Finite Automata

Prof. A. Morzenti

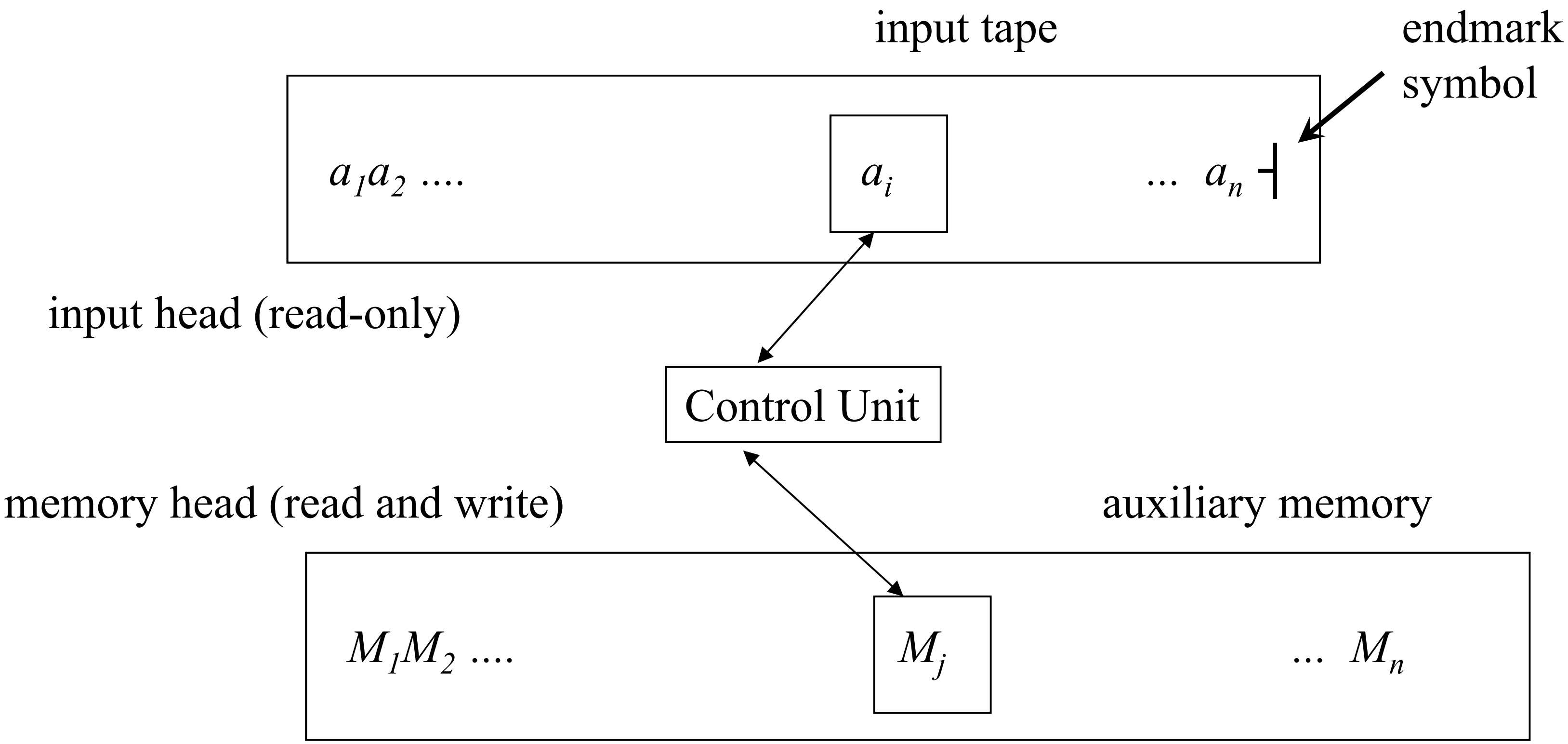
NB: up to slide 13 notions assumed to be well-known from other previous courses

Algorithms for language recognition are seen as automata, in order to:

- highlight relations between language families and grammars
- avoid early reference to software implementation details

It is then easy to provide indications to transform automata into executable code

SCHEME OF A RECOGNIZING AUTOMATON in its most general form:



AUXILIARY MEMORY

- MISSING (only that of the control unit): finite state automaton – it recognizes regular languages
- STACK MEMORY: pushdown automaton (PDA) – it recognizes CF languages

the automaton examines the source string, through a series of ***moves***:

every move depends on the symbols under the head**s** and on the state of the control unit

Effects of each move:

- moving the input head one position left or right
- write a symbol in memory and shift one position left or right
- change the state of the Control Unit

UNIDIRECTIONAL MACHINE: input head moves in one direction only

models a *single scan analysis*

instantaneous configuration determines future evolution
defined by three components:

- part of the input tape still unread
- content of the auxiliary memory and position of the memory head
- state of the control unit

initial configuration:

- input head on the first symbol
- control unit in the initial state
- memory storing the initial information (usually a special symbol)

computation: sequence of moves

deterministic – in every configuration at most one move (hence one next config.) is possible

nondeterministic otherwise

final configuration

- control unit in a final state
- input head on the string endmark (terminator) symbol '⊣'
    - sometimes alternative condition: memory contains a special symbol or empty

a string x is ***accepted*** (**recognized**) iff the automaton,
starting from initial configuration with $x \dashv$ as input
performs a computation and reaches a final configuration
(if it is nondeterministic it may do that in many ways)

A computation ends when the automaton

- reaches a final configuration ($\Rightarrow$ string is accepted), or
- no move can be executed ($\Rightarrow$ string not accepted)

language accepted or ***recognized*** by the automaton: the set of accepted strings

