INF 302 : Langages & Automates

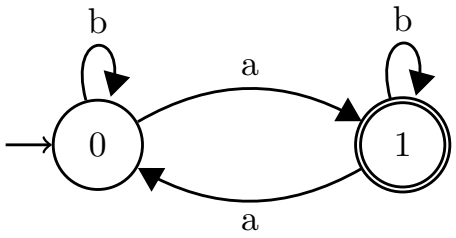
Chapter 6: Deterministic Finite-state Automata — Distinguishability, Equivalence, Minimisation

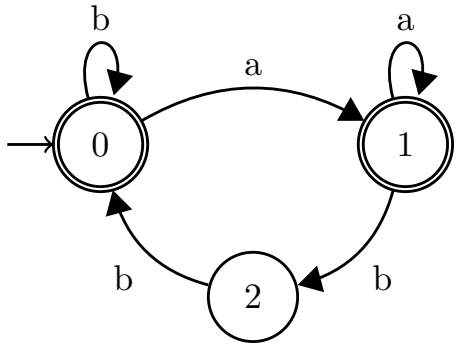
Yliès Falcone

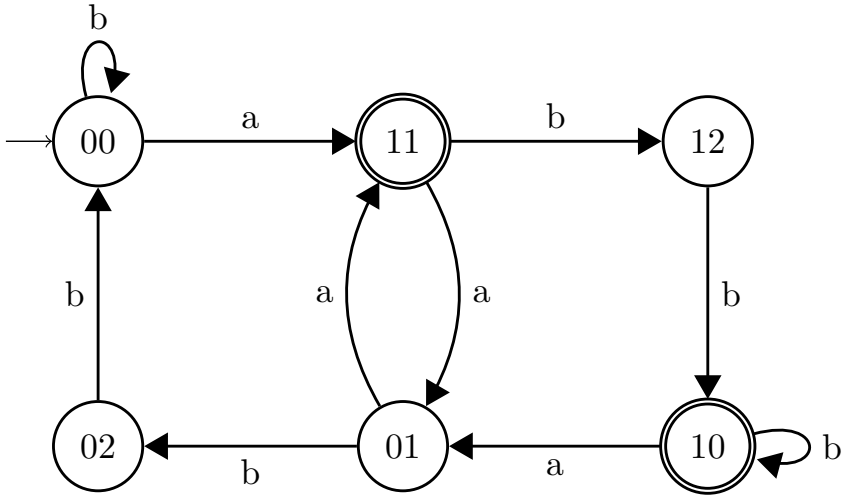
ylies.falcone@univ-grenoble-alpes.fr — www.ylies.fr

