Tik-79.148 **Spring 2001**
Introduction to Theoretical Computer Science
Tutorial 4
Solutions to Demonstration Exercises

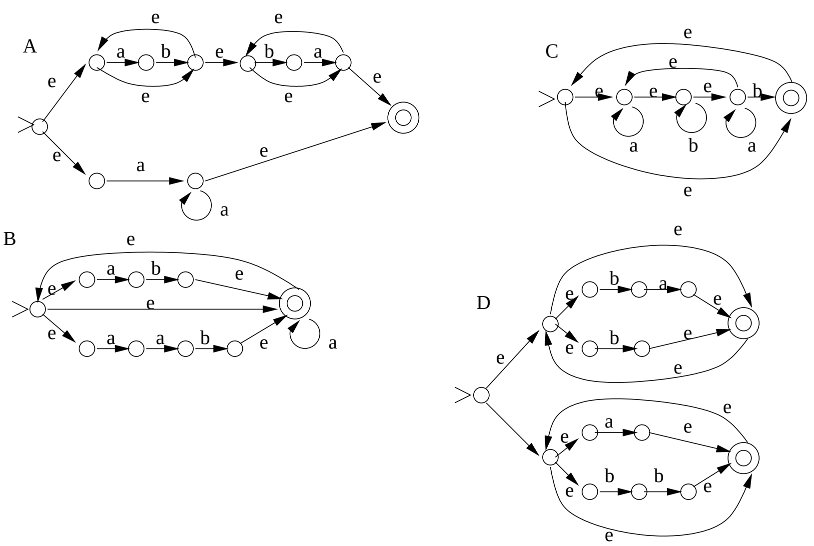
A deterministic finite automaton (DFA) is a tuple $M = (K, \Sigma, \delta, s, F)$, where K is a finite set of states, Σ is the alphabet, δ a transition function $K \times \Sigma \rightarrow K$, $s \in K$ is the initial state and $F \subseteq K$ is set of accepting states.

Each state machine corresponds to a regular language. The state machine starts from the initial state and goes through the input, letter by letter, always making a transition to a new state according to δ. If the state machine is in an accepting state, when the end of the input has been reached, the word belongs to the language (the automaton accepts the language), otherwise it does not belong to the language (the automaton rejects the word).

A nondeterministic finite automaton (NFA) $M = (K, \Sigma, \Delta, s, F)$ differs from the deterministic one in its transition relation. In a nondeterministic automaton the relation $\Delta \subseteq K \times \Sigma \cup \{e\} \times K$ allows many transitions for one input, and furthermore empty transitions (e-transitions), where the automaton can spontaneously move to a new state without consuming input, are also allowed. A nondeterministic automaton accepts a word, if there is some route from the initial state to an accepting state.

In applications it is not possible to guess the correct route to an accepting state. Thus, in practice one must go through all possible routes systematically. Nondeterministic automata are still used because there are efficient algorithms for creating a NFA from a regular expression. If execution speed is critical the nondeterministic automaton can always be converted to a corresponding deterministic automaton.

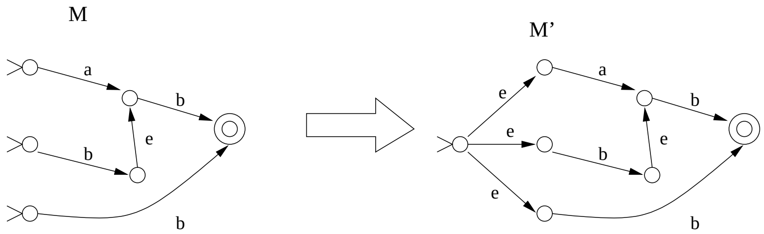
4.



5. Let $M = (K, \Sigma, \Delta, S, F)$, where $S \subseteq K$ is the set of initial states. For a NFA like this, a corresponding ordinary nondeterministic machine can be constructed:

$$M' = (K \cup \{s'\}, \Sigma, \Delta', s', F), s' \notin K, \text{ where}$$
$$\Delta' = \Delta \cup \{(s', e, s) \mid s \in S\}$$

The idea of the construction is that M' is otherwise the same automaton as M, but one state has been added. This new state is set to be the initial state, and e-transitions to all initial states of M are added. Now M' is an automaton with one initial state, and both machines clearly accept the same language, i.e. $L(M) = L(M')$. The suggested extension does not increase the expressiveness of NFA:s.