two automata accepting the same language are called ***equivalent***

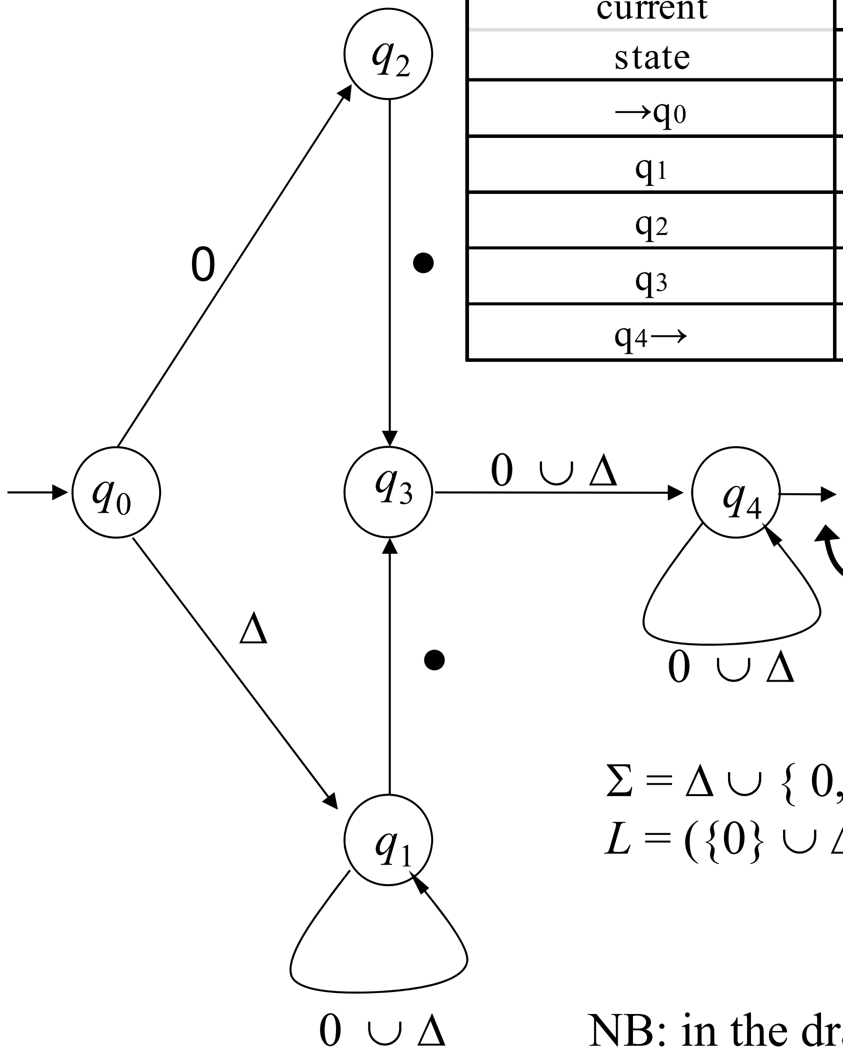
STATE-TRANSITION DIAGRAMS

it is a labeled oriented graph
nodes: states of the control unit
arcs: denote transitions, and are labeled with the input symbols

the state-transition diagram has (in the deterministic version)

- a unique initial state
- possibly many final states

Example – decimal numeric constants



current state	current input symbol				
	0	1	...	9	•
$\rightarrow q_0$	q_2	q_1	...	q_1	
q_1	q_1	q_1	...	q_1	q_3
q_2			...		q_3
q_3	q_4	q_4	...	q_4	
$q_4\rightarrow$	q_4	q_4	...	q_4	

NB: dart (little arrow) denotes final state

$\Sigma = \Delta \cup \{ 0, \bullet \}$, where $\Delta = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$
$L = (\{0\} \cup \Delta (\{0\} \cup \Delta)^*) \bullet (\{0\} \cup \Delta)^+$

NB: in the drawing several topologically equivalent arcs are merged (e.g. from q_0 to q_1 there is a bundle of 9 arcs ....).

FORMAL DEFINITION OF A DETERMINISTIC FINITE AUTOMATON

Five elements:

1. $\boldsymbol{Q}$, the set of ***states*** (non-empty, finite)
2. $\boldsymbol{\Sigma}$, the *input alphabet* (or terminal alphabet)
3. the *transition function* (possibly partial) $\delta : (Q \times \Sigma) \to Q$
4. $\boldsymbol{q_0 \in Q}$ the *initial state*
5. $\boldsymbol{F \subseteq Q}$, the set of ***final states***

the transition function encodes the moves of the automaton $\boldsymbol{M}$:

if $\boldsymbol{\delta(q, a) = r}$ ***then*** $\boldsymbol{M}$, in the state $\boldsymbol{q}$, when reading $\boldsymbol{a}$ goes into state $\boldsymbol{r}$

alternative notation to $\boldsymbol{\delta(q, a) = r}$ **:** $\boldsymbol{q \xrightarrow{a} r}$

if $\boldsymbol{\delta(q, a)}$ is undefined, the automaton stops ($\equiv$ it enters a non-final error state and rejects)

extension $\boldsymbol{\delta^*}$ of $\boldsymbol{\delta}$ for the strings of any length: $\boldsymbol{\delta^*: (Q \times \Sigma^*) \to Q}$

defined inductively as $\boldsymbol{\delta^*(q, \varepsilon) = q}$ and $\boldsymbol{\delta^*(q, xa) = \delta(\delta^*(q, x), a)}$, $\boldsymbol{x \in \Sigma^*, a \in \Sigma}$

for brevity we use $\boldsymbol{\delta}$ also to denote its extension $\boldsymbol{\delta^*}$

COMPUTATION CARRIED OUT BY AN AUTOMATON = PATH ON THE GRAPH

STRING RECOGNITION :

string x recognized (or accepted) by automaton M iff
when scanning x, M goes from the initial to a final state:

$$\boldsymbol{\delta(q_0, x) \in F}$$

Hence the empty string ε accepted iff the initial state is also final language recognized (accepted) by atomaton M : the set of all recognized strings

$$L(M) = \{x \in \Sigma^* \mid \delta(q_0, x) \in F\}$$

the family of languages accepted by finite state automata is called ***finite-state recognizable***

complexity of acceptance is called «real-time»: number of steps is equal to $|\boldsymbol{x}|$

