Mathematical Preliminaries

Mathematical Preliminaries

- Sets
- Functions
- Relations
- Graphs
- Proof Techniques

SETS

A set is a collection of elements

$$A = \{1, 2, 3\}$$

$$B = \{train, bus, bicycle, airplane\}$$

We write

$$1 \in A$$

$$ship \notin B$$

Set Representations

$C = \{ a, b, c, d, e, f, g, h, i, j, k \}$

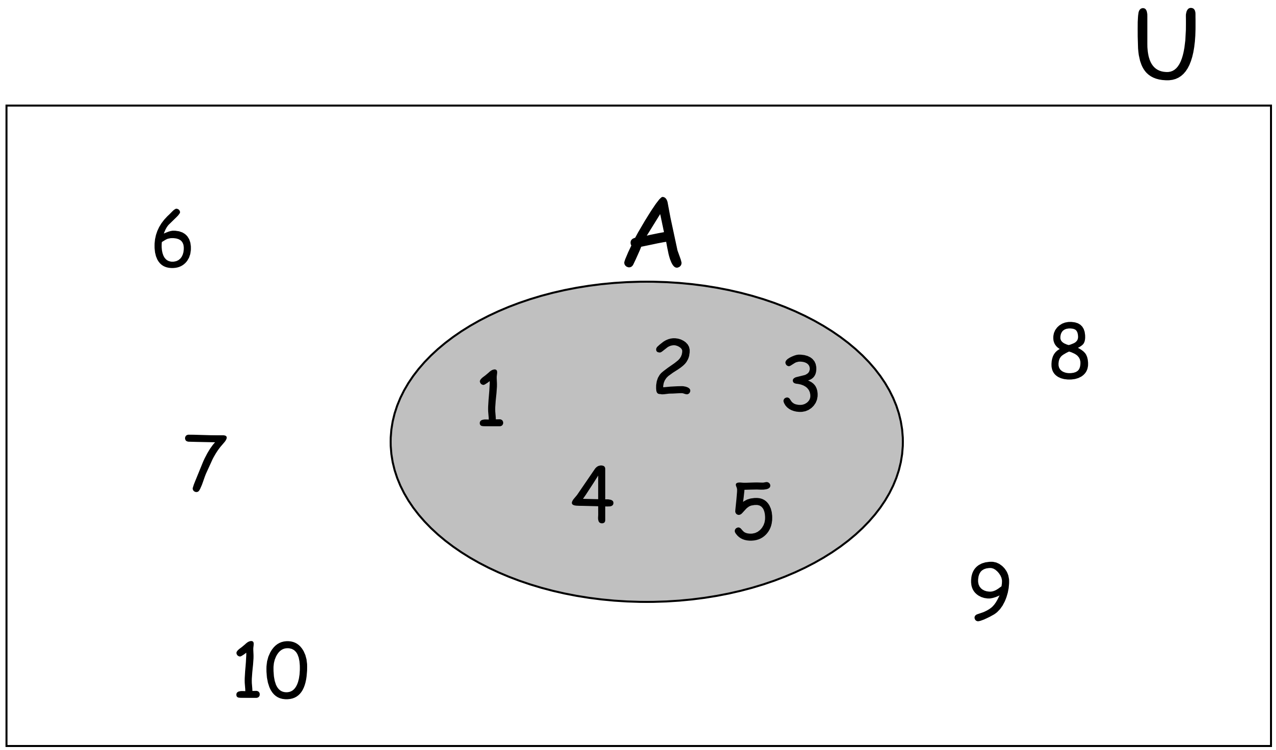
$C = \{ a, b, \ldots, k \}$ → *finite set*

$S = \{ 2, 4, 6, \ldots \}$ → *infinite set*

$S = \{ j : j > 0, \text{ and } j = 2k \text{ for some } k>0 \}$

$S = \{ j : j \text{ is nonnegative and even} \}$

$A = \{ 1, 2, 3, 4, 5 \}$



Universal Set: all possible elements

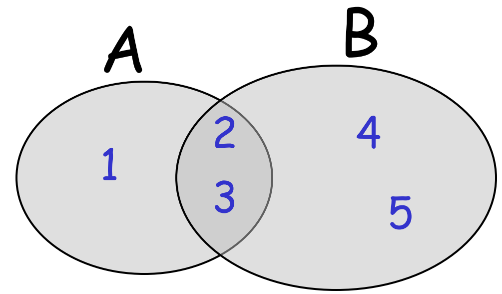
$U = \{ 1 , \ldots , 10 \}$

Set Operations

$A = \{1, 2, 3\}$ $B = \{2, 3, 4, 5\}$

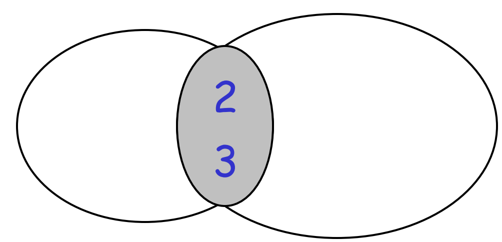
- Union

$A \cup B = \{1, 2, 3, 4, 5\}$



- Intersection

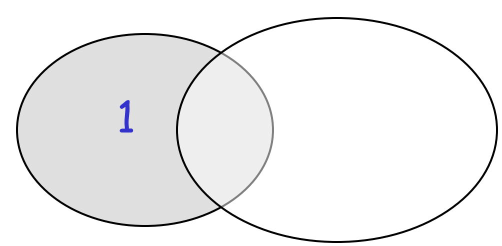
$A \cap B = \{2, 3\}$



- Difference

$A - B = \{1\}$

$B - A = \{4, 5\}$

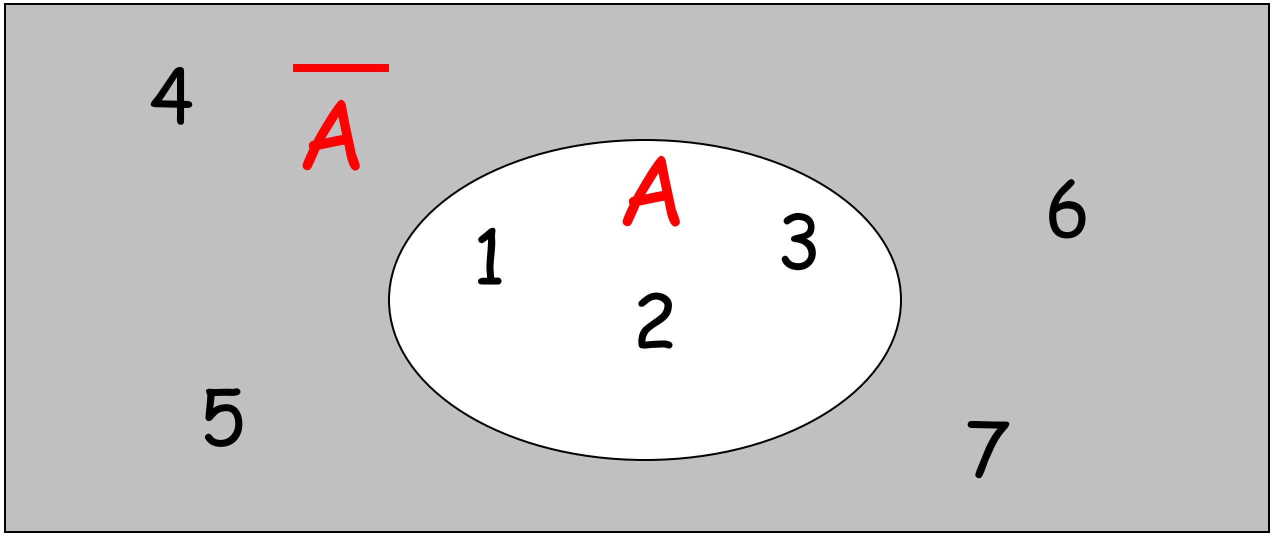


Venn diagrams

- Complement

Universal set = {1, ..., 7}

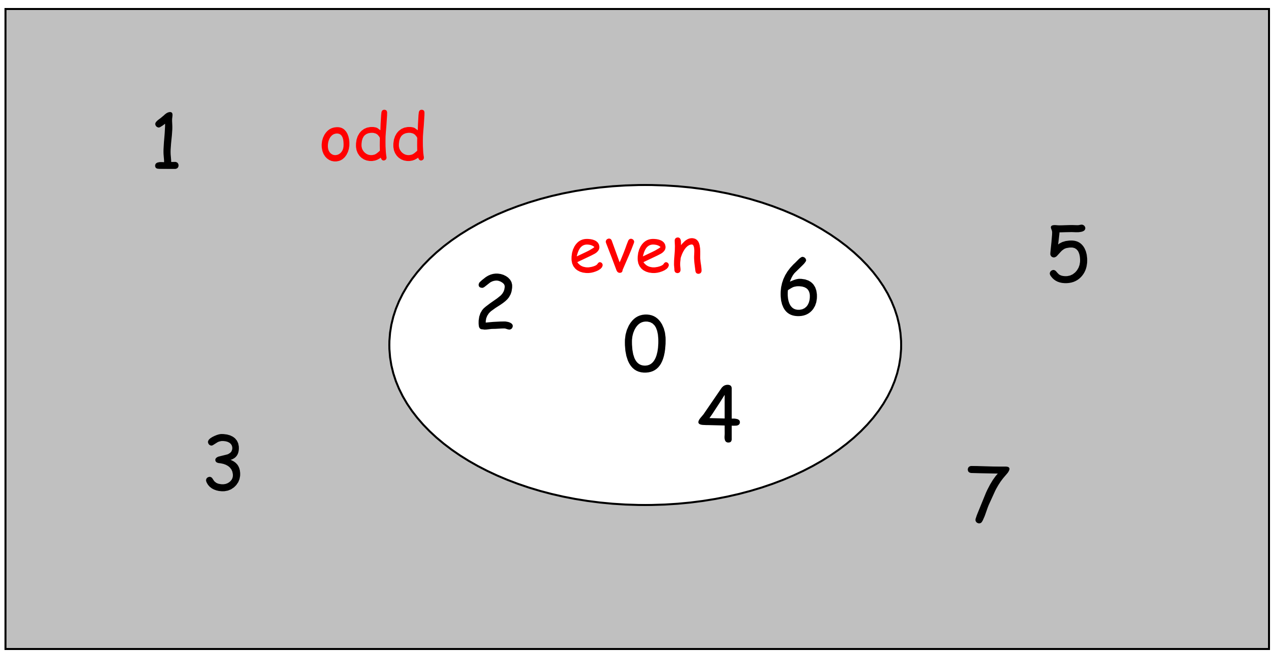
$A = \{1, 2, 3\} \Rightarrow \overline{A} = \{4, 5, 6, 7\}$



$\overline{\overline{A}} = A$

$$\overline{\{\text{ even integers }\}} = \{\text{ odd integers }\}$$

Integers



DeMorgan's Laws

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

Empty, Null Set: $\emptyset$

$\emptyset = \{ \}$

$S \cup \emptyset = S$

$S \cap \emptyset = \emptyset$

$S - \emptyset = S$

$\emptyset - S = \emptyset$

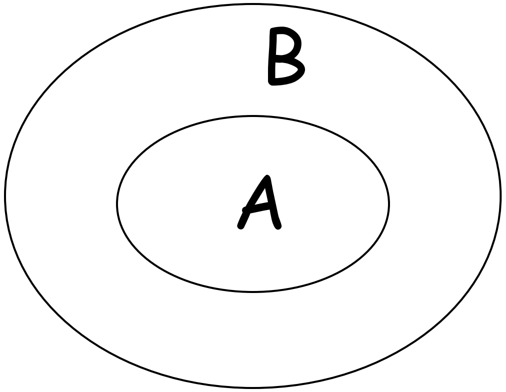
$\overline{\emptyset}$ = Universal Set

Subset

$A = \{1, 2, 3\}$ $B = \{1, 2, 3, 4, 5\}$

$$A \subseteq B$$

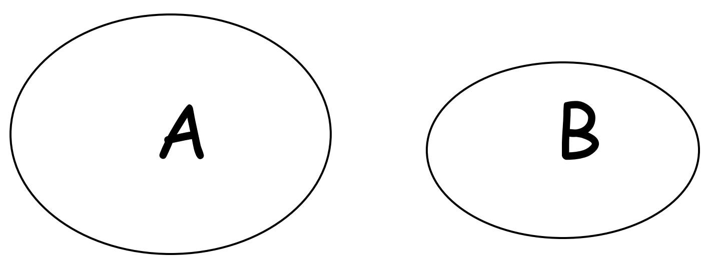
Proper Subset: $A \subset B$



Disjoint Sets

$A = \{ 1, 2, 3 \}$ $B = \{ 5, 6\}$

$$A \cap B = \emptyset$$



Set Cardinality

- For finite sets

$A = \{ 2, 5, 7 \}$

$|A| = 3$

(set size)

Powersets

A powerset is a set of sets

$S = \{ a, b, c \}$

Powerset of S = the set of all the subsets of S

$2^S = \{ \emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\} \}$

Observation: $| 2^S | = 2^{|S|}$ $\quad (8 = 2^3)$

Cartesian Product

$A = \{ 2, 4 \}$ $B = \{ 2, 3, 5 \}$

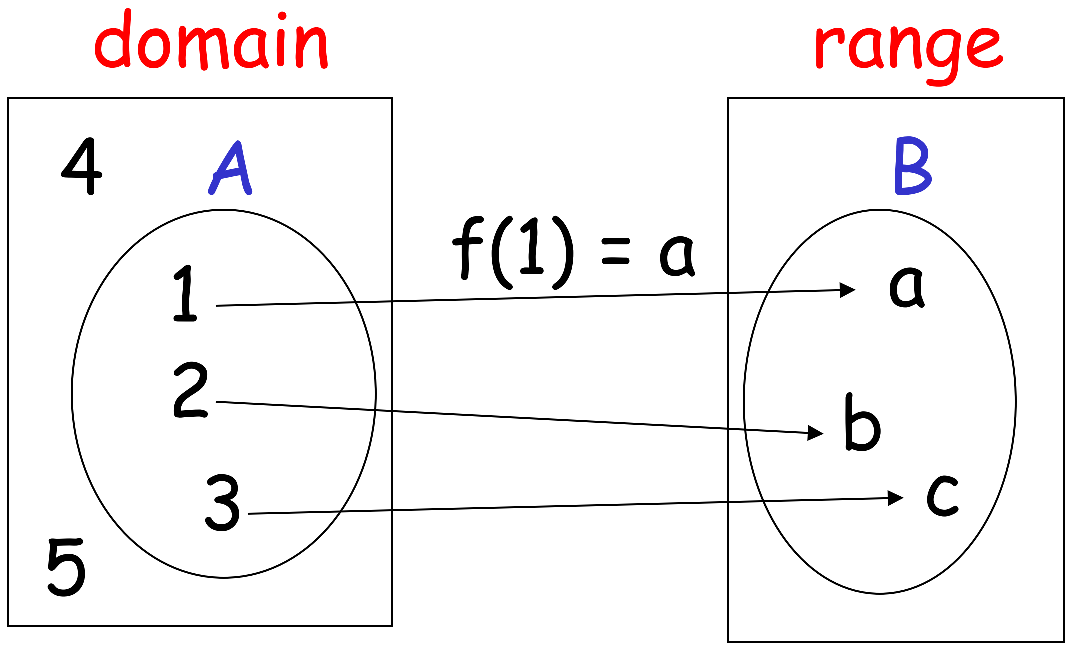
$A \times B = \{ (2, 2), (2, 3), (2, 5), (4, 2), (4, 3), (4, 5) \}$

$|A \times B| = |A|\ |B|$

Generalizes to more than two sets

$A \times B \times \ldots \times Z$

FUNCTIONS



f : A -> B

If A = domain

then f is a total function

otherwise f is a partial function

RELATIONS

$R = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\}$

$x_i \; R \; y_i$

e. g. if R = '>': $2 > 1$, $3 > 2$, $3 > 1$

Equivalence Relations

- Reflexive: x R x
- Symmetric: x R y $\Rightarrow$ y R x
- Transitive: x R y and y R z $\Rightarrow$ x R z

Example: R = '='

- x = x
- x = y $\Rightarrow$ y = x
- x = y and y = z $\Rightarrow$ x = z

Equivalence Classes

For equivalence relation R

equivalence class of $x = \{y : x \, R \, y\}$

Example:

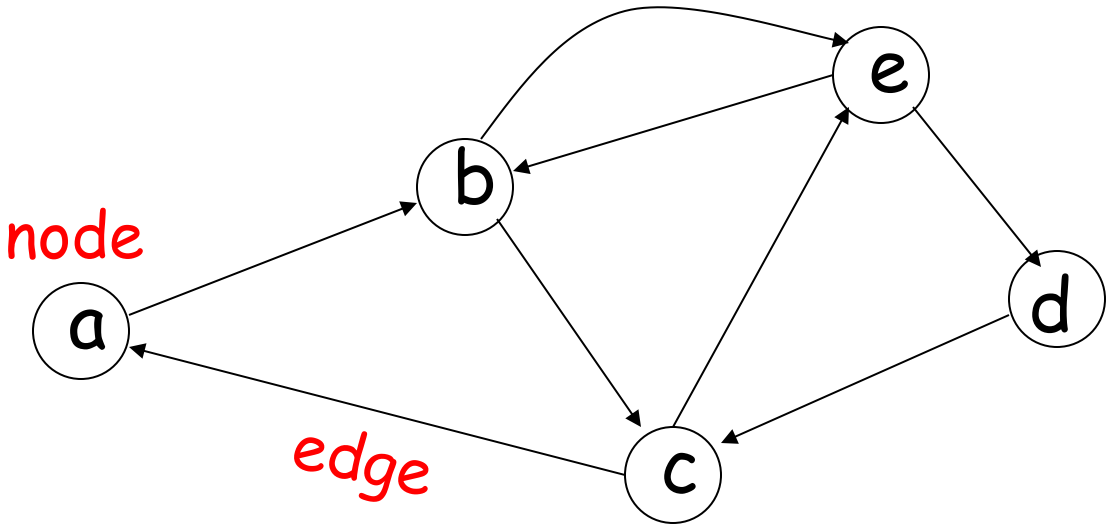
$$R = \{ (1, 1), (2, 2), (1, 2), (2, 1), (3, 3), (4, 4), (3, 4), (4, 3) \}$$

Equivalence class of $1 = \{1, 2\}$

Equivalence class of $3 = \{3, 4\}$

GRAPHS

A directed graph



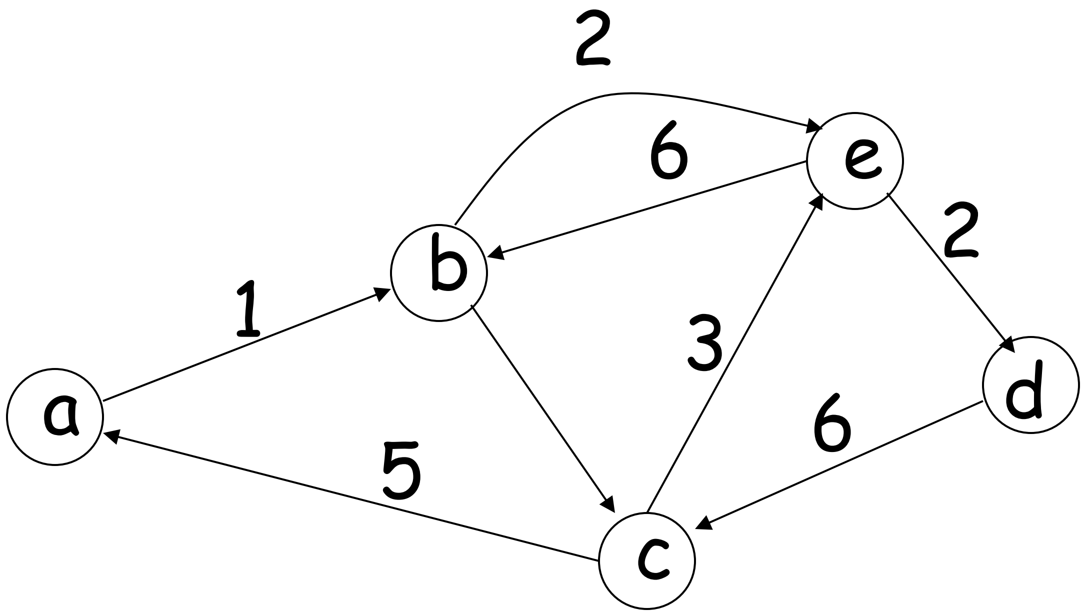
- Nodes (Vertices)

$$V = \{ a, b, c, d, e \}$$

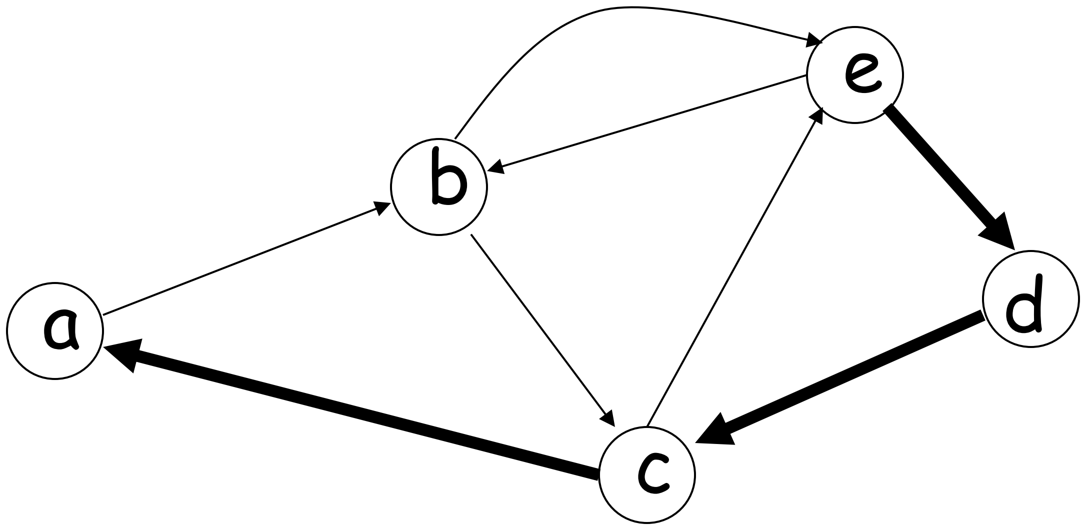
- Edges

$$E = \{ (a,b), (b,c), (b,e), (c,a), (c,e), (d,c), (e,b), (e,d) \}$$

Labeled Graph



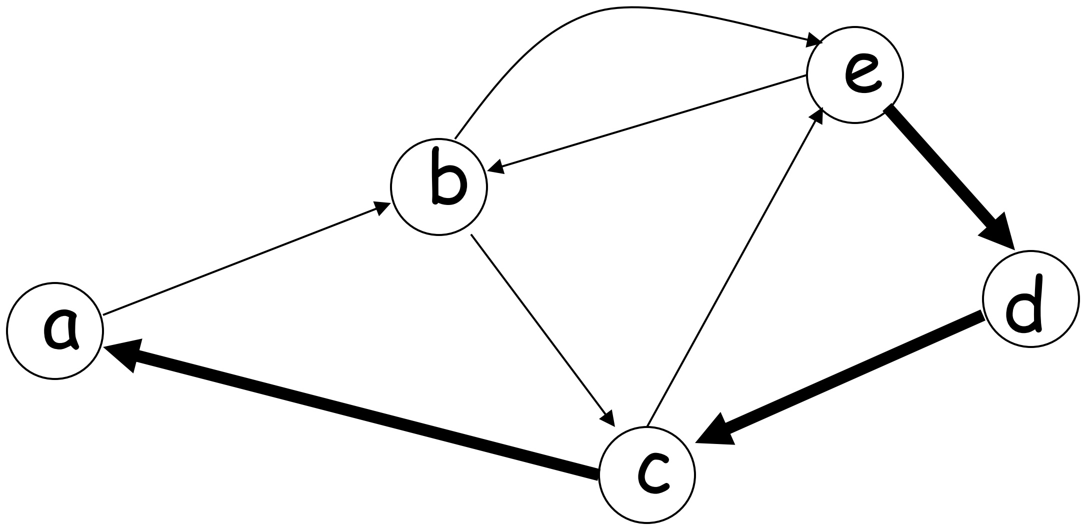
Walk



Walk is a sequence of adjacent edges

(e, d), (d, c), (c, a)

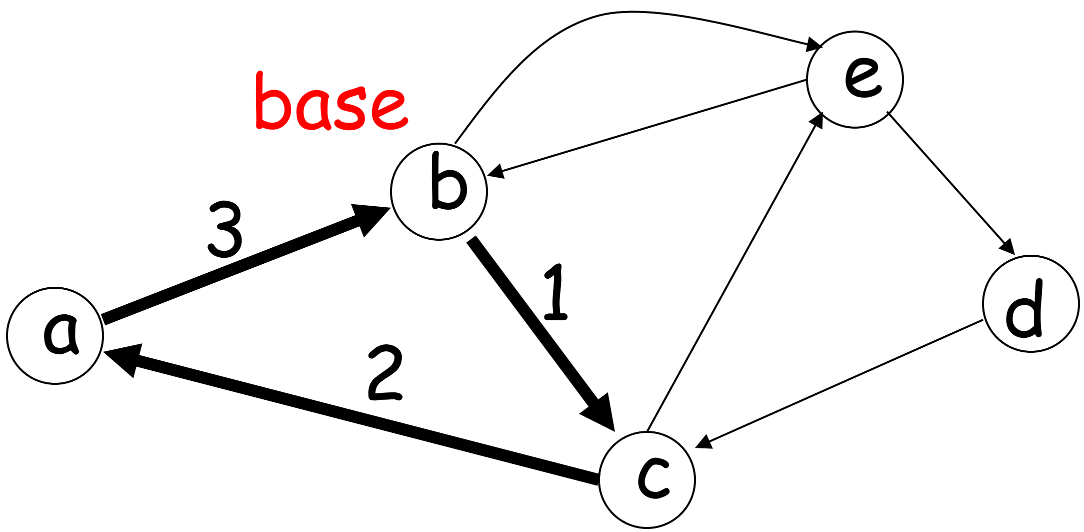
Path



Path is a walk where no edge is repeated

Simple path: no node is repeated

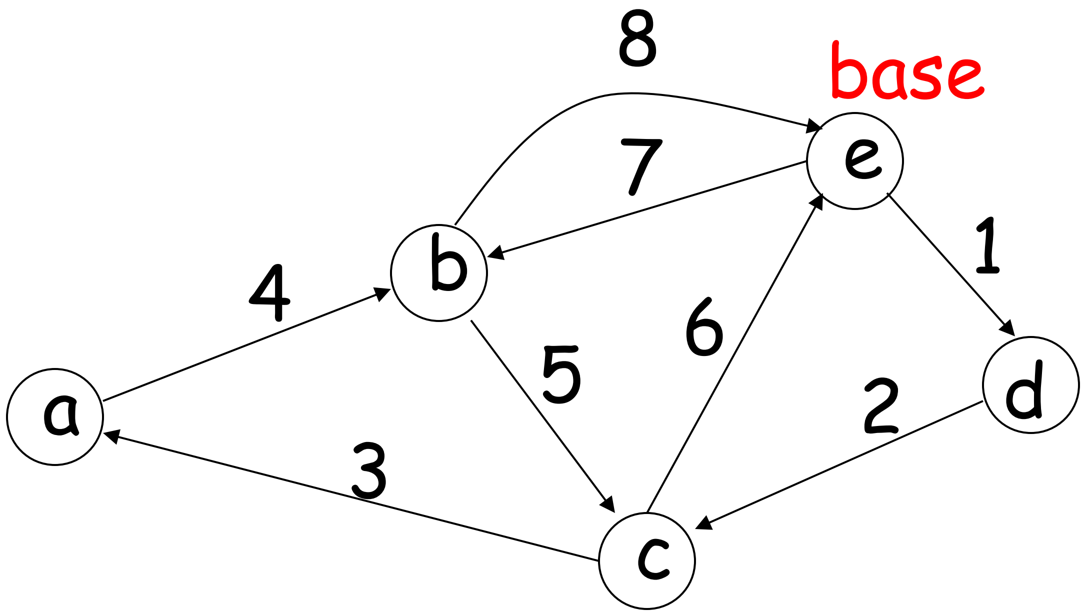
Cycle



Cycle: a walk from a node (base) to itself

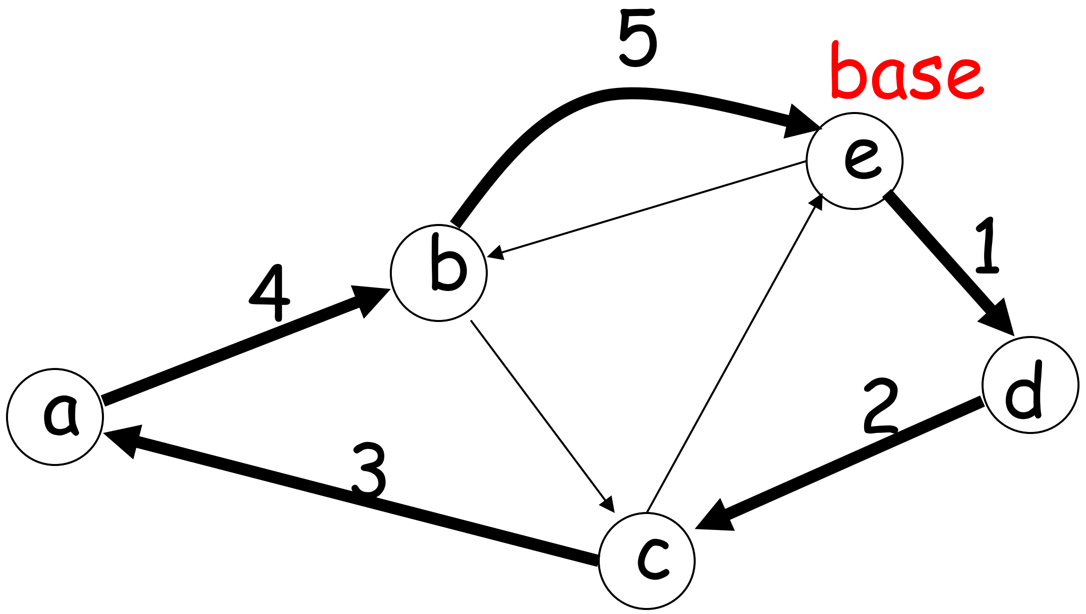
Simple cycle: only the base node is repeated