Univ. Grenoble-Alpes
-
Laboratoire d'Informatique de Grenoble - www.liglab.fr

Intuition and Objectives





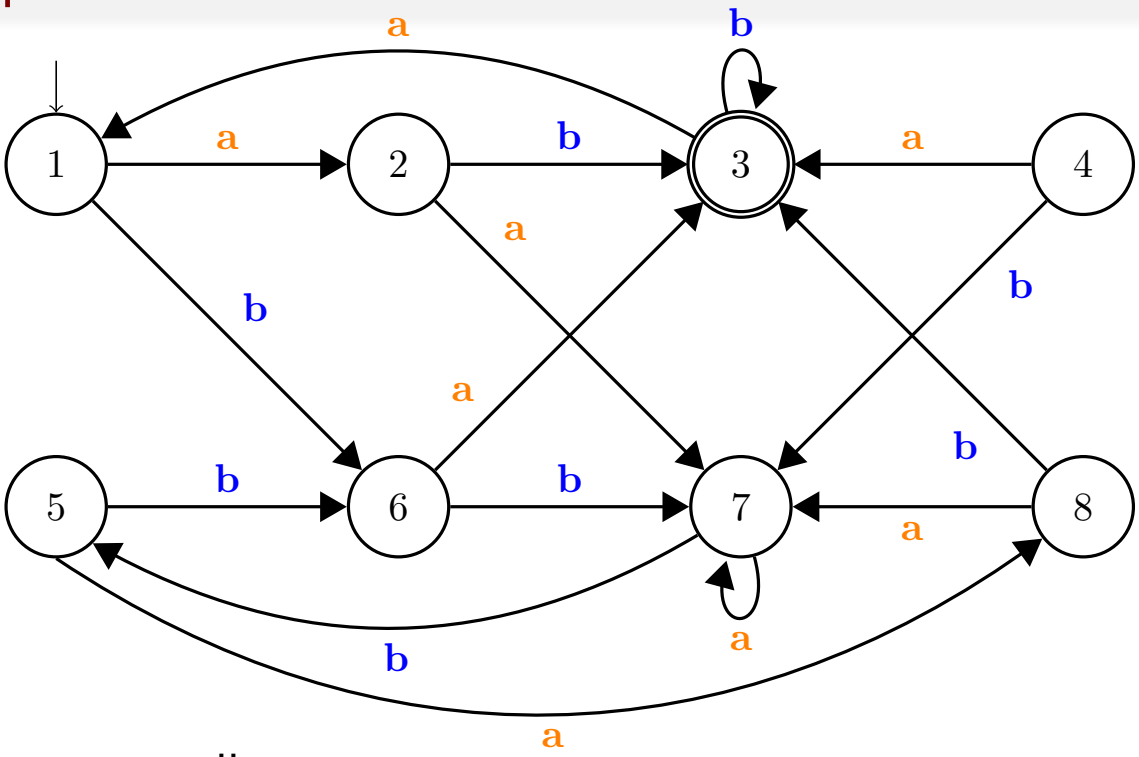


- Basic components: states (accepting states), symbols, transitions — *syntax*.
- Execution, accepted word, accepted language — *semantics*.
- Decision problems: emptiness, infiniteness.
- Automata operations / language operations: negation/complementation, product/intersection.

Outline Chap. 6 - DFA - Distinguishability, Equivalence, Minimisation

1. Testing Equivalence between States
2. Testing Automata Equivalence
3. Minimization of DFAs
4. Résumé

Equivalence and Minimization: motivation through an example



Questions

- Which states can be distinguished?
- Which states are equivalent?

More generally:

- Can we define an equivalence relation on states?
- Can we decide whether two automata are equivalent?
- Can we obtain a *canonical* (i.e., minimal) representation of an automaton?

Equivalence/distinguishability are tied to the notion of **acceptance**.

In this chapter, we consider $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ a DFA that is complete and where all states are reachable.

Outline Chap. 6 - DFA - Distinguishability, Equivalence, Minimisation

1. Testing Equivalence between States
2. Testing Automata Equivalence
3. Minimization of DFAs
4. Résumé

State Distinguishability

Definition and Example

"Two states are distinguishable if there exists a word that, starting from one state, leads to an accepting state, and starting from the other, leads to a non-accepting state."

Definition (Distinguishability relation on states)

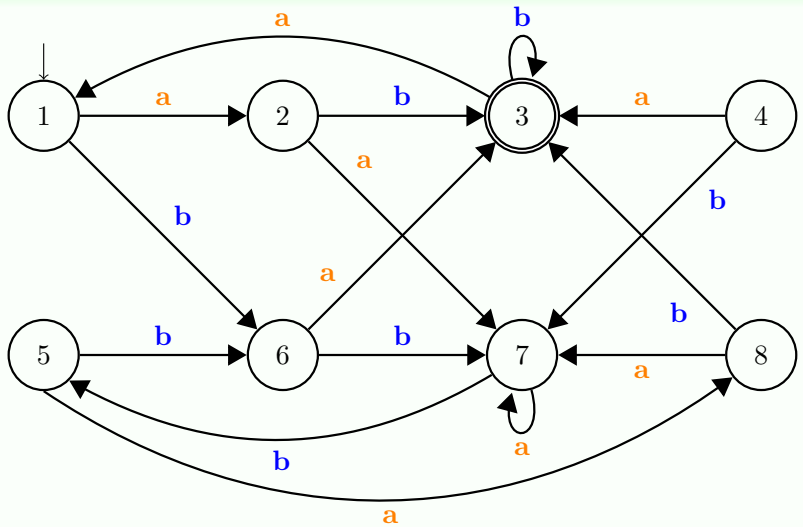
The distinguishability relation $\neq$ on Q is defined by:

$$\forall p, q \in Q : \quad \Big(p \neq q \quad \text{iff} \quad \exists u \in \Sigma^* : \big(\delta^*(p,u) \in F \iff \delta^*(q,u) \in F \big) \Big)$$

Two states that are not distinguishable are said to be *equivalent* (relation $\equiv$).

