Finite Automata

Prof. A. Morzenti

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

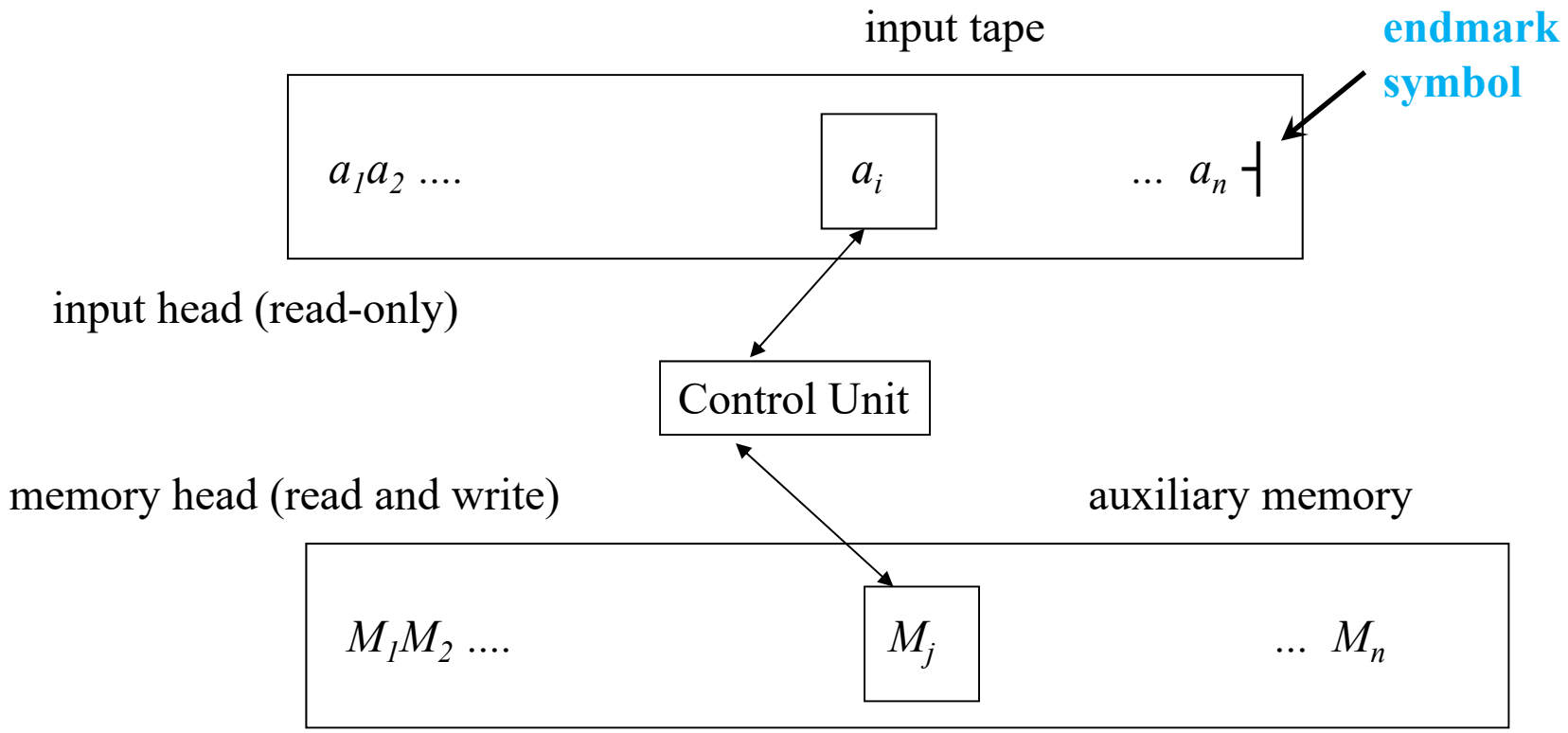
Algorithms for language recognition are modeled as automata, in order to:

- highlight relations between language families and grammars

- avoid early reference to software implementation details

It is then easy 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 '⊣'