Euler Tour



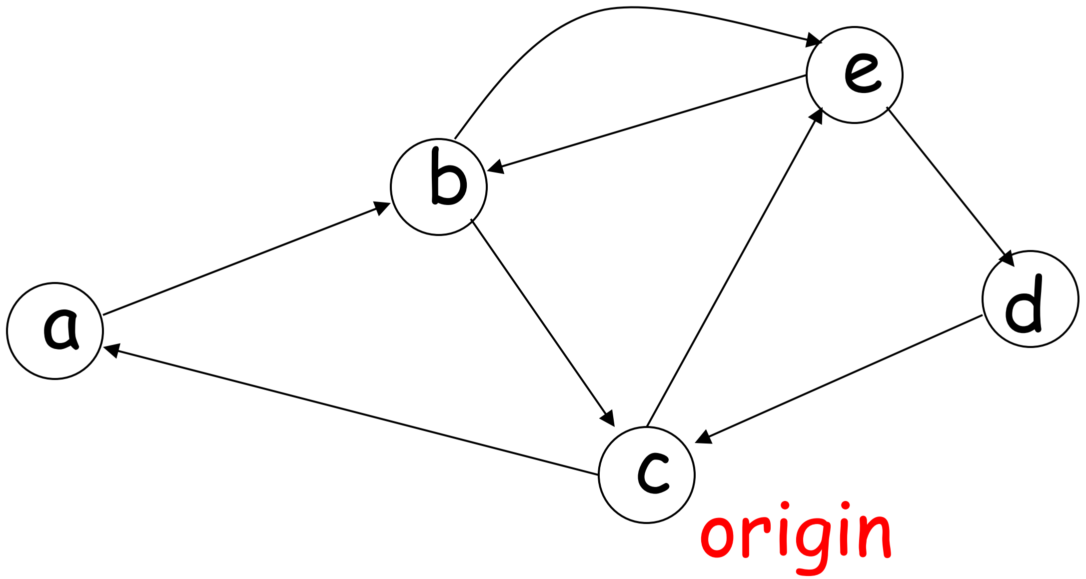
A cycle that contains each edge once

Hamiltonian Cycle

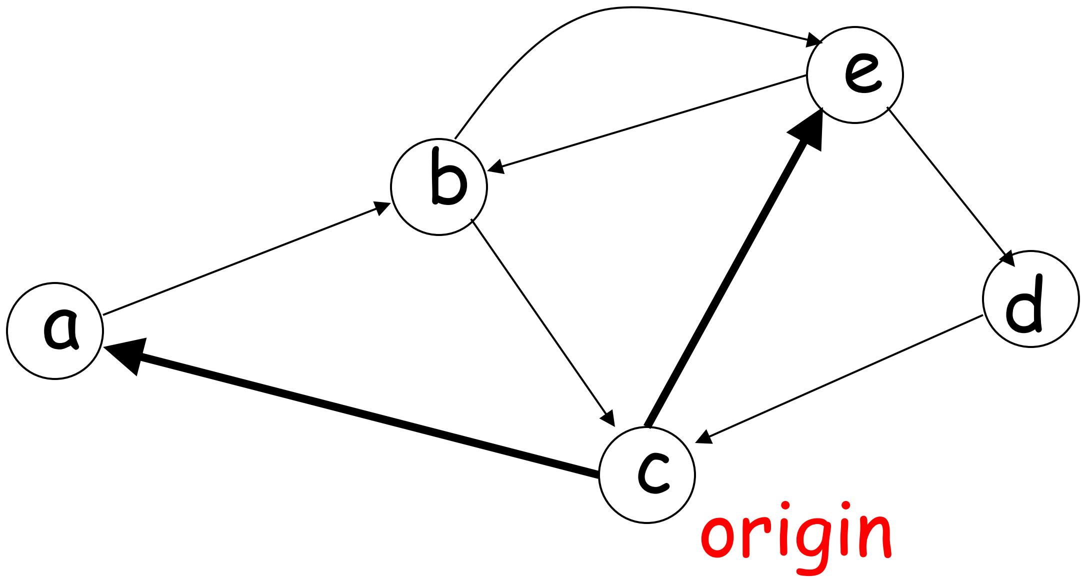


A simple cycle that contains all nodes

Finding All Simple Paths



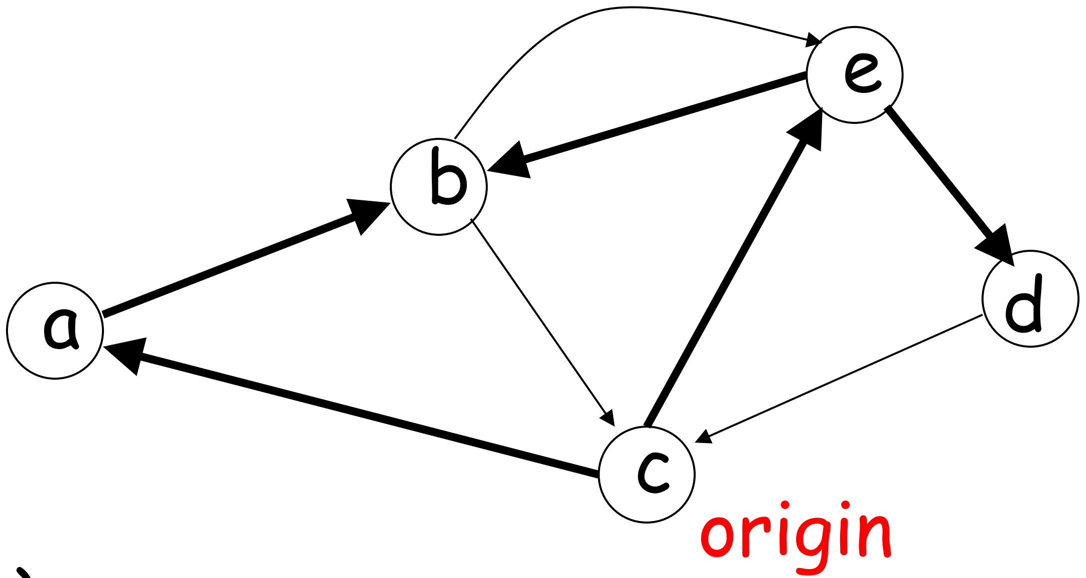
Step 1



(c, a)

(c, e)

Step 2



(c, a)

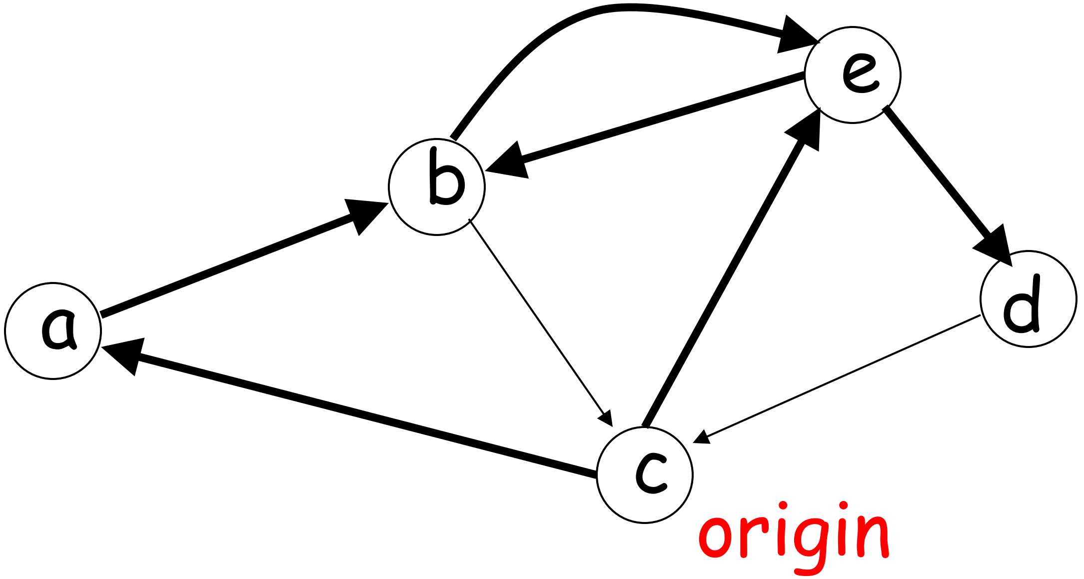
(c, a), (a, b)

(c, e)

(c, e), (e, b)

(c, e), (e, d)

Step 3



(c, a)

(c, a), (a, b)

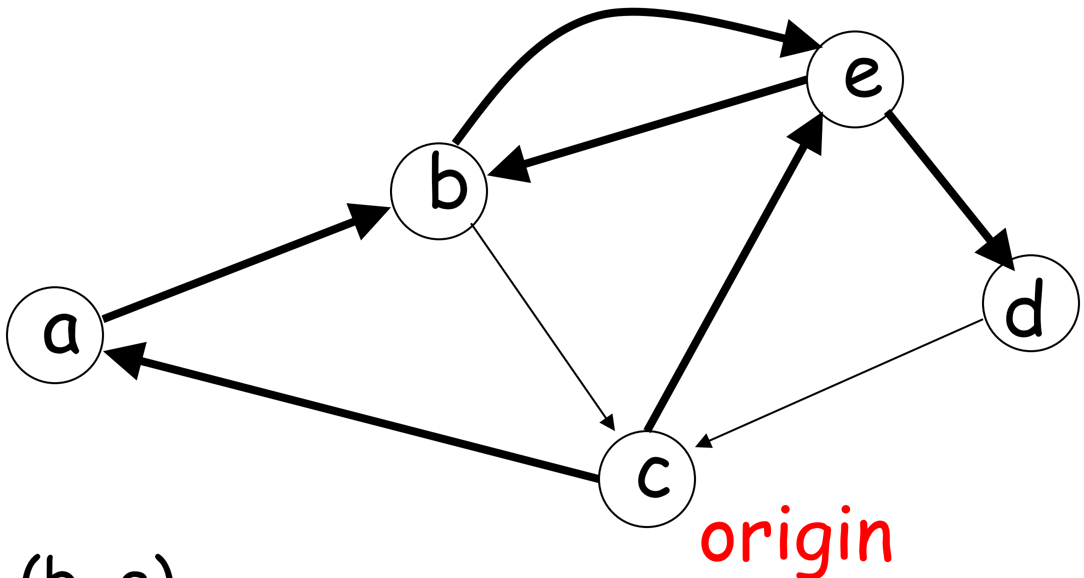
(c, a), (a, b), (b, e)

(c, e)

(c, e), (e, b)

(c, e), (e, d)

Step 4



(c, a)

(c, a), (a, b)

(c, a), (a, b), (b, e)

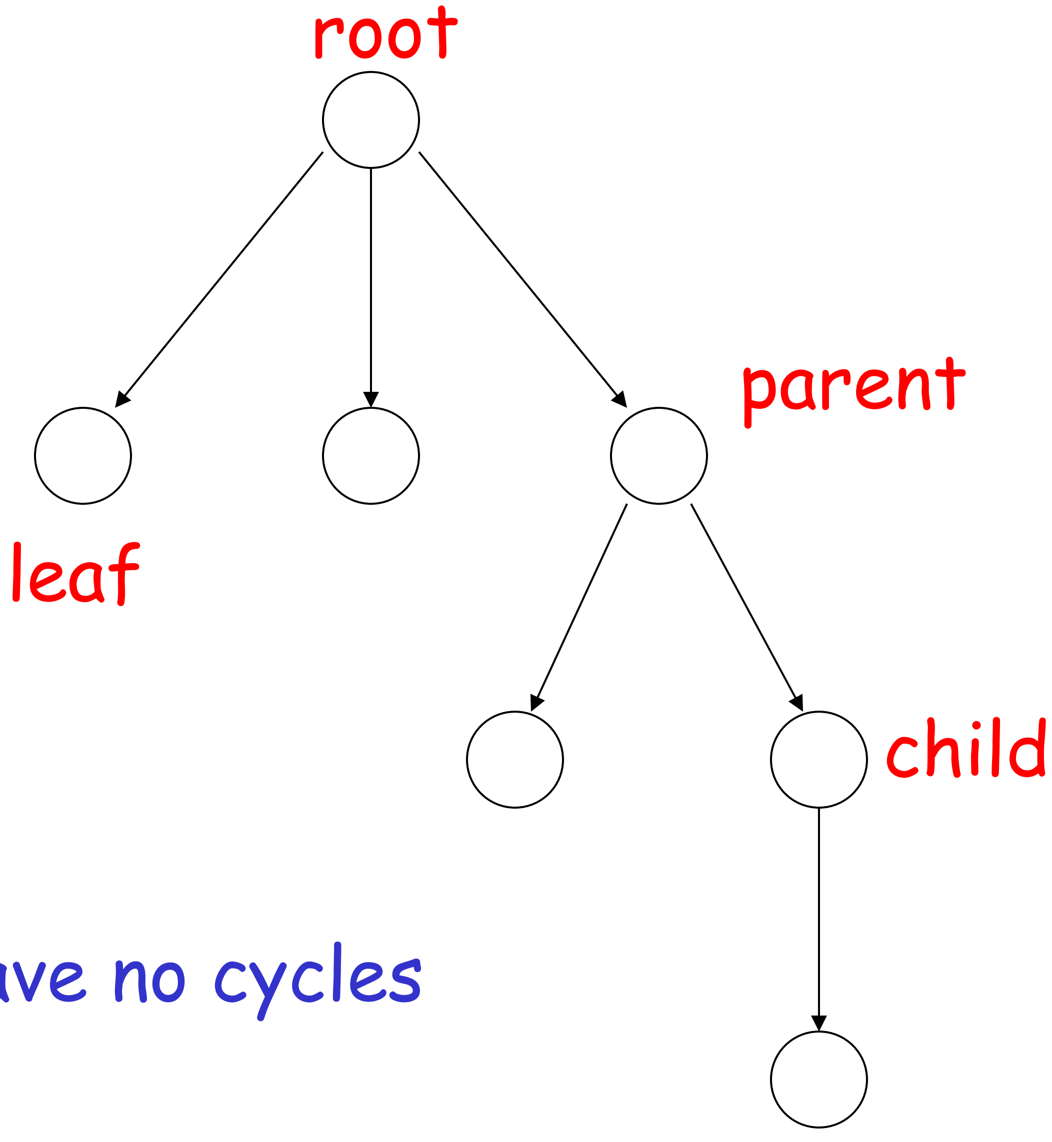
(c, a), (a, b), (b, e), (e,d)

(c, e)

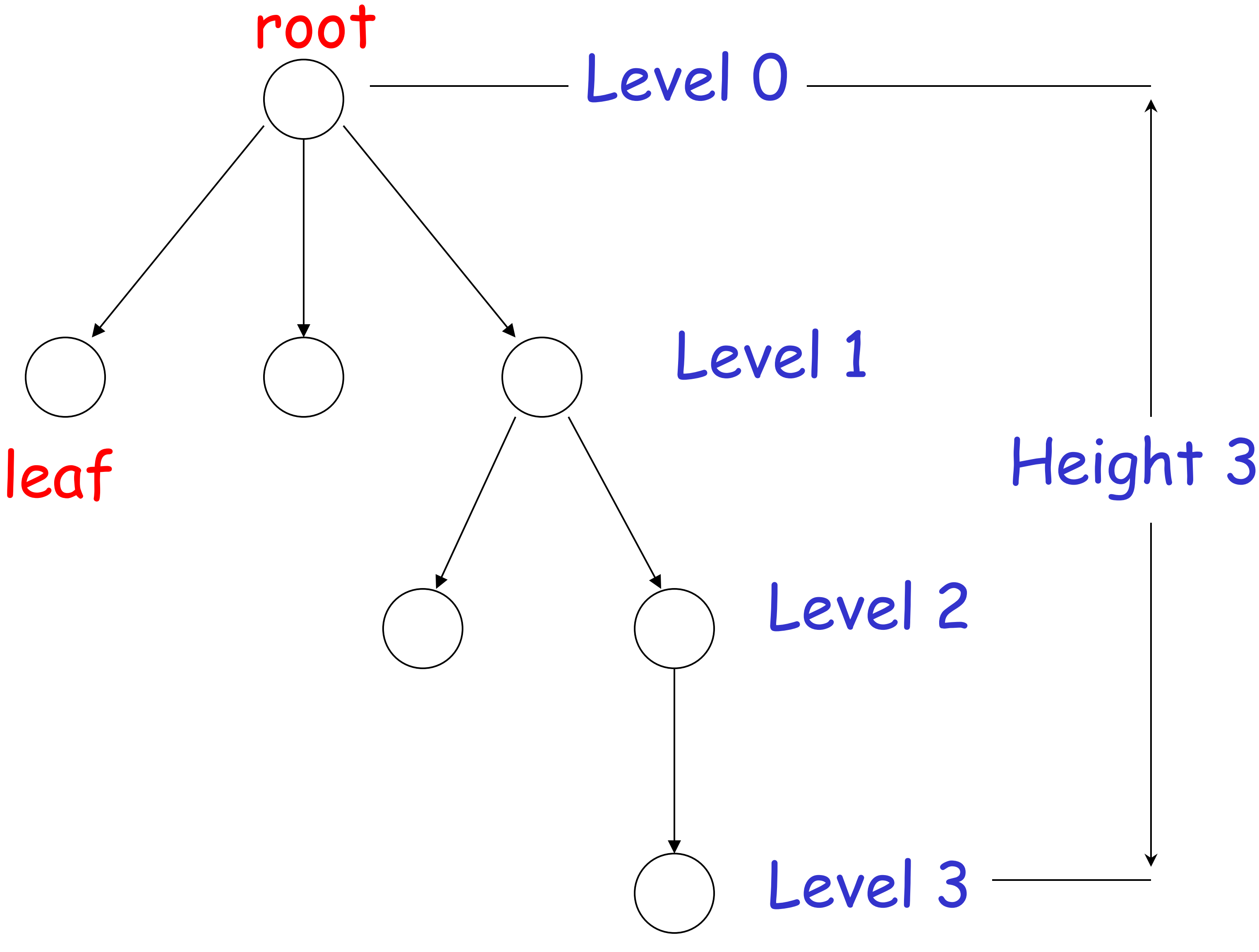
(c, e), (e, b)

(c, e), (e, d)

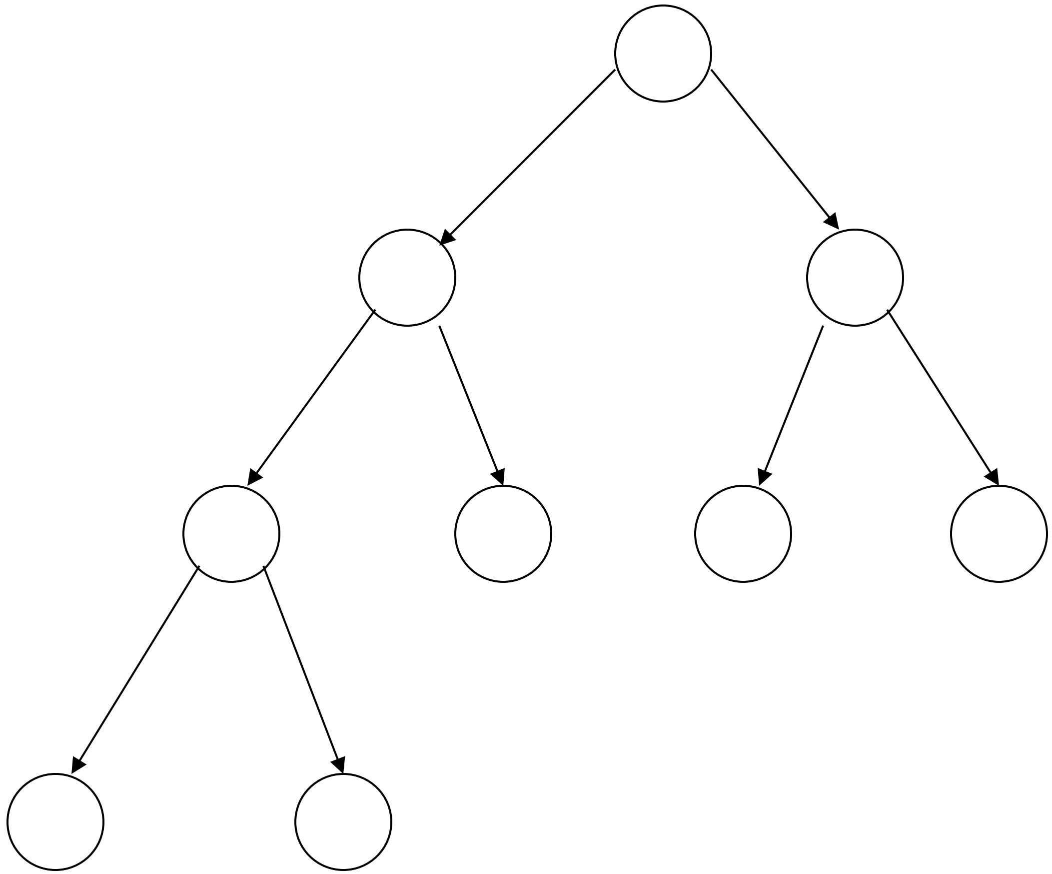
Trees



Trees have no cycles


root
Level 0
Level 1
leaf
Height 3
Level 2
Level 3

Binary Trees



PROOF TECHNIQUES

- Proof by induction

- Proof by contradiction

Induction

We have statements $P_1, P_2, P_3, \ldots$

If we know

- for some b that $P_1, P_2, \ldots, P_b$ are true
- for any k >= b that

  $P_1, P_2, \ldots, P_k$ imply P_{k+1}

Then

Every P_i is true

Proof by Induction

- Inductive basis

  Find $P_1, P_2, \ldots, P_b$ which are true

- Inductive hypothesis

  Let's assume $P_1, P_2, \ldots, P_k$ are true,

  for any $k >= b$

- Inductive step

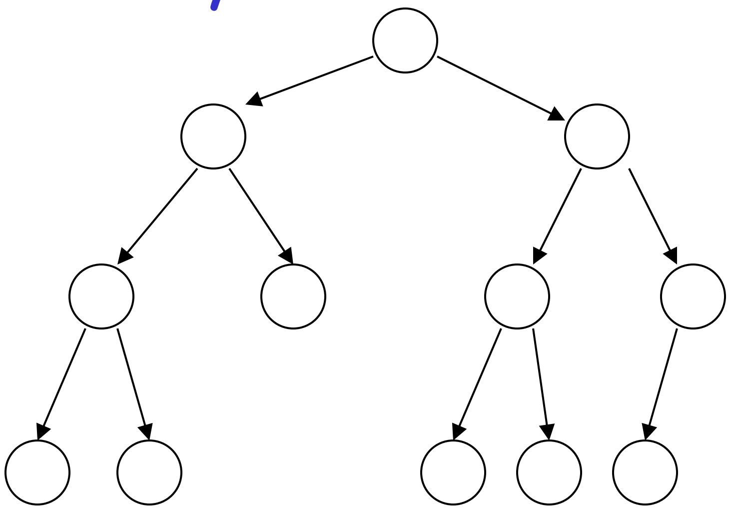
  Show that P_{k+1} is true

Example

Theorem: A binary tree of height n has at most 2^n leaves.

