COM2109

Automata

Robert Hierons

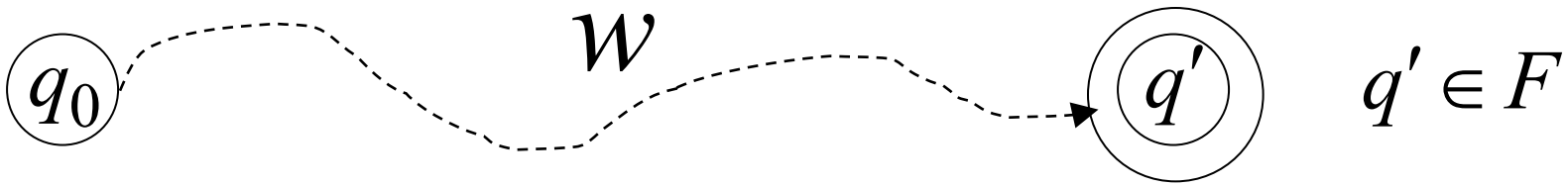
Based on the work of Mike Stannett,
Lucia Specia, M.S. Moorthy and Costas Busch

Revision: DFA, NFA, and Regular Languages

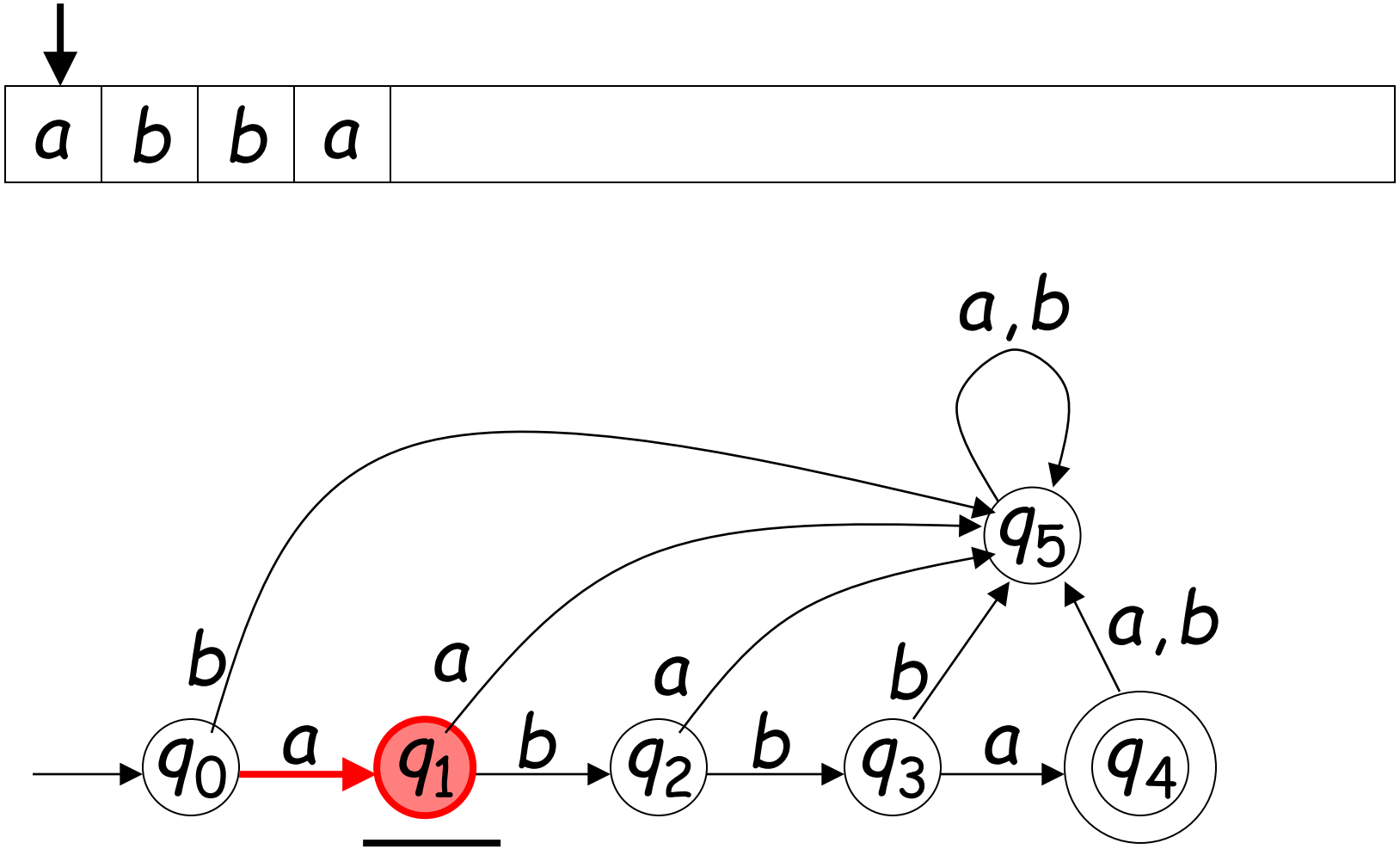
For a DFA $M = (Q, \Sigma, \delta, q_0, F)$

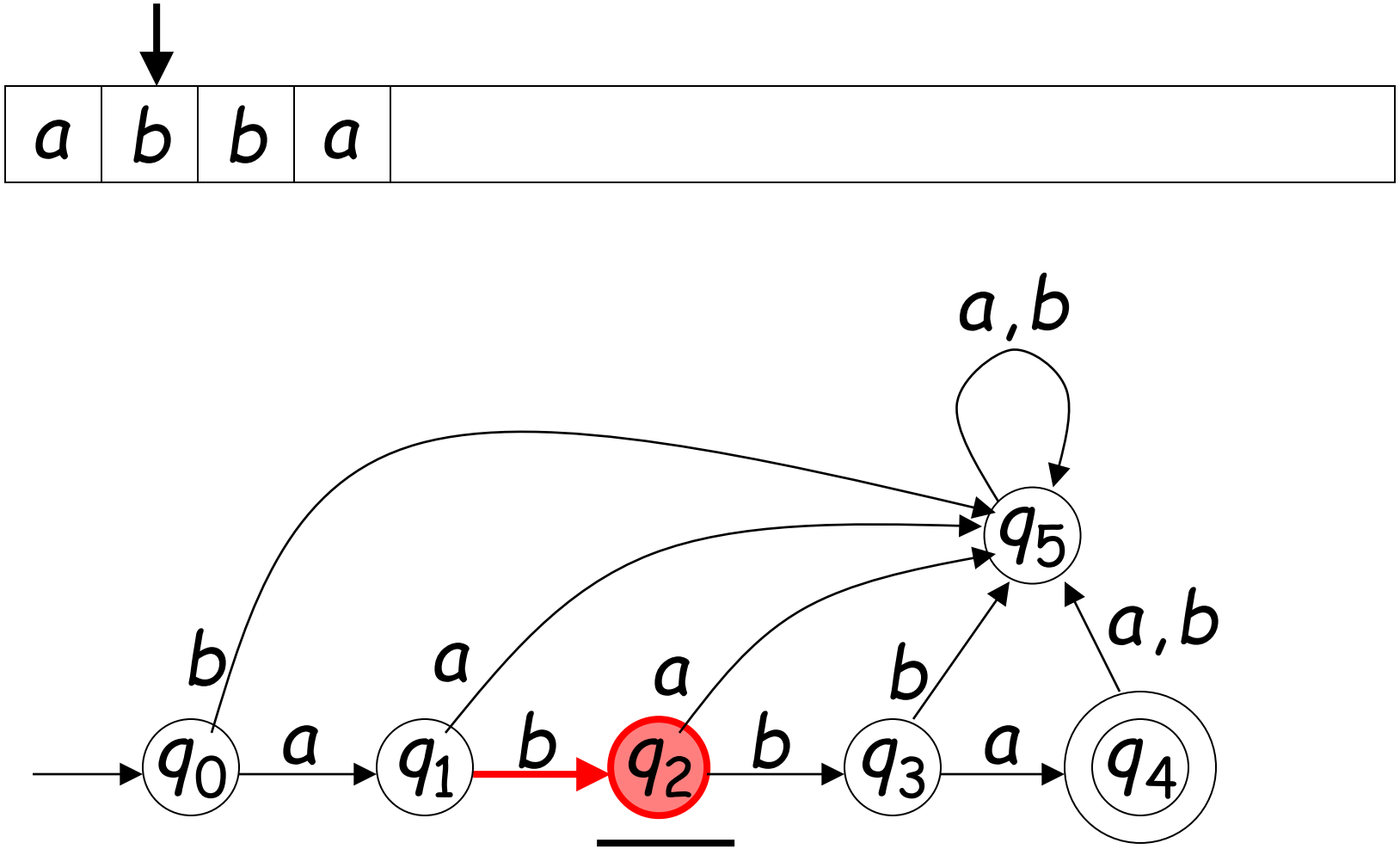
Language accepted by M :

