Aalto University
School of Science

CS-C2160 Theory of Computation

Lecture 3. Finite Automata: Minimisation and Nondeterminism

Pekka Orponen
Aalto University
Department of Computer Science

Spring 2021

Topics

Topics:

- Minimisation of finite automata
- Nondeterministic finite automata
- ε-automata

Material in Finnish:

- Sections 2.4–2.5 in Finnish lecture notes (and the concept of computation tree in these slides)

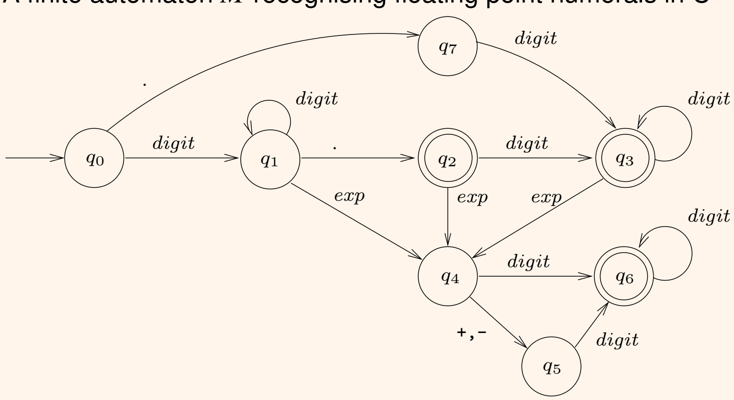
Material in English:

- Minimisation: e.g. Wikipedia
- Nondeterminism: Section 1.2 in the Sipser book

Recap: Finite automata

Example:

A finite automaton M recognising floating point numerals in C



where *digit* $= \{0,1,\ldots,9\}$ and *exp* $= \{\mathtt{E},\mathtt{e}\}$.
Now $12.3\mathtt{E}-2 \in \mathcal{L}(M)$ and $.32 \in \mathcal{L}(M)$ but $12\mathtt{E} \notin \mathcal{L}(M)$.

Finite Automata: Minimisation and Nondeterminism

3.1 Minimisation of finite automata

- One can show that, for each finite automaton, there exists a unique automaton (up to renaming of the states) with a minimum number of states that recognises the same language.
- Minimisation of automata allows us to e.g. decide whether two automata recognise the same language. (This is the case if and only if the corresponding minimal automata are the same).
- Minimisation can be done with an efficient algorithm discussed below. Its main idea is to merge all the states from which the automaton works in exactly the same way w.r.t. acceptance for all input strings.

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton.
- Extend the transition function δ of M from symbols to strings as follows: if $q \in Q$ and $x \in \Sigma^*$, define

$$\delta^*(q,x) = q' \text{ s.t. } (q,x) \vdash_M^* (q', \varepsilon).$$

- Two states, q and q', of M are *equivalent*, denoted by

$$q \equiv q',$$

  if for all $x \in \Sigma^*$ it holds that

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

  In other words, states q and q' are equivalent if the automaton accepts exactly the same strings when started from either one.

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

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

  if for all $x \in \Sigma^*$, $|x| \leq k$, it holds that

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

  In other words, q and q' are k-equivalent if no string of length k or less can distinguish them from each other.
- Obviously,

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

- The minimisation algorithm proceeds by refining equivalence classes of states induced by k-equivalence into ones induced by $(k+1)$-equivalence until full equivalence is reached.

The algorithm is based on the following simple lemma:

Lemma 3.1

(i) Two k-equivalent states, q_1 and q_2, are $(k+1)$-equivalent if and only if $\delta(q_1,a) \overset{k}{\equiv} \delta(q_2,a)$ for all $a \in \Sigma$.

(ii) If for some k it holds that *all* mutually k-equivalent states are also $(k+1)$-equivalent, then they are fully equivalent as well.

(Claim 2 follows by induction from claim (i) and observation 1(ii) on the previous slide.)

