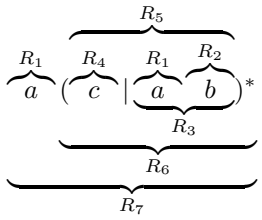
Coco - Assignment 2

S.R.W. van Kampen en R.A.J. Wacanno

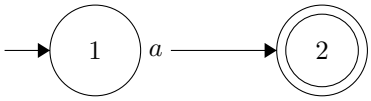
21 februari 2020

2.1 Thompson's Construction

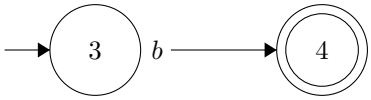
Staring from the atomic components, the algorithm constructs automatons $R_{1,2,4}$. For each of the operators present in the regular expression, the algorithm combines these automatons using prescribed constructions which eventually results in automaton R_7



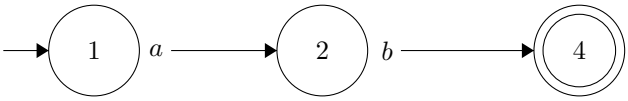
R_1



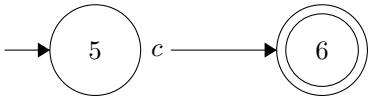
R_2



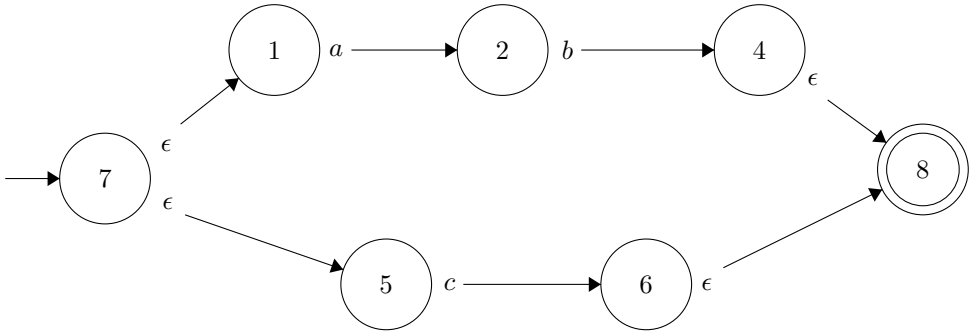
R_3



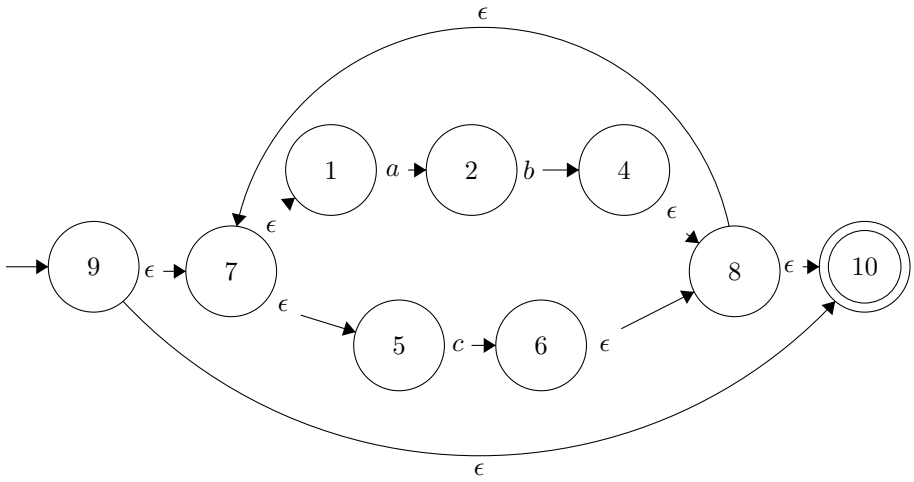
R_4



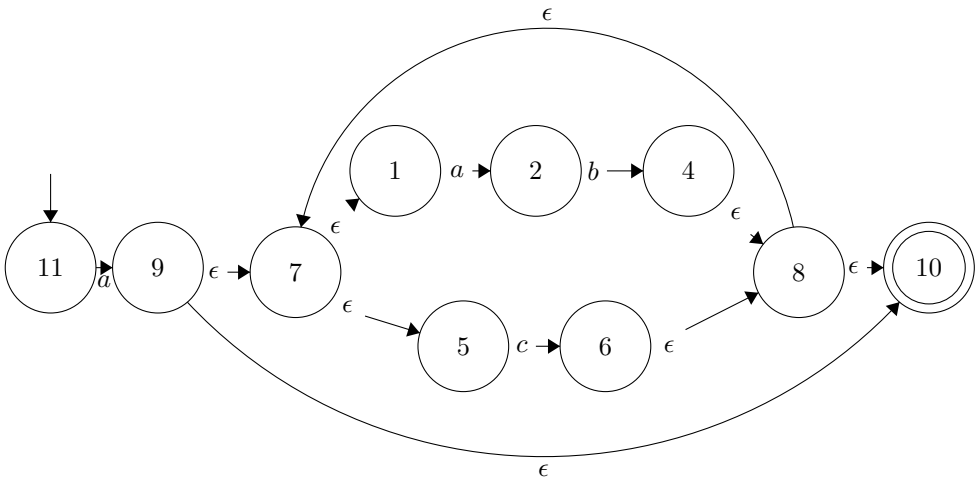
R_5



R_6

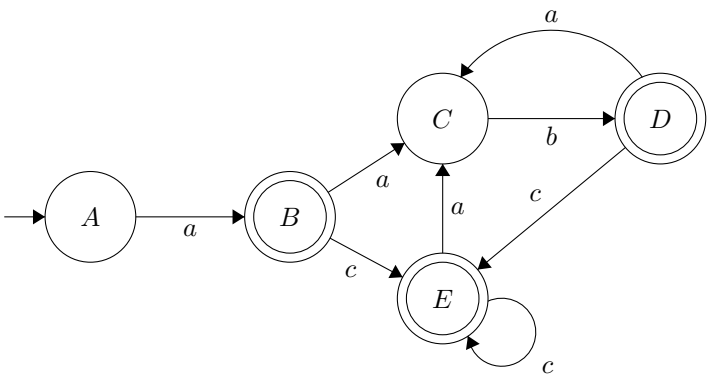


R_7



2.2 Subset Construction

NFA state	DFA state	Next state a	b	c
{11}	A	B	-	-
{9, 7, 1, 5, 10}	B	C	-	E
{2}	C	-	D	-
{10, 4, 8, 1, 7, 5}	D	C	-	E
{6, 8, 10, 1, 5, 7}	E	C	-	E



2.3 Hopcroft's Algorithm

In Hopcroft's algorithm, we look for sets of states that perform the same actions given any character of the alphabet, and are of the same type (either accepting or non-accepting). We start with the following sets:

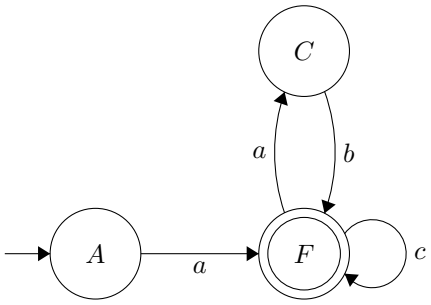
$$P = \{\{B, D, E\}, \{A, C\}\}$$

Now we look at the first set, $\{B, D, E\}$. When we give it elements of our alphabet, do these states perform the same? Yes! An `a` always sends us to state C, a `b` always causes the DFA to reject its input and a `c` always sends us to state E. Therefore, we return the whole set $\{B, D, E\}$. Setting T equal to $T \cup \text{Split}(\{B, D, E\})$ yields $T = \{B, D, E\}$.

We now look at $\{A, C\}$. These states do behave differently on different inputs; for instance, A rejects immediately on the input `b`, while C sends us to D on a `b`. Therefore, we return two sets, $\{\{A\}, \{C\}\}$. Setting T equal to $T \cup \text{Split}(\{A, C\})$ yields $T = \{\{B, D, E\}, \{A\}, \{C\}\}$.

