Programmation Avancée et Compilation 2001
(Solutions)

4 Finite-State Automata

Exercise 6

(a) The automaton is deterministic: there is no state with equally labelled outgoing transitions or ϵ-productions.

(b) The *start-state* of the automaton is state 0.

(c) The *final states* are given by states 1 and 2.

(d) The string `110100` is accepted by a run passing through states $0, 2, 3, 3, 2, 2$.

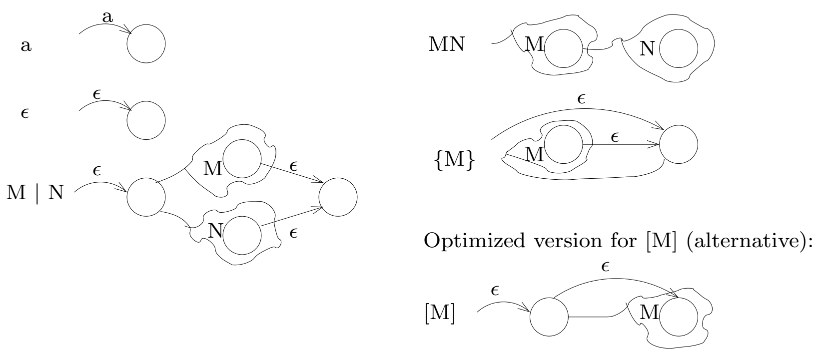
(e) The two shortest strings to be accepted are 0 and 1.

(f) There is no upper bound on the length of the strings the automaton accepts. This is due to the presence of loops containing the final state 2, which are reachable from the start-state.

(g) The automaton accepts all strings made up of 0s and 1s containing zero or an odd number of 1s, where in the latter case the string has to start with a 1. That is, it accepts all binary numbers without superfluous leading 0s that contain zero or an odd number of 1s.

Exercise 7

The lecture presents the following conversion algorithm of regular expressions into NFAs without start-state:



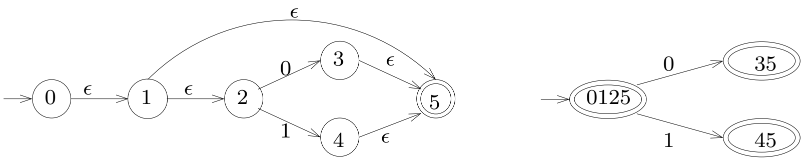
The final states are still not marked as such. In order to obtain an NFA once the transformation is finished, one attaches a start-state to the front and marks the final state as such.

In order to obtain a DFA from an NFA, we apply a *closure* construction, uniting all those states that can be reached by the same transitions. For this, two operations are necessary:

- **closure**(s) computes all states that are reachable from s by any number of ϵ-transitions, and
- **DFAedge**(s, c) computes all states that are reachable by a c-transition ($c \neq \epsilon$) from s.

We obtain the new start-state by computing **closure**(old start-state). Then we successively apply **closure**(**DFAedge**(s, c)), in order to obtain the other new states. Those new states containing one or more old final states yield the new final states. A common way when performing this construction by hand is to use tables.

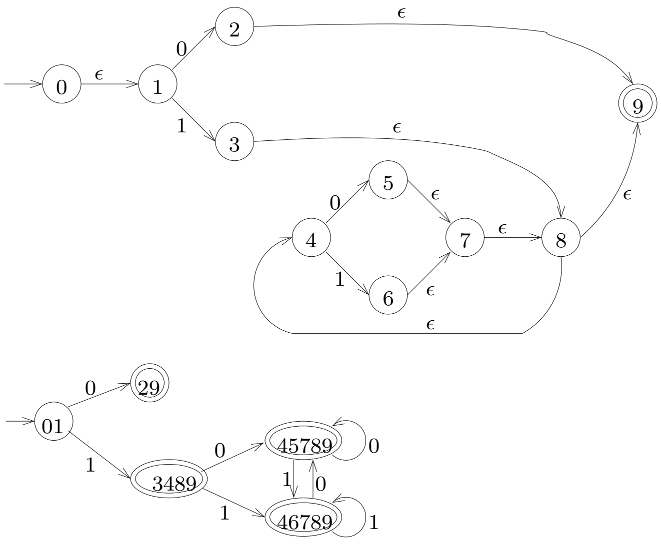
(a) The conversion algorithms yield the following NFA and DFA:



Remark: We have used the alternative version for [M]. Otherwise, when expressing [M] in terms of $\epsilon \mid \mathsf{M}$, the NFA has six states with an ϵ-transition between states 5 and 6, and state 6 being final instead of state 5.

(b) The conversion algorithms yield the following NFA and DFA:

(c) The conversion algorithms yield the following NFA and DFA:



Remark: Note that for every regular expression, there exists an arbitrary number of equivalent NFAs. Different construction algorithms may well yield different NFA's. These are *equivalent*, however, that is, they accept the same language.

Note further that the DFAs we obtain from the NFAs are not yet *minimal*, that is they still can have more states than the smallest equivalent DFA. For each DFA there exists a unique minimal DFA accepting the same language, and it can be constructed by yet another algorithm.