Algorithm MIN-FA [Minimisation of finite automata]

- *Input:* A finite automaton $M = (Q, \Sigma, \delta, q_0, F)$.
- *Output:* A finite automaton $\widehat{M}$ that is (i) equivalent to M, meaning that it recognises the same language, and (ii) has a minimum number of states.
- *Procedure:*
  1. [Removal of redundant states]
     Remove all the states of M that cannot be reached from the initial state q_0 with any input string.
  2. [0-equivalence]
     Partition the remaining states of M into two equivalence classes: non-accept and accept states.

- Procedure continues...
  3. [from k-equivalence to $(k+1)$-equivalence]
     Check if it is the case that, for each alphabet symbol a, from all the states in the same equivalence class there is a transition with the symbol a to states in the same "successor equivalence class".

     If this is the case, the algorithm terminates and the states of the minimal automaton $\widehat{M}$ correspond to the *equivalence classes* of the states of M. For each alphabet symbol σ, there is a σ-transition from class state $\hat{q}_1$ to class state $\hat{q}_2$ in $\widehat{M}$ if there is a σ-transition from any state in class $\hat{q}_1$ to any state in class $\hat{q}_2$ in M.

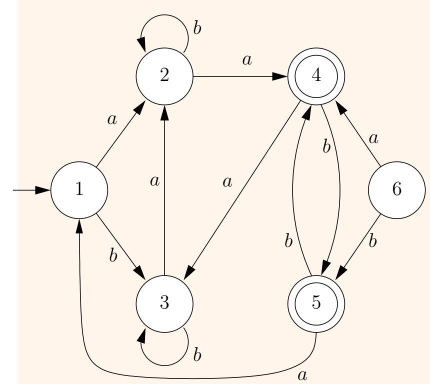
     Otherwise, refine the partitioning by splitting each equivalence class of states that violates the above condition into smaller classes according to the respective successor classes. Repeat step 3.

Given a finite automaton M, algorithm MIN-FA constructs a finite automaton $\widehat{M}$ that recognises the same language and has a minimum number of states. The resulting automaton is unique (up to renaming of the states).

- The algorithm always terminates: each time Step 3 is executed, at least one of the (finitely many) equivalence classes is split in two non-empty smaller classes.
- Step 3 refines k-equivalence classes into $(k+1)$-equivalence classes. [Lemma 3.1(i)]
- When all k-equivalence classes are $(k+1)$-equivalence classes as well, the states in each of them are mutually fully equivalent. [Lemma 3.1(ii)]
- Each equivalence class contains at least one state and any such state is reachable from the initial state, meaning that all the classes are necessary.
- The minimality and uniqueness proofs for the resulting automaton can be found in the Finnish lecture notes.

Example

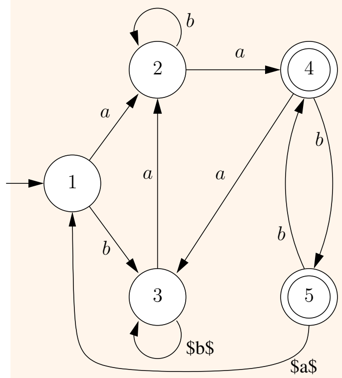
Let us minimise the automaton shown below.



		a	b
$\rightarrow$	1	2	3
	2	4	2
	3	2	3
$\leftarrow$	4	3	5
$\leftarrow$	5	1	4
	6	4	5

Example

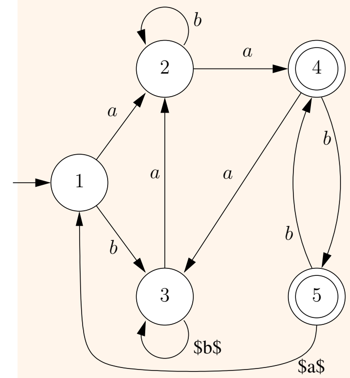
In step 1, state 6 is removed as it cannot be reached from the initial state with any input string.



		a	b
$\rightarrow$	1	2	3
	2	4	2
	3	2	3
$\leftarrow$	4	3	5
$\leftarrow$	5	1	4

Example

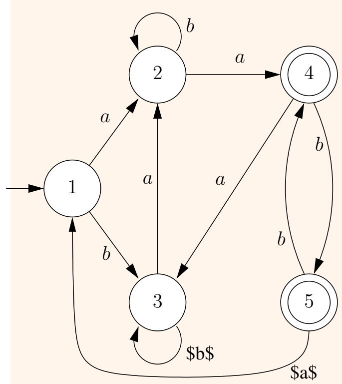
In step 2, the remaining states $\{1,2,3,4,5\}$ are partitioned into class I that contains the non-accept states and class II that contains the accept states.