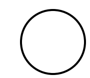
Proof by induction:

let $L(i)$ be the maximum number of leaves of any subtree at height i



We want to show: $L(i) <= 2^i$

- Inductive basis

  $L(0) = 1$ (the root node) 

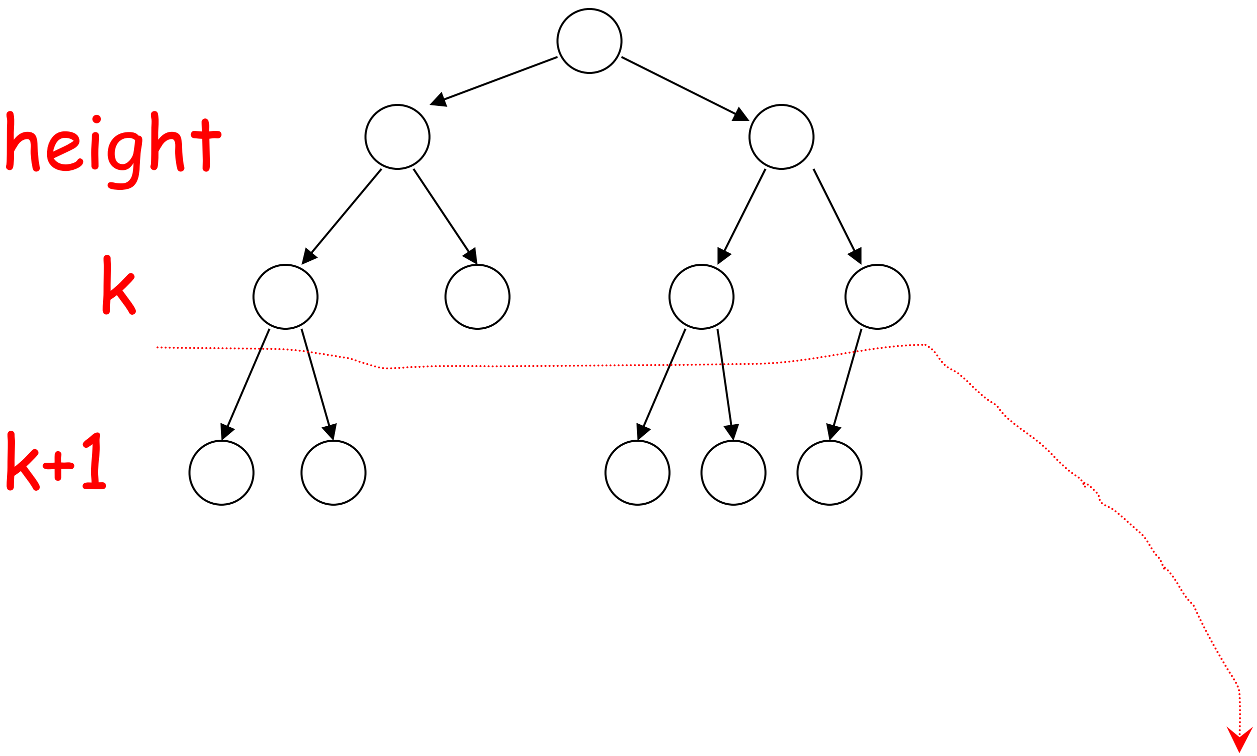
- Inductive hypothesis

  Let's assume $L(i) <= 2^i$ for all $i = 0, 1, \ldots, k$

- Induction step

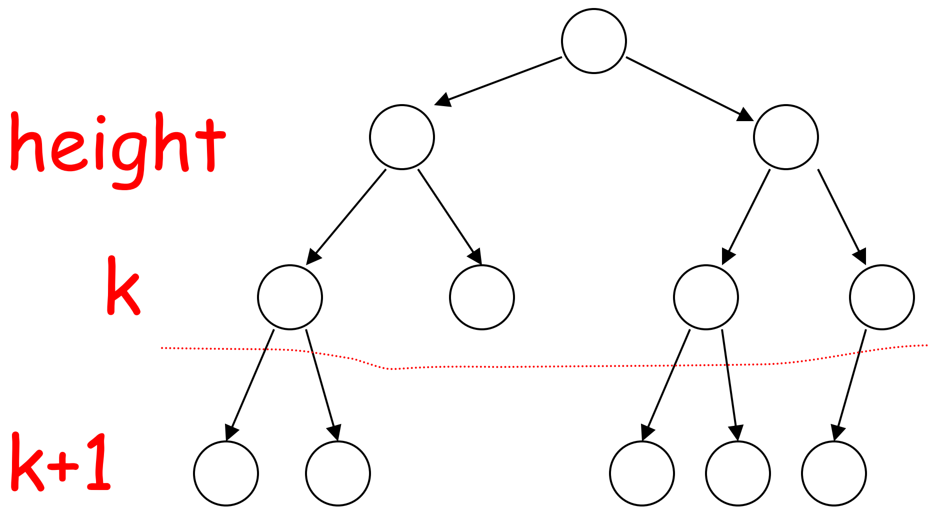
  we need to show that $L(k + 1) <= 2^{k+1}$

Induction Step



From Inductive hypothesis: $L(k) <= 2^k$

Induction Step



$L(k) <= 2^k$

$$L(k+1) <= 2 * L(k) <= 2 * 2^k = 2^{k+1}$$

(we add at most two nodes for every leaf of level k)

Remark

Recursion is another thing

Example of recursive function:

$$f(n) = f(n-1) + f(n-2)$$

$$f(0) = 1, \quad f(1) = 1$$

Proof by Contradiction

We want to prove that a statement P is true

- we assume that P is false
- then we arrive at an incorrect conclusion
- therefore, statement P must be true

Example

Theorem: $\sqrt{2}$ is not rational

Proof:

Assume by contradiction that it is rational

$\sqrt{2}$ = n/m

n and m have no common factors

We will show that this is impossible

$\sqrt{2} = n/m \Rightarrow 2m^2 = n^2$

Therefore, n^2 is even $\Rightarrow$ n is even

$n = 2k$

$2m^2 = 4k^2 \Rightarrow m^2 = 2k^2 \Rightarrow$ m is even

$m = 2p$

Thus, m and n have common factor 2

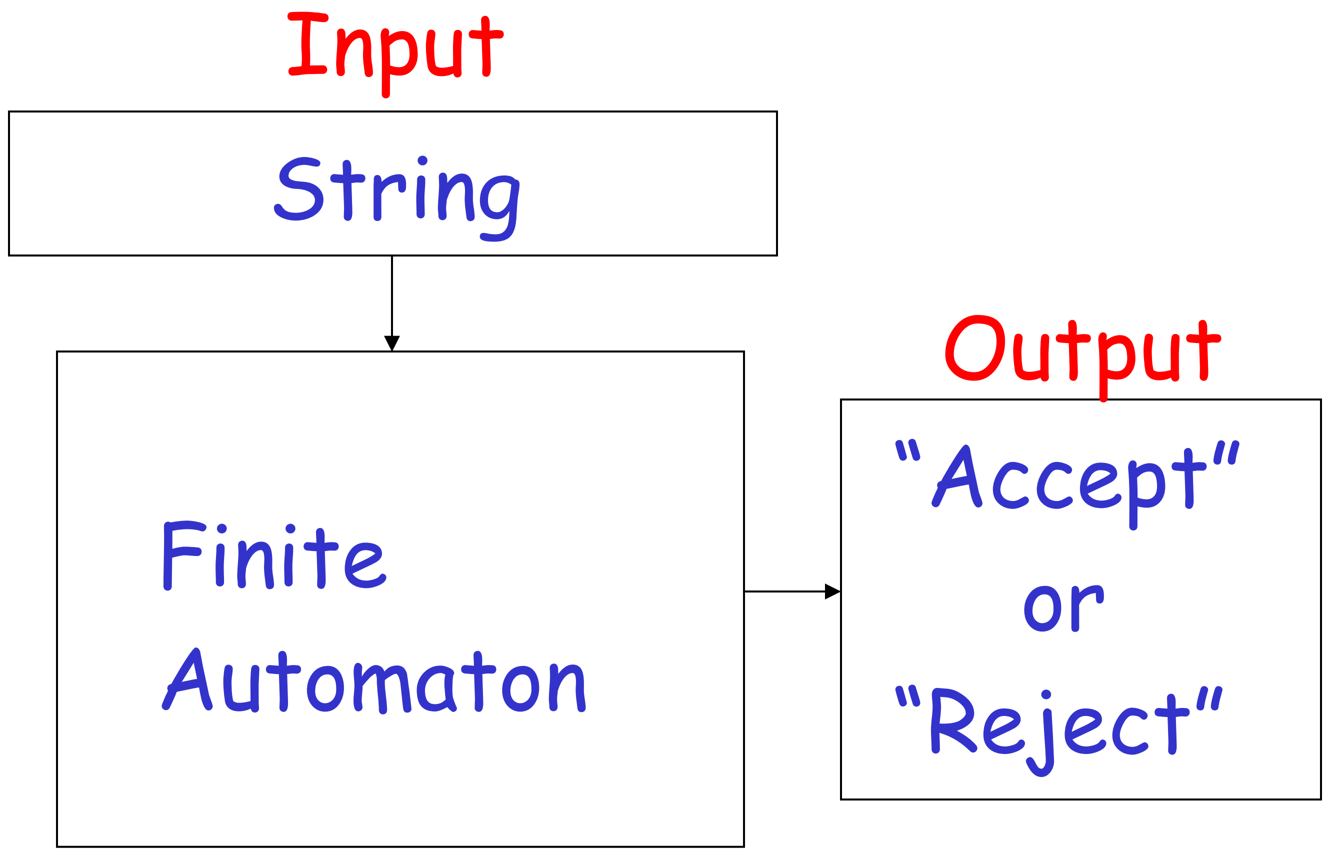
Contradiction!