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

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***

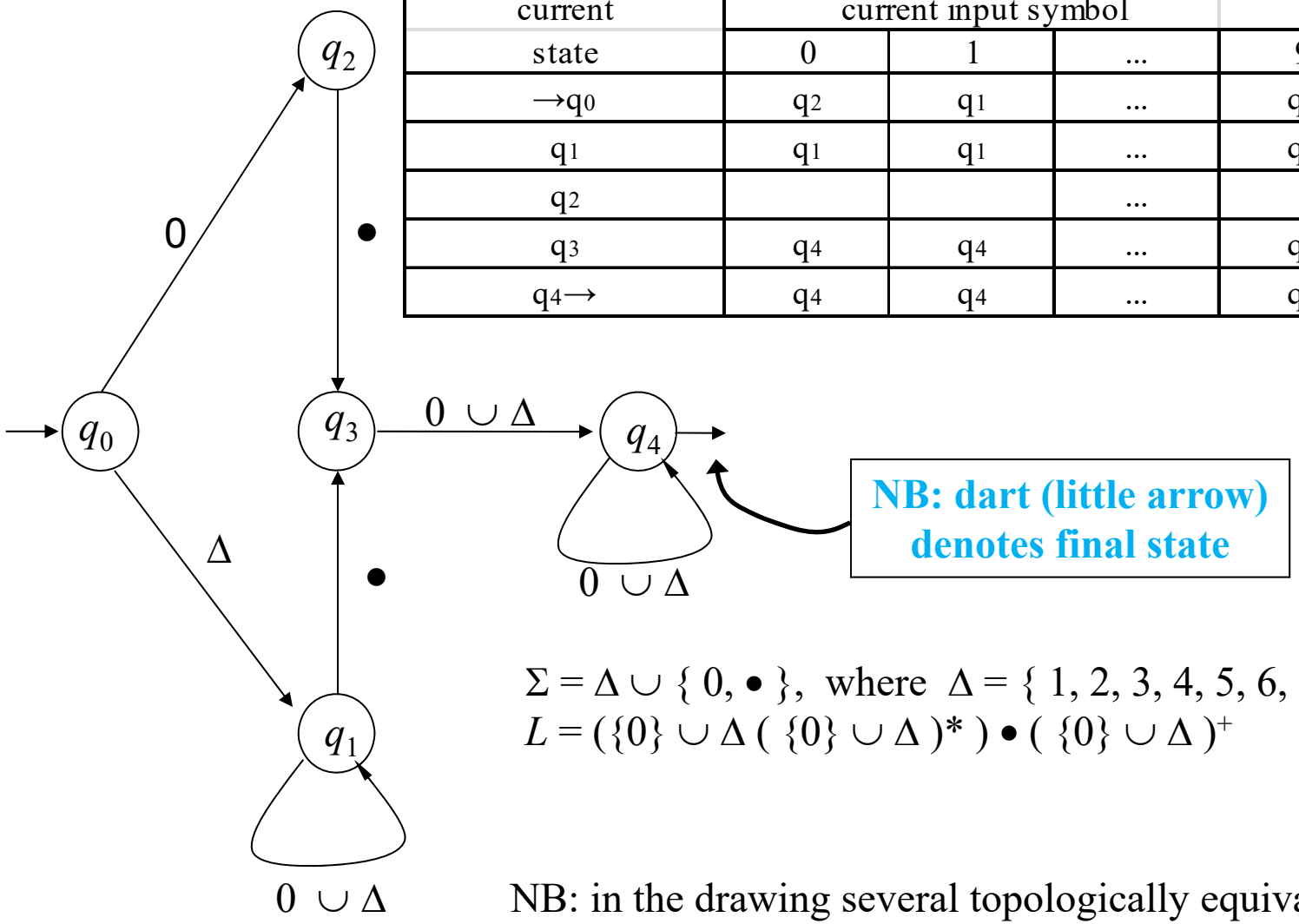
STATE-TRANSITION DIAGRAMS

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	•
→q0	q2	q1	...	q1	
q1	q1	q1	...	q1	q3
q2			...		q3
q3	q4	q4	...	q4	
q4→	q4	q4	...	q4	

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

the transition function encodes the moves of the automaton M:

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

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

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

extension δ^* of δ for the strings of any length: $\delta^* : (Q \times \Sigma^*) \rightarrow Q$

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

for brevity we use δ also to denote its extension δ^*

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:

$$\delta(q_0, x) \in F$$

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 $|x|$

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:

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

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

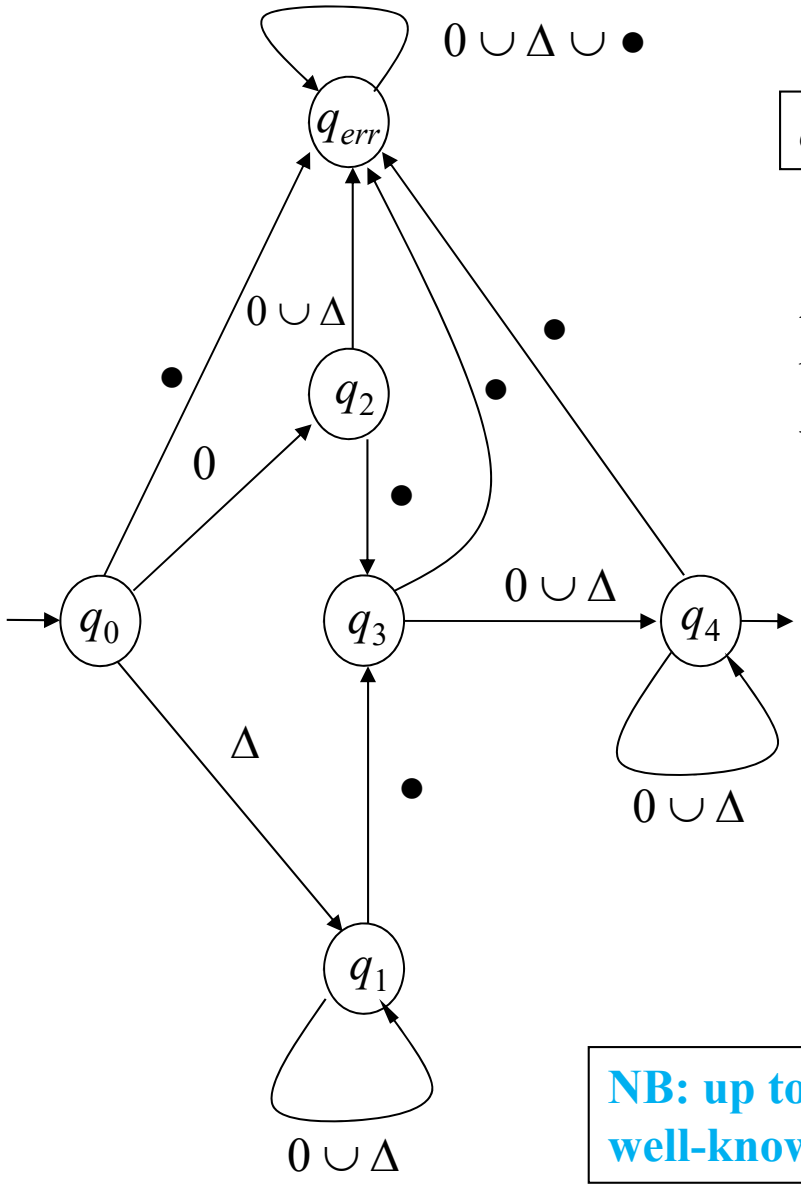
$q_4 \in F$ the string $3 \bullet 1$ is accepted

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

($\Rightarrow$ NB: the error state q_{err} is not postaccessible)