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

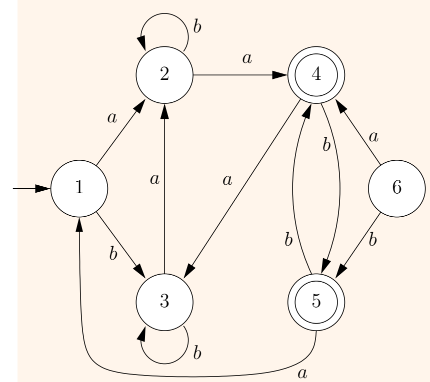
Example

In step 3, it is marked to which class the transition function maps each (state,symbol)-pair:



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

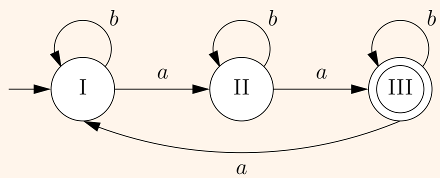
Since class I contains two kinds of states, $\{1,3\}$ and $\{2\}$, the partitioning is refined and the transition function again inspected w.r.t the new partitioning:



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

Now the states in each class behave similarly for each input symbol and the algorithm terminates.

The resulting minimal finite automaton is:

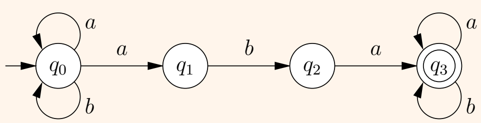


3.2 Nondeterministic finite automata

- Nondeterministic automata are similar to deterministic ones except that their transition function δ does not associate a (state,input symbol) pair to a single successor state but to *a set of possible successor states*.
- A nondeterministic automaton accepts an input if *at least one* possible execution path leads to an accepting final state.
- Nondetermistic automata cannot be, as such, implemented with computer programs but they are an important *description formalism* for decision problems.
- *Note:* In these slides, as well as in the Finnish lecture notes, nondeterministic automata are introduced in two stages, first without allowing ε-transitions and then with them. The Sipser book allows ε-transitions at once.

Example:

A nondeterministic automaton that detects whether the input string contains substring aba:



The automaton accepts the string $aaba$ because it is *possible* for it to proceed as follows:

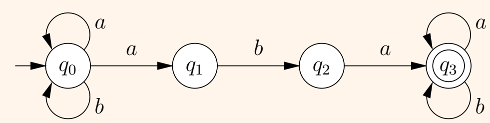
$$(q_0, aaba) \vdash (q_0, aba) \vdash (q_1, ba) \vdash (q_2, a) \vdash (q_3, \varepsilon)$$

The automaton could also have ended 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 does not matter: intuitively we can think that the automaton is capable of always taking "the best possible choice".

A nondeterministic automaton that detects whether the input string contains substring aba:



Notice: The automaton does not accept e.g. the string $aabbab$ because it is *not possible* to reach the (only) accepting final configuration (q_3, ε) from the initial configuration $(q_0, aabbab)$ *in any way*.

Formally, we define nondeterministic FA (NFA) as follows:

Definition 3.1 (Nondeterministic finite automata)

A nondeterministic finite automaton is a tuple

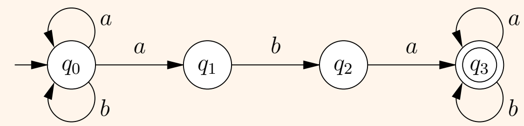
$$M = (Q, \Sigma, \delta, q_0, F),$$

where

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

Example:

An "*aba*-automaton" that detects whether the input string contains a substring *aba*:



The transition function is

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

For example, $\delta(q_0, a) = \{q_0, q_1\}$ and $\delta(q_1, a) = \emptyset$.