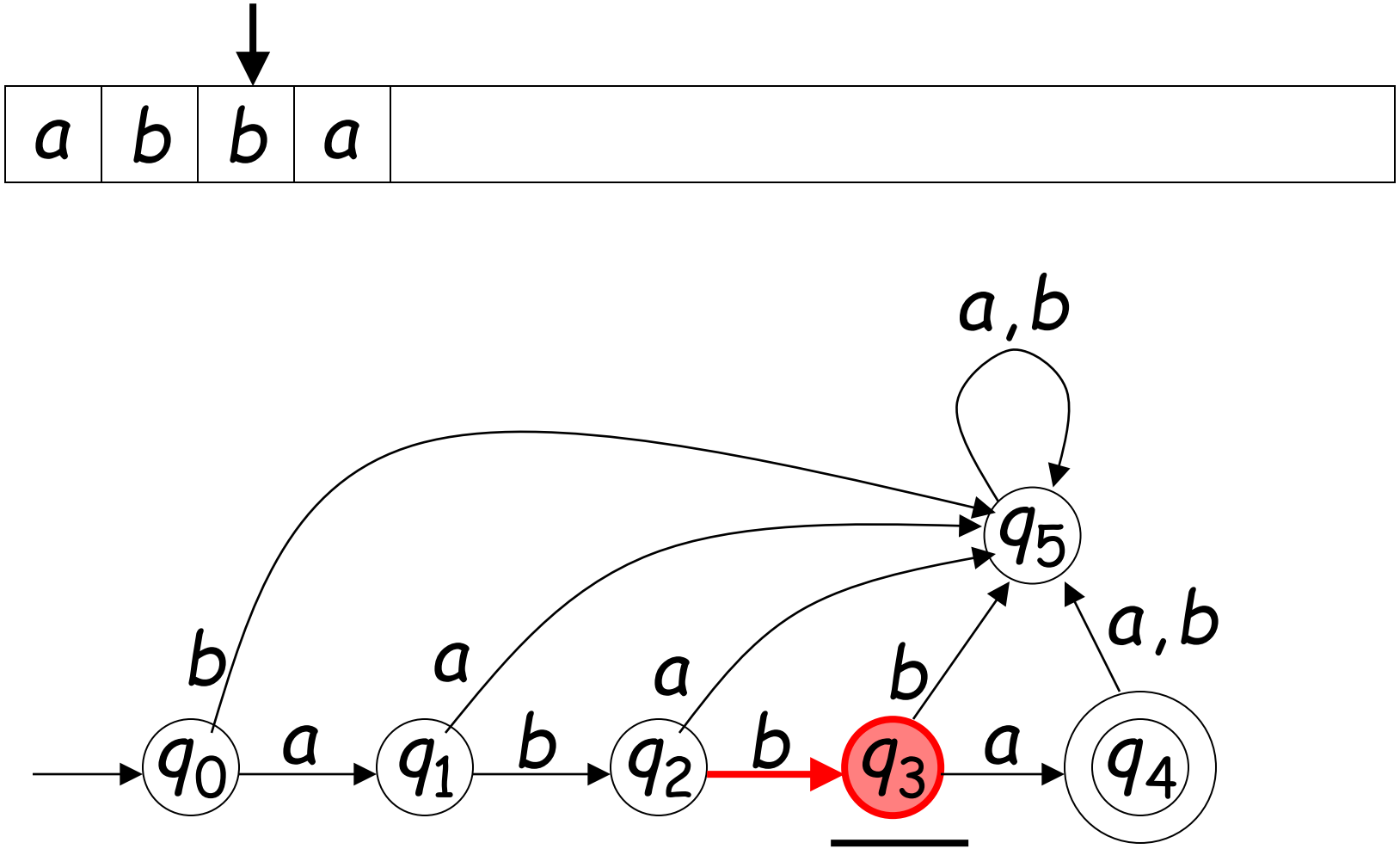
$$L(M) = \{ w \in \Sigma^* : \hat{\delta}(q_0, w) \in F \}$$