Example (Distinguishable vs. equivalent states)

- distinguishable: $1 \neq 2$, $1 \neq 3$, $1 \neq 4$, $1 \neq 6$, $2 \neq 3$, ...
- equivalent: $4 \equiv 6$, $2 \equiv 8$, and also $1 \equiv 5$, as well as $q \equiv q$ for every state q.



State Distinguishability

Properties of the Relation

Recall: definition of the distinguishability relation

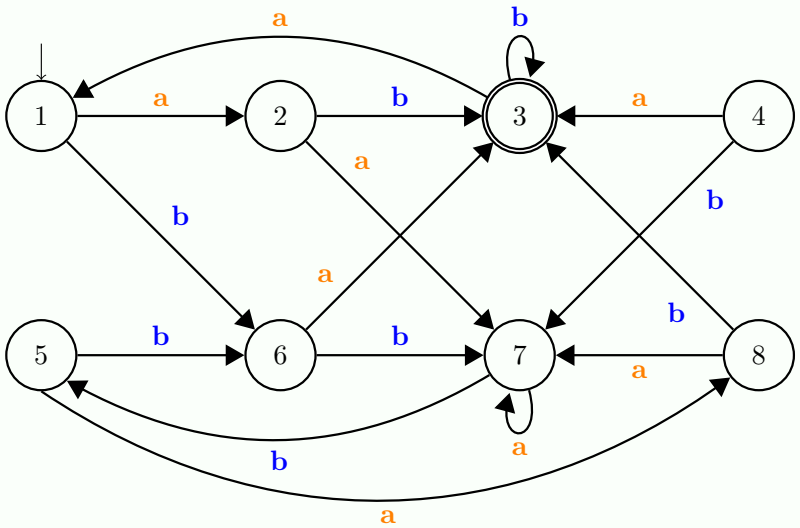
$$p \neq q \quad \text{iff} \quad \exists u \in \Sigma^* : \left(\delta^*(p,u) \in F \iff \delta^*(q,u) \in F\right)$$

Theorem: properties of the distinguishability relation

The distinguishability relation $\neq$ on states of Q is:

- irreflexive: $\forall q \in Q : \neg(q \neq q)$,
- symmetric: $\forall p, q \in Q : p \neq q \implies q \neq p$.

Illustration of the theorem



- irreflexivity: $q \neq q$ is false, for every state q,
- symmetry: $6 \neq 1$ and $1 \neq 6$.

Computing the Distinguishability Relation

Why not compute $\neq$ directly?

The definition of the distinguishability relation is not directly usable for computation:

- It requires reading words of arbitrary length.
- It involves searching for words in an infinite set (Σ^*).

Technique for computing $\neq$

$\hookrightarrow$ Restrict the distinguishability relation to k symbols.

$\hookrightarrow$ Compute $\neq$ in an *iterative* way.