Example – Decimal numeric constants (follows) – The automaton *M* (see next) defined as:

$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

$$\Sigma = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, \bullet\}$$

$$q_0 = q_0$$

$$F = \{q_4\}$$

examples of transitions:

$$\delta(q_0, 3\bullet 1) = \delta(\delta(q_0, 3\bullet), 1) = \delta(\delta(\delta(q_0, 3), \bullet), 1) =$$

$$= \delta(\delta(q_1, \bullet), 1) = \delta(q_3, 1) = q_4$$

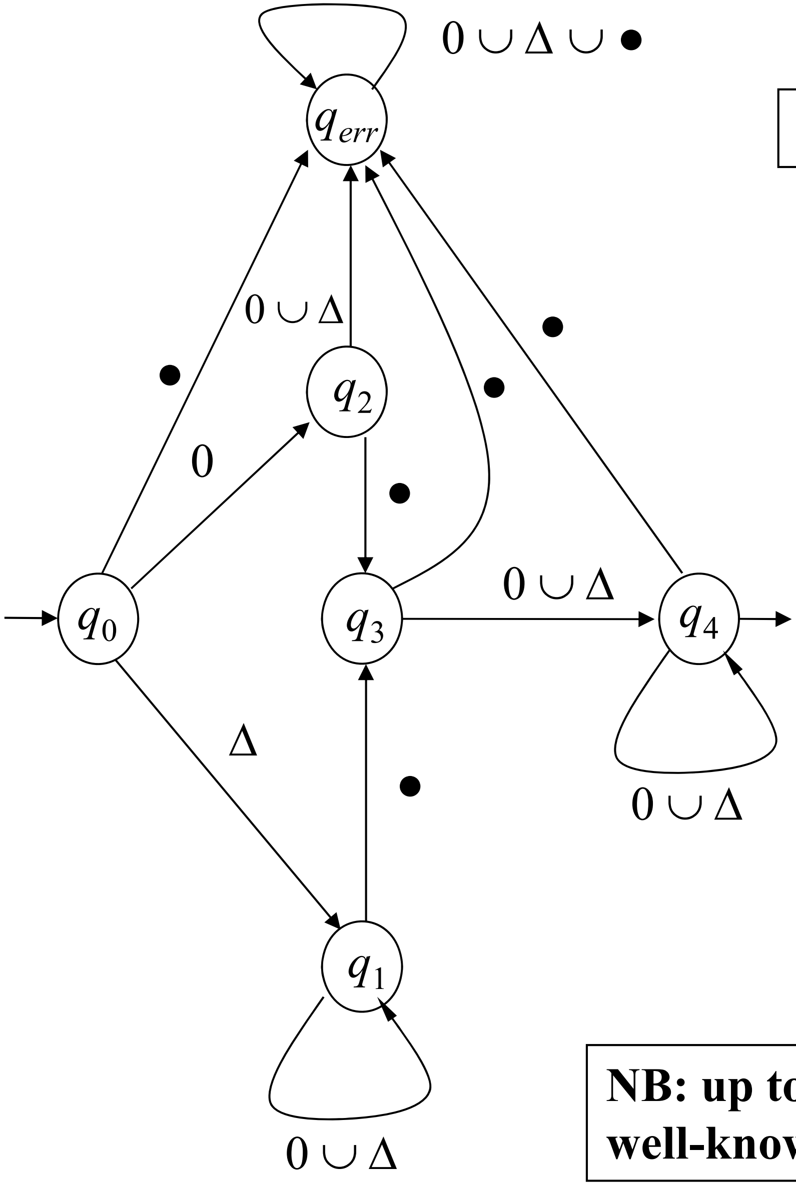
$q_4 \in F$ the string $3\bullet 1$ is accepted

strings that are not accepted: $3\bullet$ and 02

$\delta(q_0, 3\bullet) = q_3$ - q_3 not final - $3\bullet \notin L$

$\delta(q_0, 02) = \delta(\delta(q_0, 0), 2) = \delta(q_2, 2)$ - undefined - $02 \notin L$

