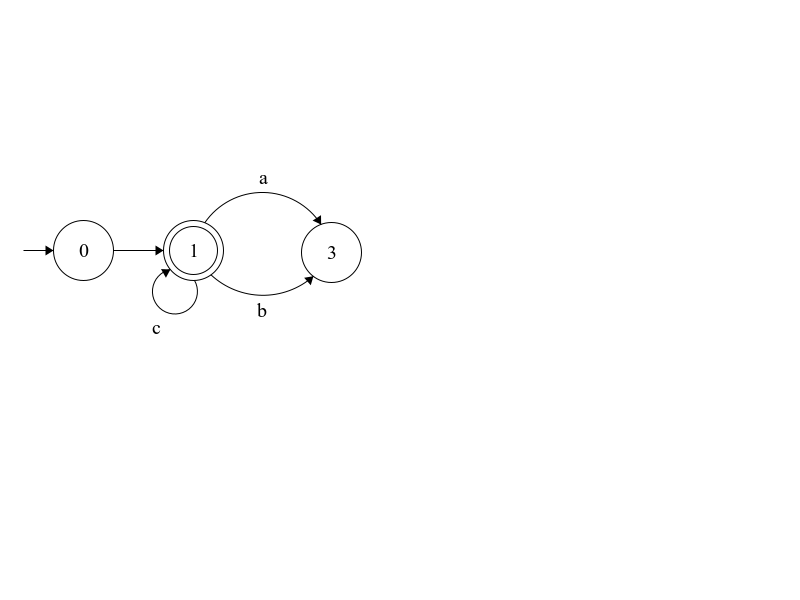


# DFA to Min DFA (based on Hopcroft’s Algorithm)

Group states that behave identically

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | N |  | a | a | b | c |
| -> | 0 | [0] | x | - | - | - |
| \* | 1 | [1, 3, 4] | - | x | - | x |
|  | 2 | [2] | - | - | x | - |

### Min DFA diagram



# Direct-coded Scanner

Now that we created the min-DFA we can create a DC scanner based on that data.

/\*\*

\* Scanner for the a(c|ab)\* regex

\*

\* @param char \* the string to parse

\* @param int the length of the string

\* @return char \*|NULL

\*/

char \*scanner(char \*stream, int pos = 0)

{

char c;

// label to scan for a character

state\_init:

c = stream[pos++];

if (c == 'a') goto state\_ca;

if (pos == 1) goto state\_err;

else goto state\_done;

// label to scan for a|ab characters recursively

state\_ca:

c = stream[pos++];

if (c == 'c') goto state\_ca;

else if (c == 'a') goto state\_ab;

else goto state\_done;

// label to scan for b character

state\_ab:

c = stream[pos++];

if (c == 'b') goto state\_ca;

else goto state\_err;

// label to check if the end of the regex was reached

state\_done:

c = stream[pos-1];

if (c == 32) goto state\_succ;

else goto state\_err;

// label for success

state\_succ:

return stream;

// label for errors

state\_err:

return NULL;

}