State Distinguishability **at** k **steps**

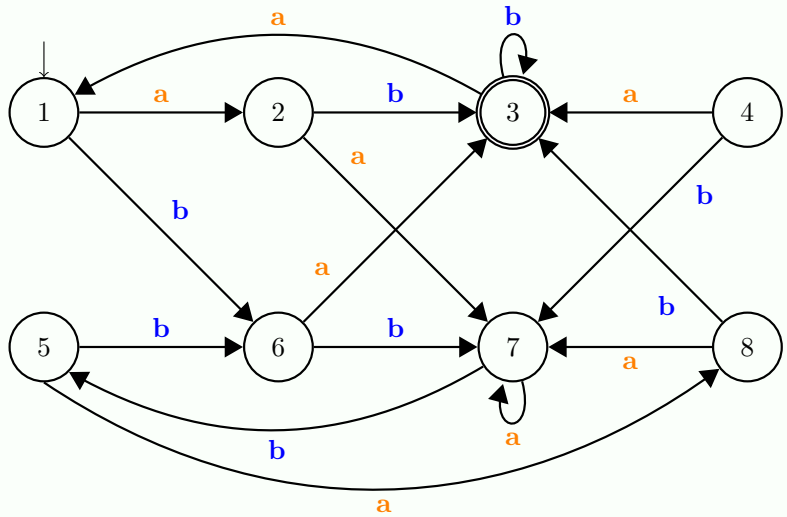
Definition (k-step distinguishability)

For each $k \in \mathbb{N}$, we define the relation $\neq_k$ on Q:

1. $q \neq_0 q'$ iff $q \in F \nLeftrightarrow q' \in F$;
2. For $k \in \mathbb{N}$, $p \neq_{k+1} q$ iff $(p \neq_k q) \vee (\exists a \in \Sigma : \delta(p, a) \neq_k \delta(q, a))$.

Example (States distinguishable at k steps)

- $k = 0$: $x \neq_0 3$ and $3 \neq_0 x$, with $x \in \{1, 2, 4, \ldots, 8\}$;
- $k = 1$: $1 \neq_1 \{2, 6, 8\}$, $2 \neq_1 \{1, 4, 5, 7\}$, $\ldots$, and $x \neq_1 y$ whenever $x \neq_0 y$;
- $k = 2$: $x \neq_2 y$ whenever $x \neq_1 y$.



Building $\neq$ from $\neq_k$

Lemma

For all $k \in \mathbb{N}$, $q \neq_k q'$ iff

$$\exists u \in \Sigma^* : |u| \leq k \wedge \left(\delta^*(q,u) \in F \iff \delta^*\left(q',u\right) \in F\right).$$

Corollary

$$\bigcup_{k \in \mathbb{N}} \neq_k \quad = \quad \neq$$

Using the definition of k-step distinguishability, we obtain the following lemma.

Lemma

For all $k \in \mathbb{N}$,

$$\neq_{k+1} \quad = \quad \neq_k \cup \bigcup_{a \in \Sigma} \left\{(q,q') \mid \left(\delta(q,a), \delta(q',a)\right) \in \neq_k\right\}$$

From this we can derive an algorithm for computing distinguishable states.

State Distinguishability

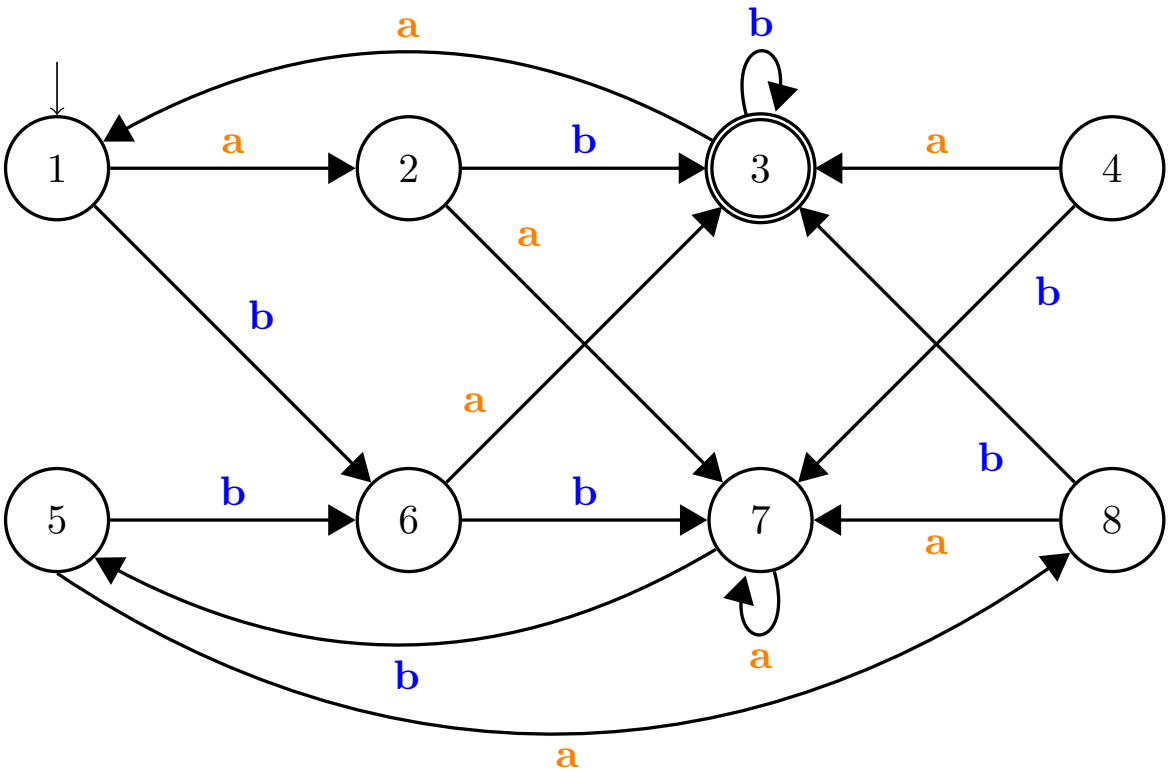
Algorithme 1 Computing distinguishable states

