2.4 Minimizing finite state automata

It can be shown that every finite state automaton has a unique equivalent (recognizes the same language) minimal (with respect to number of states) finite state automaton.

Minimizing a given finite state automaton (determining the minimal equivalent automaton) is an important problem for both theory and practice: one can thereby e.g. check, whether two given automata recognize the same language.

The following efficient method can be used to solve the problem. The basic idea is to identify those states, starting from which there is no difference in the behaviour of the automaton.

Let

$$M = (Q, \Sigma, \delta, q_0, F)$$

be a finite state automaton.

Let us expand the transition function of M from from single symbols to strings: if $q \in Q$, $x \in \Sigma^*$, denoted by

$$\delta^*(q, x) = \text{ the } q' \in Q, \text{ for which } (q, x) \vdash_M^* (q', \varepsilon).$$

The states q and q' are *equivalent*, denoted by

$$q \equiv q',$$

if for all $x \in \Sigma^*$ we have

$$\delta^*(q, x) \in F \quad \text{if and only if} \quad \delta^*(q', x) \in F;$$

in other words, if the automaton accepts exactly the same strings starting from q and q'.

A milder equivalence condition: the states q and q' are *k-equivalent*, denoted by

$$q \overset{k}{\equiv} q',$$

if for all $x \in \Sigma^*$, $|x| \leq k$, we have

$$\delta^*(q, x) \in F \quad \text{if and only if} \quad \delta^*(q', x) \in F;$$

that is, if no string of length at most k can differentiate q and q'.

Clearly we have:

$$\begin{array}{lll} \text{(i)} & q \overset{0}{\equiv} q' & \text{iff} \quad \text{both } q \text{ and } q' \text{ are final states or neither is; and} \\ \text{(ii)} & q \equiv q' & \text{iff} \quad q \overset{k}{\equiv} q' \text{ for all } k = 0, 1, 2, \ldots \end{array} \qquad (1)$$

The minimization algorithm presented is based on refining the k-equivalence classes of the states of the automaton to $(k+1)$-equivalence classes until complete equivalence is attained.

Algorithm MIN-FA [Minimizing a finite state automaton]

Input: A finite state automaton $M = (Q, \Sigma, \delta, q_0, F)$.

Output: The finite state automaton $\widehat{M}$ that is equivalent to M and has the minimum possible number of states.

Method:

1. [Removing unnecessary states.] Remove from M all states that cannot be reached with any input string.

2. [0-equivalence.] Partition the remaining states of M into two classes: final states and non-final states.

3. [k-equivalence $\rightarrow$ $(k+1)$-equivalence.] Check whether the transitions from the states in the same equivalence class always lead to the same equivalence class with the same input symbol. If yes, the algorithm terminates and the states $\widehat{M}$ correspond to the *classes* of states of M.

Otherwise refine the partition by splitting each class that violated the previous condition into new, smaller equivalence classes according to which classes the transitions lead to with each possible input symbol, and repeat step 3 with the new partition.

It is easy to show that at the start of the $(k+1)$th iteration of step 3 ($k = 0, 1, \ldots$) two states are in the same equivalence class if and only if they are k-equivalent.

It follows that as the algorithm terminates when the partition can no longer be refined, the classes are exactly the $\equiv$-equivalence classes of the states of M (cf. property (1.ii)).

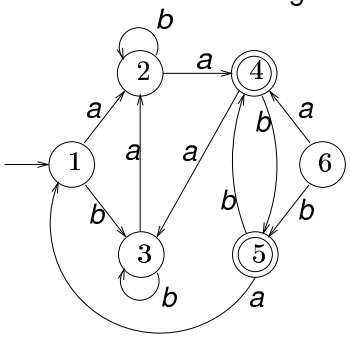
The algorithm always terminates, since in each iteration of step 3 except in the last one at least one equivalence class is split.