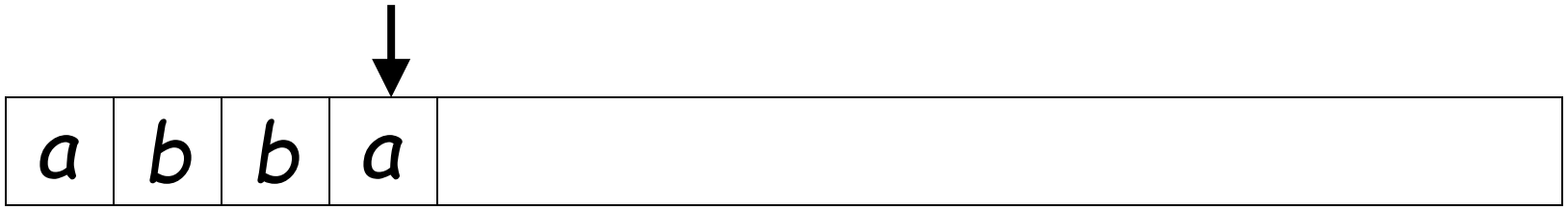
Scanning the Input

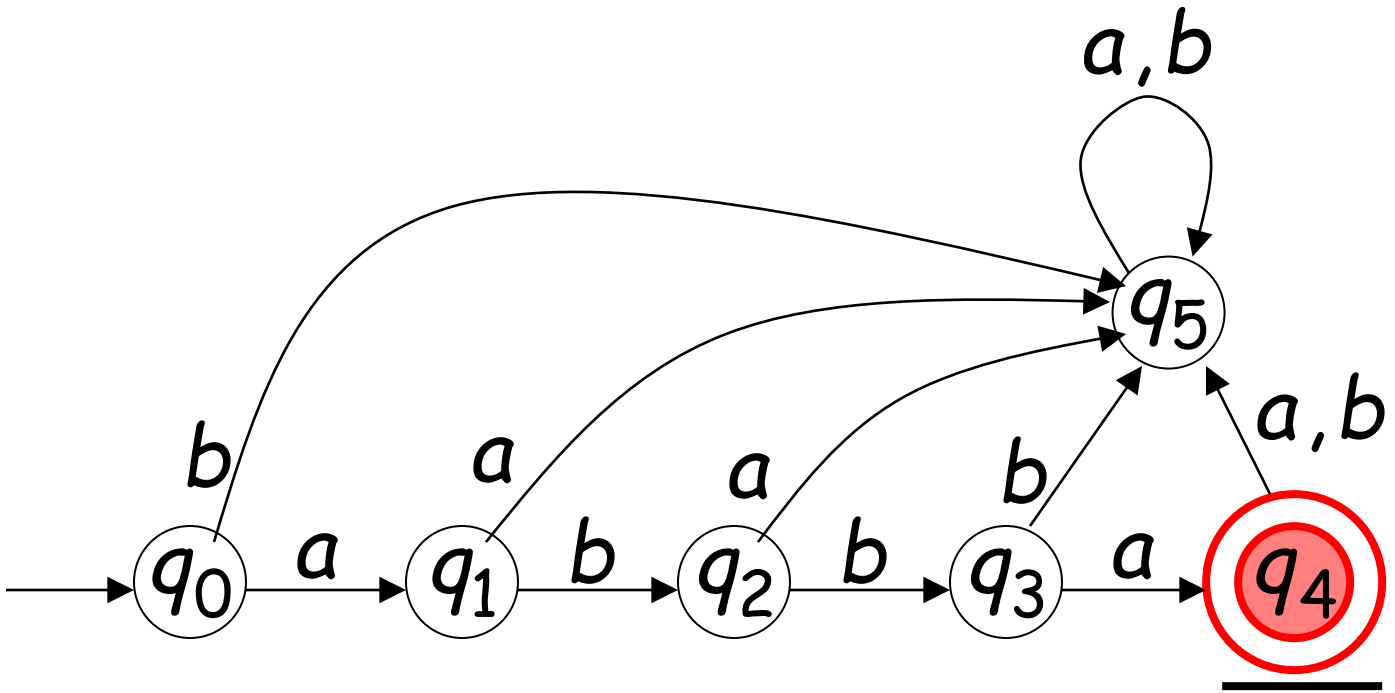



a b b a
a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4


a b b a
a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4

Input finished





accept

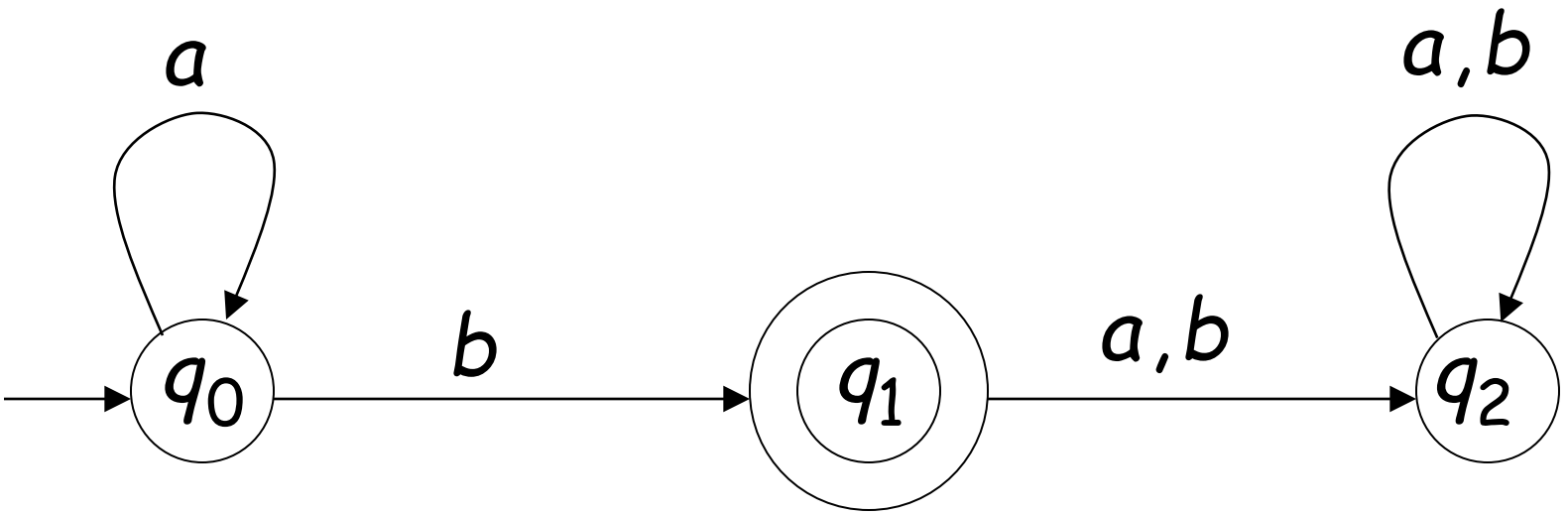
To accept a string:

all the input string is scanned
and the last state is an accepting state

To reject a string:

all the input string is scanned
and the last state is not an accepting state

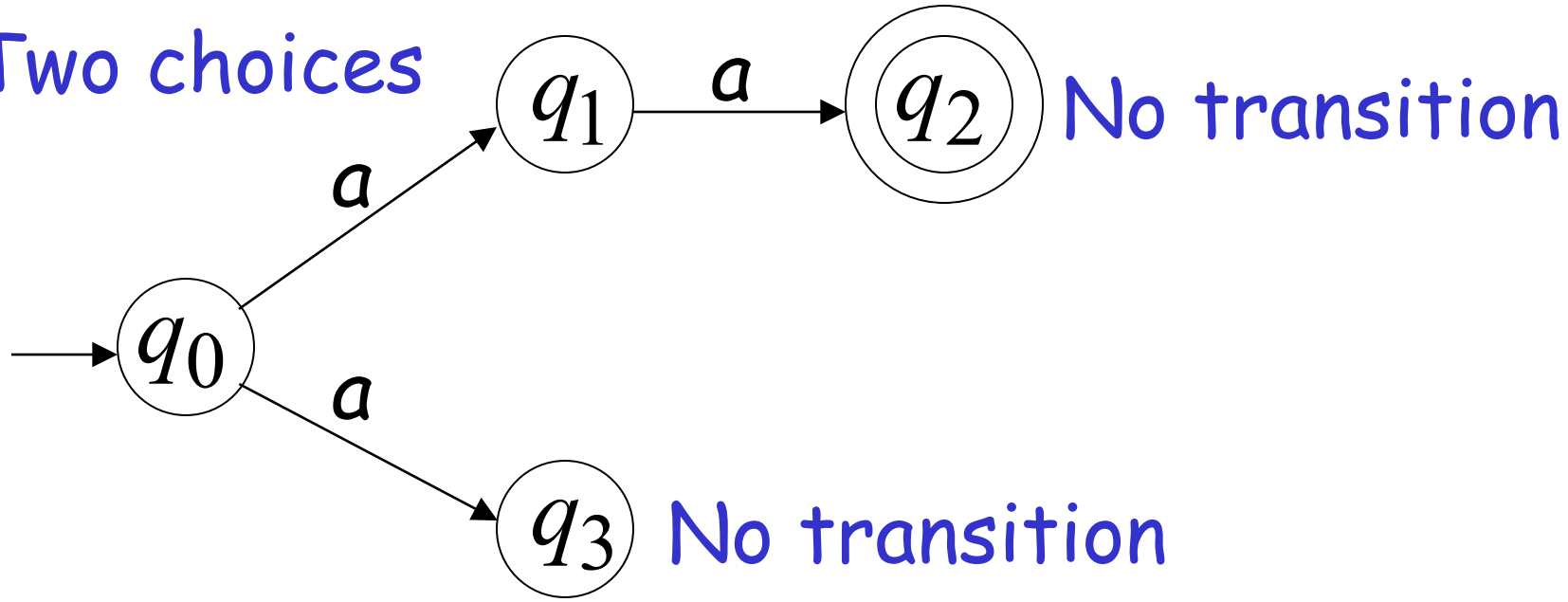
Language Accepted: $L = \{a^n b : n \geq 0\}$



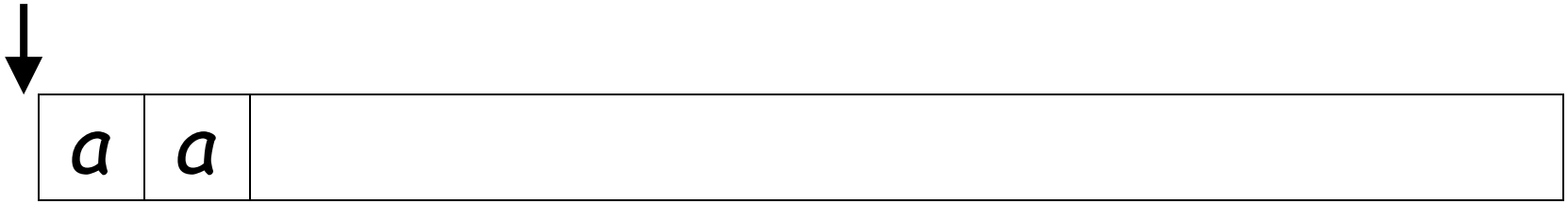
Non-Deterministic Finite Automata

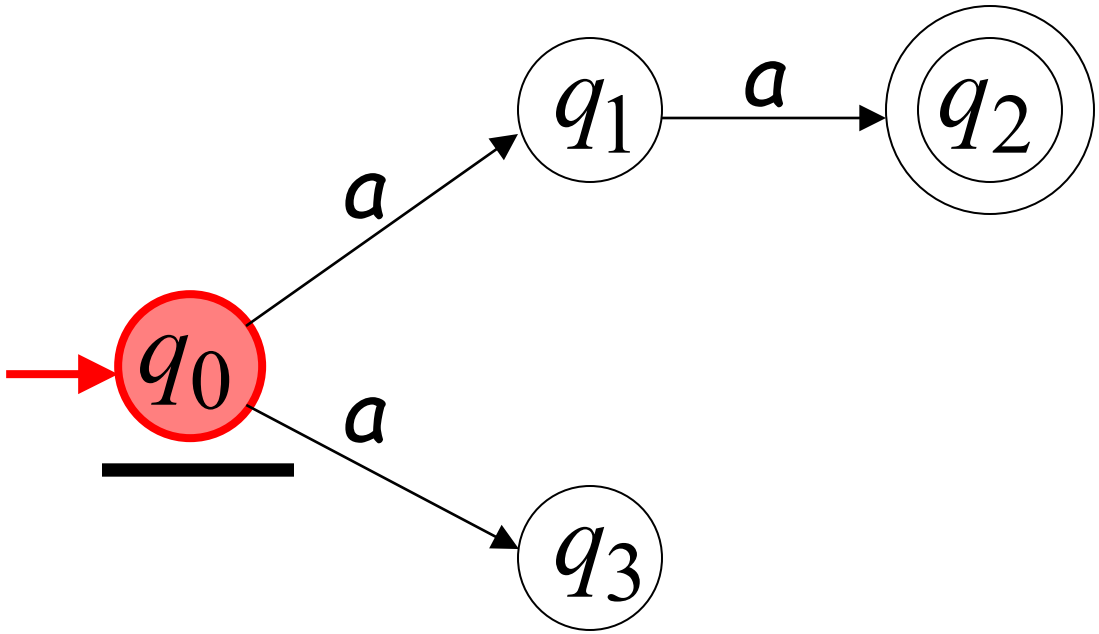
Nondeterministic Finite Automaton (NFA)