6. The claim:

$$(q, xy) \vdash_M^* (p, y) \text{ if and only if } (q, x) \vdash_M^* (p, e)$$

By definition $(q, w) \vdash_M (q', w')$ if and only if there exists $u \in \Sigma \cup \{e\}$ such that $w = uw'$ and $(q, u, q') \in \Delta$. In other words, from the configuration (q, w), the configuration (q', w') is reachable in one step only if $w = w'$ and there is an empty transition from q to q' or w' is the suffix of w, and from the state q there is transition to q' with the first symbol of w. The notation $\vdash_M^*$ denotes the reflexive and transitive closure of $\vdash_M$.

If $(q, xy) \vdash_M^* (p, y)$, then by definition there exists a sequence of configurations $(q, xy) \vdash_M (q_1, w_1 y), \vdash_M (q_2, w_2 y) \vdash_M \cdots \vdash_M (q_n, w_n y) \vdash_M (p, y)$, where

$$
\begin{aligned}
w_0 &= x \\
w_i &= u w_{i+1}, u \in \Sigma \cup \{e\} \ .
\end{aligned}
$$

Because on each step the new state is defined by the first symbol in the input, the other symbols of the string do not affect the transition. Furthermore, as the length of the input cannot grow at any point, it is not possible to peek at some symbol of the string y, and write them back at the tape. Therefore, during the execution above the string y is not used in any transition and all transitions are chosen on basis of the letters x.

Because only x affects the execution of the automaton, exactly the same transitions could be performed if y where replaced with the empty string e. So

$$(q, xy) \vdash_M^* (p, y) \Rightarrow (q, x) \vdash_M^* (p, e)$$

The proof in the other direction can be done using the same arguments.

7. The states of the joint state machine are all possible pairs, which can be constructed from the states of the component machines. E.g., if $K_1 = \{q_1, q_2, q_3\}$ and $K_2 = \{p_1, p_2\}$, then the states of the joint machine are

$$K = \{(q_1, p_1), (q_2, p_1), (q_3, p_1), (q_1, p_2), (q_2, p_2), (q_3, p_2)\} \ .$$

Conceptually the transitions of the machines are split into two classes, internal and external. A transition is internal if the label does not belong to the alphabet of the other machine. In the exercise the only internal transition of M_1 is $q_3 \stackrel{int_1}{\to} q_0$. M_2 has four internal transitions, those with the label int_2. If the label of a transition appears in the alphabet of both machines the transition is external.

The rules for forming the joint transition relation essentially say that the state machines synchronize on external transitions. A state machine can always perform an internal transition, but an external transition can performed only if both machines perform the same transition at the same time.

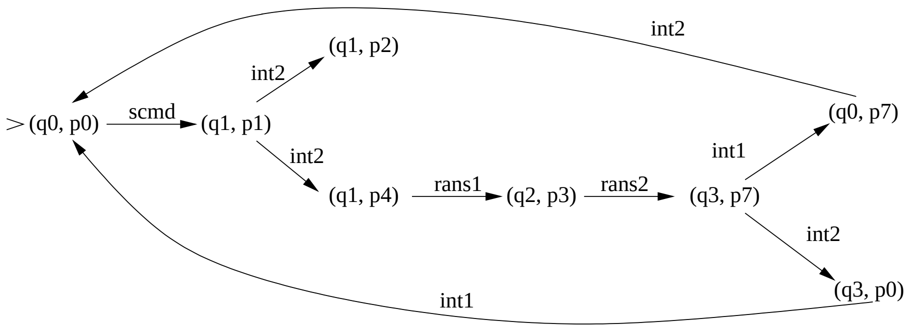
The states of component state machines are:

$$
\begin{aligned}
K_1 &= \{q_0, q_1, q_2, q_3\} \\
K_2 &= \{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7\}
\end{aligned}
$$

The joint state machine contains $4 \times 8 = 32$ states. Many of these states can never be reached, so the state space shrinks to:

$$K = \{(q_0, p_0), (q_1, p_1), (q_1, p_2), (q_1, p_4), (q_2, p_3), (q_3, p_7), (q_0, p_7), (q_3, p_0)\}$$

The transitions of the joint machine are shown in the picture below.



We can see from the picture that there is no arc forward from the state (q_1, p_2). This state can be reached from the initial state with the sequence

$$(q_0, p_0) \stackrel{scmd}{\rightarrow} (q_1, p_1) \stackrel{int_2}{\rightarrow} (q_1, p_2) \ ,$$

so the system contains a reachable deadlock.

There are no final states defined for the machines in this exercise because usually when protocols are analyzed we are not interested simple executions but we consider infinite sequences of messages. Ordinary state machines only allow finite input (albeit as long as we want), and to recognize infinite words stronger classes of automata are used, usually Büchi automata. They, however, are not a topic of this course.