Theorem 2.1 Algorithm MIN-FA determines the minimal finite state automaton $\widehat{M}$ equivalent to the given finite state automaton M. Further, this automaton is unique up to the naming of the states. □

Example. We minimize the following automaton:

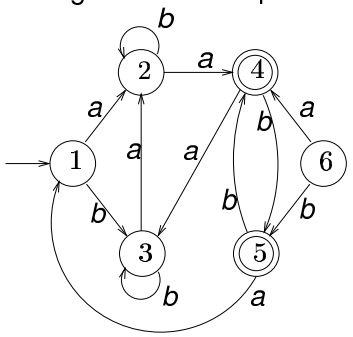
In state 1 the unreachable state 6 is removed.

In stage 2 the states 1–5 of the automaton are partitioned into non-final states (class I) and final states (class II) and the transitions are examined with regard to the partition:



			a	b
I :	$\rightarrow$	1	2, I	3, I
		2	4, II	2, I
		3	2, I	3, I
II :	$\leftarrow$	4	3, I	5, II
	$\leftarrow$	5	1, I	4, II

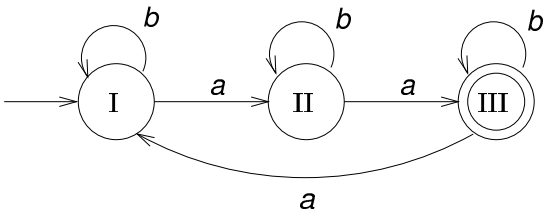
There are now two kinds of states ({1,3} and {2}) in class I, so the partition must be refined and the transitions again examined with regard to the new partition:



			a	b
I :	$\rightarrow$	1	2, II	3, I
		3	2, II	3, I
II :		2	4, III	2, II
III :	$\leftarrow$	4	3, I	5, III
	$\leftarrow$	5	1, I	4, III

Now the states in each equivalence class are similar, so the minimization algorithm terminates.

The state chart of the minimum automaton is:

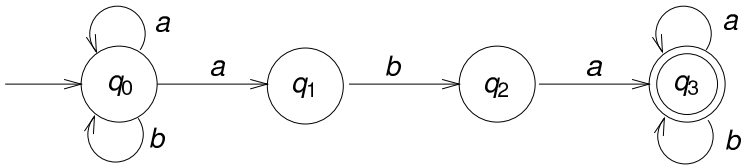


2.5 Nondeterministic finite state automata

Nondeterministic finite state automata are in other respects like deterministic ones, but their transition function δ does not yield a single new state given the old state and an input symbol, but rather a *set* of possible next states. A nondeterministic automaton accepts its input, if *some* possible sequence of states leads to an accepting final state.

Even though nondeterministic automata cannot be directly realized as computer programs, they are an important *description formalism* for decision problems.

Example. A nondeterministic automaton that examines whether the input contains the substring *aba*:



The automaton accepts e.g. the input string *aaba*, since it can proceed as follows:

$$(q_0, aaba) \vdash (q_0, aba) \vdash (q_1, ba) \vdash (q_2, a) \vdash (q_3, \varepsilon).$$

The automaton could also end up in a non-accepting state:

$$(q_0, aaba) \vdash (q_0, aba) \vdash (q_0, ba) \vdash (q_0, a) \vdash (q_0, \varepsilon),$$

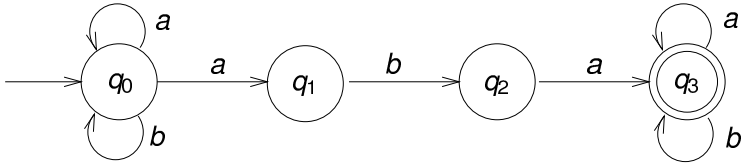
but this is of no importance — one may think that the automaton can "see the future" and always choose the best possible alternative.