Entrée : $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ (* a complete DFA with all states reachable *)
Sortie : $D \subseteq Q \times Q$ (* distinguishability relation on states of Q *)

1: **ensemble de** pairs of states D, D_{pre}; (* D contains distinguishable pairs of states *)
(* D_{pre} stores D from the previous iteration *)
2: **ensemble de** pairs of states X; (* newly found distinguishable pairs at each iteration *)
3: $D := \big(F \times (Q \setminus F)\big) \cup (Q \setminus F) \times F$; (* initialize with accepting vs. non-accepting states, i.e. $\neq_0$ *)
4: $D_{pre} := \emptyset$; (* update D_{pre} (backup of D) *)
5: **tant que** $D_{pre} \neq D$ **faire**
6: $D_{pre} := D$;
7: $X := \big\{(p,q), (q,p) \in Q \times Q \mid \exists a \in \Sigma : \big(\delta(p,a), \delta(q,a)\big) \in D\big\}$;
(* compute new distinguishable pairs *)
8: $D := D \cup X$; (* add new pairs to D *)
9: **fin tant que**
10: **retourner** D; (* final result: $D = \neq$ *)

Remarque In an implementation and its execution trace, it is useful to exploit the irreflexivity and symmetry of D, D_{pre} (and X). □

State distinguishability: example



	1	2	3	4	5	6	7
2	x						
3	x	x					
4	x	x	x				
5		x	x	x			
6	x	x	x		x		
7	x	x	x	x	x	x	
8	x		x	x	x	x	x

State Distinguishability: Algorithm Correctness – Proof

Proof.

Suppose the theorem is false (i.e., there exists a counterexample automaton).
Then, there exists at least one "bad pair" of states $\{p, q\}$ such that:

- p and q are distinguishable: there exists $w \in \Sigma^*$ such that either $\delta^*(p, w) \in F$ or $\delta^*(q, w) \in F$ (but not both),
- the algorithm does not distinguish these states.

Let $w = a_1 a_2 \cdots a_n$ be the shortest word distinguishing a bad pair $\{p, q\}$.

- $w \neq \epsilon$, by the initialization of the algorithm (line 5),
- let $p' = \delta(p, a_1)$ and $q' = \delta(q, a_1)$
  - p' and q' are distinguished by $a_2 \cdots a_n$ since $\delta^*(p', a_2 \cdots a_n) = \delta^*(p, w)$ and $\delta^*(q', a_2 \cdots a_n) = \delta^*(q, w)$,
  - $a_2 \cdots a_n$ is shorter than any word distinguishing a bad pair,
  - therefore $\{p', q'\}$ cannot be a bad pair.
- The algorithm will thus mark $\{p', q'\}$ as distinguishable.
- By the body of the loop, in the worst case, at the next iteration, $\{p, q\}$ will be marked.