- A configuration (q, w) of a nondeterministic FA *can lead directly* to a configuration (q', w'), denoted by

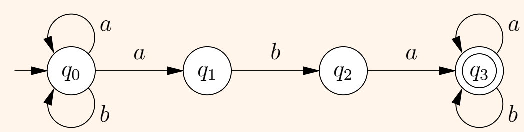
$$(q, w) \underset{M}{\vdash} (q', w'),$$

  if (i) $w = aw'$ for an $a \in \Sigma$ and (ii) $q' \in \delta(q, a)$.

  In such a case we may also say that (q', w') is a possible *immediate successor* of the configuration (q, w).
- Non-immediate successor configurations, acceptance of strings etc. are defined similarly to those for deterministic finite automata discussed in the previous lecture.

Example:

For the "*aba*-automaton"



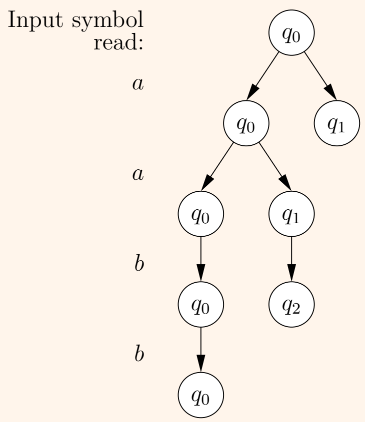
the possible computations on input string *aabb* are

- $(q_0, aabb) \underset{M}{\vdash} (q_0, abb) \underset{M}{\vdash} (q_0, bb) \underset{M}{\vdash} (q_0, b) \underset{M}{\vdash} (q_0, \varepsilon)$
- $(q_0, aabb) \underset{M}{\vdash} (q_0, abb) \underset{M}{\vdash} (q_1, bb) \underset{M}{\vdash} (q_2, b)$
- $(q_0, aabb) \underset{M}{\vdash} (q_1, abb)$

None of these is ending in a configuration with an accept state and empty remaining string. Therefore, the string *aabb* is not accepted and does not belong to the language recognised by the automaton.

Example:

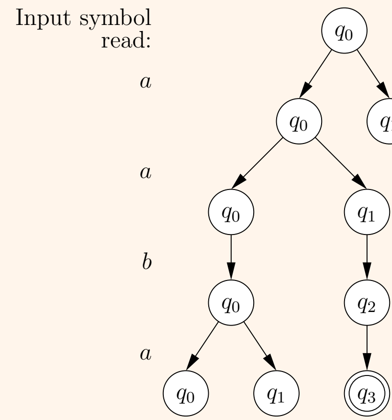
The computations of the "*aba*-automaton" on the input *aabb* can also be illustrated as a *computation tree*



where each path from the root to a leaf describes one computation.

Example:

The computation tree for the "*aba*-automaton" on input *aaba*:



Now at least one computation path ends in an accept state when the input is fully processed, and thus the string *aaba* belongs to the language recognised by the automaton.

Theorem 3.2 (Determinisation of NFA)

Let $A = L(M)$ be a language recognised by a nondeterministic FA M. Then there exists also a deterministic FA $\widehat{M}$ such that $A = L(\widehat{M})$.

Proof

Let $A = L(M)$ for some nondeterministic FA $M = (Q, \Sigma, \delta, q_0, F)$. The idea is to construct a deterministic FA $\widehat{M}$ that simulates the operation of M in all states possible at each step *in parallel*.

Formally, the states of $\widehat{M}$ are *sets* of states of M:

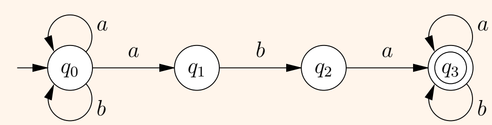
$$\widehat{M} = (\widehat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \widehat{F}),$$

where

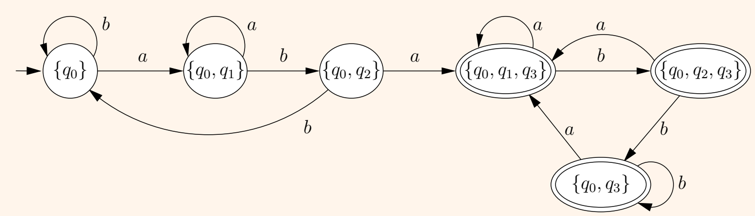
$$\begin{aligned}
\widehat{Q} &= \mathcal{P}(Q) = \{S \mid S \subseteq Q\}, \\
\hat{q}_0 &= \{q_0\}, \\
\widehat{F} &= \{S \subseteq Q \mid S \cap F \neq \emptyset\}, \\
\hat{\delta}(S, a) &= \bigcup_{q \in S} \delta(q, a).
\end{aligned}$$