14B11CI171

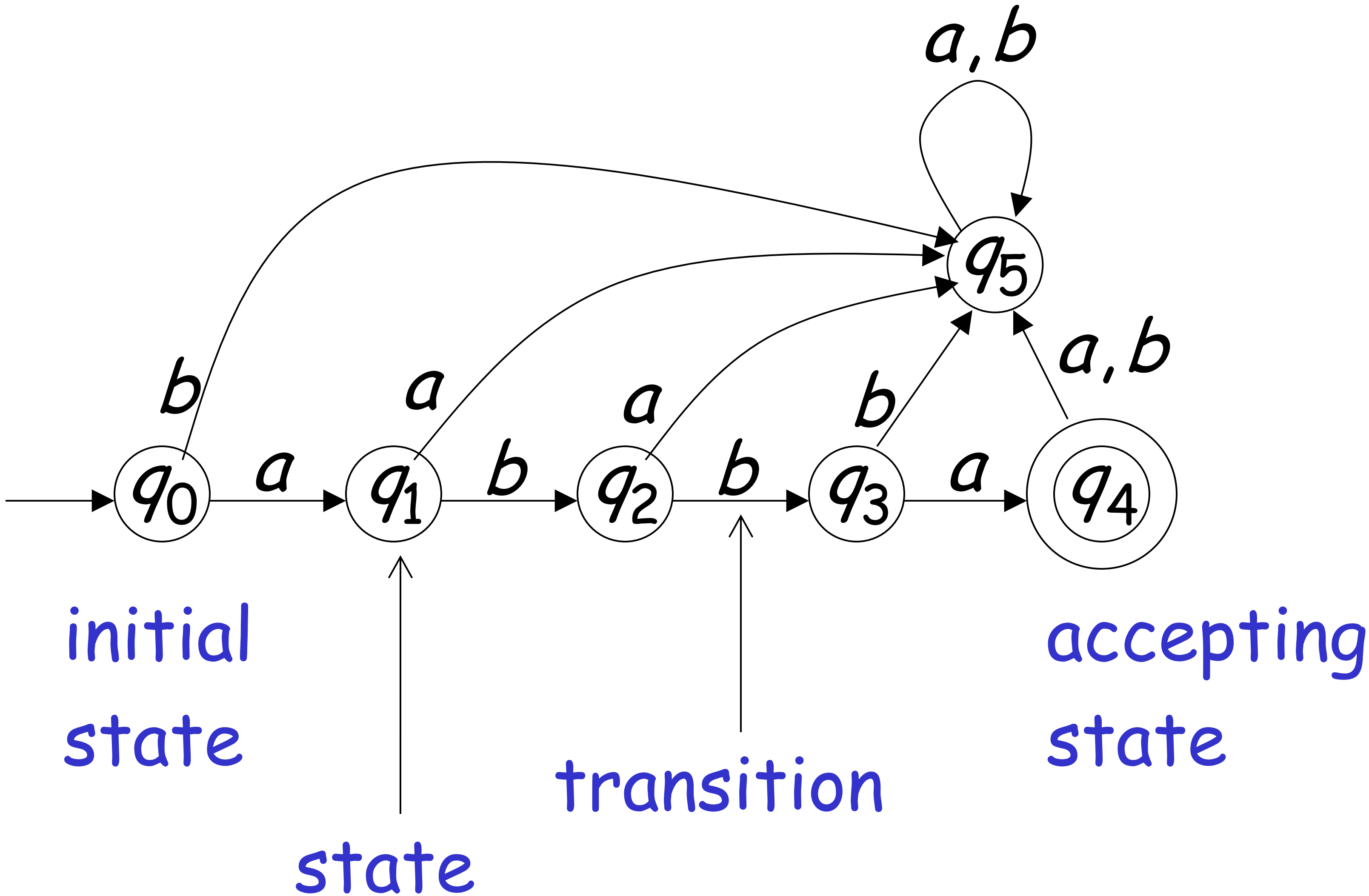
Theory of Computation

Finite Automata

Finite Automaton



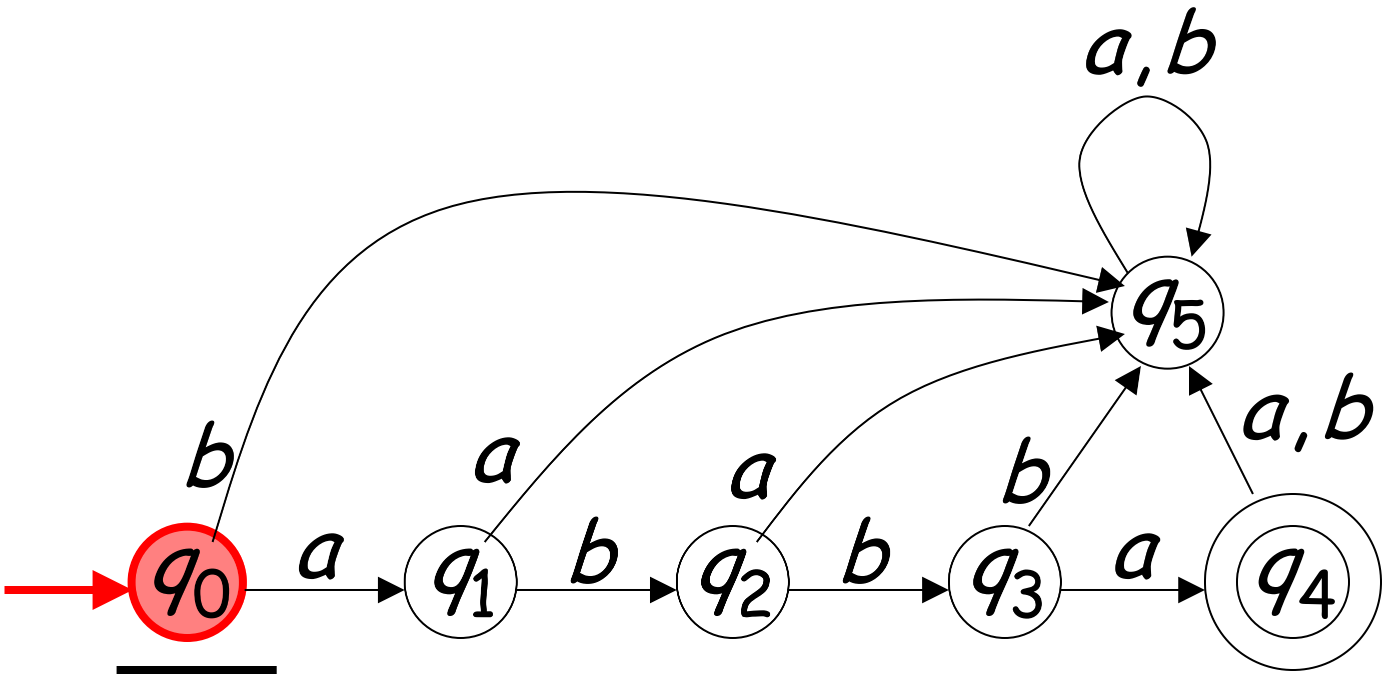
Transition Graph



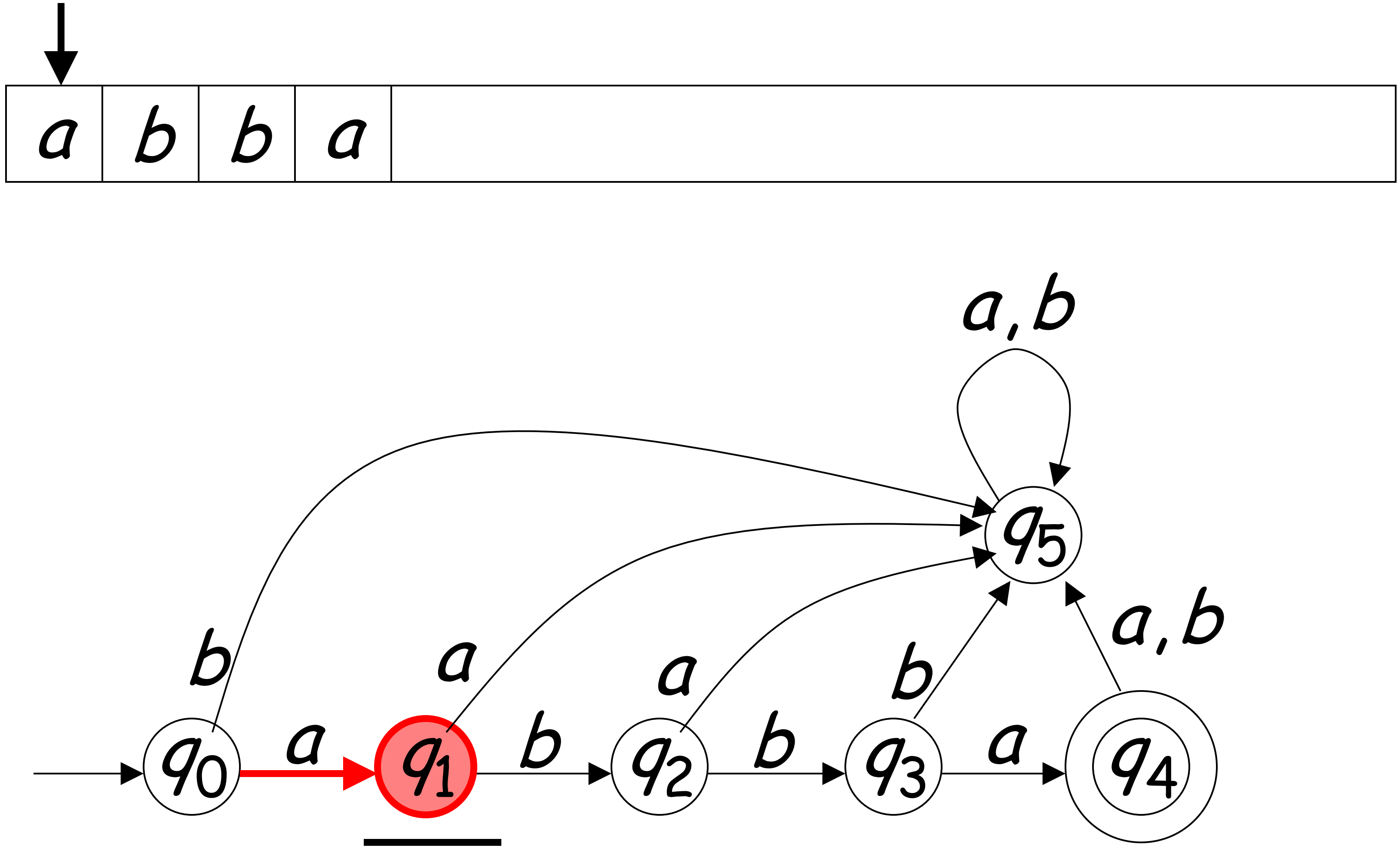
Initial Configuration

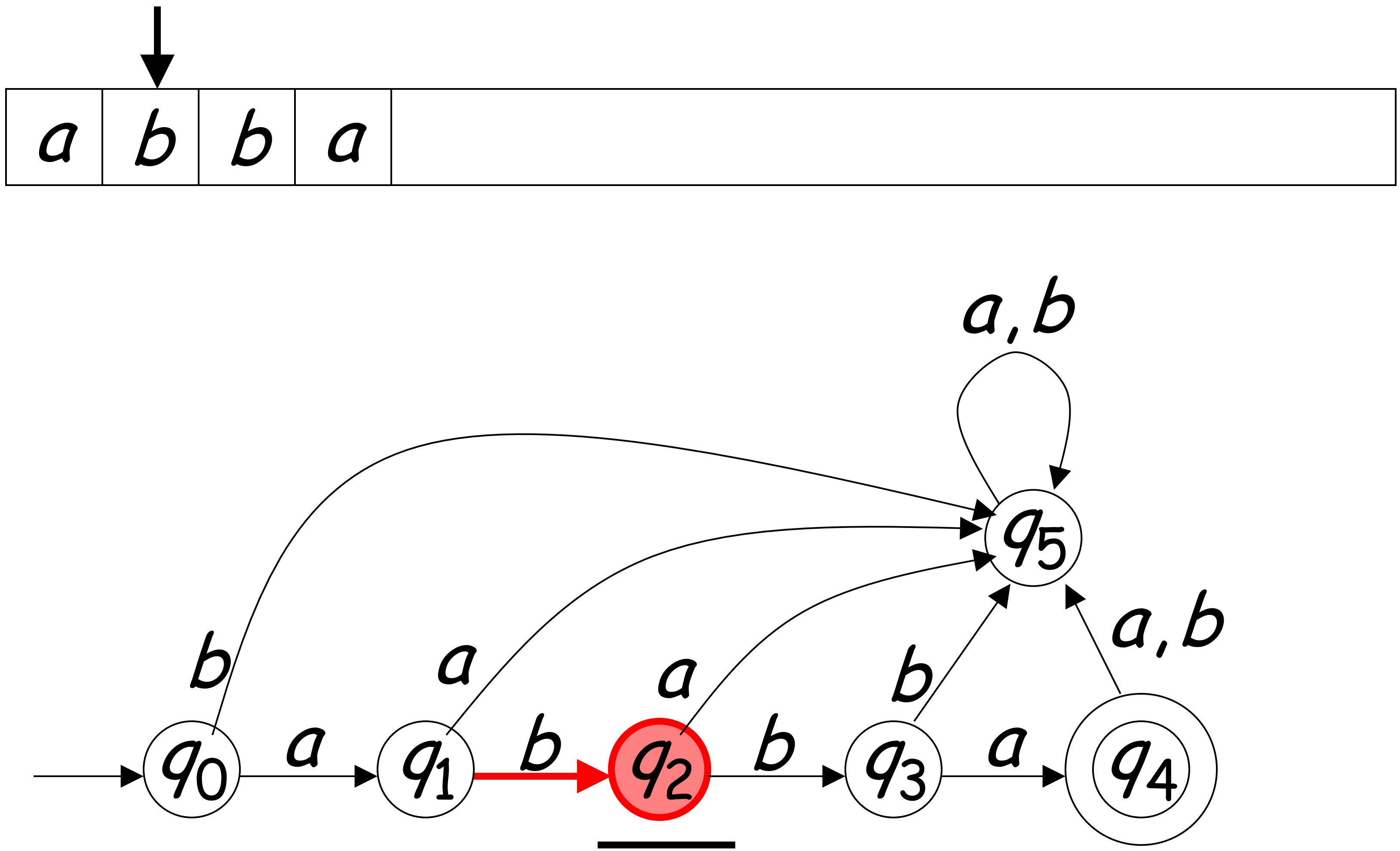
Input String

a	b	b	a	

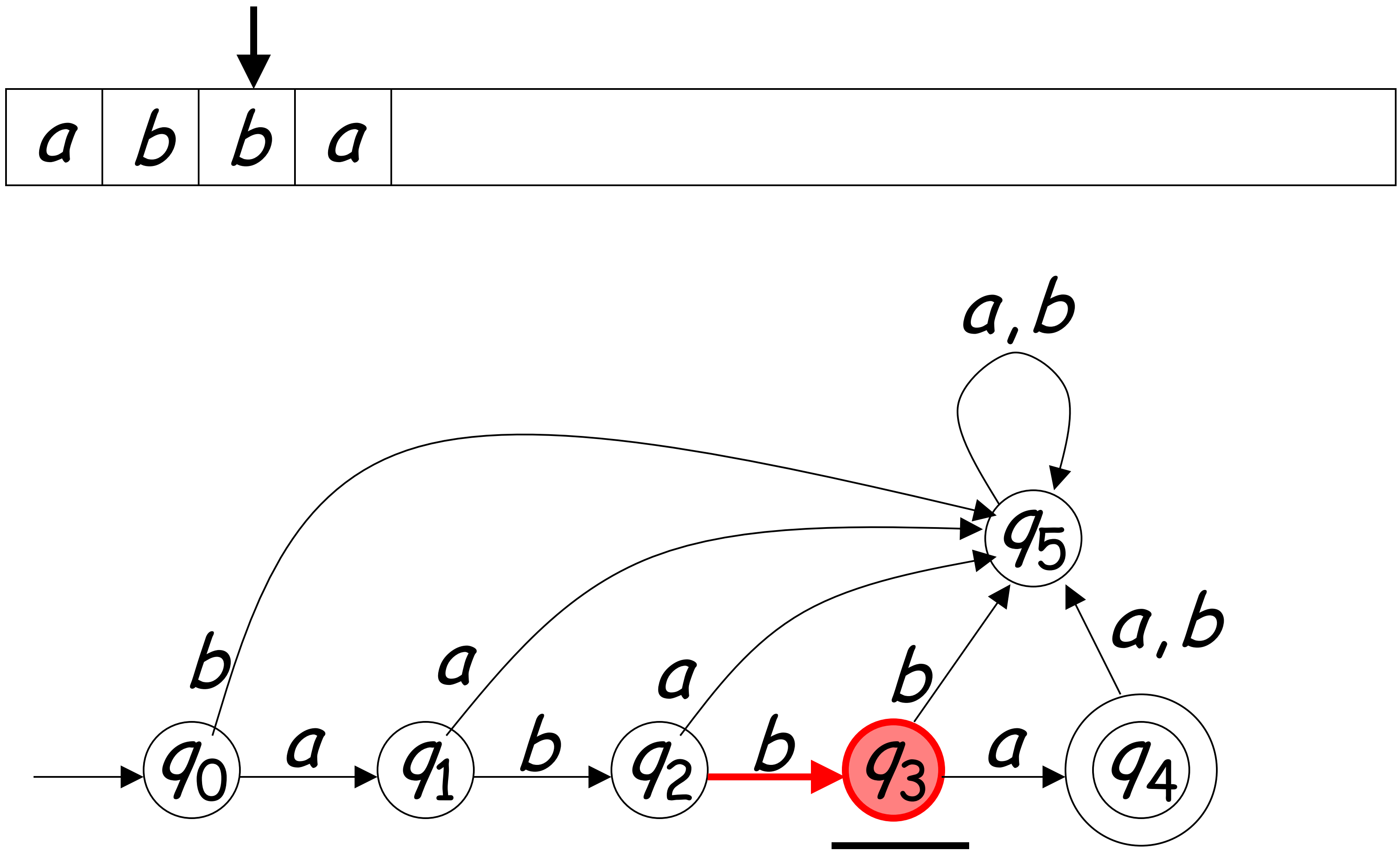


Reading the Input

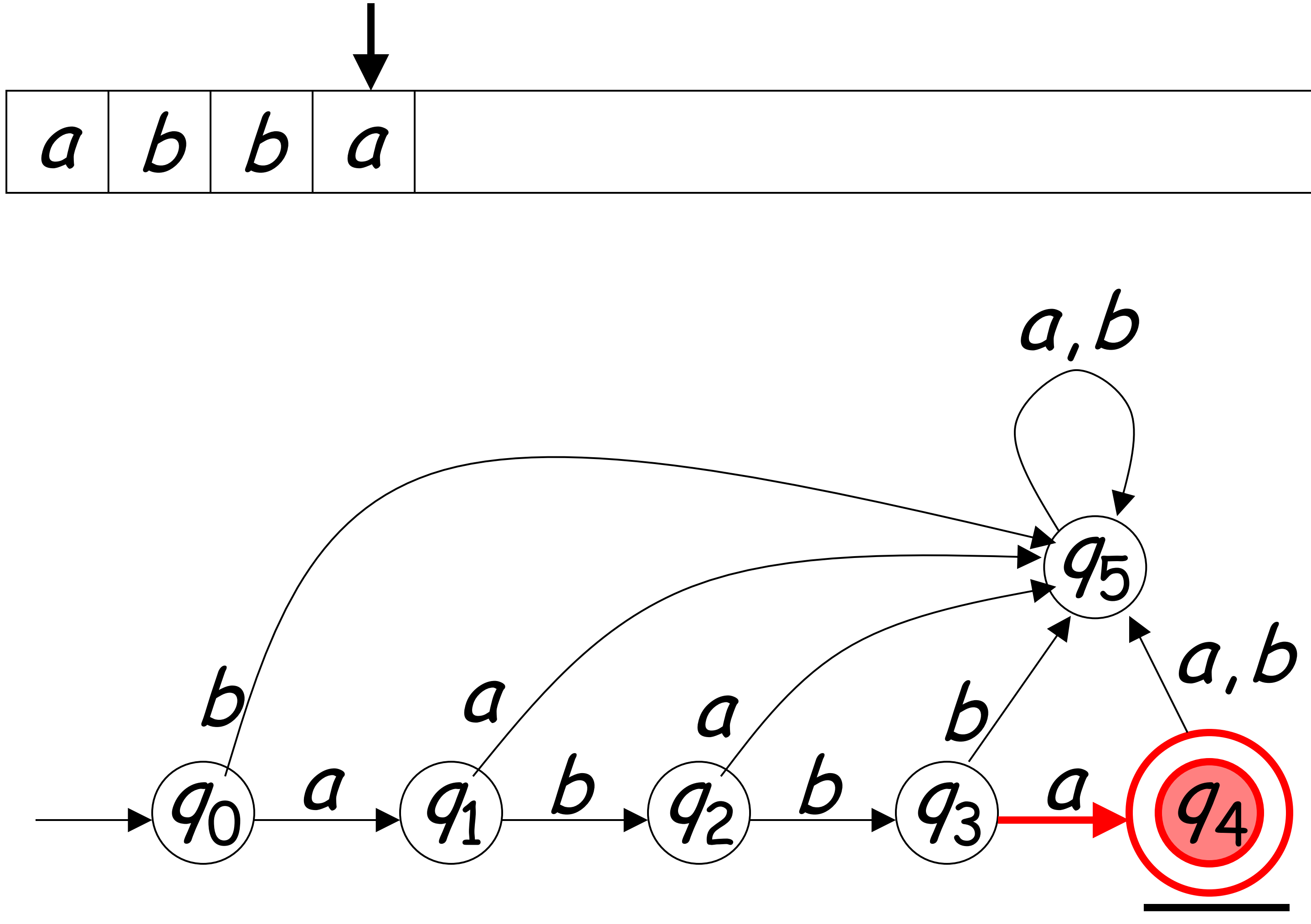




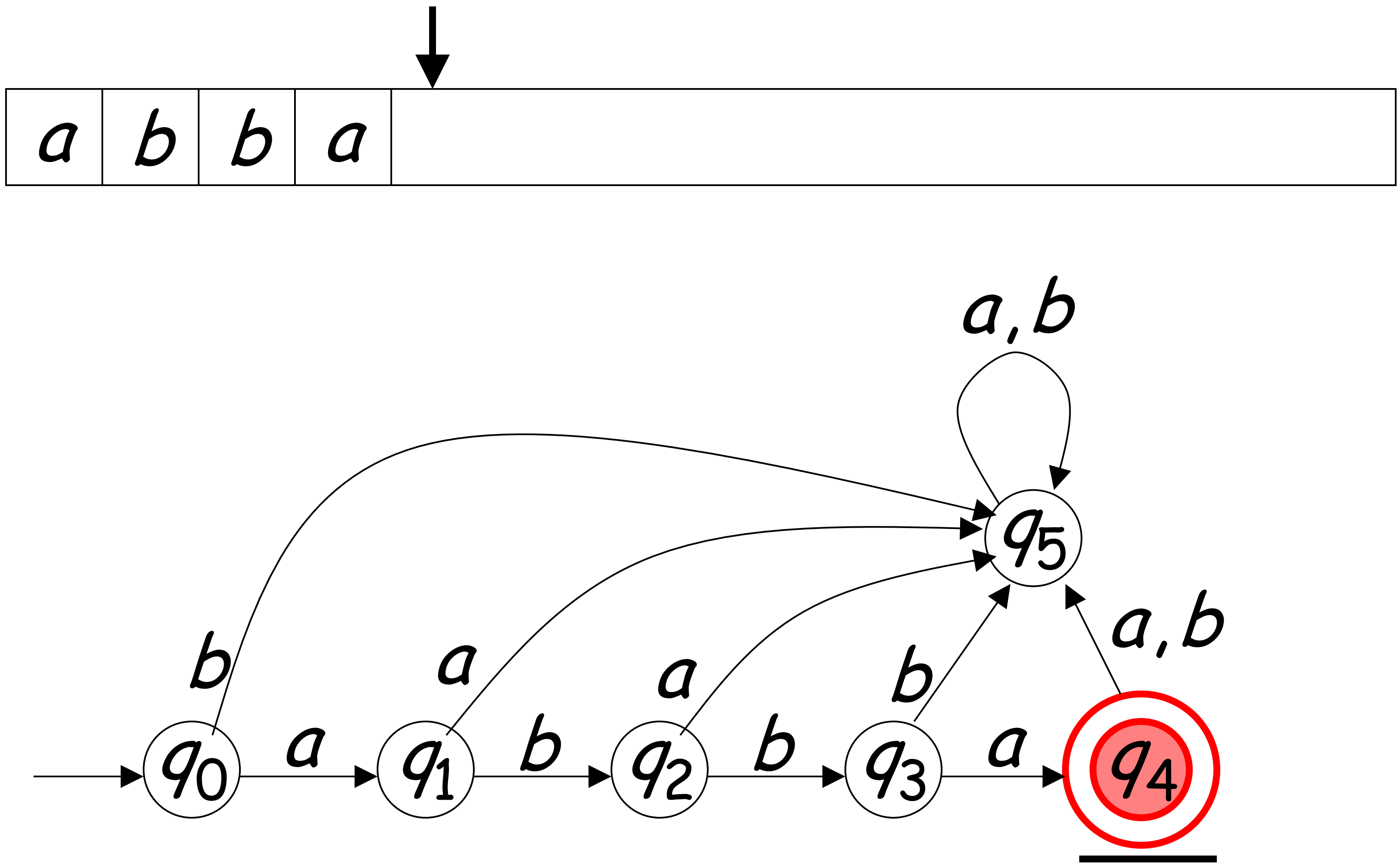
a
b
b
a
a,b
q_5
b
a
a
b
a,b
q_0
a
q_1
b
q_2
b
q_3
a
q_4