COMPLETING THE AUTOMATON WITH THE ERROR STATE



q_{err} = error «sink» state

the transition function can always be made complete by means of an ***error state*** without changing the accepted language

$\forall$ state $q \in Q$ and $\forall$ symbol $a \in \Sigma$

if $\delta(q,a)$ is undefined

let $\delta(q,a) = q_{err}$

and $\forall$ symbol $a \in \Sigma$ let $\delta(q_{err},a) = q_{err}$

NB: up to here notions assumed to be well-known from other previous courses

CLEAN AUTOMATA

An automaton can include useless parts that do not contribute to string recognition

they must be eliminated

A state ***q*** is ***reachable*** from a state ***p*** if there exists a computation going from p to q

A state is ***accessible*** if it is reachable from the initial state

A state is ***postaccessible*** if a final state can be reached from it

(⇒ final states are postaccessible by definition

⇒ NB: the error state q_{err} is not postaccessible)

A state is ***useful*** if it is accessible and postaccessible

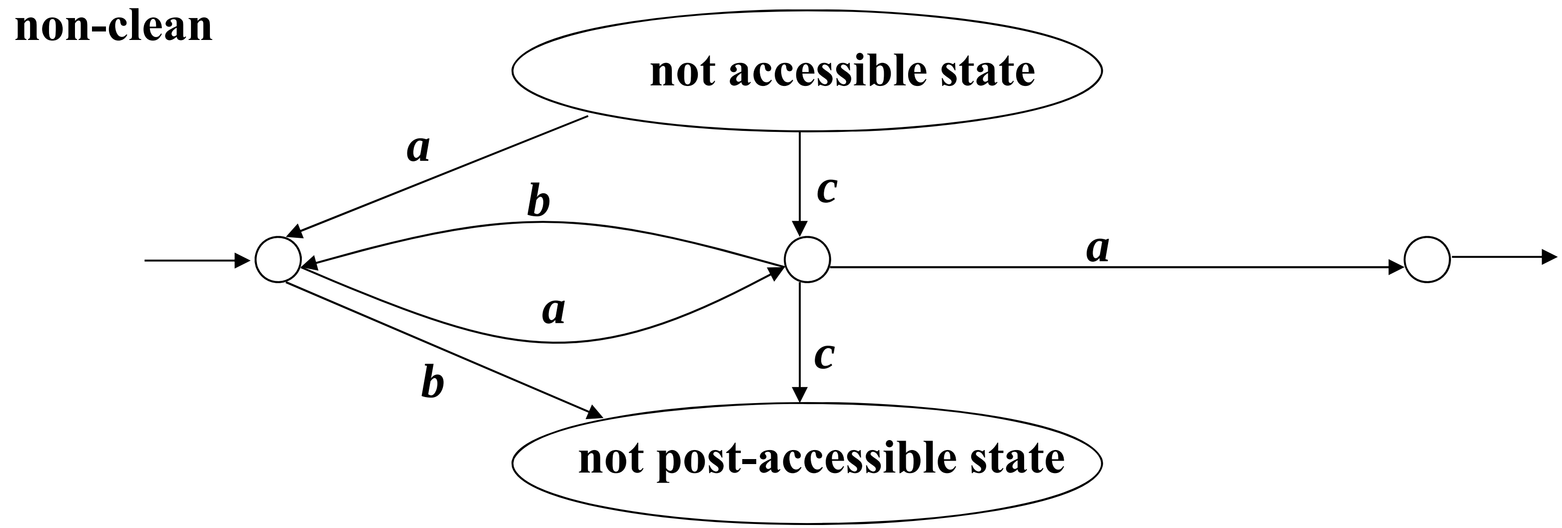
(it lays on some path from initial to final state)