□

Outline Chap. 6 - DFA - Distinguishability, Equivalence, Minimisation

1 Testing Equivalence between States

2 Testing Automata Equivalence

3 Minimization of DFAs

4 Résumé

Testing Equivalence Between Two Automata

Consider two *complete* DFAs:

- $A = (Q^A, \Sigma, q^A_{\text{init}}, \delta^A, F^A)$,
- $B = (Q^B, \Sigma, q^B_{\text{init}}, \delta^B, F^B)$.

Question

How can we determine whether A and B accept the same language?

Procedure for testing equivalence

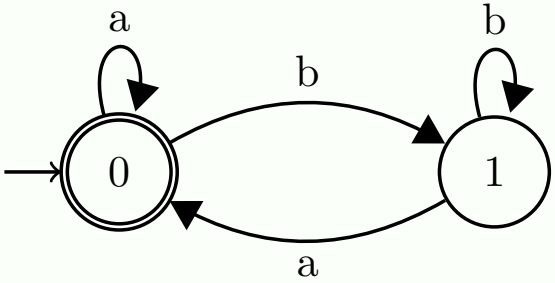
1. Build the automaton $E = (Q^A \cup Q^B, \Sigma, q^A_{\text{init}}, \delta^A \cup \delta^B, F^A \cup F^B)$.
2. Test whether q^A_{init} and q^B_{init} are distinguishable in E.

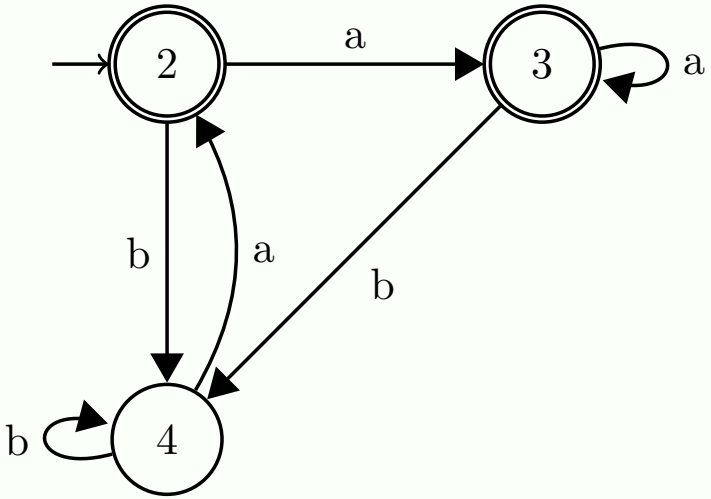
Remarque An alternative approach: construct the product automaton for the symmetric difference $(L(A) \setminus L(B)) \cup (L(B) \setminus L(A))$ and check whether its language is empty. □

Testing Equivalence Between Two Automata

Example

Example (Two equivalent automata)





1	x			
2		x		
3		x		
4	x		x	x
	0	1	2	3

(a x in cell (i, j) indicates that states i and j are distinguishable)

Outline Chap. 6 - DFA - Distinguishability, Equivalence, Minimisation

1 Testing Equivalence between States

2 Testing Automata Equivalence

3 Minimization of DFAs

4 Résumé

Equivalence Between States

Recall: $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ is a complete DFA where all states are reachable.

Definition (Equivalence relation on states)

The equivalence relation $\equiv$ on Q is defined by:

$$\forall p, q \in Q : \qquad p \equiv q \quad \text{iff} \quad \forall u \in \Sigma^* : \left(\delta^*(p, u) \in F \iff \delta^*(q, u) \in F\right)$$

$\equiv$ is indeed an equivalence relation:

- reflexive,
- symmetric,
- transitive.

Notation:

- $[q]$: the equivalence class of state q,
- $Q_{/\equiv}$: the set of equivalence classes (in an automaton with state set Q).

Equivalence and Distinguishability are dual

Two states are equivalent if and only if they are not distinguishable.

The Relation $\equiv_k$

Let $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ be a DFA where all states are reachable.