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

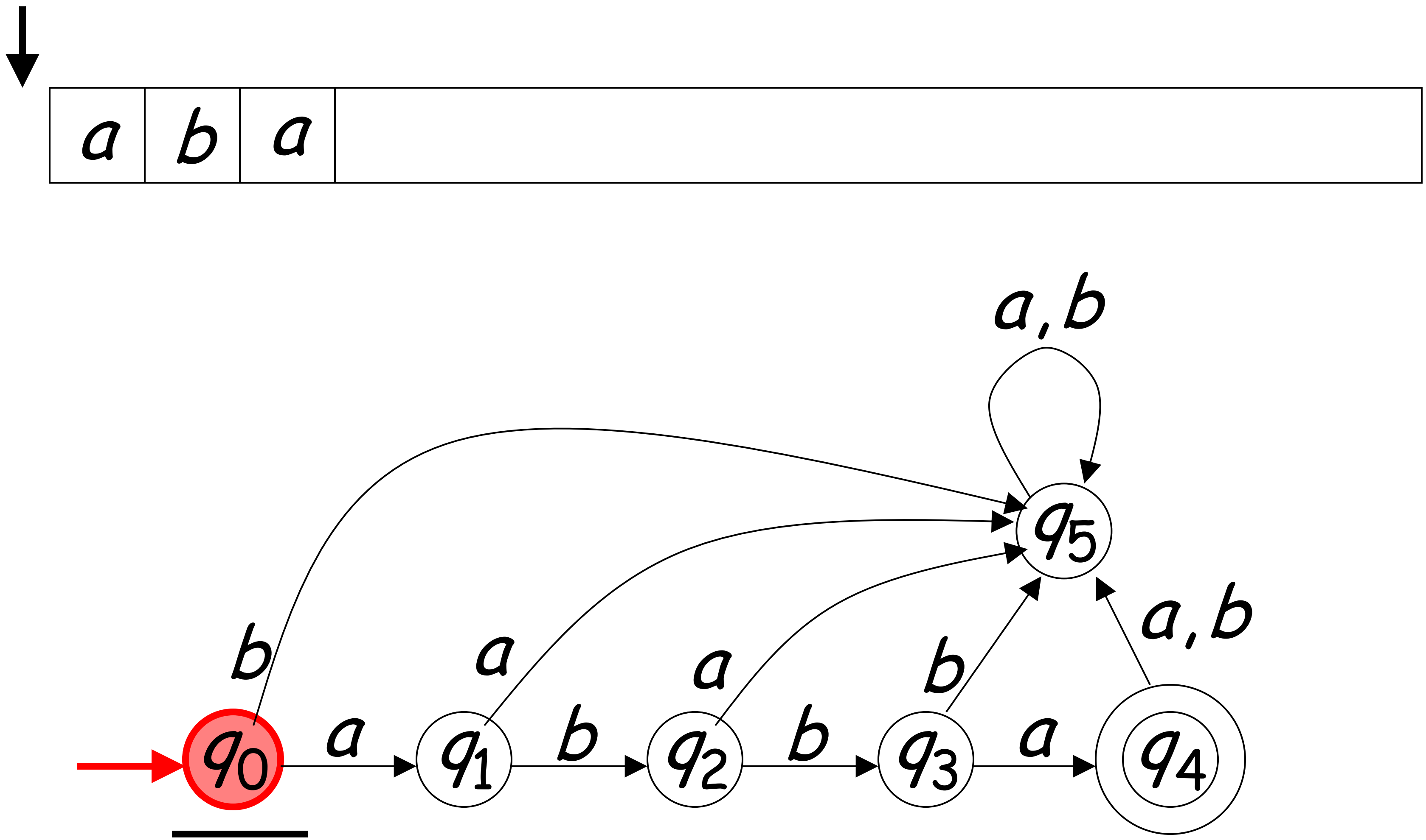

a
b
b
a
a,b
a,b
b
a
a
b
a
b
b
a
q_0
q_1
q_2
q_3
q_4
q_5

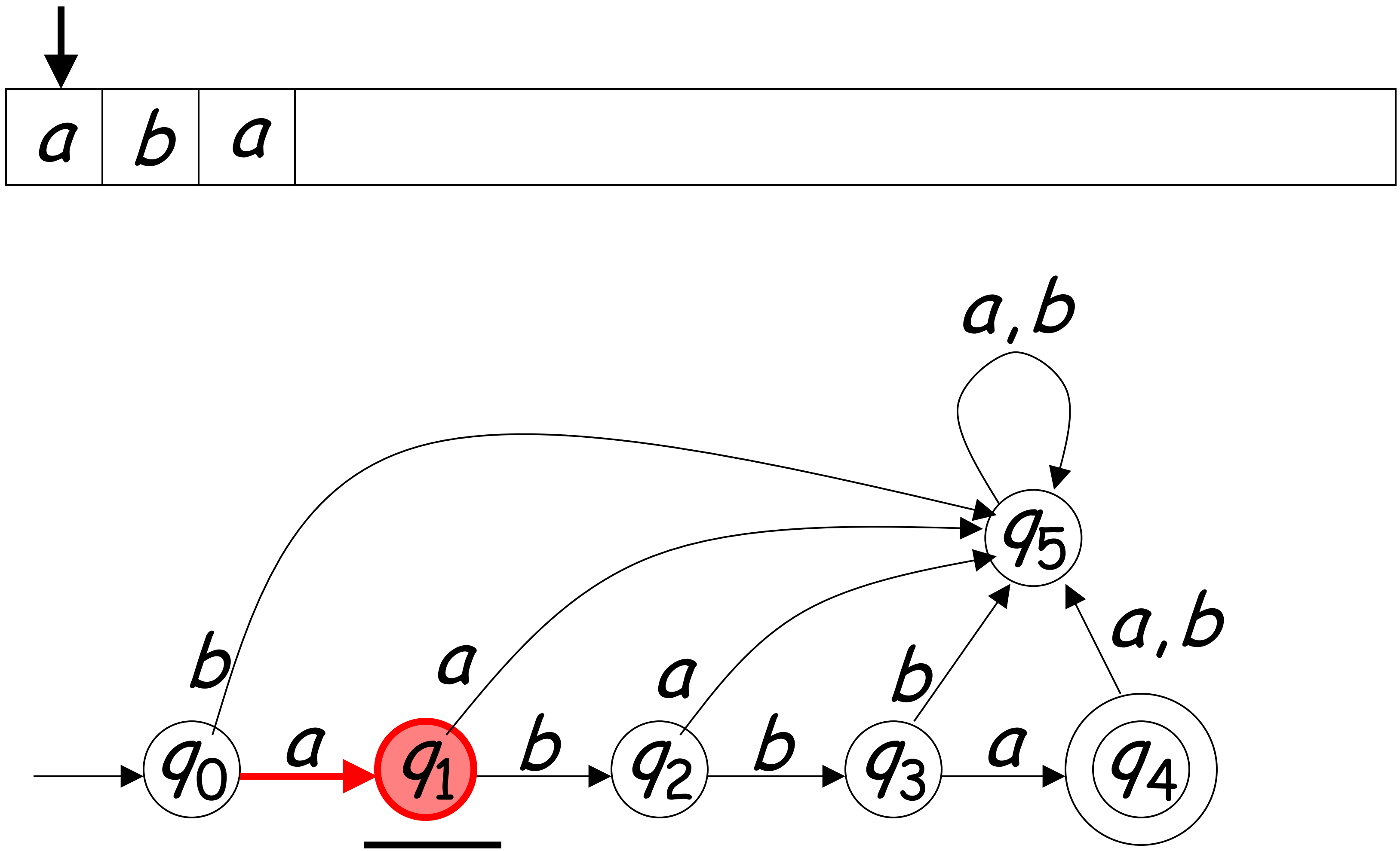
Input finished



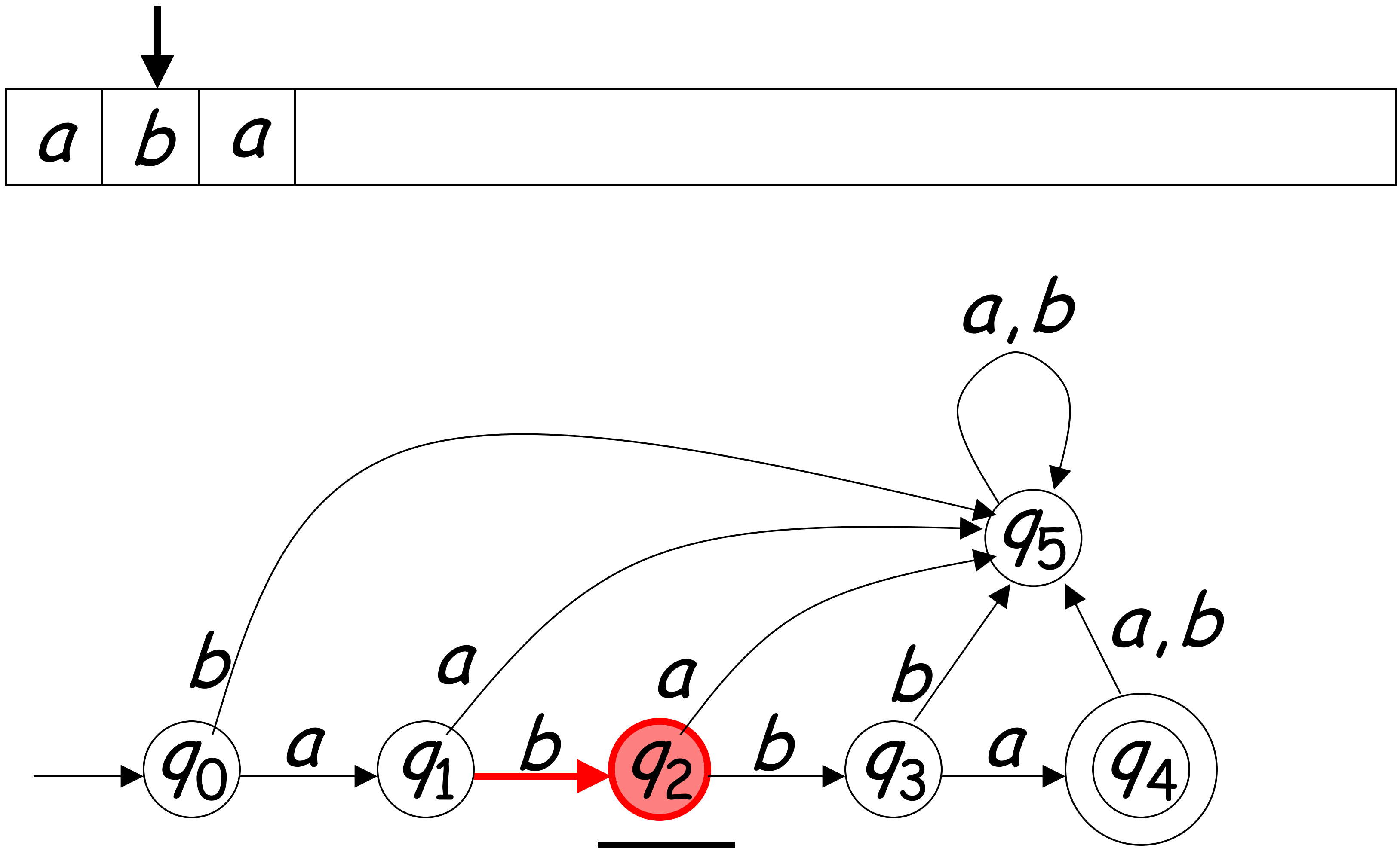
accept

Rejection

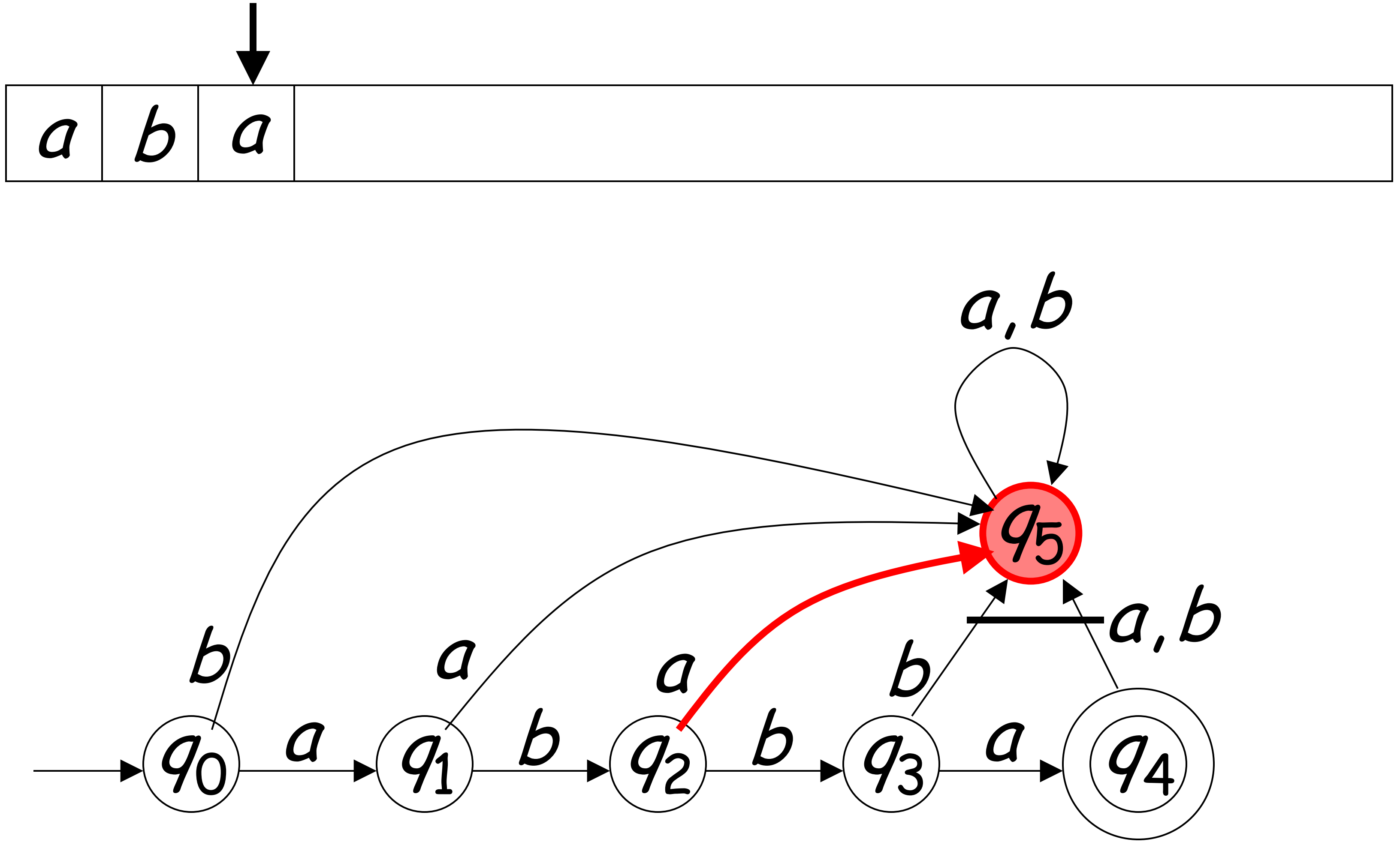




a
b
a
a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4

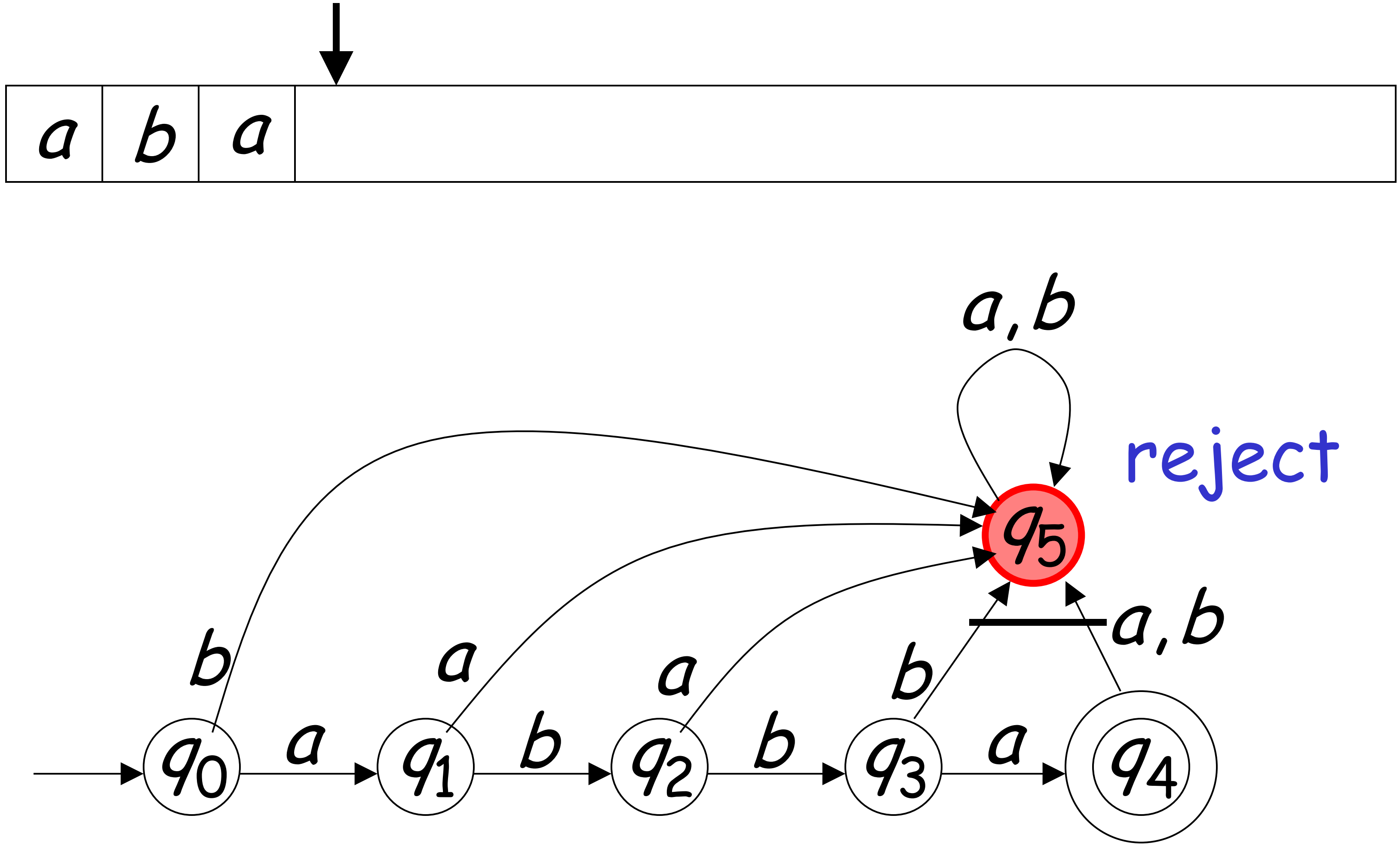


a
b
a
a,b
a,b
b
a
a
b
q0
a
q1
b
q2
b
q3
a
q4
q5

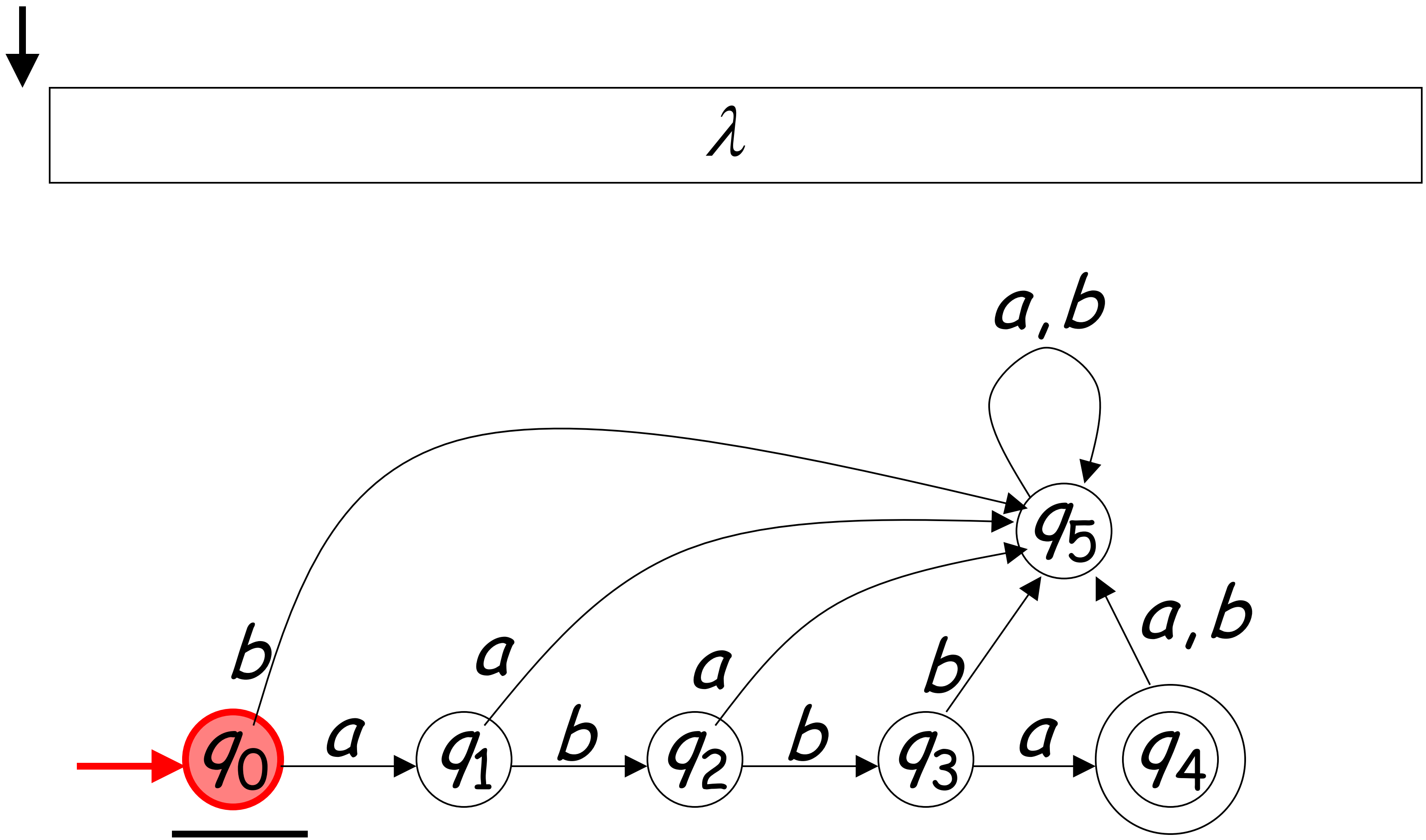


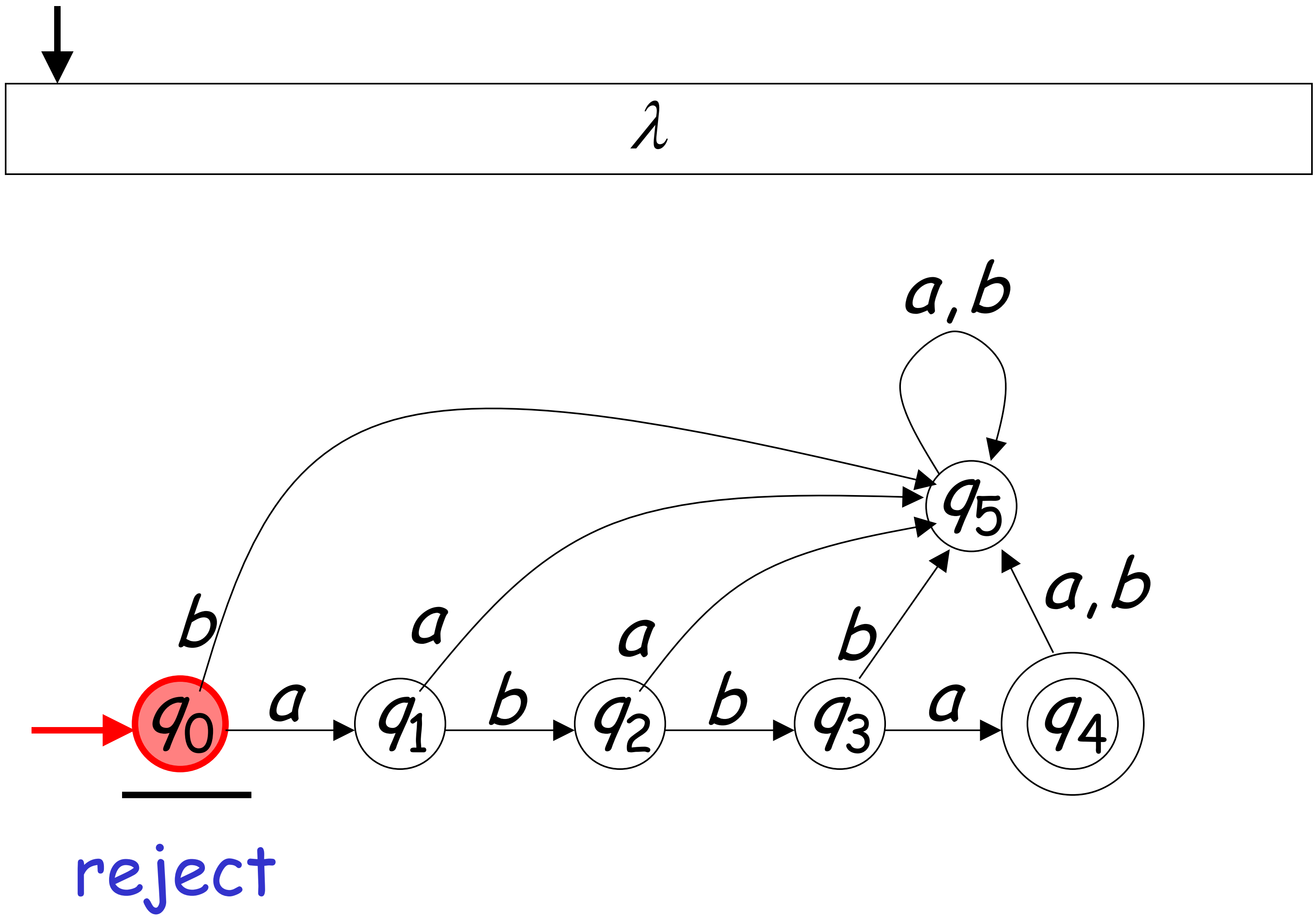
a
b
a
a, b
b
a
a
b
a, b
q_0
a
q_1
b
q_2
b
q_3
a
q_4
q_5

Input finished



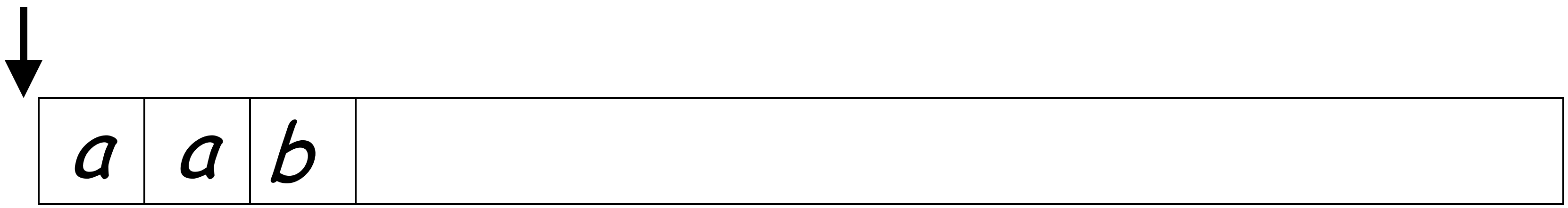
Another Rejection

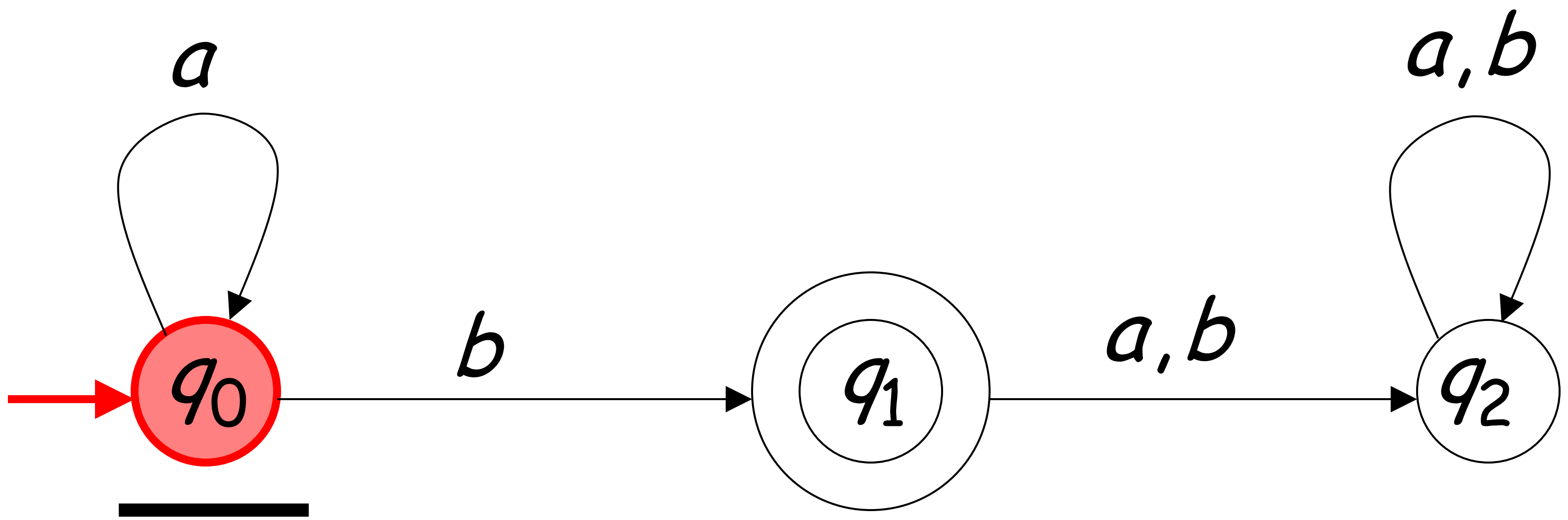


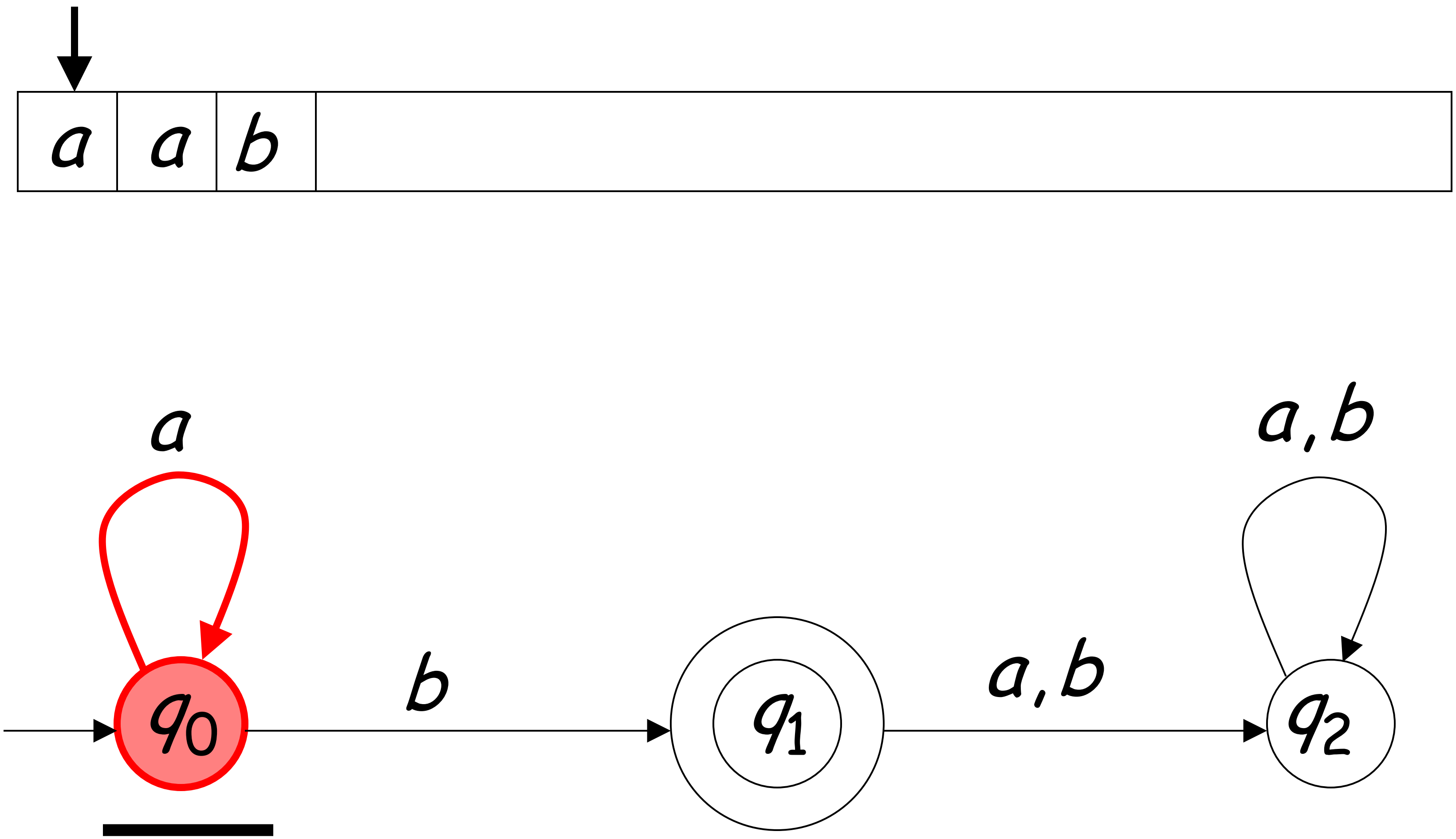


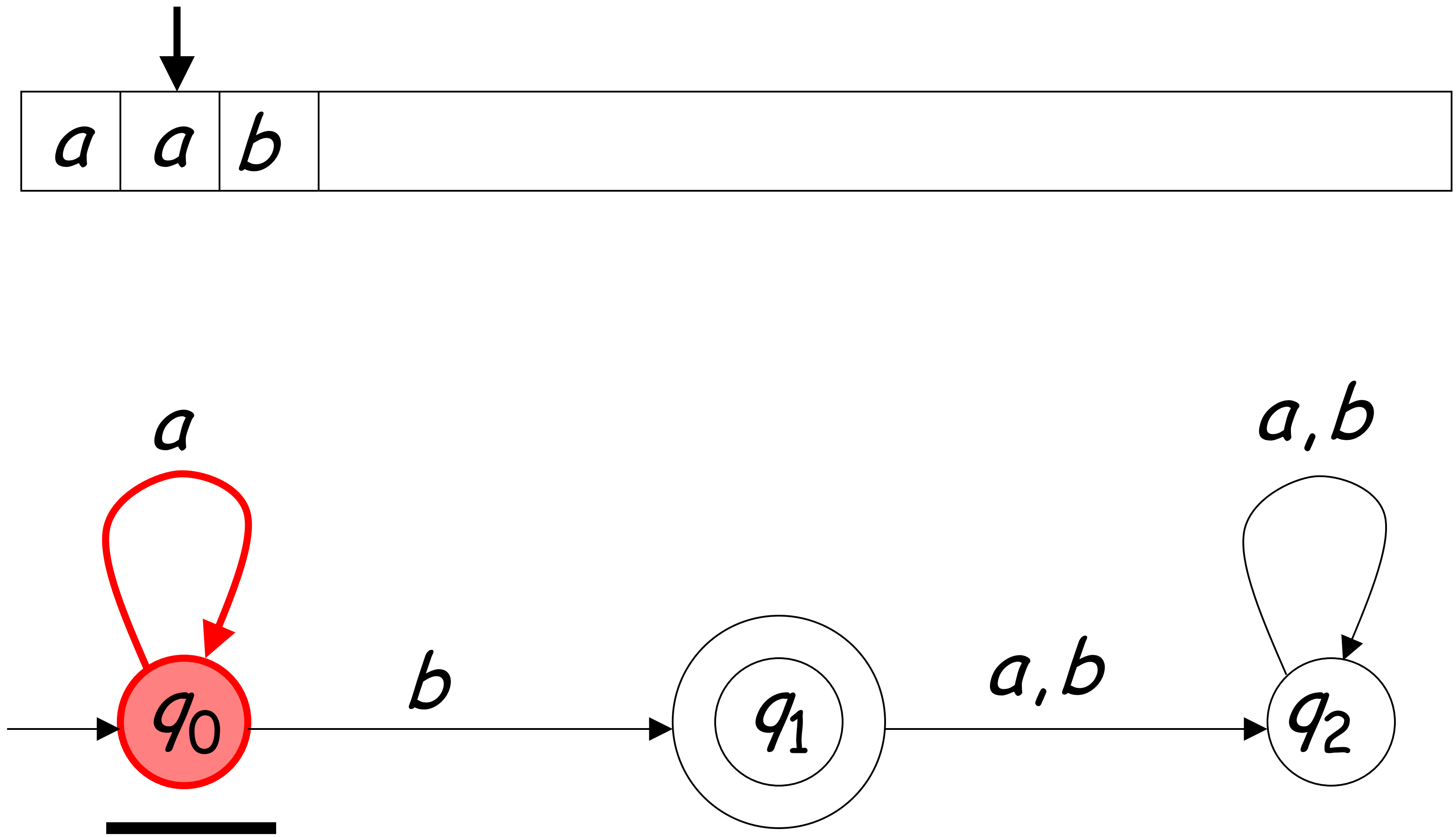
λ
a, b
b
a
a
b
a, b
a
b
b
a
q_0
q_1
q_2
q_3
q_4
q_5
reject