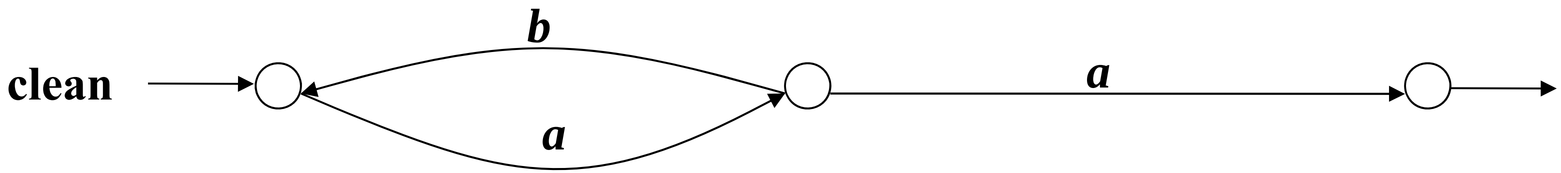
An automaton is ***clean*** if every state is useful

PROPERTY – Every finite automaton admits an equivalent clean automaton.

Cleaning an automaton: identify useless states, delete them and all the incident arcs

Example – useless state elimination





MINIMAL AUTOMATON

PROPERTY – For every finite-state language, the finite recognizer is minimal w.r.t. the number of states ***exists and is unique*** (apart from a renaming of states)

We provide a procedure for minimizing the number of states
assuming the automaton is clean except for the possible presence of error state q_{err}

INDISTINGUISHABLE STATES – state p is indistinguishable from state q,
iff, $\forall$ **string** $\boldsymbol{x}$, $\delta(p, x)$ and $\delta(q, x)$ ar both final, or both nonfinal
(i.e., scannig x from p and from q, one ***cannot*** reach two states, one final and the other not)

indistinguishability is a ***binary relation***; it is ***reflexive***, ***symmetric***, and ***transitive***

hence it is an ***equivalence relation***

two indistinguishable states can be ***merged***, thus reducing the number of states,
with no change in the language recognized by the automaton

it is a typical construction: the new set of states is the ***quotient set*** w.r.t. the equivalence class

Impossible to compute the indistinguishability relation ***directly*** from its definition

one should consider the whole accepted language, which may be infinite

we compute the indistinguishability relation through its complement:

the ***distinguishability*** relation

it can be computed through its ***inductive definition***

p is ***distinguishable*** from q iff

1. p is final and q is not, or viceversa, or
2. $\exists a$: $\delta(p, a)$ is distinguishable from $\delta(q, a)$

$\Rightarrow$ q_{err} is distinguishable from every postaccessible state p, because

$\exists$ string x : $\delta(p, x) \in F$ (just because p is postaccessible)

whereas $\forall$ string x: $\delta(q_{err}, x) = q_{err}$

$\Rightarrow$ p is distinguishable from q (both assumed postaccessible)

if the set of labels on arcs outgoing from p and q are different

(NB: not necessarily disjointed)

In fact, if $\exists a$ such that $\delta(p, a) = p'$, with p' postaccessible, whereas $\delta(q, a) = q_{err}$,