Alphabet = $\{a\}$

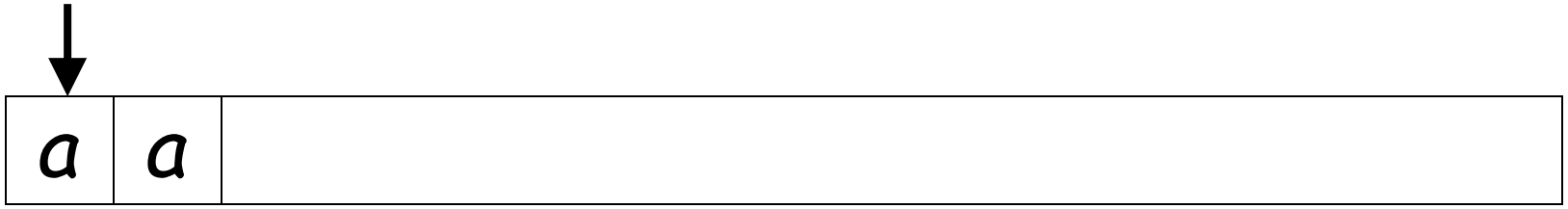


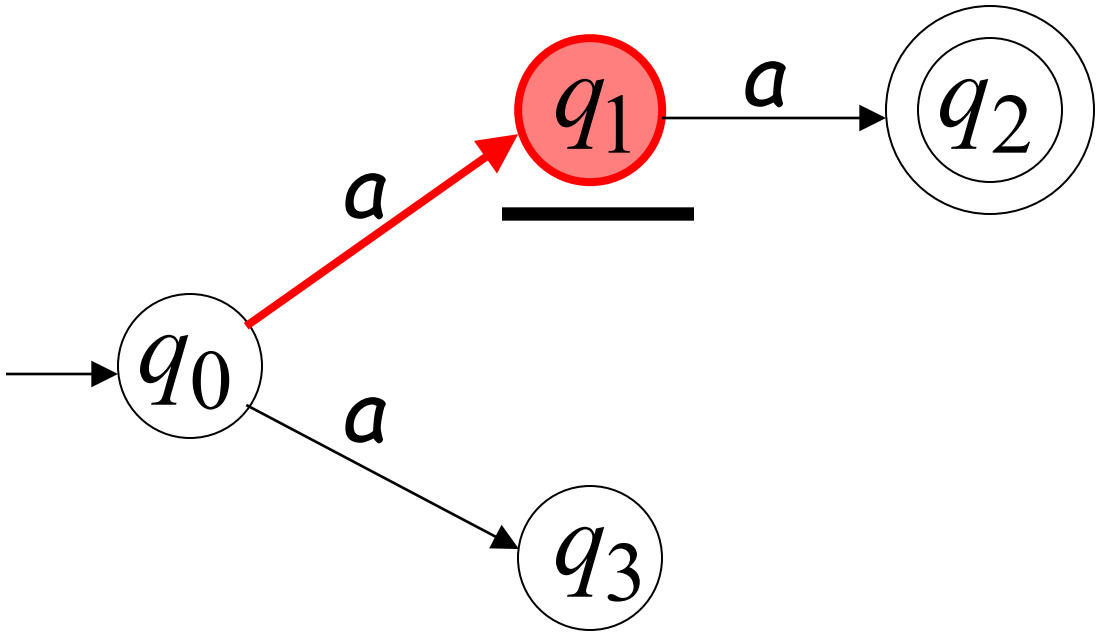
First Choice



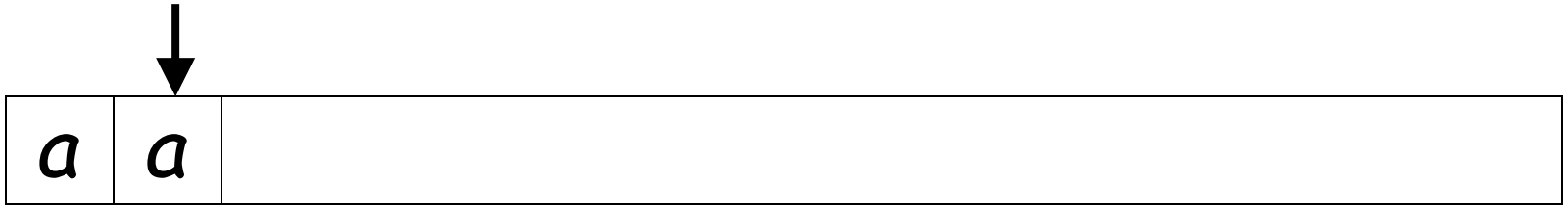


First Choice

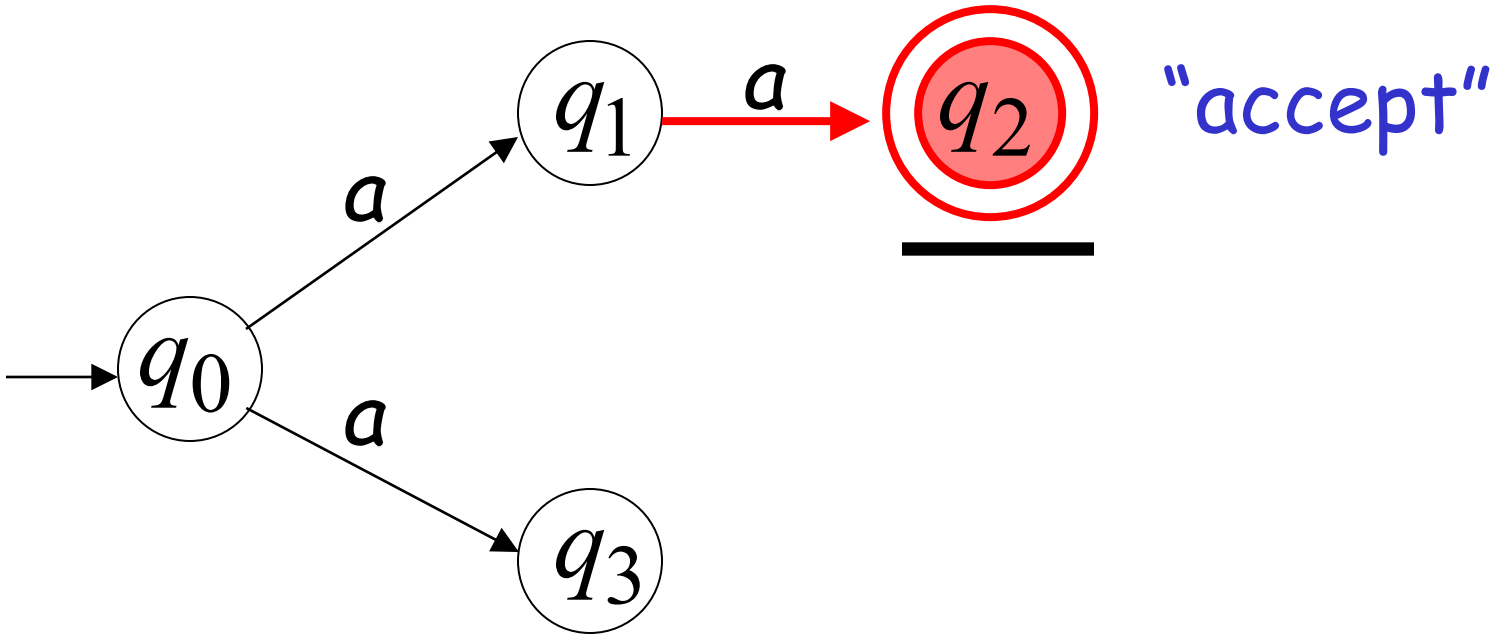




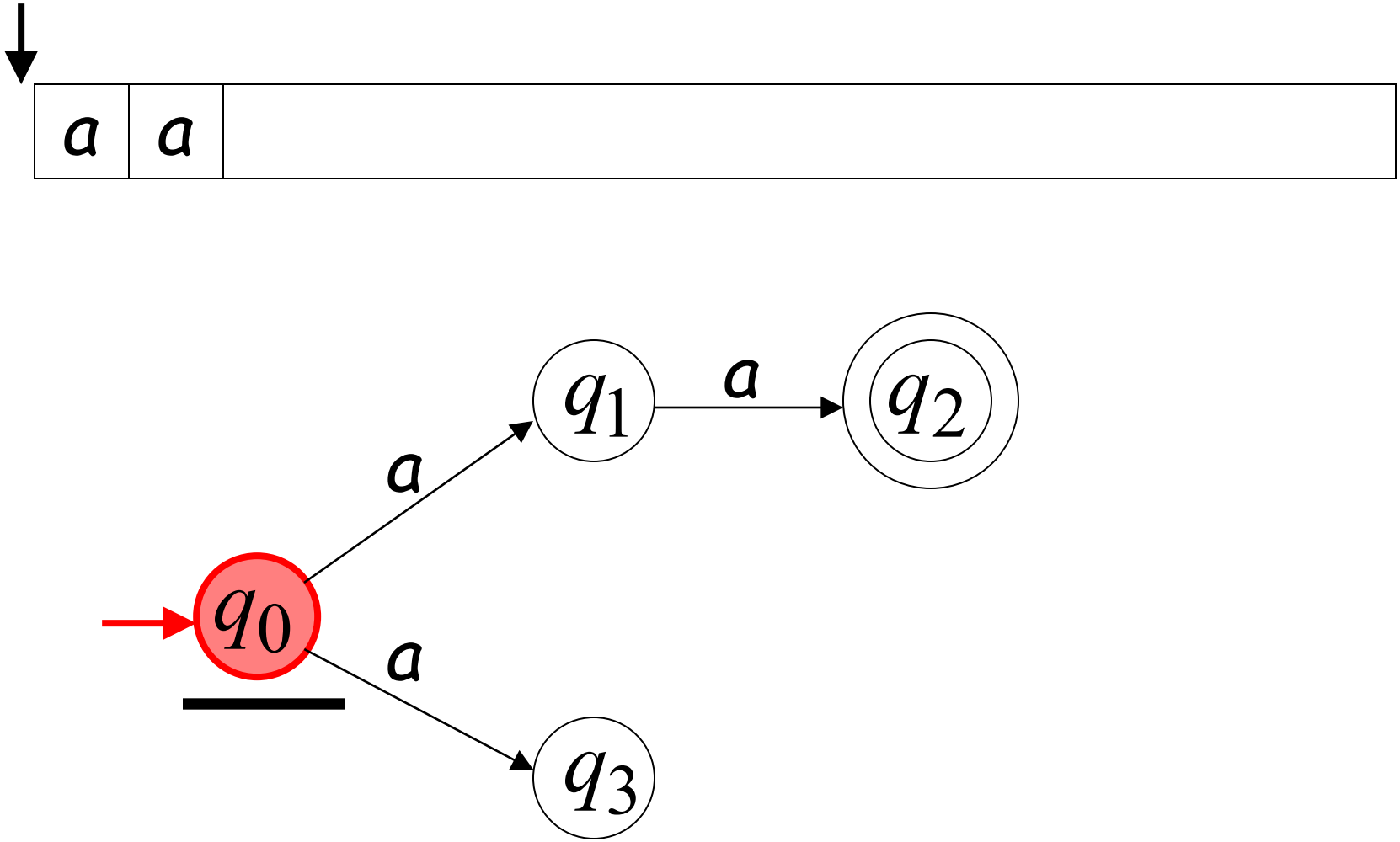
First Choice



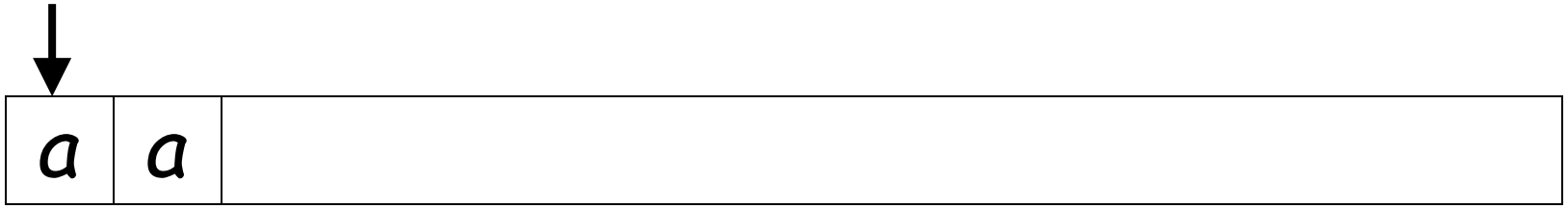
All input is consumed



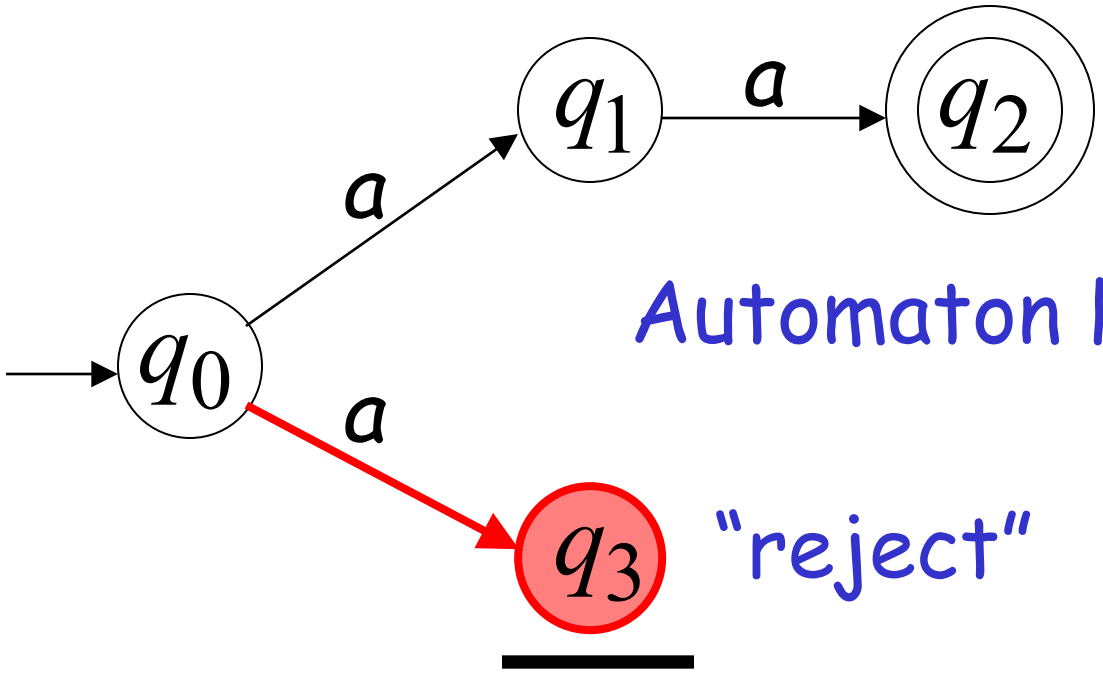
Second Choice



Second Choice



Input cannot be consumed



Automaton Halts

"reject"

An NFA accepts a string:

if there is **at least one** computation of the NFA that accepts the string

i.e., all the input string is processed and the automaton is in an accepting state

An NFA rejects a string:

if there is **no** computation of the NFA that accepts the string.