Another Example

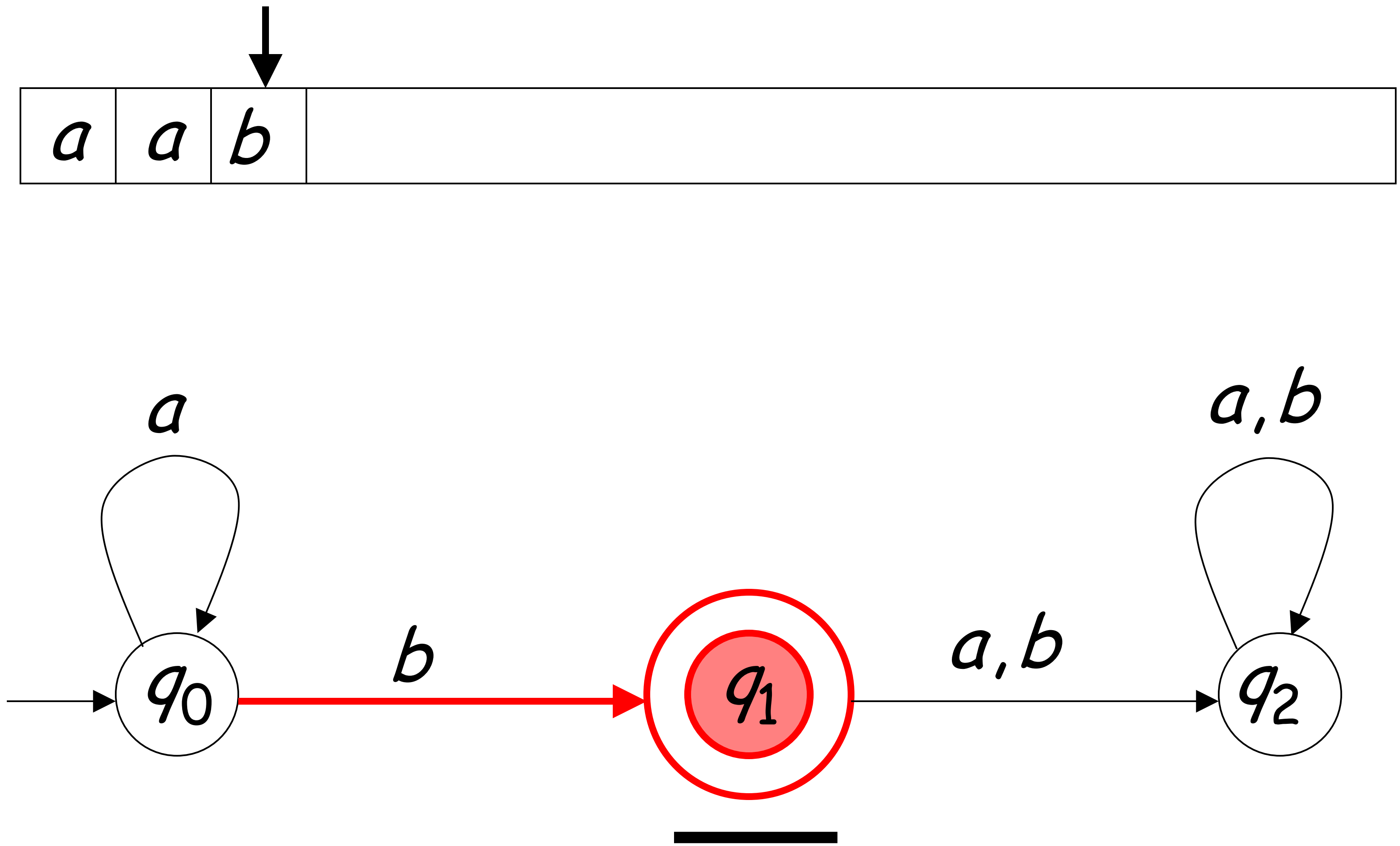


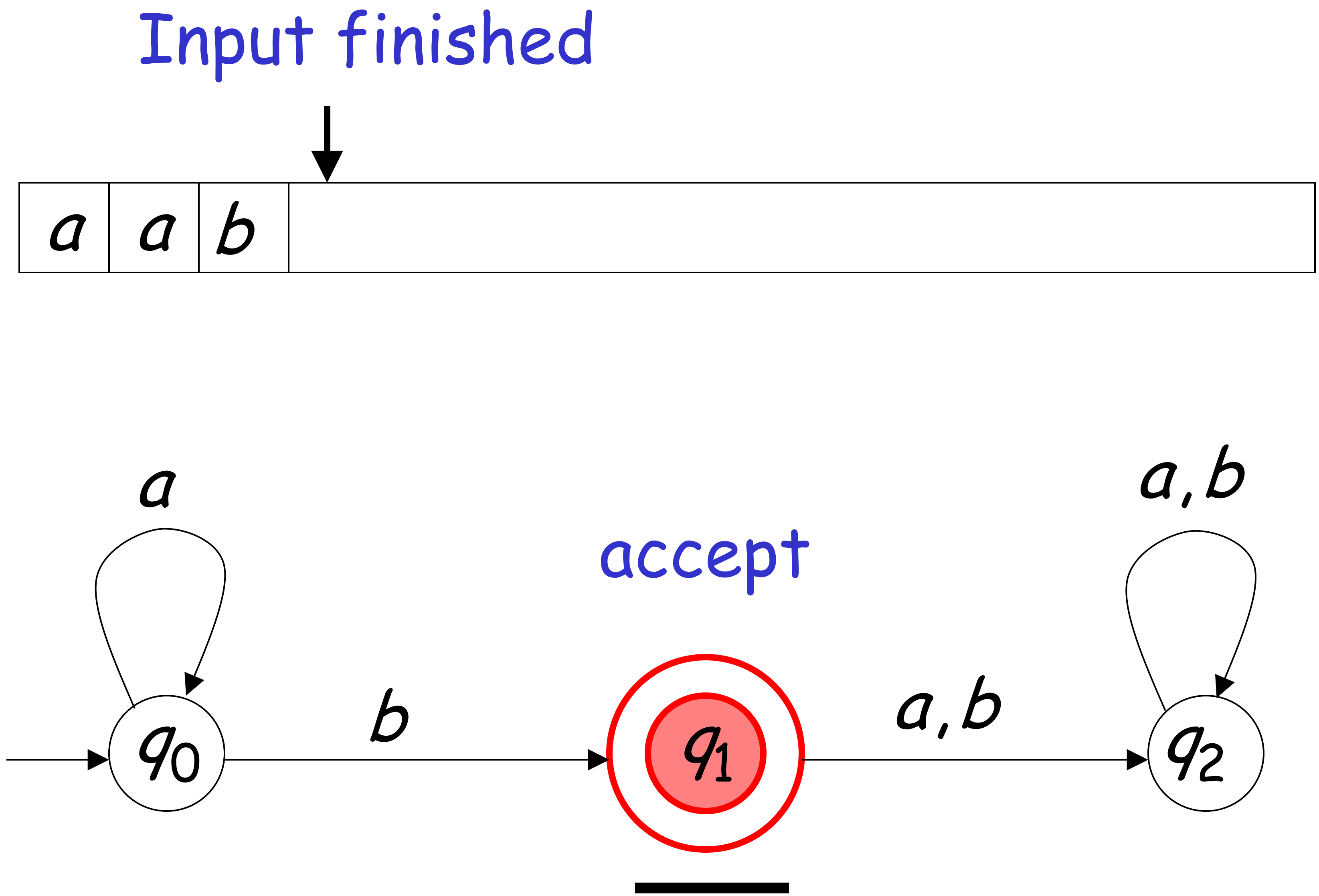



a
a
b
a
a, b
b
a, b
q_0
q_1
q_2

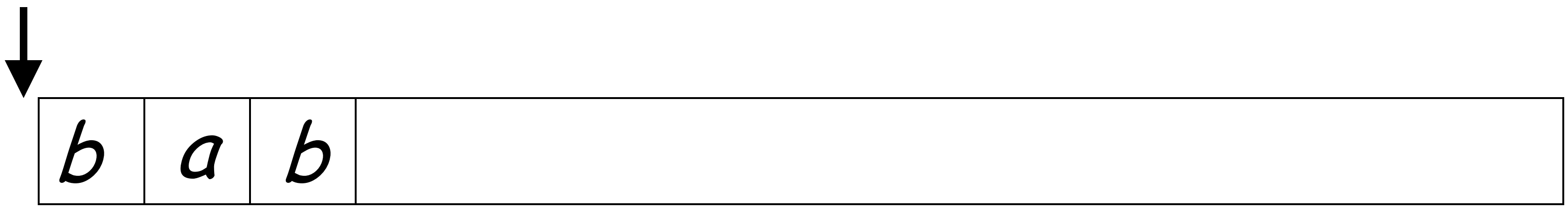


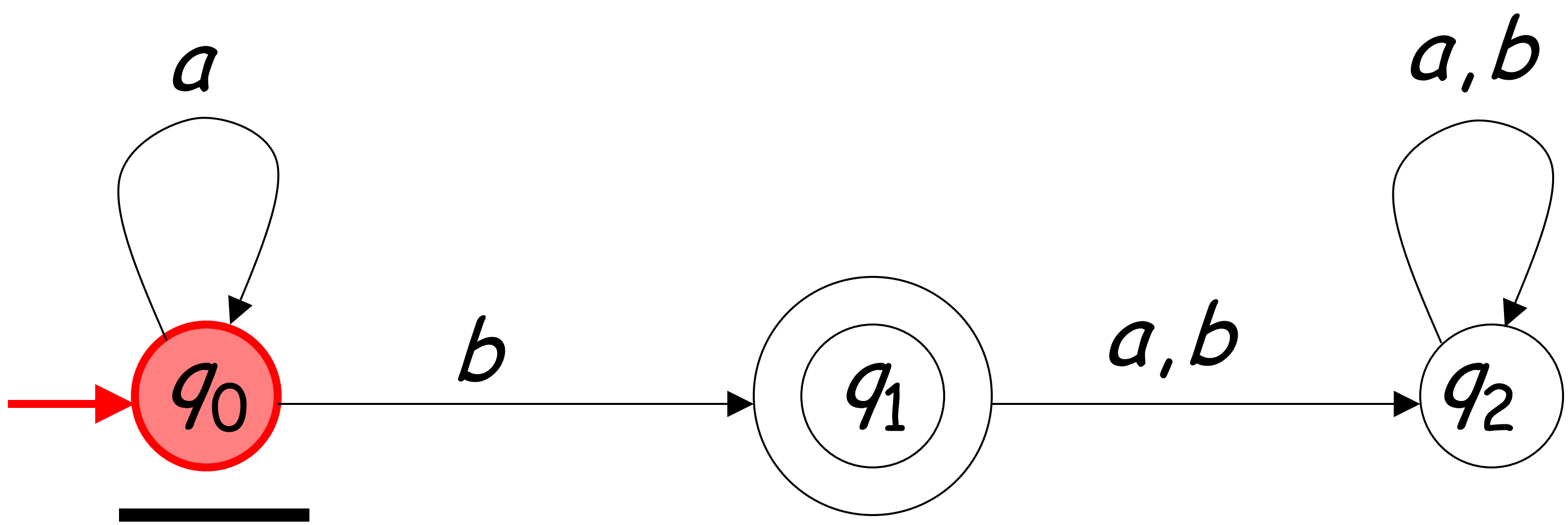
a
a
b
a
a,b
b
a,b
q_0
q_1
q_2

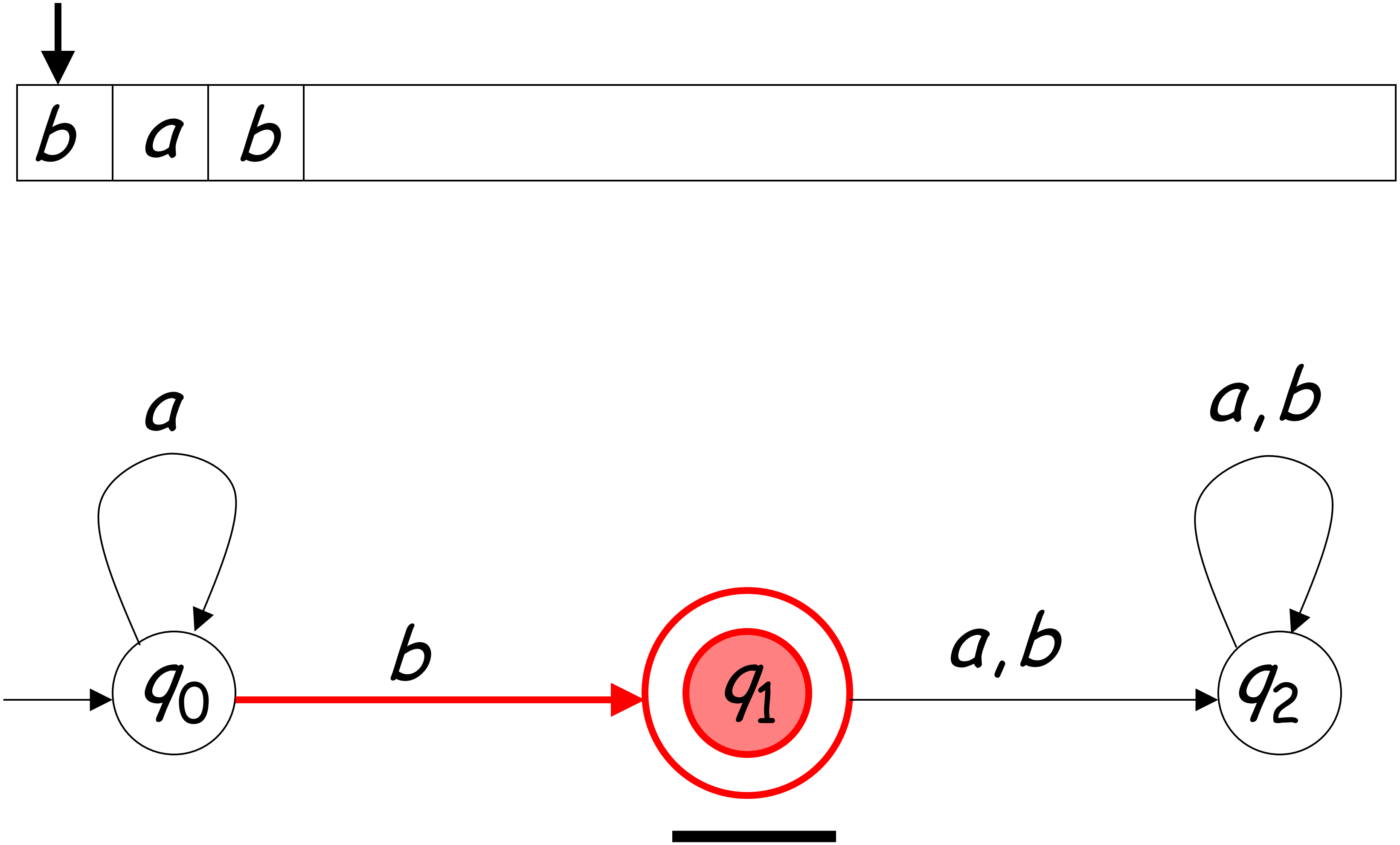

a
a
b
a
a,b
b
a,b
q_0
q_1
q_2

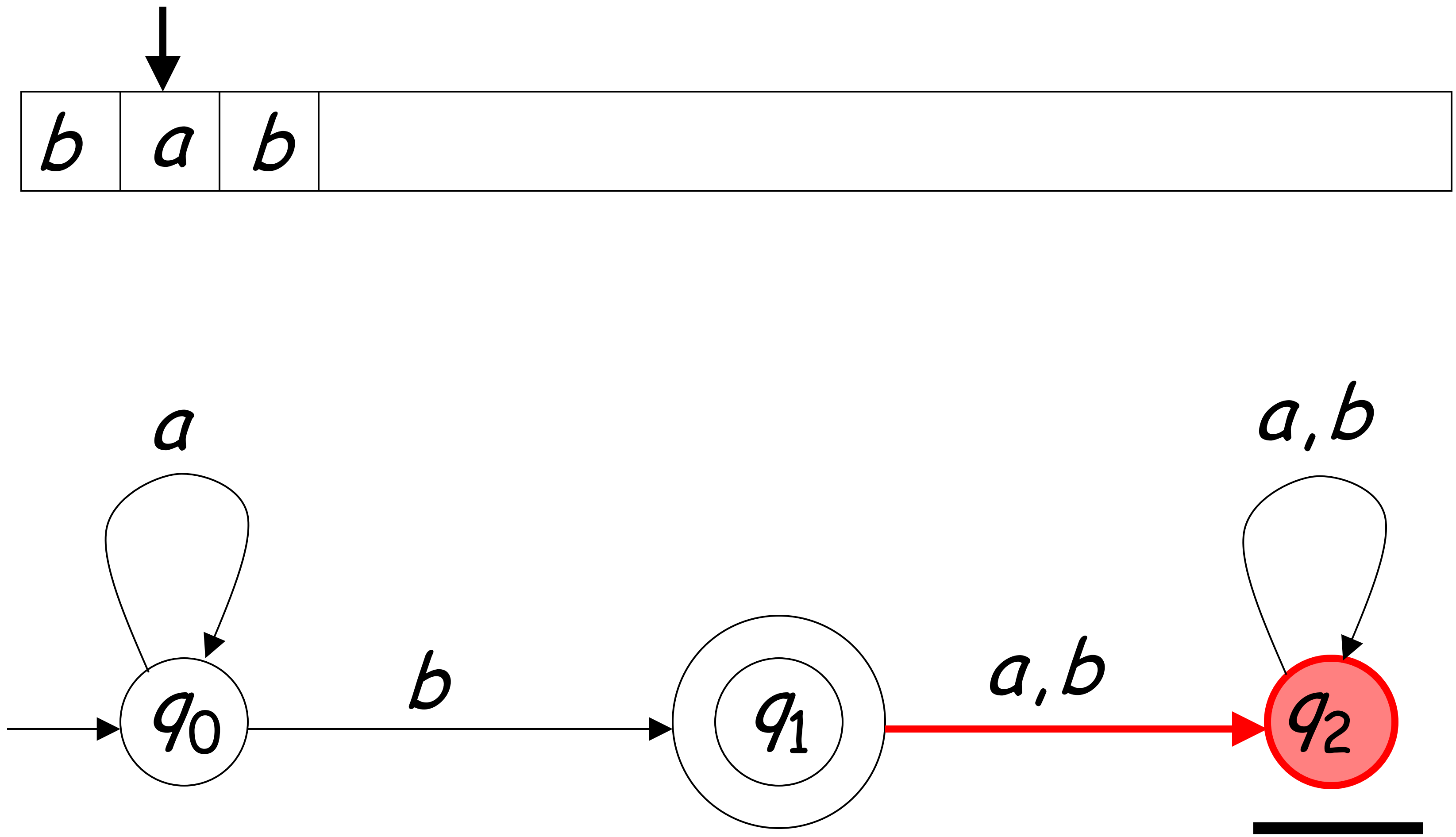

Input finished
a
a
b
a
a,b
accept
b
a,b
q_0
q_1
q_2

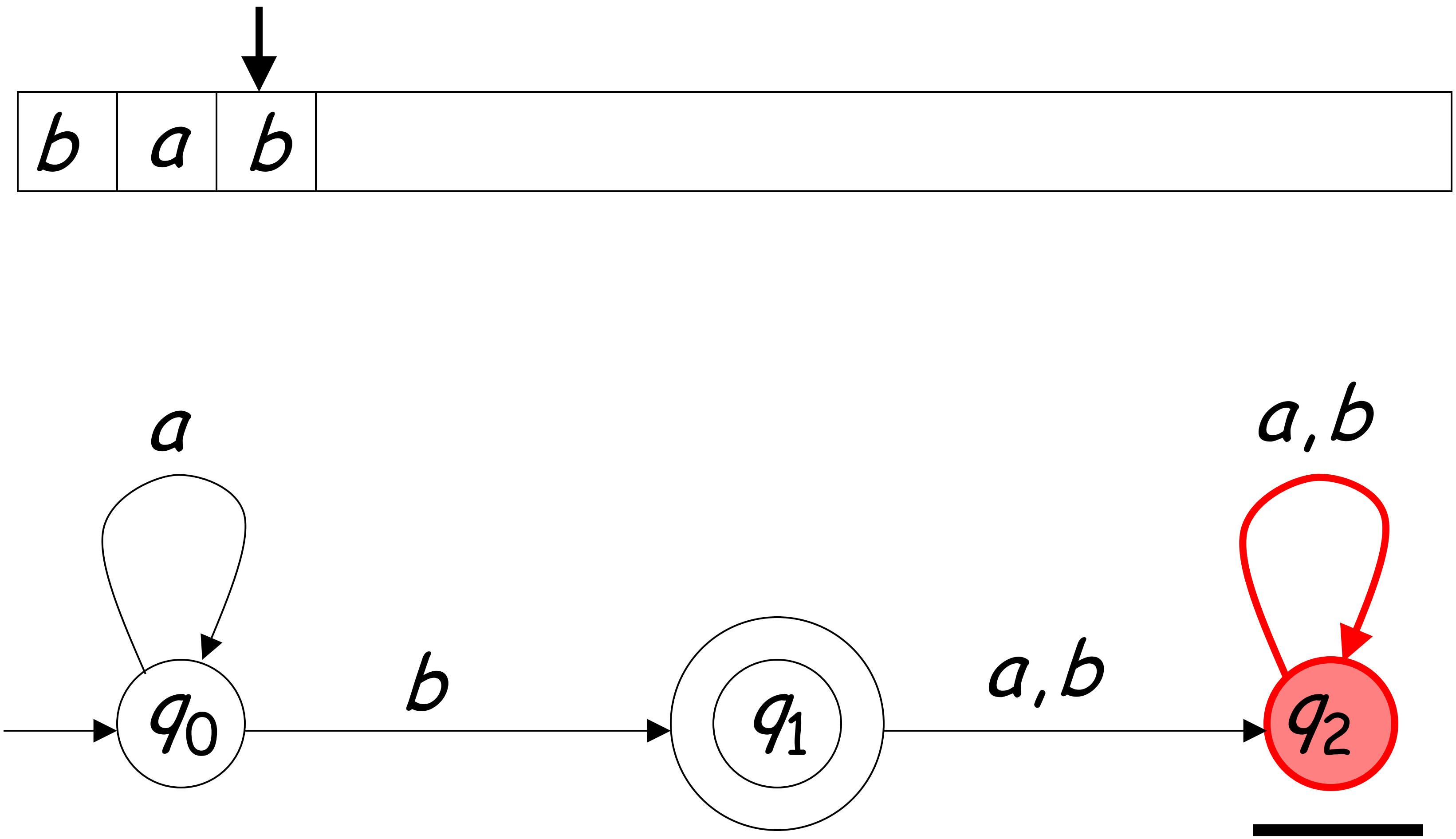
Rejection Example





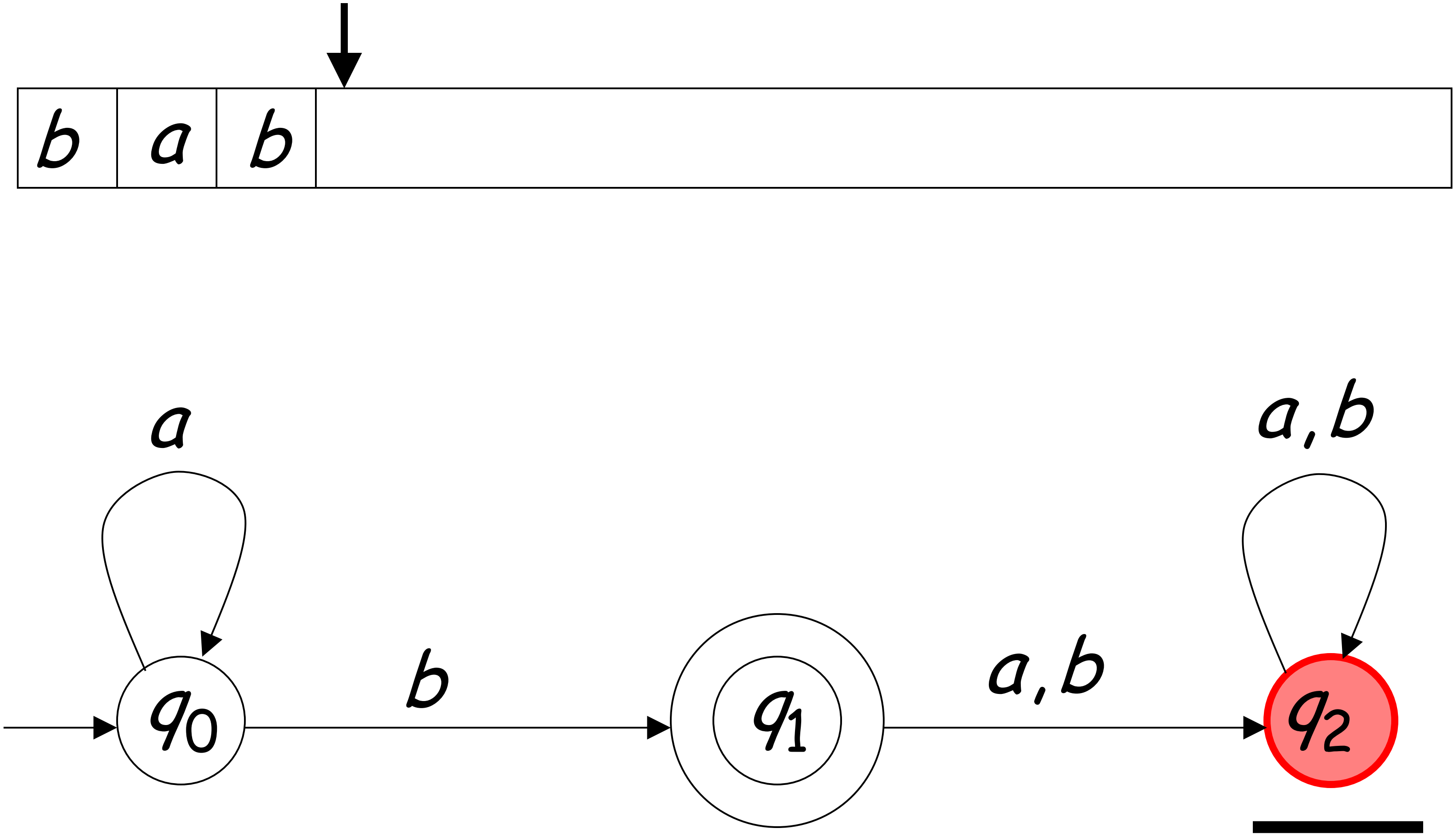

b
a
b
a
a,b
b
a,b
q_0
q_1
q_2


b
a
b
a
a,b
b
q0
q1
a,b
q2



b a b
a
a, b
b
a, b
q_0
q_1
q_2

Input finished



reject

Languages Accepted by FAs

FA M

Definition:

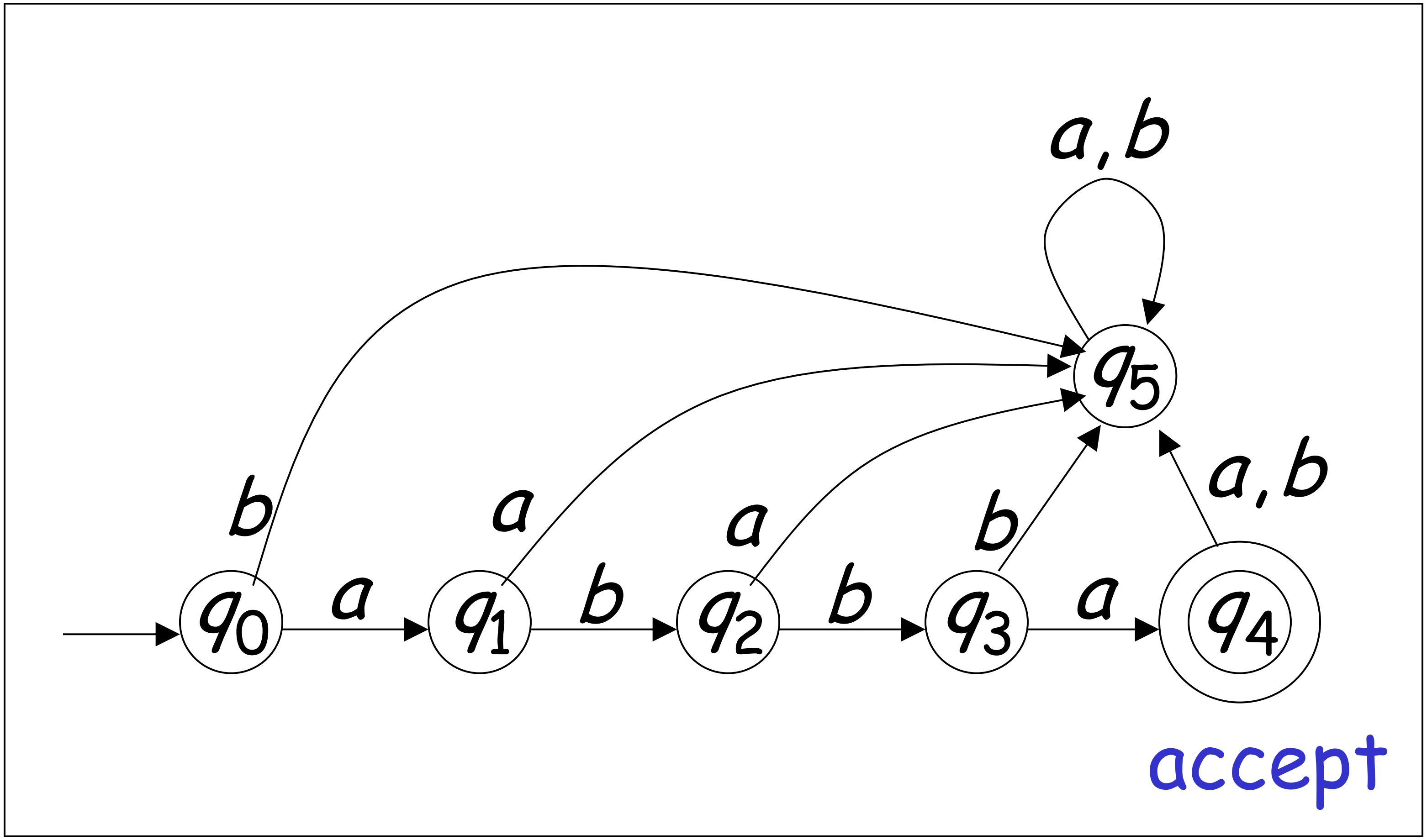
The language $L(M)$ contains all input strings accepted by M

$L(M)$ = { strings that bring M to an accepting state}

Example

$L(M) = \{abba\}$

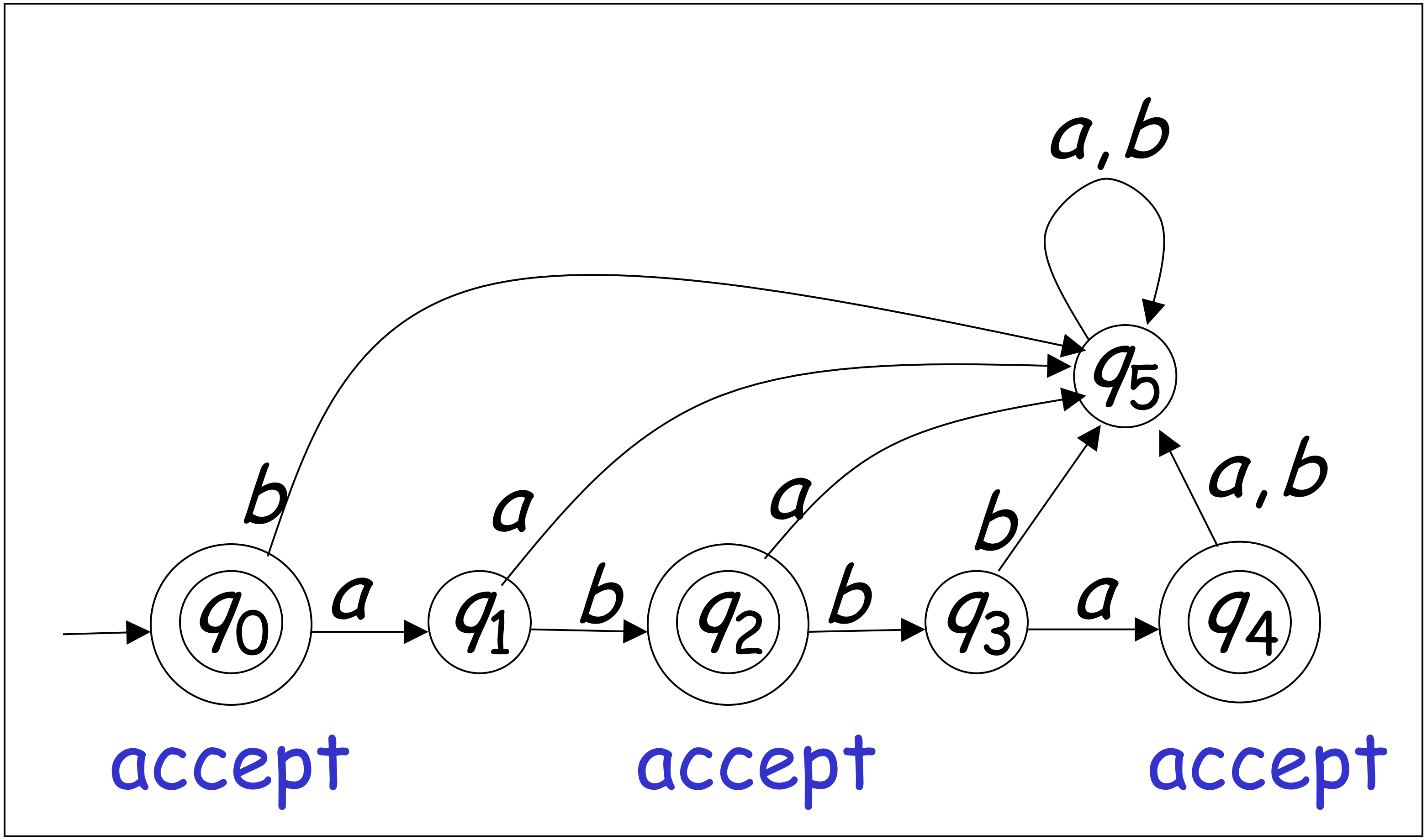
M



Example

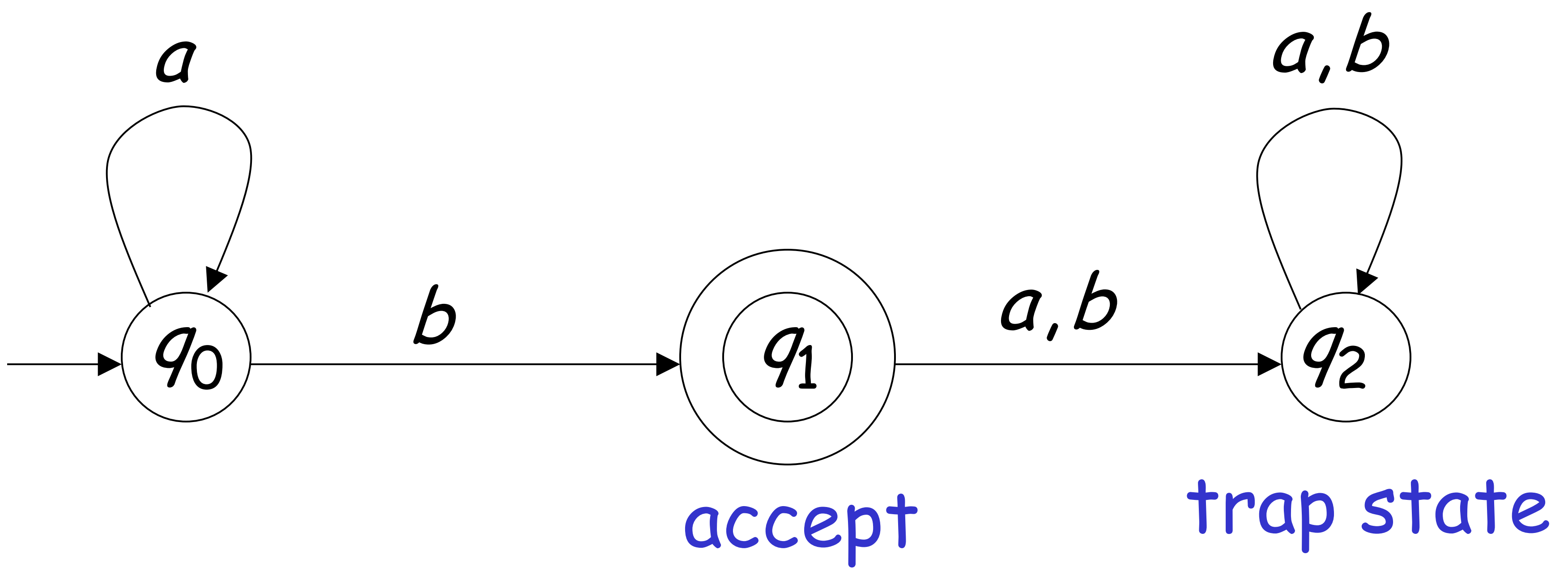
$$L(M)=\{\lambda, ab, abba\}$$

M



Example

$$L(M) = \{a^n b : n \geq 0\}$$



Formal Definition

Finite Automaton (FA)