Question

How can we compute $\equiv$?

Definition (k-step equivalence)

For each $k \in \mathbb{N}$, we define the relation $\equiv_k$ on Q:

1. $q \equiv_0 q'$ iff $q \in F \Leftrightarrow q' \in F$.
2. For $k \in \mathbb{N}$, $q \equiv_{k+1} q'$ iff

$$q \equiv_k q' \wedge \forall a \in \Sigma : \delta(q, a) \equiv_k \delta(q', a).$$

Building $\equiv$ from $\equiv_k$

Lemma

For all $k \in \mathbb{N}$, $q \equiv_k q'$ iff

$$\forall u \in \Sigma^* : |u| \leq k \implies (\delta^*(q,u) \in F \iff \delta^*(q',u) \in F).$$

Corollary

$$\bigcap_{k \in \mathbb{N}} \equiv_k = \equiv$$

Using the definition of k-step equivalence, we obtain the following lemma.

Lemma

For all $k \in \mathbb{N}$,

$$\equiv_{k+1} \quad = \quad \equiv_k \cap \bigcap_{a \in \Sigma} \left\{ (q,q') \mid \big(\delta(q,a), \delta(q',a)\big) \in \equiv_k \right\}$$

From this we can derive an algorithm to compute equivalence classes.

Algorithm for Computing $Q_{/\equiv}$

Algorithme 2 Computing equivalence classes

Entrée : $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ (* a complete DFA where all states are reachable *)

Sortie : $R = Q_{/\equiv}$

1: **ensemble de** pairs of states R; (* R is the final "finest" partition sought *)

2: **ensemble de** pairs of states R_{pre} (* R_{pre} stores R from the previous iteration *)

3: **ensemble de** pairs of states X;
(* temporary set per iteration, newly found distinguishable pairs *)

4: $R := (F \times F) \cup \Big((Q \setminus F) \times (Q \setminus F)\Big)$;
(* initialize partition into accepting vs. non-accepting states, i.e. $R = \equiv_0$ *)

5: $R_{pre} := \emptyset$;

6: **tant que** $R_{pre} \neq R$ **faire**

7: $\quad R_{pre} := R$; (* update R_{pre} *)

8: $\quad X := \Big\{(p, q) \in R \mid \exists a \in \Sigma : (\delta(p, a), \delta(q, a)) \notin R\Big\}$;
(* compute new distinguishable pairs *)

9: $\quad R := R \setminus X$; (* remove distinguishable pairs from R *)

10: **fin tant que**

11: **retourner** R;

Remarque In an implementation and its execution trace, it is useful to exploit the reflexivity and symmetry of R, R_{pre} (and X). □

Minimization: Minimized Automaton and Equivalence

Let $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ be a complete DFA where all states are reachable.

Definition (Minimized automaton – also called quotient automaton)

The minimization of A is the automaton $A_{/\equiv} = (Q_{/\equiv}, \Sigma, [q_{\text{init}}], \delta_{/\equiv}, F_{/\equiv})$ where:

- $\delta_{/\equiv}$ is the transition function defined by:

$$\begin{array}{rcl} \delta_{/\equiv} & : & Q_{/\equiv} \times \Sigma \to Q_{/\equiv} \\ \delta_{/\equiv}([q], a) & \stackrel{\text{def}}{=} & [\delta(q, a)] \end{array}$$

- $F_{/\equiv} = \{[q] \mid q \in F\}$.