We've processed all sets in P, and now let it equal T. P will now no longer change, and we've now got sets of states that can be coalesced. A and C can stay as they are, and we'll coalesce B, D, and E to a new state, F. Observe that F should be an accepting state that sends us to itself on input `c` and sends us to C on input `a`. Instead of state A sending us to B when it receives an `a`, it now needs to send us to F, and similarly C needs to send us to F on receiving a `b`.

The resulting DFA is as follows:



2.4 Direct-coded scanner

Below, a direct-coded scanner is implemented as a C program.

The states are represented here as C labels, and transitions between states are achieved using `goto` statements. The behaviour of each state depends on the `c` retrieved from the input `stream`. If the input matches any of the characters specified in an `if` statement, the corresponding `goto` to another state label is taken, i.e. if a valid character from the alphabet is encountered, the code advances to the correct state and, in case of accepting states, when an *end of file* character is received, the code returns `EXIT_SUCCESS` (similarly to when a separation character is found in the example in the slides). If the character is not matched by any of the aforementioned statements, the code returns `EXIT_FAILURE`.

```
int main()
{
    FILE *stream = stdin;
    int c;
    state_A:
        c = getc(stream);
        if (c == 'a') goto state_F;
        else goto state_err;

    state_F:
        c = getc(stream);
        if (c == 'a') goto state_C;
        if (c == 'c') goto state_F;
        if (c == EOF) goto state_succ;
        else goto state_err;

    state_C:
        c = getc(stream);
        if (c == 'b') goto state_F;
        else goto state_err;


    state_succ: return EXIT_SUCCESS;
    state_err: return EXIT_FAILURE;
}
```