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

Q : set of states

Σ : input alphabet

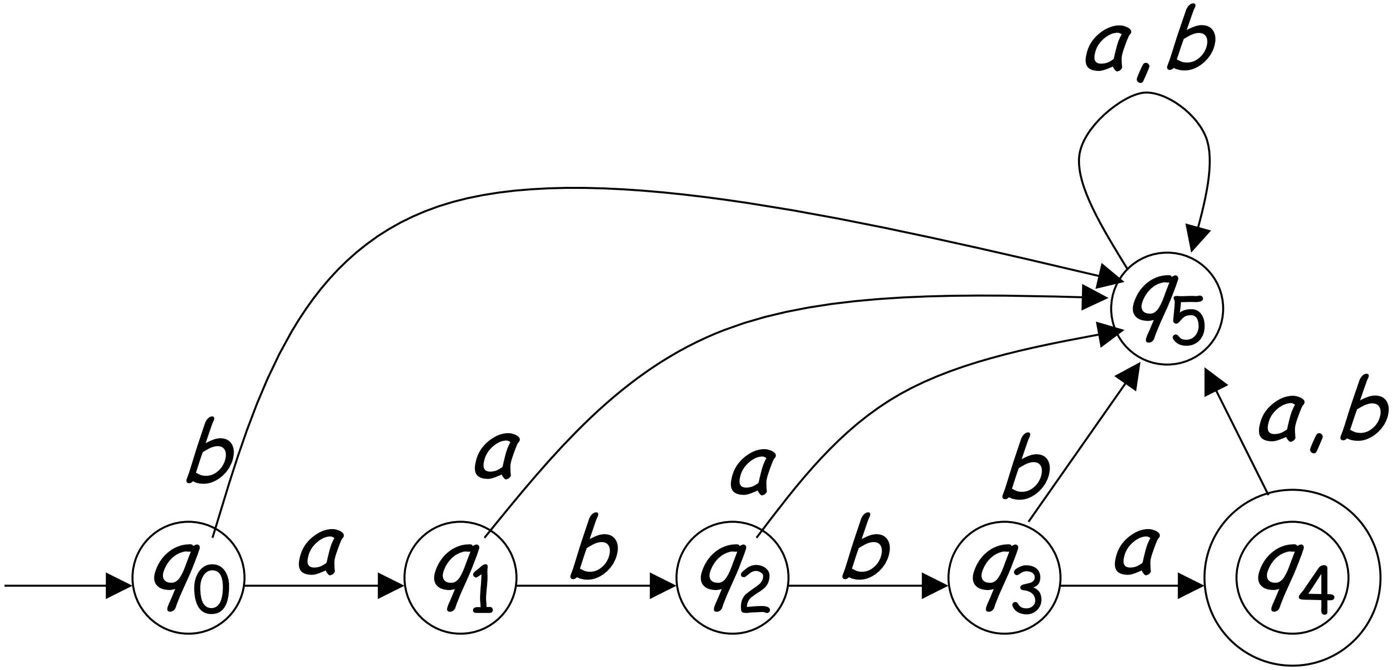
δ : transition function

q_0 : initial state

F : set of accepting states

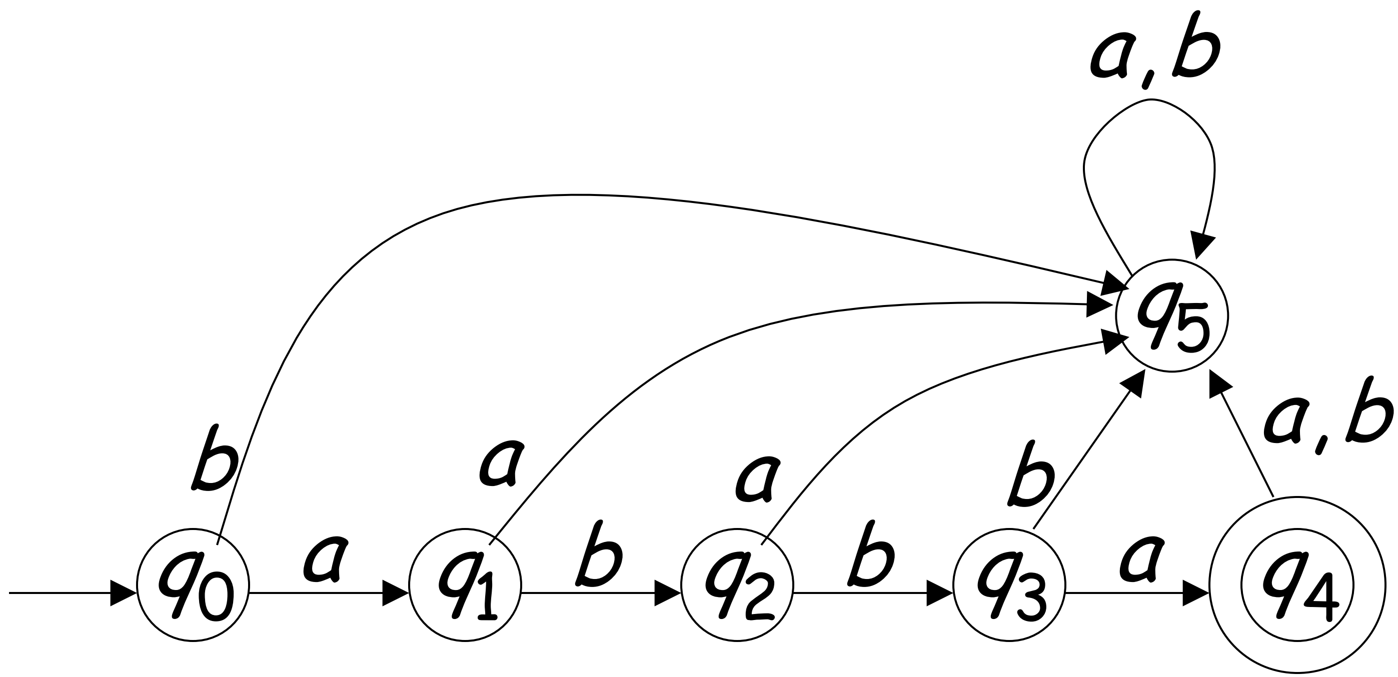
Input Alphabet Σ

$$\Sigma = \{a, b\}$$

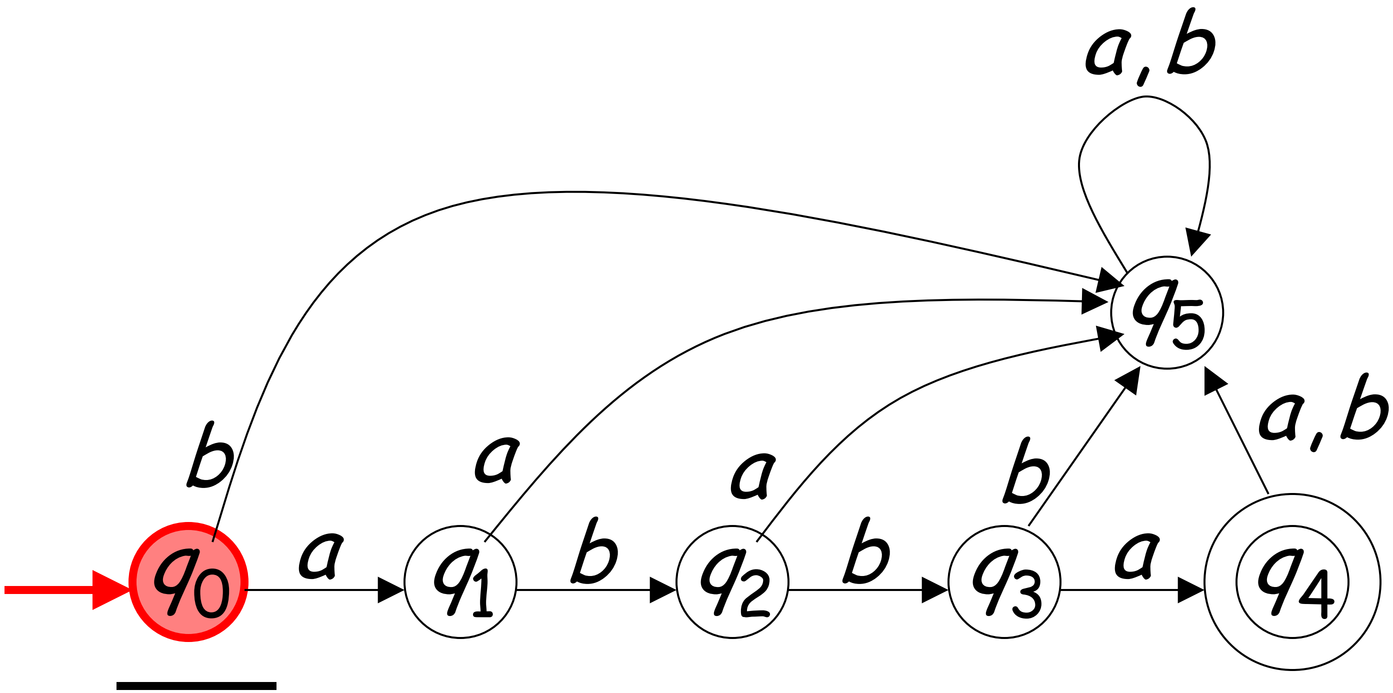


Set of States Q

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

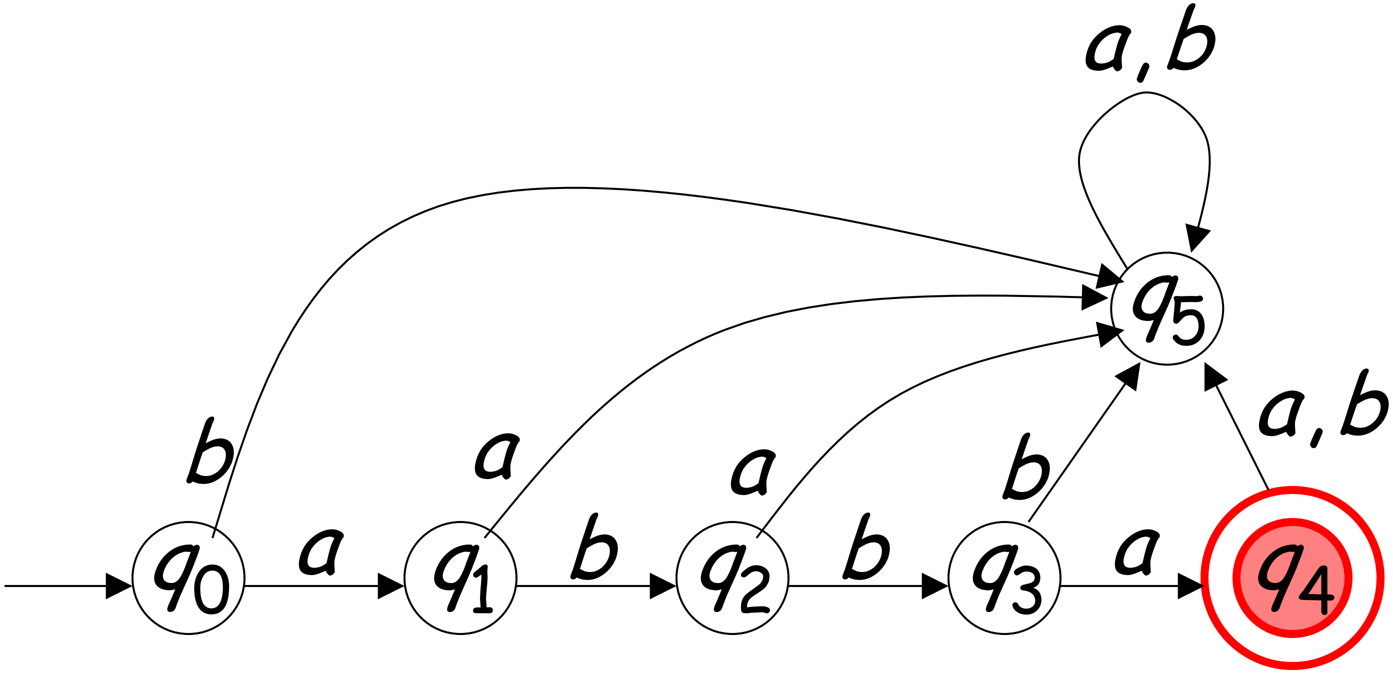


Initial State q_0



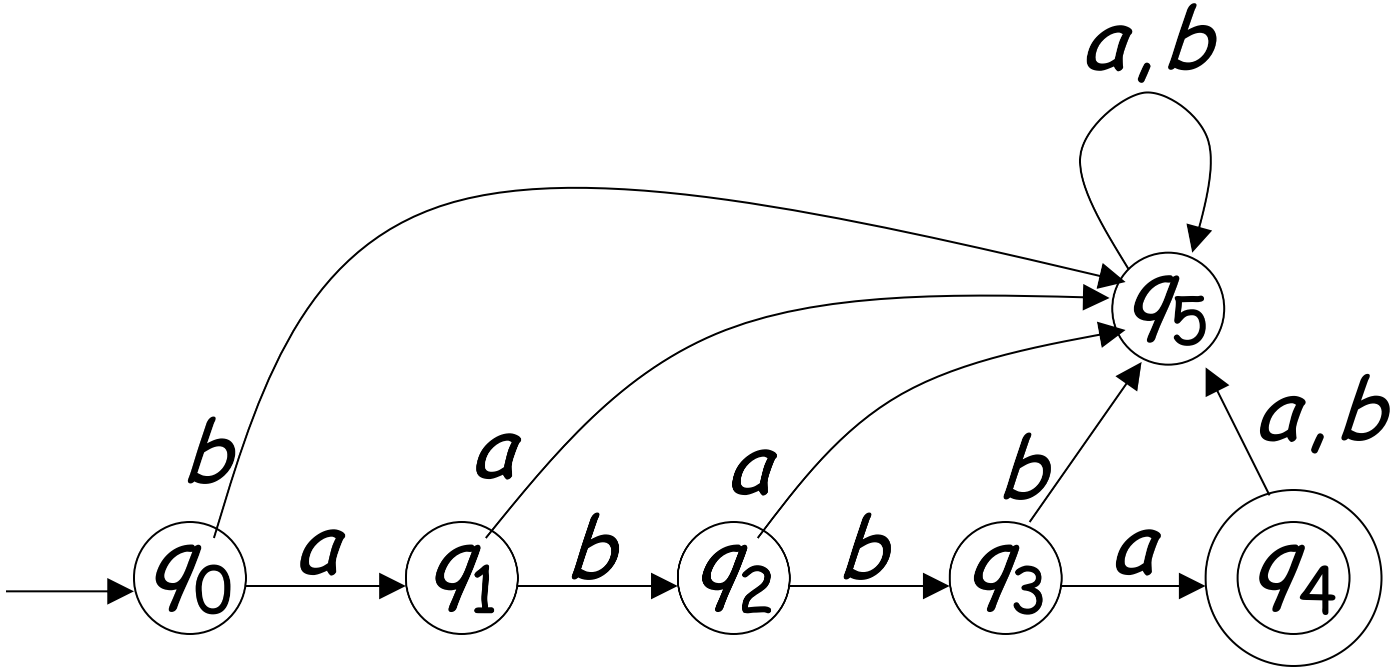
Set of Accepting States F

$$F = \{q_4\}$$

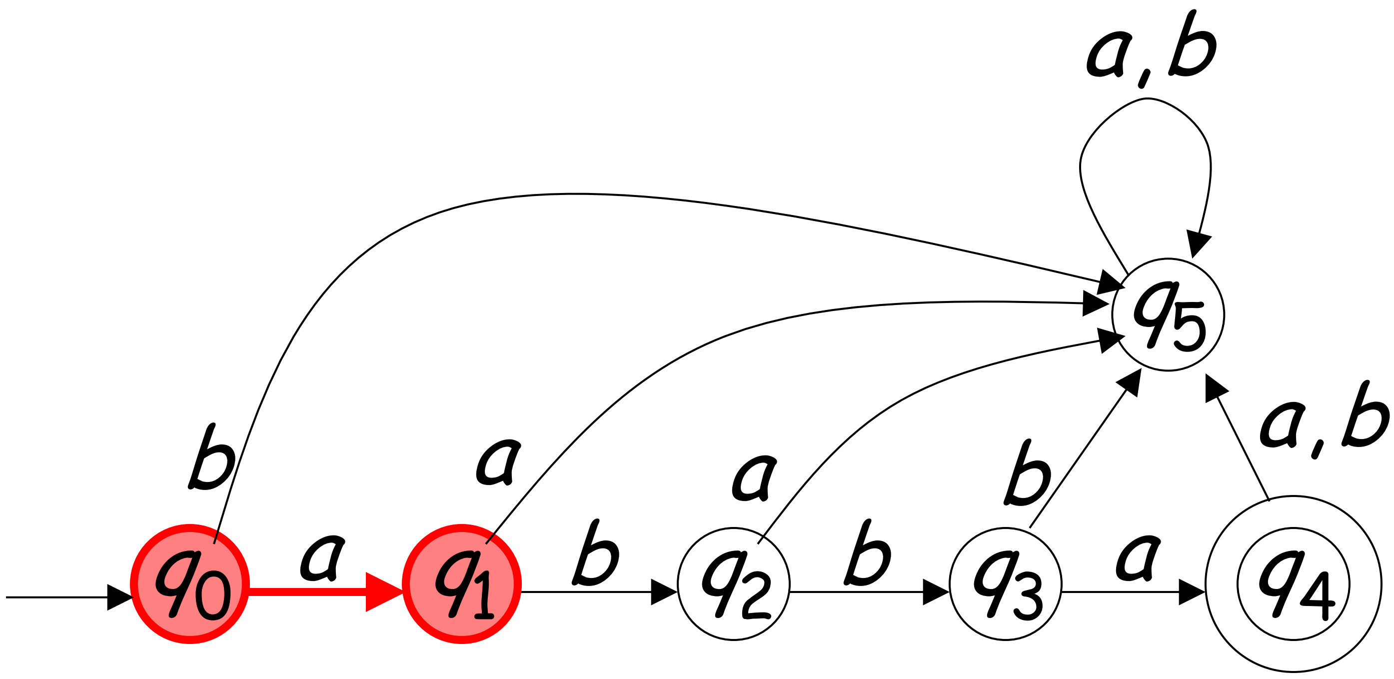


Transition Function δ

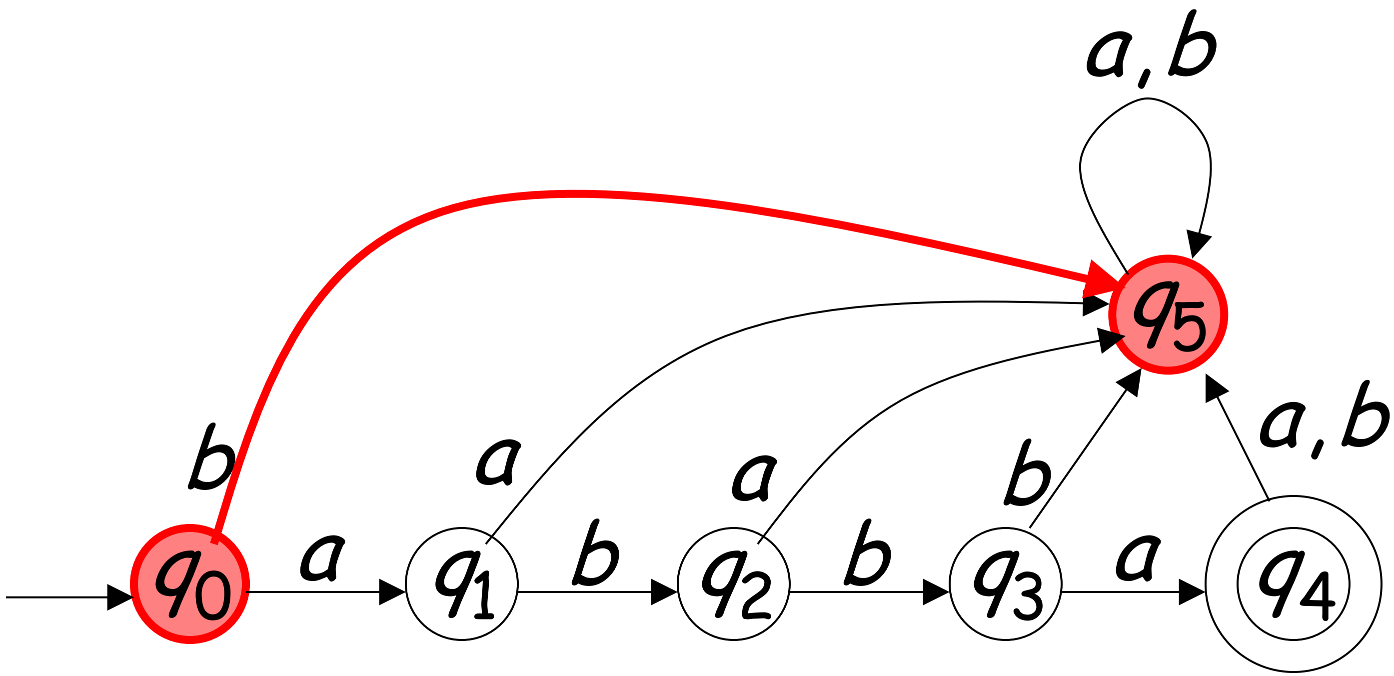
$$\delta : Q \times \Sigma \rightarrow Q$$