Definition 2.2 A nondeterministic finite state automaton is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F),$$

where

- Q is a finite set of *states*;
- Σ is the *input alphabet*;
- $\delta : Q \times \Sigma \to \boxed{\mathcal{P}(Q)}$ is the set-valued *transition function*;
- $q_0 \in Q$ is the *initial state*;
- $F \subseteq Q$ is the set of *(accepting) final states*.



The transition function of the *aba*-automaton for example:

		a	b
$\rightarrow$	q_0	$\{q_0, q_1\}$	$\{q_0\}$
	q_1	$\emptyset$	$\{q_2\}$
	q_2	$\{q_3\}$	$\emptyset$
$\leftarrow$	q_3	$\{q_3\}$	$\{q_3\}$

We see from the array for example that $\delta(q_0, a) = \{q_0, q_1\}$ and $\delta(q_1, a) = \emptyset$.

The configuration (q, w) of a nondeterministic automaton may *lead directly to* configuration (q', w'), denoted with

$$(q, w) \underset{M}{\vdash} (q', w'),$$

if $w = aw'$ $(a \in \Sigma)$ and $q' \in \delta(q, a)$. It is also said that (q', w') is a possible *immediate successor* of (q, w).

Configuration derivations of several steps, accepting and rejecting strings etc. are defined exactly as for deterministic finite state automata. Since the single step successor relation is different now, they take a slightly different meaning.

Theorem 2.2 Let $A = L(M)$ be a language recognized by some nondeterministic finite state automaton M. Then there is also a deterministic finite state automaton $\widehat{M}$, for which $A = L(\widehat{M})$.

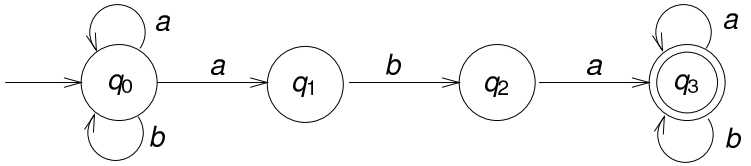
Proof. Let $A = L(M)$, $M = (Q, \Sigma, \delta, q_0, F)$. The idea of the proof is to design a deterministic finite state automaton $\widehat{M}$, that simulates M in all states possible at a given moment *in parallel*.

Formally: the states of $\widehat{M}$ correspond to *sets* of states of M:

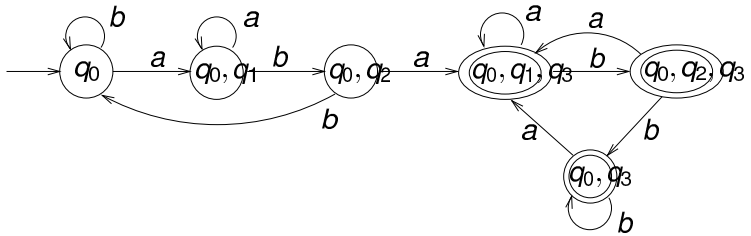
$$\widehat{M} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \widehat{F}),$$

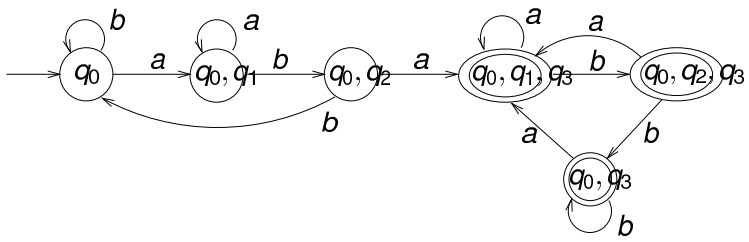
where

$$\begin{aligned} \hat{Q} &= \mathcal{P}(Q) = \{S \mid S \subseteq Q\}, \\ \hat{q}_0 &= \{q_0\}, \\ \widehat{F} &= \{S \subseteq Q \mid S \text{ contains some } q_f \in F\}, \\ \hat{\delta}(S, a) &= \bigcup_{q \in S} \delta(q, a). \end{aligned}$$