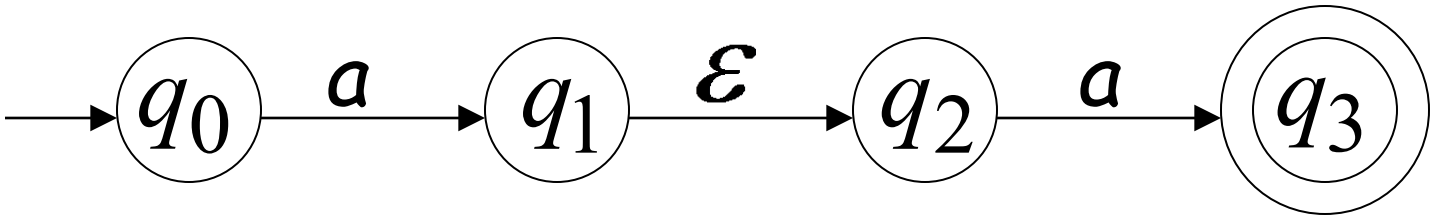
For each computation:

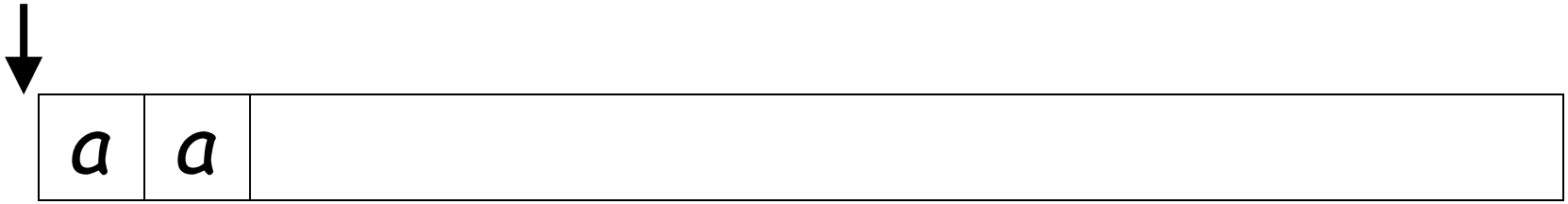
- All the input is consumed and the automaton is in a non final state