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

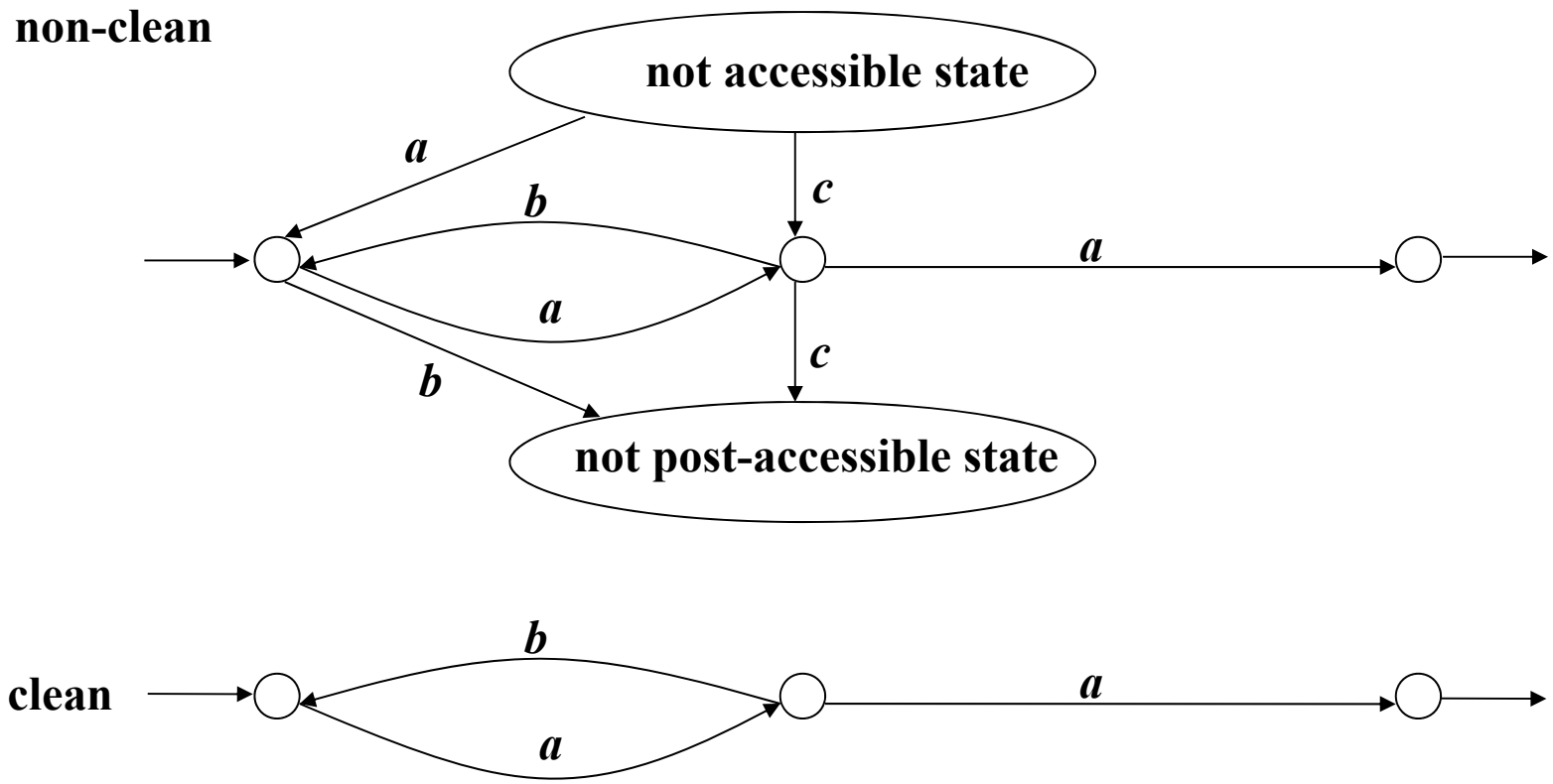
(it lays on some path from initial to final state)

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 $\delta(q, x)$ are 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: $\delta(q_{err}, x) = q_{err} \notin F$

$\Rightarrow p$ 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 p' (as from all postaccessible states)

Example

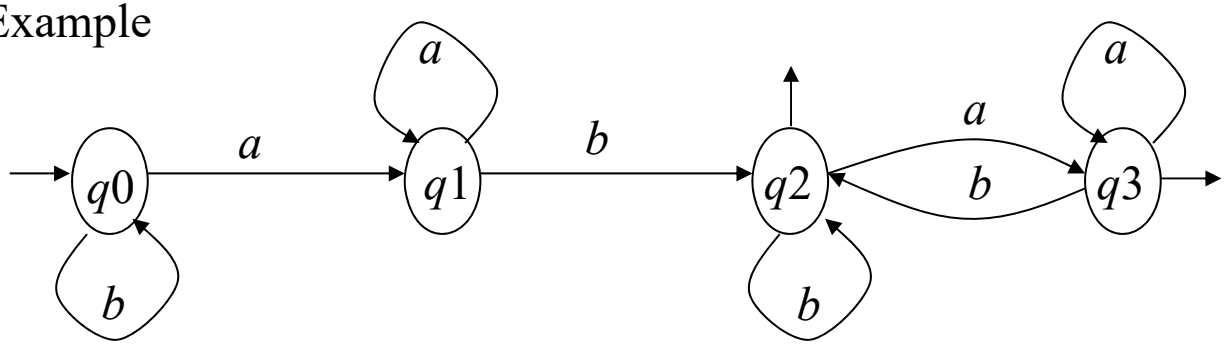


Table of indistinguishable states: first of all, final and nonfinal states are distinguishable

q1			
q2	X	X	
q3	X	X	
	q0	q1	q2

$(\delta(q0,a), \delta(q1,a))=(q1,q1)$
$(\delta(q0,b), \delta(q1,b))=(q0,q2)$
q0 dist. q2 ➔ q0 dist. q1
$(\delta(q2,a), \delta(q3,a))=(q3,q3)$
$(\delta(q2,b), \delta(q3,b))=(q2,q2)$
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:

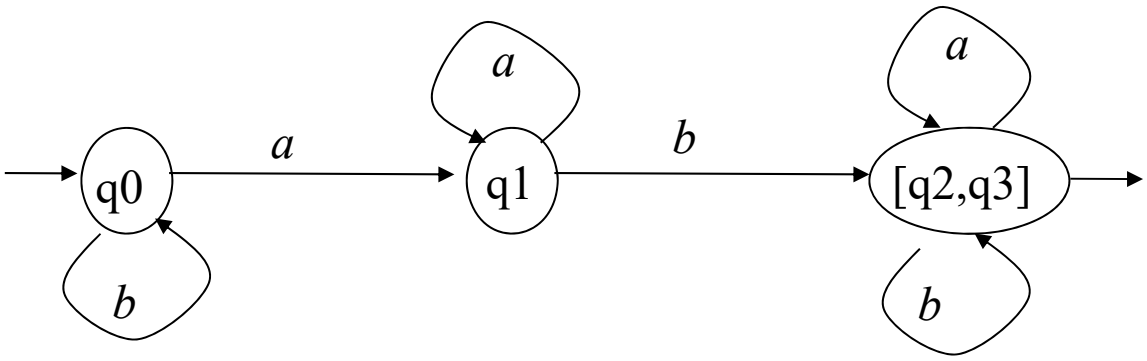
$$[\ldots, p_r, \ldots] \xrightarrow{b} [\ldots, q_s, \ldots]$$

iff in *M* there is an arc:

$$p_r \xrightarrow{b} q_s$$

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:

$\delta\,(q2, a) = q3$ and $\delta(q3, a) = q_{err}$ and $q3$ 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)

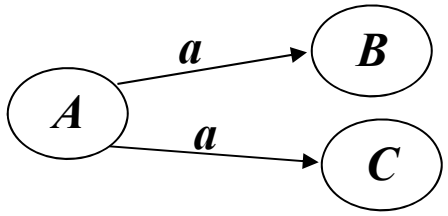
State minimization provides a method for ***checking*** **(*deterministic*)** ***automata equivalence***:

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

Nondeterministic Finite Automata

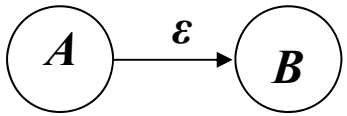
FORMS OF NONDETERMINISM

1) alternative moves for a unique input

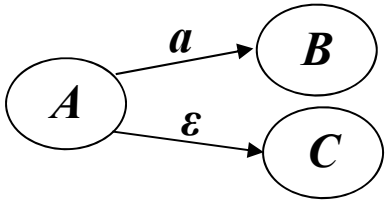


$$\delta(A,a)=\{B,C\}$$

2) spontaneous move (or ε-move): automaton changes its state without "consuming" input



$$\delta(A,\varepsilon)=B$$



$$\delta(A,a)=B \quad \delta(A,\varepsilon)=C$$

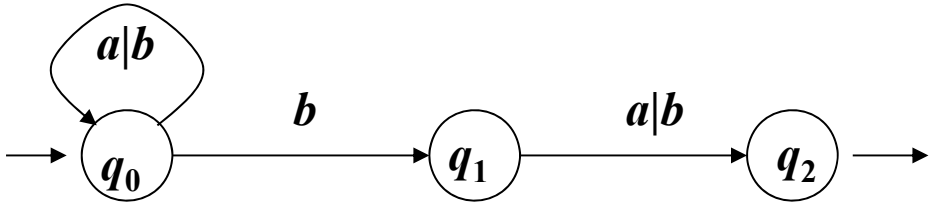
3) more than one initial state (the automaton may start the computation from any of them)

THREE MOTIVATIONS FOR NONDETERMINISM IN FINITE STATE AUTOMATA

1) **conciseness**: language definitions through ND automata are more compact and readable