then p is distinguishable from q

because q_{err} is distinguishable from all postaccessible states

Example

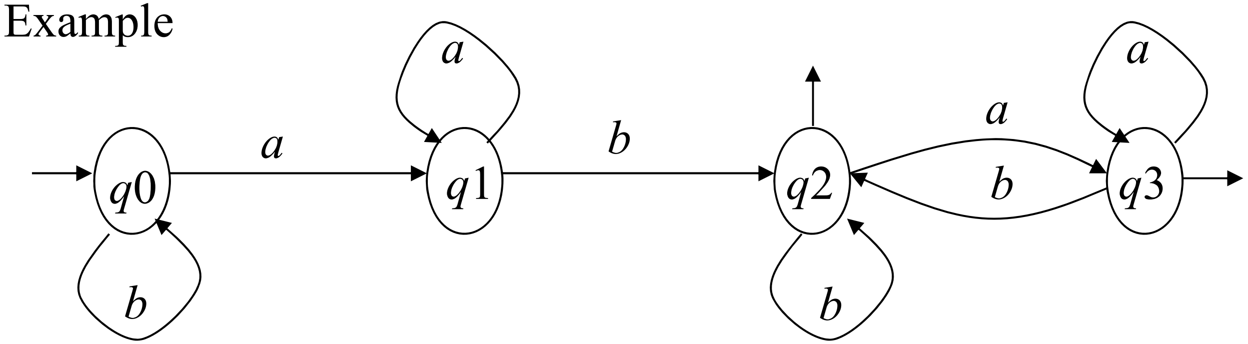


Table of indistinguishable states: initially final and nonfinal states are distinguishable

q1			
q2	X	X	
q3	X	X	
	q0	q1	q2

(δ(q0,a), δ(q1,a))=(q1,q1)
(δ(q0,b), δ(q1,b))=(q0,q2)
q0 dist. q2 ➔ q0 dist. q1
(δ(q2,a), δ(q3,a))=(q3,q3)
(δ(q2,b), δ(q3,b))=(q2,q2)
q3=q3, q2=q2 ➔ q2 indist. q3

q1	(1,1)(0,2)		
q2	X	X	
q3	X	X	(3,3)(2,2)
	q0	q1	q2

indistinguishable pairs: q_2 and q_3
equivalence classes of indistinguishable states:
[q0], [q1], [q2, q3].

q1	X		
q2	X	X	
q3	X	X	
	q0	q1	q2

MINIMIZATION

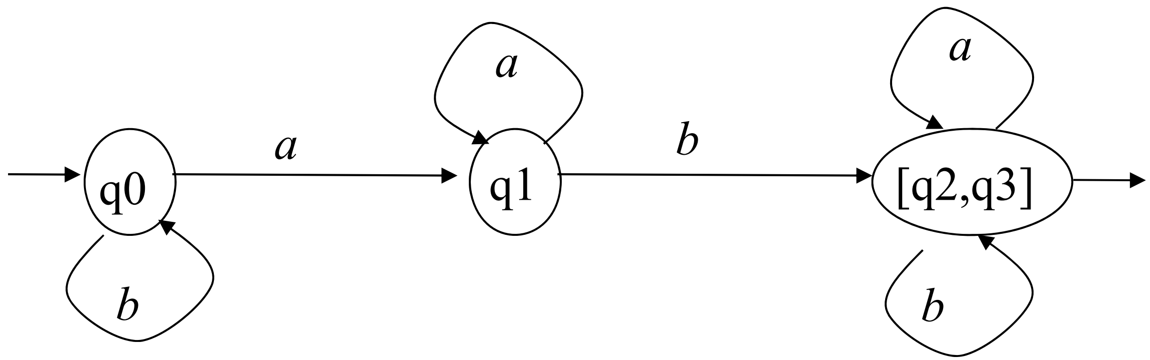
states of the minimal automaton *M'*: the equivalence classes of the indistinguishability relation

transition function: arcs among equivalence classes:

$\boxed{[\ldots, p_r, \ldots] \xrightarrow{b} [\ldots, q_s, \ldots]}$ iff in *M* there is an arc: $\boxed{p_r \xrightarrow{b} q_s}$

Example (follows)

Result of minimization:



Example with non-total transition function

Suppose to modify the previous automaton M by removing the move $\delta(q3, a) = q3$.

We redefine as $\delta(q3, a) = q_{err}$

$q2$ and $q3$ are now distinguishable:

$$\delta(q2, a) = q3 \text{ and } \delta(q3, a) = q_{err} \text{ and } q3 \text{ is distinguishable from } q_{err}$$

M is therefore already minimal

The minimization procedure provides a proof of the existence and unicity of a minimum automaton equivalent to any given one.

NB: This property does not hold, in general, for ***nondeterministic automata*** (coming next)

State minimization provides a method for ***checking* (*deterministic*) *automata equivalence***:

- clean the automata,
- minimize them,
- check if they are identical (apart from a renaming of states)