Example:

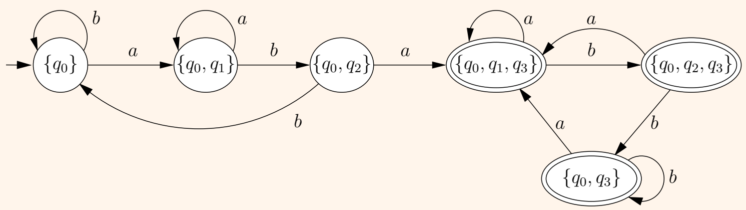
When applied to the *aba*-automaton



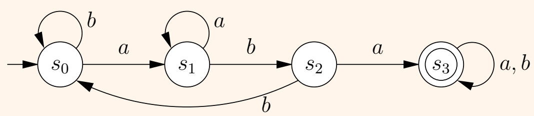
the algorithm produces the following deterministic automaton (only those states that can be reached from the new initial state are drawn):



Minimising the deterministic automaton



leads to this final result (where also the states have been renamed):



[Proof continues.] Let us verify that the automaton $\widehat{M}$ is equivalent to M, meaning that $\mathcal{L}(\widehat{M}) = \mathcal{L}(M)$.
By definition,

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

and

$$x \in \mathcal{L}(\widehat{M}) \text{ iff } (\{q_0\}, x) \underset{\widehat{M}}{\vdash^*} (S, \varepsilon) \text{ and } S \text{ contains an accept state } q_f \in F$$

Thus it is enough to prove that, for all $x \in \Sigma^*$ and all $q \in Q$, it holds that

$$(q_0, x) \underset{M}{\vdash^*} (q, \varepsilon) \text{ iff } (\{q_0\}, x) \underset{\widehat{M}}{\vdash^*} (S, \varepsilon) \text{ and } q \in S. \qquad (2)$$

Claim (2):

$$(q_0, x) \underset{M}{\vdash^*} (q, \varepsilon) \text{ iff } (\{q_0\}, x) \underset{\widehat{M}}{\vdash^*} (S, \varepsilon) \text{ and } q \in S.$$

Proof of claim (2) is by induction on the length of the string x.

(i) Base case $|x| = 0$:

$(q_0, \varepsilon) \underset{M}{\vdash^*} (q, \varepsilon)$ iff $q = q_0$;

similarly, $(\{q_0\}, \varepsilon) \underset{\widehat{M}}{\vdash^*} (S, \varepsilon)$ iff $S = \{q_0\}$.

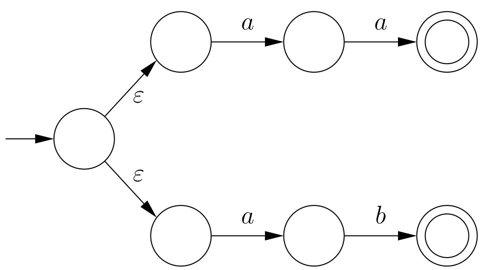
(ii) Induction step: Let $x = ya$ for some $a \in \Sigma$ and assume that claim (2) holds for y. Now