Remarque Since δ is a function, $\delta_{/\equiv}$ is also a function. □

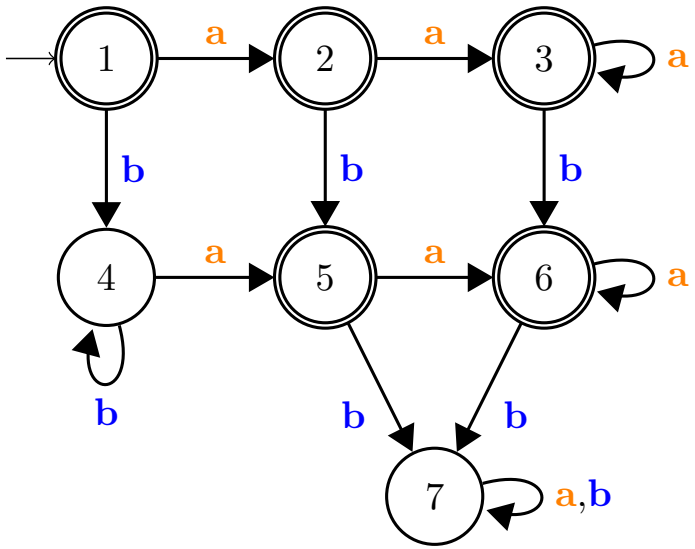
Theorem

Given A and its minimization $A_{/\equiv}$:

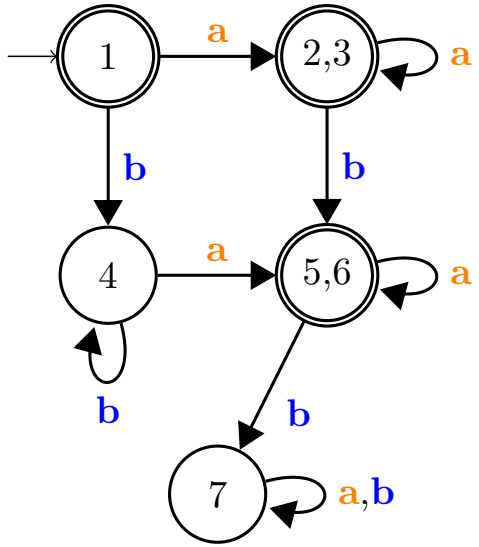
1. $L(A_{/\equiv}) = L(A)$;
2. $A_{/\equiv}$ is minimal for $L(A)$: there is no complete DFA recognizing $L(A)$ with fewer states than $A_{/\equiv}$.

Remarque The minimized automaton is unique up to isomorphism: any two minimal DFAs for the same language differ only by a renaming of states. □

Minimization: example



$\equiv_0$	$\equiv_1$	$\equiv_2$	$\equiv_3$
1	2	2	2
2	3	3	3
3	1	1	1
5	5	5	5
6	6	6	6
4	4	4	4
7	7	7	7



Why is the Minimization Algorithm Optimal?

Let $A = (Q, \Sigma, \delta, q_{\text{init}}, F)$ be a complete DFA where all states are reachable.

Let M be the minimized automaton obtained by the minimization algorithm.

Suppose there exists an automaton N that recognizes the same language as A but has fewer states than M.

- Apply the equivalence-testing procedure to M and N.
- The initial states of M and N are indistinguishable because $L(M) = L(N)$.
- If p and q are indistinguishable, then all their successors on any symbol are also indistinguishable (otherwise p and q would be distinguishable).
- Every state of M is indistinguishable from at least one state of N:
  - take p in M; there exists $w \in \Sigma^*$ from the initial state of M to p,
  - following w in N from its initial state reaches some state,
  - by induction, p and that state in N are indistinguishable.
- Since N has fewer states than M, two distinct states of M must be indistinguishable from the same state of N.
- By transitivity of indistinguishability, those two states of M are indistinguishable from each other.
- Contradiction: by correctness of the minimization algorithm, all states of M are pairwise distinguishable.

Outline Chap. 6 - DFA - Distinguishability, Equivalence, Minimisation

1. Testing Equivalence between States
2. Testing Automata Equivalence
3. Minimization of DFAs
4. Résumé

Summary of Chapter 6: *Equivalence and Minimization* of DFAs

Equivalence and Minimization of DFAs

- Distinguishability ($\neq$) and equivalence ($\equiv$) between states,
- Distinguishability and equivalence between automata,
- DFA minimization ($A_{/\equiv}$).

Bonus

- Explain why the algorithms for computing state distinguishability and equivalence relations always terminate.

Remarque Termination is guaranteed because the number of state pairs and partitions is finite. Each iteration strictly increases the set of marked pairs (for $\neq$) or refines the partition (for $\equiv$), so the process cannot continue indefinitely. □