Example – language with second last symbol = *b*

$$L_2 = (a \mid b)^* b(a \mid b)$$

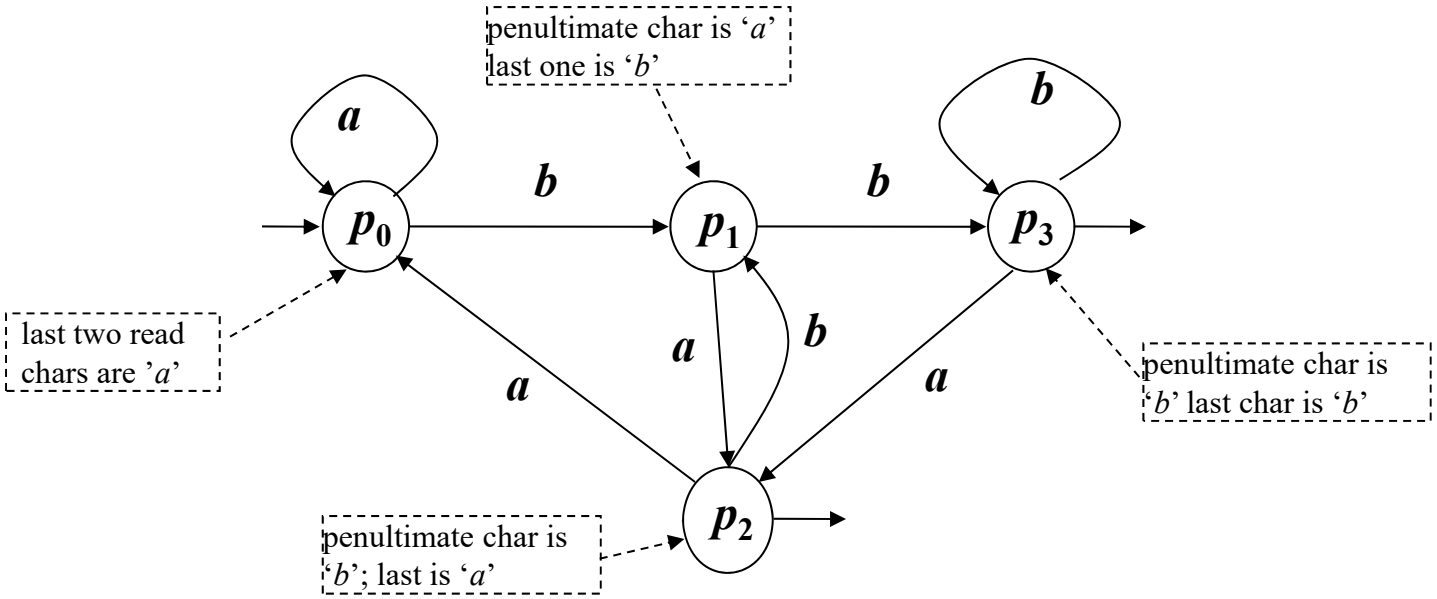


recognition of *baba*
two computations, only one accepting

$$q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_2$$

$$q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0$$

the same language accepted by a ***deterministic*** automaton *M*2
in *M*2 the condition that the symbol before the last one is a *b* is not so evident



Exercise: show/prove that
Generalizing the example, from language L_2 to language L_k where the *k*-th last element ($k \geq 2$) is *b*, the ***nondeterministic*** automaton has ***k + 1 states***,
the number of states of the ***deterministic*** one grows ***exponentially*** with ***k*** **($\approx 2^k$)**

nondeterminism can make many definitions much more concise

2) **left – right duality**

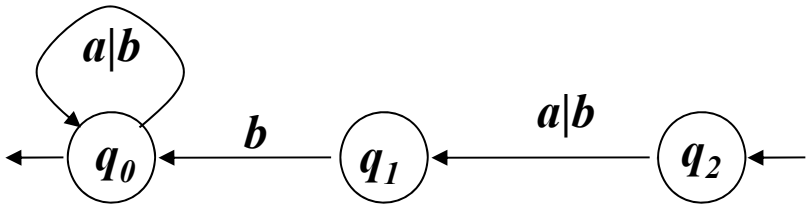
going from a (deterministic) automaton for lang. L to that for L^R requires to:

1. switch initial and final states
2. reverse the arrows direction

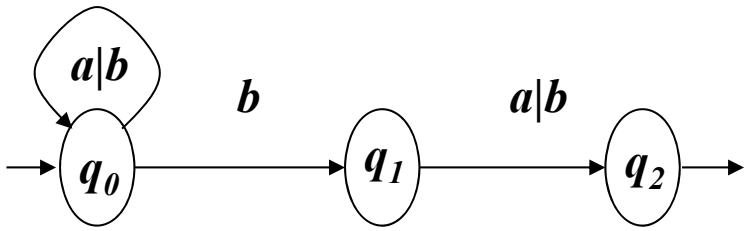
Both operations may introduce nondeterminism

Example - The language L_2 of strings having b as the second last symbol is the reverse image of language L' of strings having b as the second symbol

$$L' = \{x \mid b \text{ is the second symbol of } x\} \quad L' = (L_2)^R$$



L': ***b*** second char: deterministic



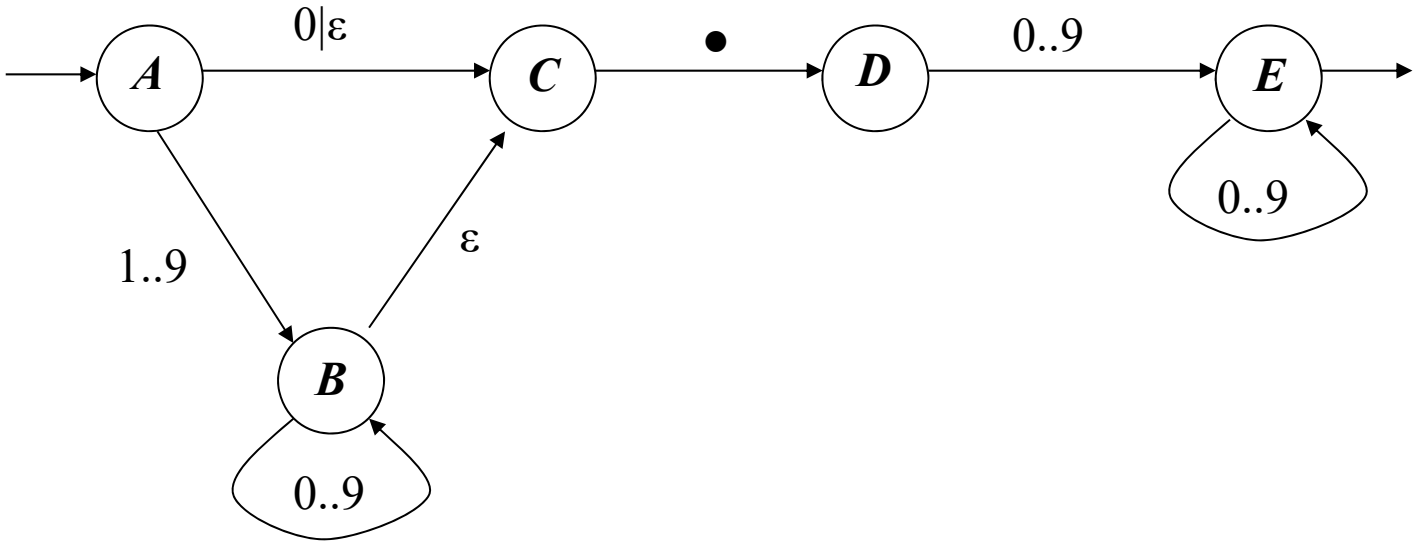
L_2 : ***b*** second last char: nondeterministic

3) **automata design**: the transition through nondeterministic automata is useful in the construction of the finite recognizer of a language defined by a regular expression (see coming lessons)

AUTOMATA WITH SPONTANEOUS MOVES

Example – decimal constants (with or without 0 before the dot, with no leading 0's in the integer part)

$$L = \left(0 \mid \varepsilon \mid \left((1..9)(0..9)^{*}\right)\right) \bullet (0..9)^{+}$$



When the automaton has spontaneous moves, the computation can be longer than the string (NB: the property of real-time analysis does not hold)

Computation accepting string 34●5: $A \xrightarrow{3} B \xrightarrow{4} B \xrightarrow{\varepsilon} C \xrightarrow{\bullet} D \xrightarrow{5} E$

Computation accepting string ●02: $A \xrightarrow{\varepsilon} C \xrightarrow{\bullet} D \xrightarrow{0} E \xrightarrow{2} E$