$(q_0, x) = (q_0, ya) \underset{M}{\vdash^*} (q, \varepsilon)$ iff
$\exists q' \in Q$ s.t. $(q_0, ya) \underset{M}{\vdash^*} (q', a)$ and $(q', a) \underset{M}{\vdash} (q, \varepsilon)$ iff
$\exists q' \in Q$ s.t. $(q_0, y) \underset{M}{\vdash^*} (q', \varepsilon)$ and $(q', a) \underset{M}{\vdash} (q, \varepsilon)$ iff **[induction]**
$\exists q' \in Q$ s.t. $(\{q_0\}, y) \underset{\widehat{M}}{\vdash^*} (S', \varepsilon)$ and $q' \in S'$ and $q \in \delta(q', a)$ iff
$(\{q_0\}, y) \underset{\widehat{M}}{\vdash^*} (S', \varepsilon)$ and $\exists q' \in S'$ s.t. $q \in \delta(q', a)$ iff
$(\{q_0\}, y) \underset{\widehat{M}}{\vdash^*} (S', \varepsilon)$ and $q \in \bigcup_{q' \in S'} \delta(q', a) = \hat{\delta}(S', a)$ iff
$(\{q_0\}, ya) \underset{\widehat{M}}{\vdash^*} (S', a)$ and $q \in \hat{\delta}(S', a) = S$ iff
$(\{q_0\}, ya) \underset{\widehat{M}}{\vdash^*} (S', a)$ and $(S', a) \vdash_{\widehat{M}} (S, \varepsilon)$ and $q \in S$ iff
$(\{q_0\}, x) = (\{q_0\}, ya) \underset{\widehat{M}}{\vdash^*} (S, \varepsilon)$ and $q \in S$. $\square$

3.3 ε-automata

In the future we will also need one more extension of FAs: nondeterministic FA that allow *ε-transitions*. Such transitions allow an automaton to make nondeterministic choices without reading any symbols from the input.

For instance, the language $\{aa, ab\}$ can be recognised with the following ε-automaton:



Definition 3.2 (ε-automata)

An *ε-automaton* is a tuple

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

where the transition function δ is a function

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

The other definitions are as for standard nondeterministic FA except that the "leads directly" relation is now defined so that

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

if

- $w = aw'$ for an $a \in \Sigma$ and $q' \in \delta(q, a)$, or
- $w = w'$ and $q' \in \delta(q, \varepsilon)$.

Theorem 3.3 (Eliminating ε-transitions)

Let $A = L(M)$ for an ε-automaton M. Then there exists also a standard nondeterministic FA $\widehat{M}$ such that $L(\widehat{M}) = A$.

Proof

Let $M = (Q, \Sigma, \delta, q_0, F)$ be any ε-automaton. Intuitively, the automaton $\widehat{M}$ we construct below works otherwise exactly as M except that it "jumps over" ε-transitions by taking only those "real" transitions from each state that may follow immediately after a sequence of ε-transitions.

Formally, given a state $q \in Q$, we define its *ε-closure* $\varepsilon^*(q)$ in M by

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

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

The automaton $\widehat{M}$ can now be defined as follows:

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

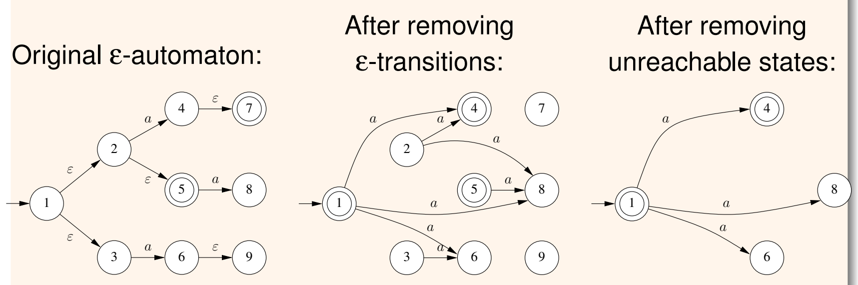
where

$$\begin{aligned} \hat{\delta}(q, a) &= \bigcup_{q' \in \varepsilon^*(q)} \delta(q', a), \\ \widehat{F} &= \{q \in Q \mid \varepsilon^*(q) \cap F \neq \emptyset\} \end{aligned}$$

□

Example:

By removing ε-transitions from an ε-automaton with the above construction we get a standard nondeterministic automaton:



Consider the automaton after removing the ε-transitions:

1. Why is the initial state “1” now an accepting final state?
2. Why does it not matter that the accepting final state “7” is not anymore reachable from the initial state?