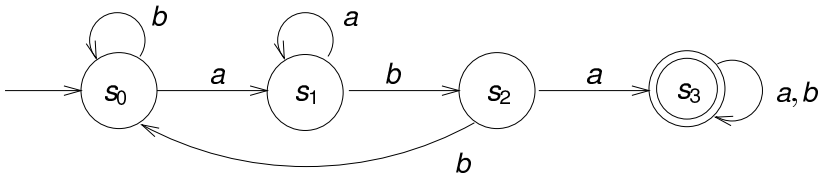


For example, when applied to the *aba*-automaton the above construction gives the following deterministic automaton (only the states reachable from the initial state are presented):





By minimizing and renaming the states this simplifies to



[Proof continues.] Let us verify that $\widehat{M}$ truly is equivalent with M, that is, $L(\widehat{M}) = L(M)$.
By definition we have

$$x \in L(M) \text{ iff } (q_0, x) \vdash_M^* (q_f, \varepsilon) \text{ for some } q_f \in F$$

and

$$x \in L(\widehat{M}) \text{ iff } (\{q_0\}, x) \vdash_{\widehat{M}}^* (S, \varepsilon)$$

ja S contains some $q_f \in F$.

We show that for all $x \in \Sigma^*$ and $q \in Q$ we have:

$$(q_0, x) \vdash_M^* (q, \varepsilon) \text{ iff } (\{q_0\}, x) \vdash_{\widehat{M}}^* (S, \varepsilon) \text{ ja } q \in S. \qquad (2)$$

Claim (2):

$$(q_0, x) \vdash_M^* (q, \varepsilon) \text{ iff } (\{q_0\}, x) \vdash_{\widehat{M}}^* (S, \varepsilon) \text{ ja } q \in S.$$

Claim (2) is proven by induction on the length of the string x.
(i) Case $|x| = 0$*:*
$(q_0, \varepsilon) \vdash_M^* (q, \varepsilon)$ iff $q = q_0$;
also $(\{q_0\}, \varepsilon) \vdash_{\widehat{M}}^* (S, \varepsilon)$ iff $S = \{q_0\}$.

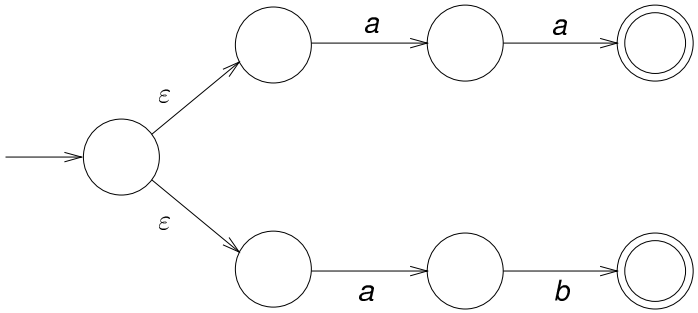
Claim (2): $(q_0, x) \vdash_M^* (q, \varepsilon)$ iff $(\{q_0\}, x) \vdash_{\widehat{M}}^* (S, \varepsilon)$ and $q \in S$.
(ii) Induction step:
Let $x = ya$; assume that Claim (2) holds for y. Then:

$(q_0, x) = (q_0, ya) \vdash_M^* (q, \varepsilon)$ iff
$\exists q' \in Q$ s.e. $(q_0, ya) \vdash_M^* (q', a)$ and $(q', a) \vdash_M (q, \varepsilon)$ iff
$\exists q' \in Q$ s.e. $(q_0, y) \vdash_M^* (q', \varepsilon)$ and $(q', a) \vdash_M (q, \varepsilon)$ iff (induction assumption)
$\exists q' \in Q$ s.e. $(\{q_0\}, y) \vdash_{\widehat{M}}^* (S', \varepsilon)$ and $q' \in S'$ and $q \in \delta(q', a)$ iff
$(\{q_0\}, y) \vdash_{\widehat{M}}^* (S', \varepsilon)$ and $\exists q' \in S'$ s.e. $q \in \delta(q', a)$ iff
$(\{q_0\}, y) \vdash_{\widehat{M}}^* (S', \varepsilon)$ and $q \in \bigcup_{q' \in S'} \delta(q', a) = \hat{\delta}(S', a)$ iff
$(\{q_0\}, ya) \vdash_{\widehat{M}}^* (S', a)$ and $q \in \hat{\delta}(S', a) = S$ iff
$(\{q_0\}, ya) \vdash_{\widehat{M}}^* (S', a)$ and $(S', a) \vdash_{\widehat{M}} (S, \varepsilon)$ and $q \in S$ iff
$(\{q_0\}, x) = (\{q_0\}, ya) \vdash_{\widehat{M}}^* (S, \varepsilon)$ and $q \in S$. □

ε-automata

We need one more extension of finite state automata: a nondeterministic automaton, with *ε-transitions*. By such a transition the automaton may make a nondeterministic choice between possible successor states without reading an input symbol.

For example, the language $\{aa, ab\}$ could be recognized by the following ε-automaton:



Formally: an ε-automaton is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F),$$

where the transition function δ is a mapping

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q).$$

Other definitions are as for ordinary nondeterministic finite state automata, except for the direct successor relation: for ε-automata the relation

$$(q, w) \underset{M}{\vdash} (q', w')$$

holds, if we have

(i) $w = aw'$ ($a \in \Sigma$) and $q' \in \delta(q, a)$; or
(ii) $w = w'$ and $q' \in \delta(q, \varepsilon)$.

Lemma 2.4 Let $A = L(M)$ for some ε-automaton M. Then there also exists a nondeterministic automaton $\widehat{M}$ without ε-transitions, for which $A = L(\widehat{M})$.

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an ε-automaton. The automaton $\widehat{M}$ functions like M except that it "skips over" the ε-transitions by only performing those "real" transitions, that can be reached from the current state by a sequence of ε-transitions.

Formally the *ε-closure* $\varepsilon^*(q)$ of state $q \in Q$ in automaton M is defined as

$$\varepsilon^*(q) = \{q' \in Q \mid (q, \varepsilon) \underset{M}{\vdash^*} (q', \varepsilon)\},$$

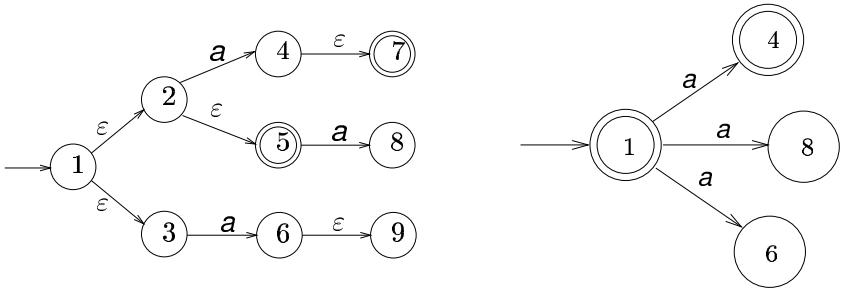
that is, the set $\varepsilon^*(q)$ contains all such states of M that can be reached from q by ε-transitions only.

The automaton $\widehat{M}$ can now be described as

$$\widehat{M} = (Q, \Sigma, \hat{\delta}, q_0, \widehat{F}),$$

where

$$\hat{\delta}(q, a) = \bigcup_{q' \in \varepsilon^*(q)} \delta(q', a);$$

$$\widehat{F} = \{q \in Q \mid \varepsilon^*(q) \cap F \neq \emptyset\}.$$

By removing the ε-transitions by the previous construction an ordinary nondeterministic automaton can be obtained from an ε-automaton, e.g.