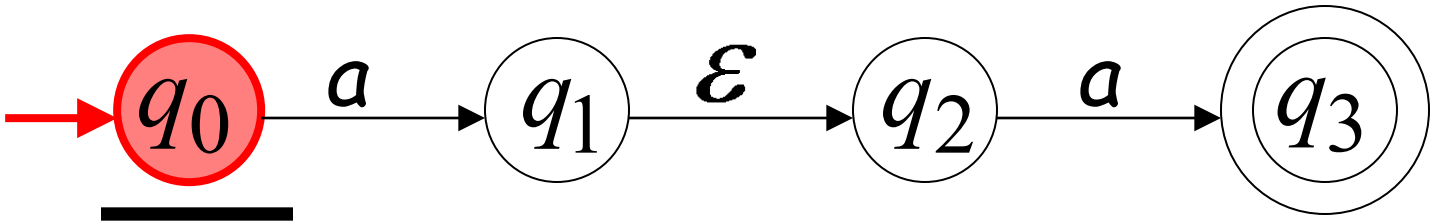
OR

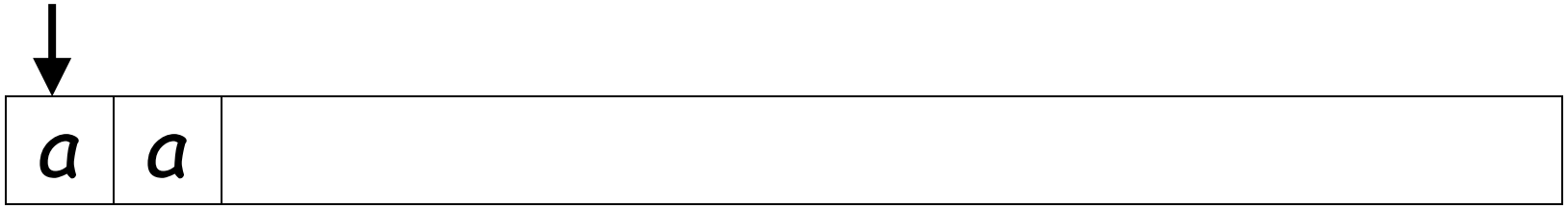
- The input cannot be consumed

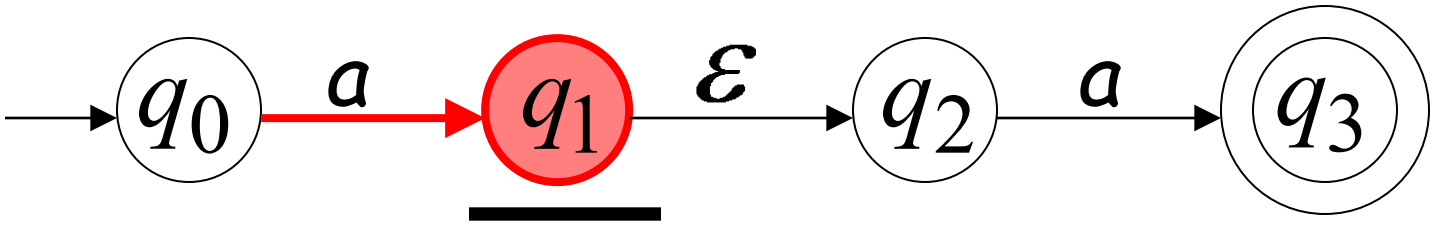
Epsilon (Lambda) Transitions



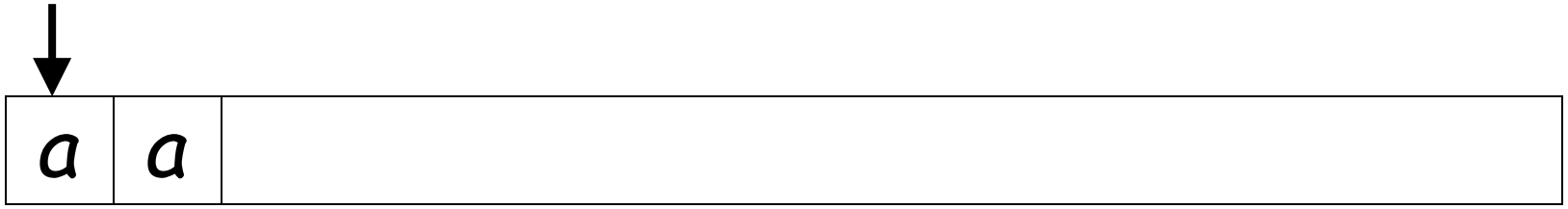

a
a

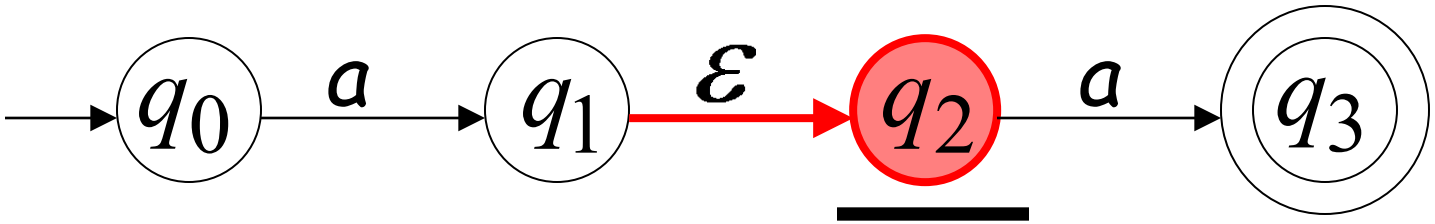

q_0
a
q_1
ε
q_2
a
q_3


a
a

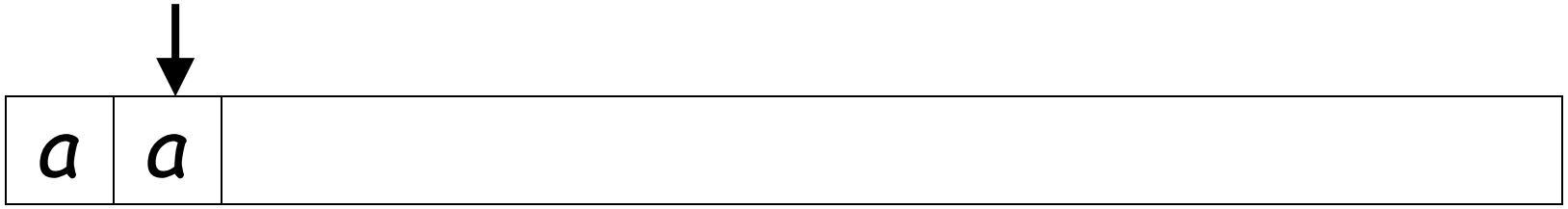

q_0
a
q_1
ε
q_2
a
q_3

input tape head does not move

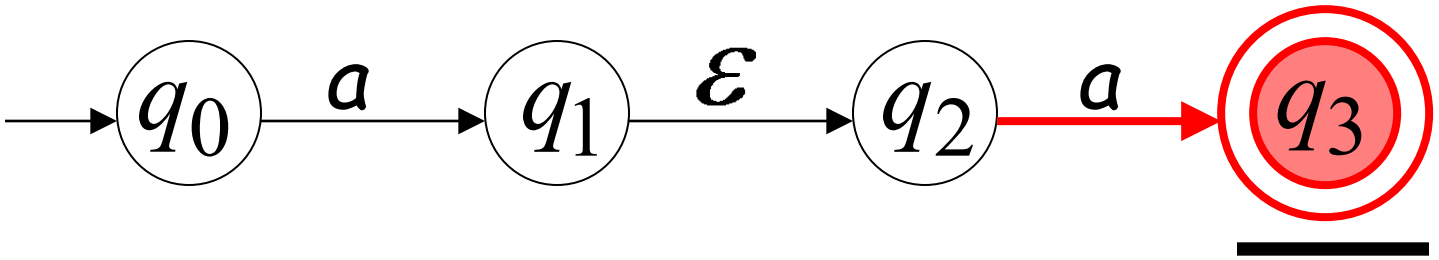




all input is consumed



"accept"

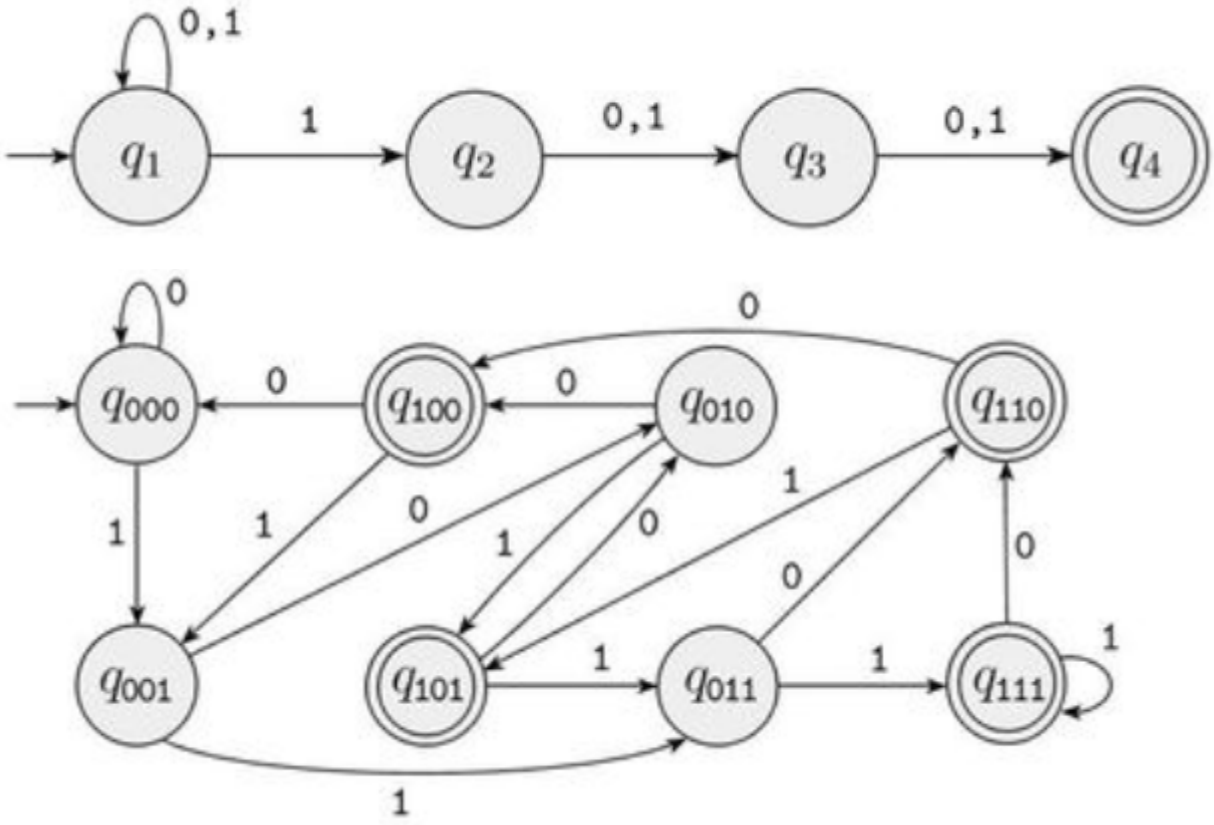


String aa is accepted

NFAs are interesting because they can be more concise than DFAs

NFA M_1

DFA M_2



Language: 1 in the third position from the end

Summary: DFA vs NFA

- DFA: every state always has exactly one exiting transition arrow for each symbol of the alphabet
  - NFA: states can have 0, 1 or *n* exiting arrows
  - NFA: can have empty transitions
  - NFA: from a given state, if a given symbol x is read, exiting transitions to more than one state are possible

Non-determinism: behaviour is "not determined"

DFA or NFA?

Each has benefits:

NFA can be more concise (exponentially so).

It is easier to define some operations on DFA (eg complement, intersection).

Remember – if we have a DFA (NFA) we can convert this into an equivalent NFA (DFA)

Regular Expressions

Recursive Definition

Primitive regular expressions: $\emptyset, \quad \varepsilon, \quad \alpha$

Given regular expressions r_1 and r_2

$r_1 + r_2$ — also written as → $r_1 \cup r_2$

$r_1 \cdot r_2$

r_1^*

(r_1)

Are regular expressions

Language defined

Primitive regular expressions:

$$L(\varnothing) = \varnothing$$

$$L(\varepsilon) = \{\varepsilon\}$$

$$L(a) = \{a\}$$

Language defined (continued)

For regular expressions r_1 and r_2

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$

$$L(r_1 *) = (L(r_1))*$$

$$L((r_1)) = L(r_1)$$

Regular Languages

Regular Languages

A **language** is a set of strings/words

A language L is a **regular language** if there exists a DFA M that accepts L

Equivalent definitions

We have seen that every NFA can be converted into an equivalent DFA.

Thus: a language L is a **regular language** ifand only if there exists an NFA M that accepts L

In addition: a language L is a **regular language** if and only if there exists a regular expression r that accepts L