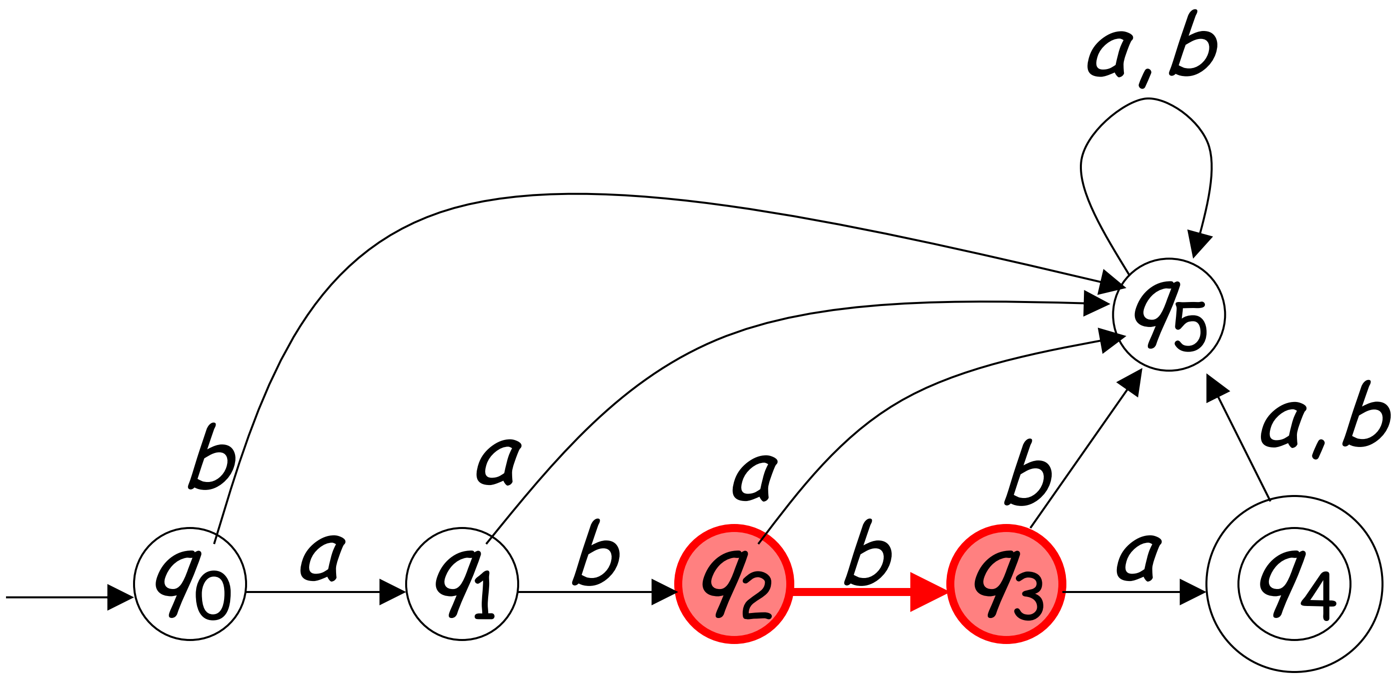
$$\delta(q_0, a) = q_1$$



$$\delta(q_0, b) = q_5$$

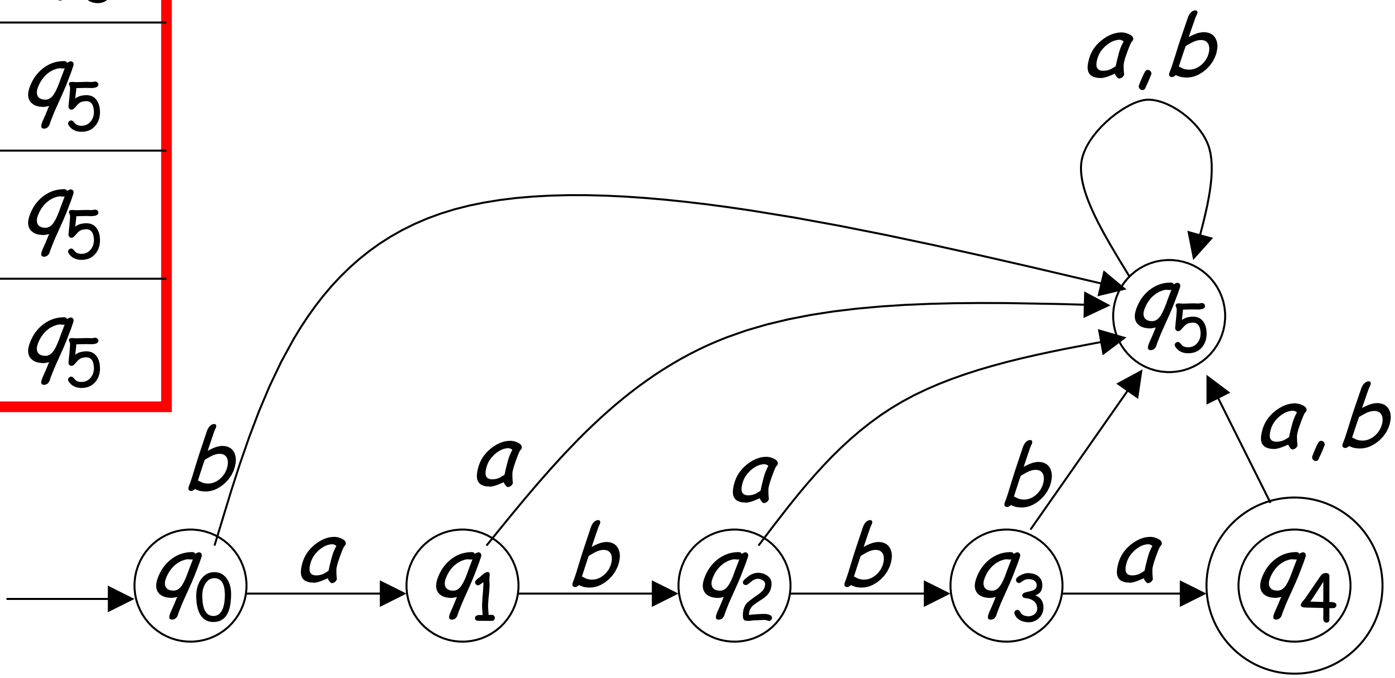


$$\delta(q_2, b) = q_3$$



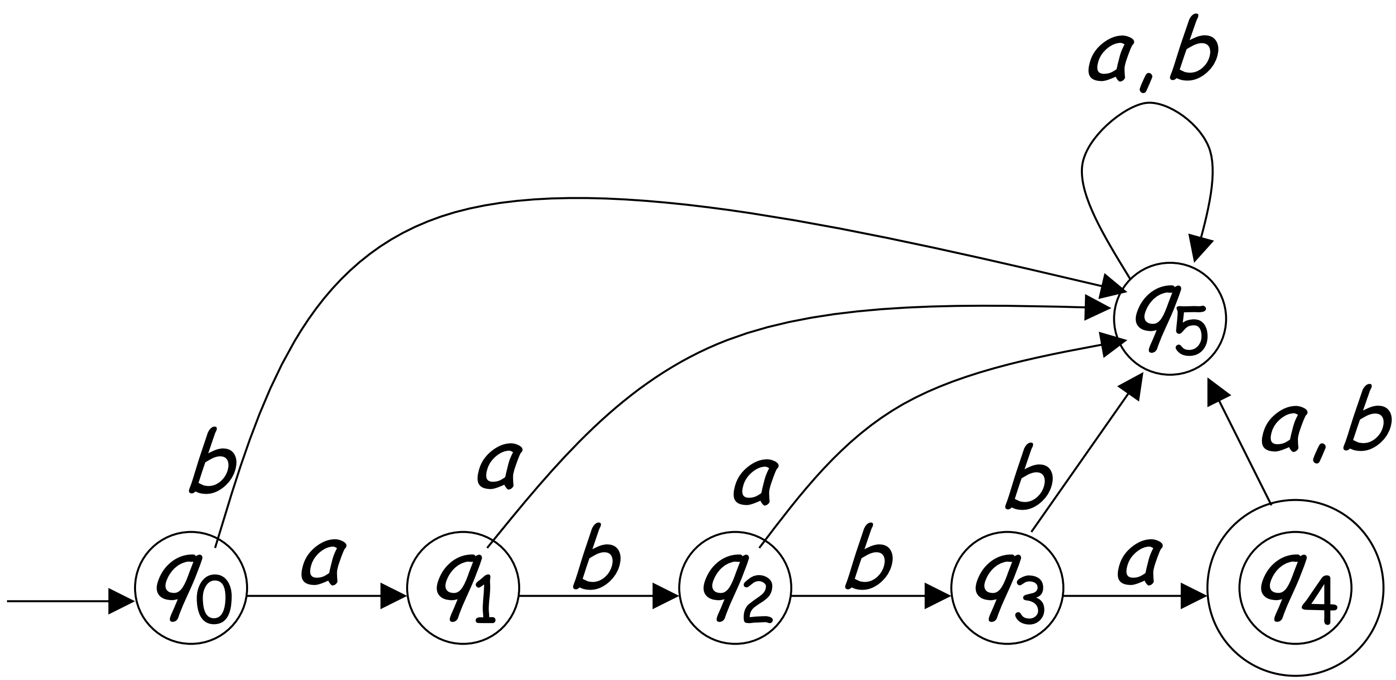
Transition Function δ

δ	a	b
q_0	q_1	q_5
q_1	q_5	q_2
q_2	q_5	q_3
q_3	q_4	q_5
q_4	q_5	q_5
q_5	q_5	q_5

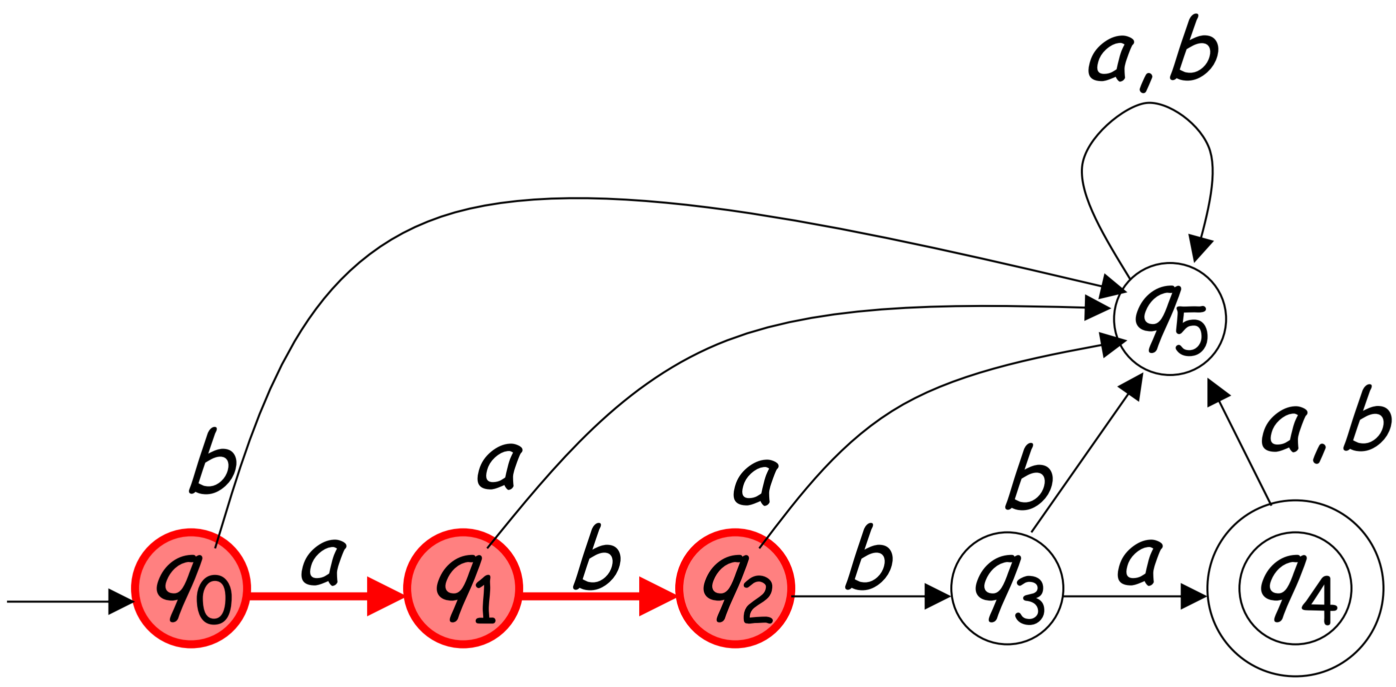


Extended Transition Function $\delta *$

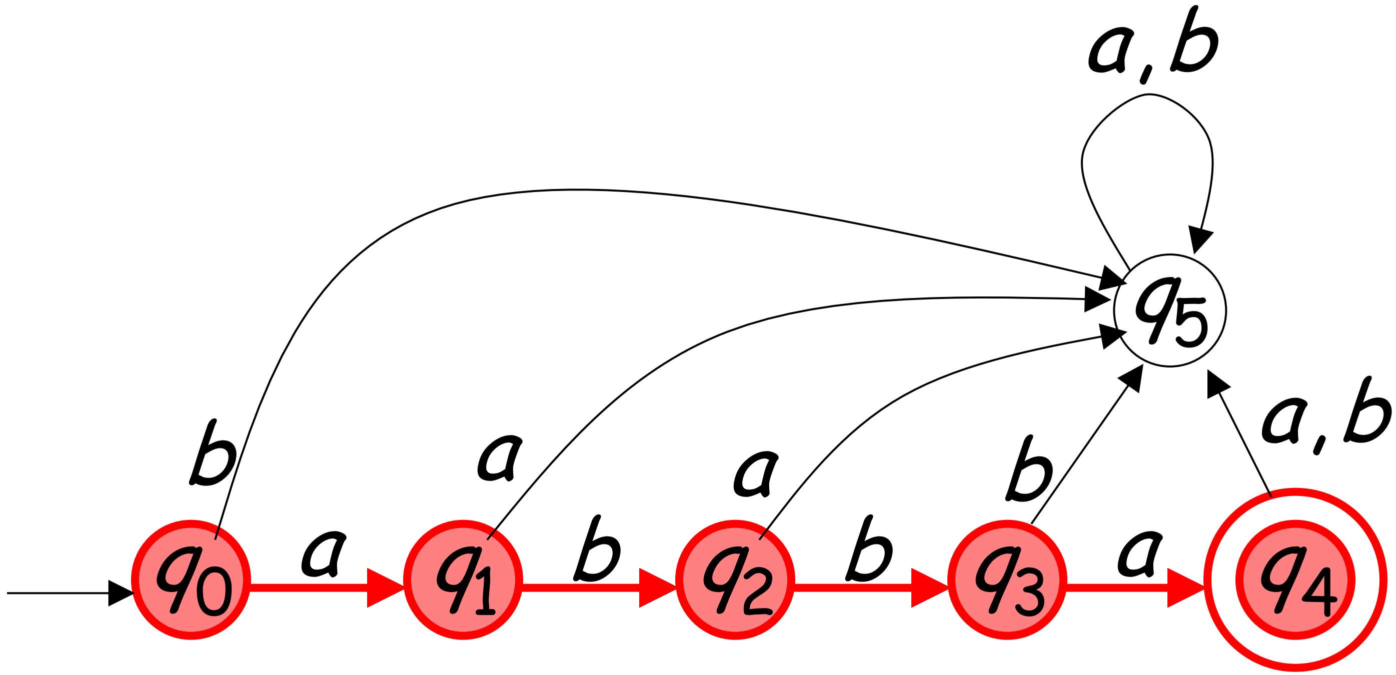
$$\delta^* : Q \times \Sigma^* \rightarrow Q$$



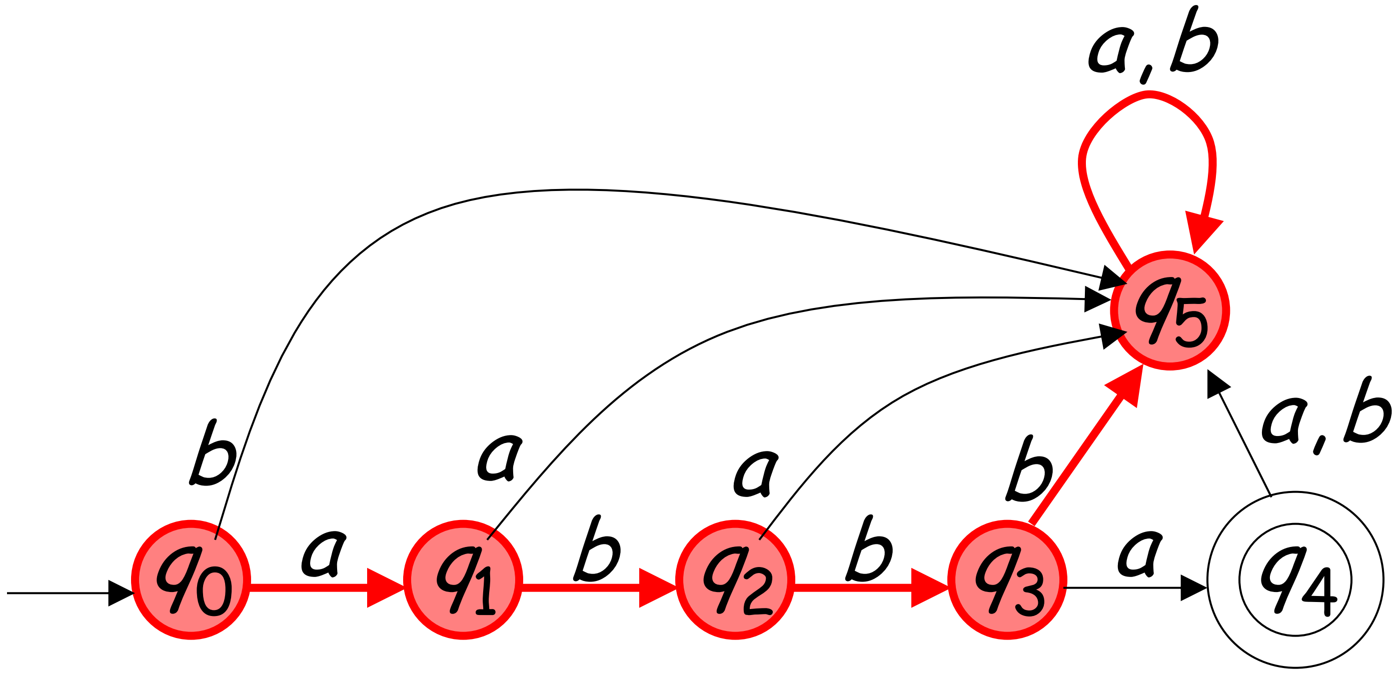
$$\delta^*(q_0, ab) = q_2$$



$$\delta^*(q_0, abba) = q_4$$

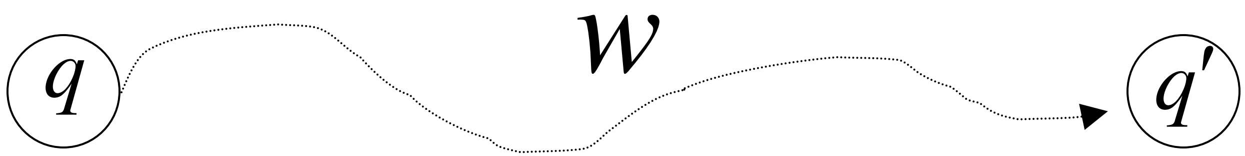


$$\delta^*(q_0, abbbaa) = q_5$$

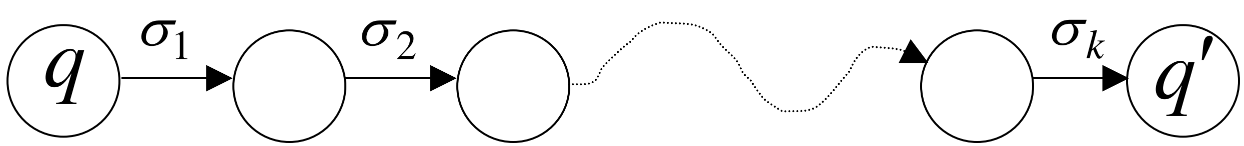


Observation: if there is a walk from q to q' with label w then

$$\delta^*(q,w) = q'$$

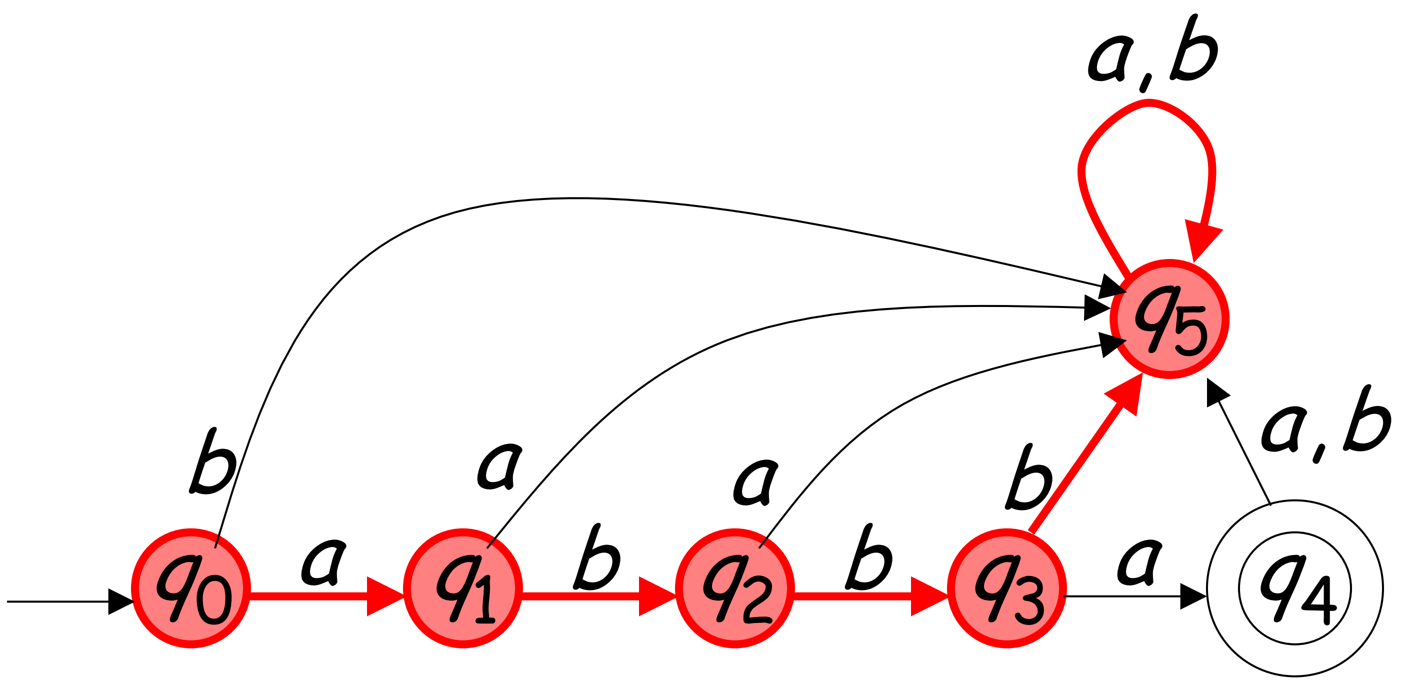


$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



Example: There is a walk from q_0 to q_5 with label $abbbaa$

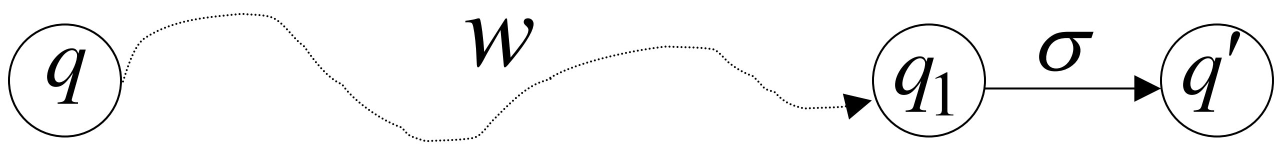
$$\delta^*(q_0, abbbaa) = q_5$$



Recursive Definition

$$\delta^*(q,\lambda)=q$$

$$\delta^*(q,w\sigma)=\delta(\delta^*(q,w),\sigma)$$



$$\left.\begin{array}{l}\delta^*(q,w\sigma)=q' \\ \\ \delta(q_1,\sigma)=q'\end{array}\right\} \Rightarrow \delta^*(q,w\sigma)=\delta(q_1,\sigma)$$

$$\left.\begin{array}{l}\delta^*(q,w\sigma)=\delta(q_1,\sigma) \\ \\ \delta^*(q,w)=q_1\end{array}\right\} \Rightarrow \delta^*(q,w\sigma)=\delta(\delta^*(q,w),\sigma)$$

$$\delta^*(q_0, ab) =$$

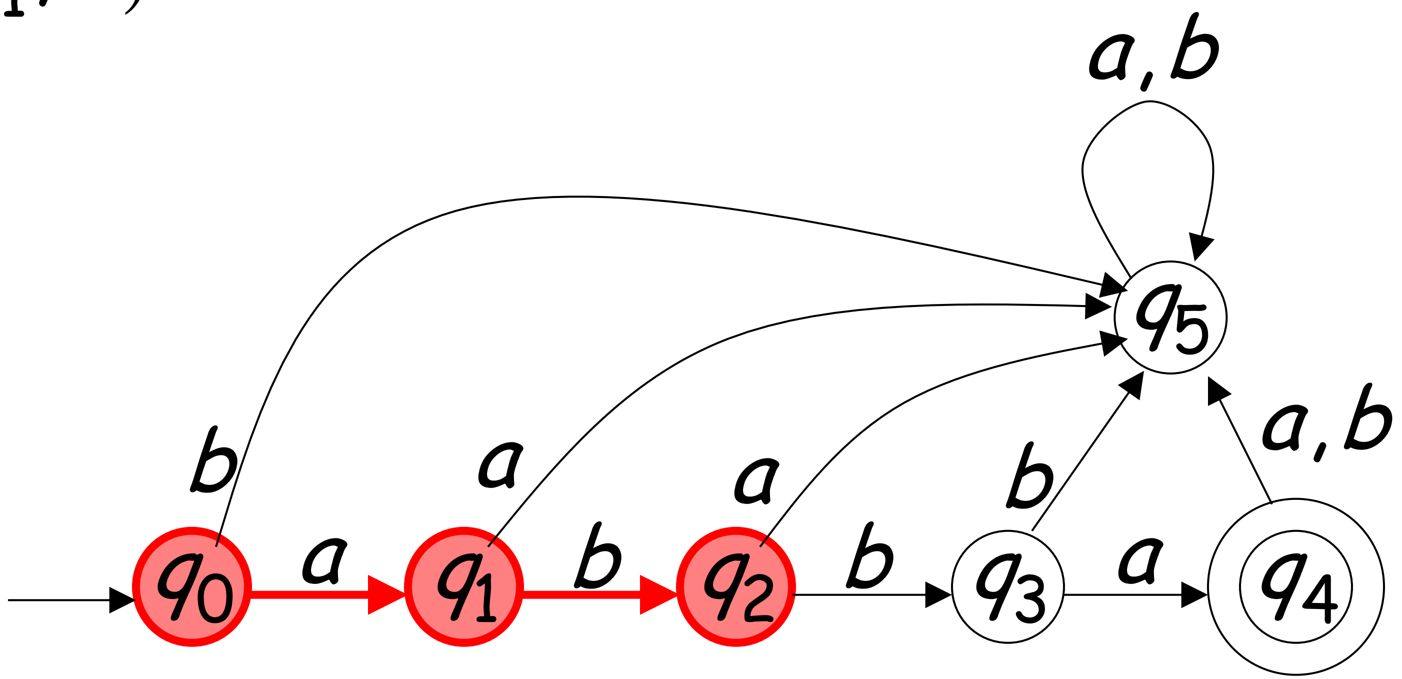
$$\delta(\delta^*(q_0, a), b) =$$

$$\delta(\delta(\delta^*(q_0, \lambda), a), b) =$$

$$\delta(\delta(q_0, a), b) =$$

$$\delta(q_1, b) =$$

$$q_2$$

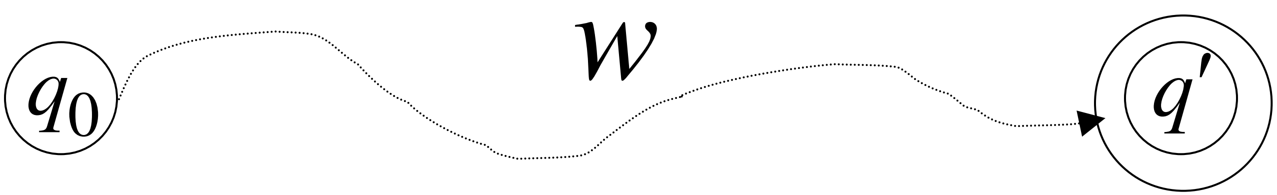


Language Accepted by FAs

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

Language accepted by M :

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

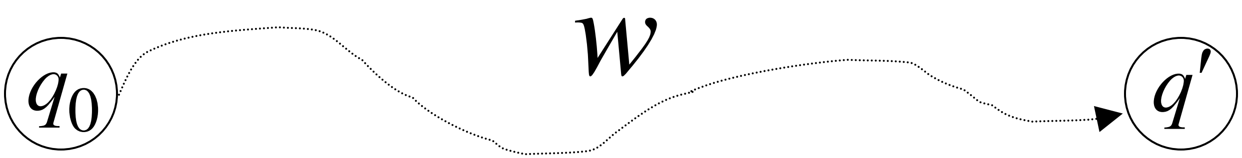


$q' \in F$

Observation

Language rejected by M :

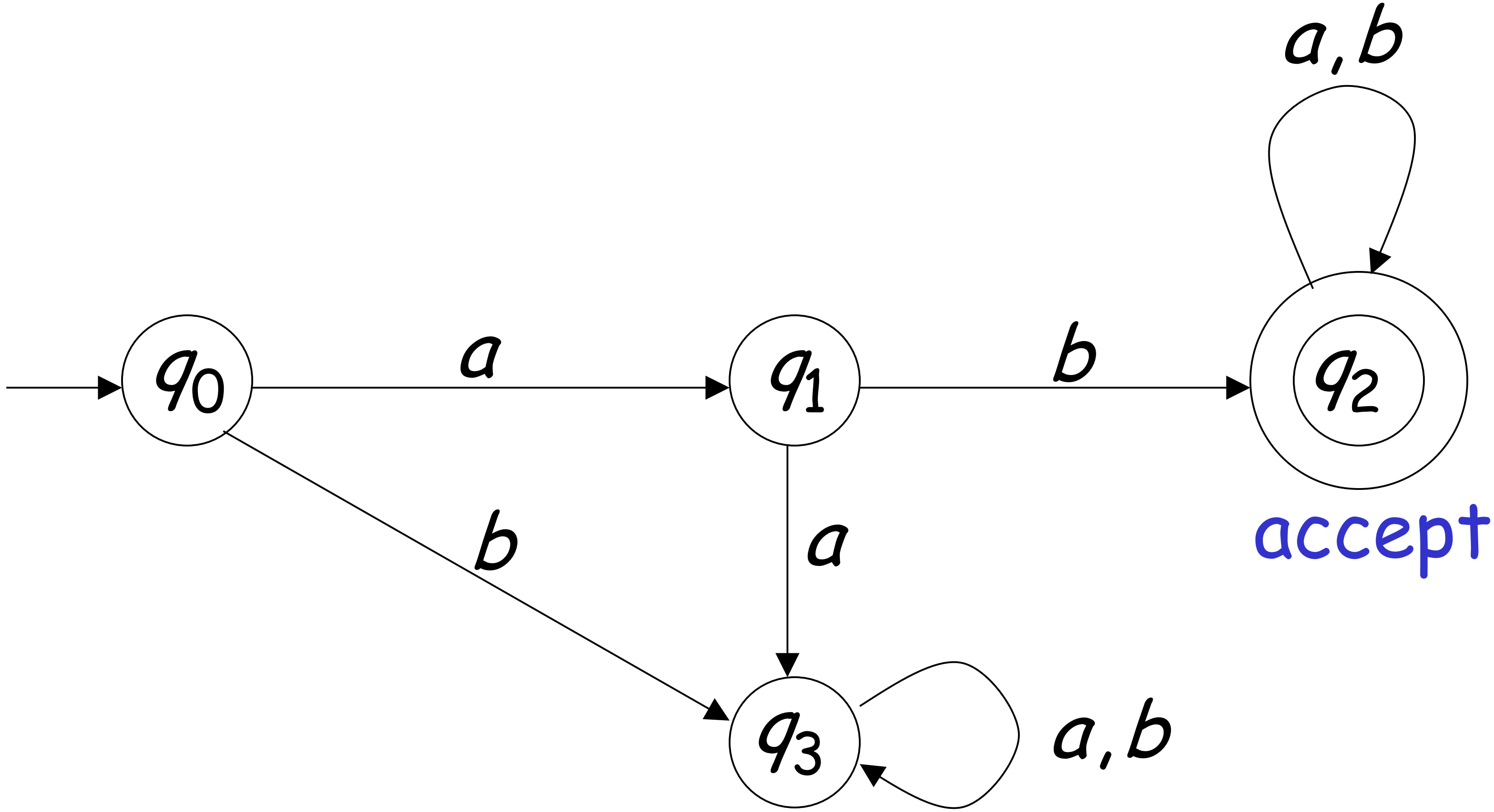
$$\overline{L(M)} = \{w \in \Sigma^* : \delta^*(q_0, w) \notin F\}$$



$q' \notin F$

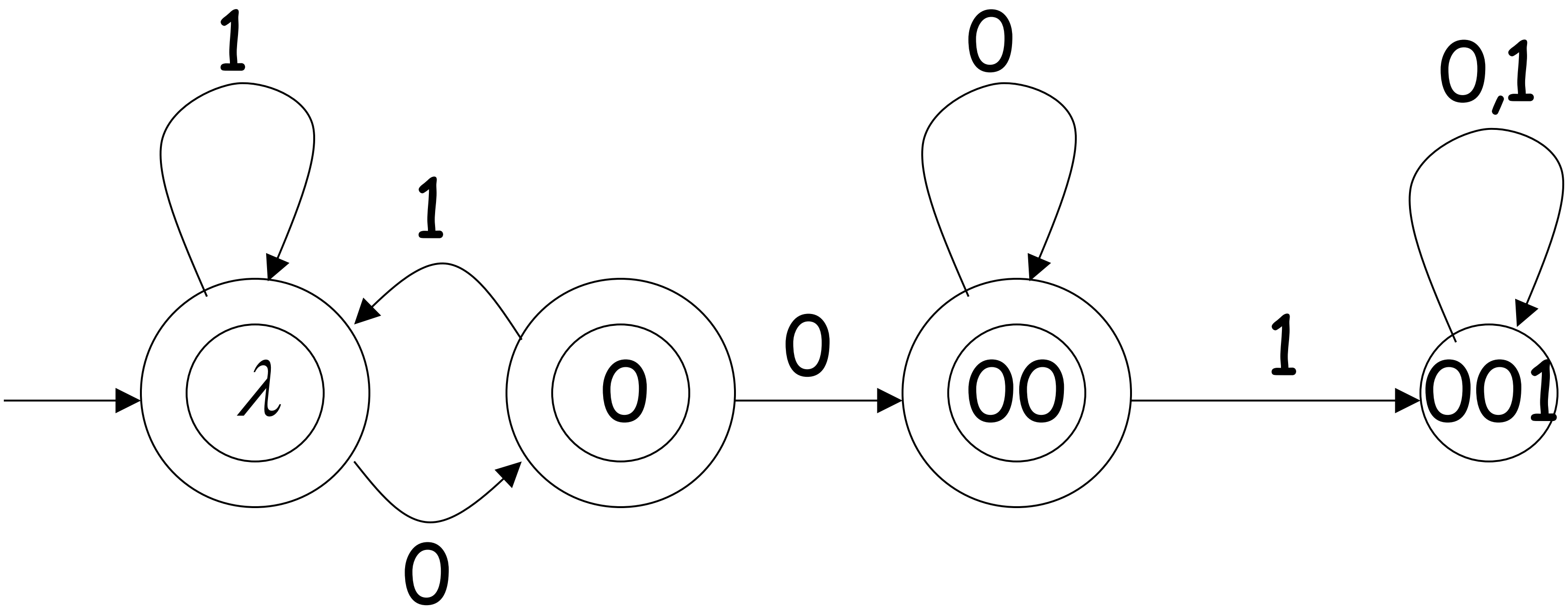
Example

$L(M) = \{$ all strings with prefix ab $\}$



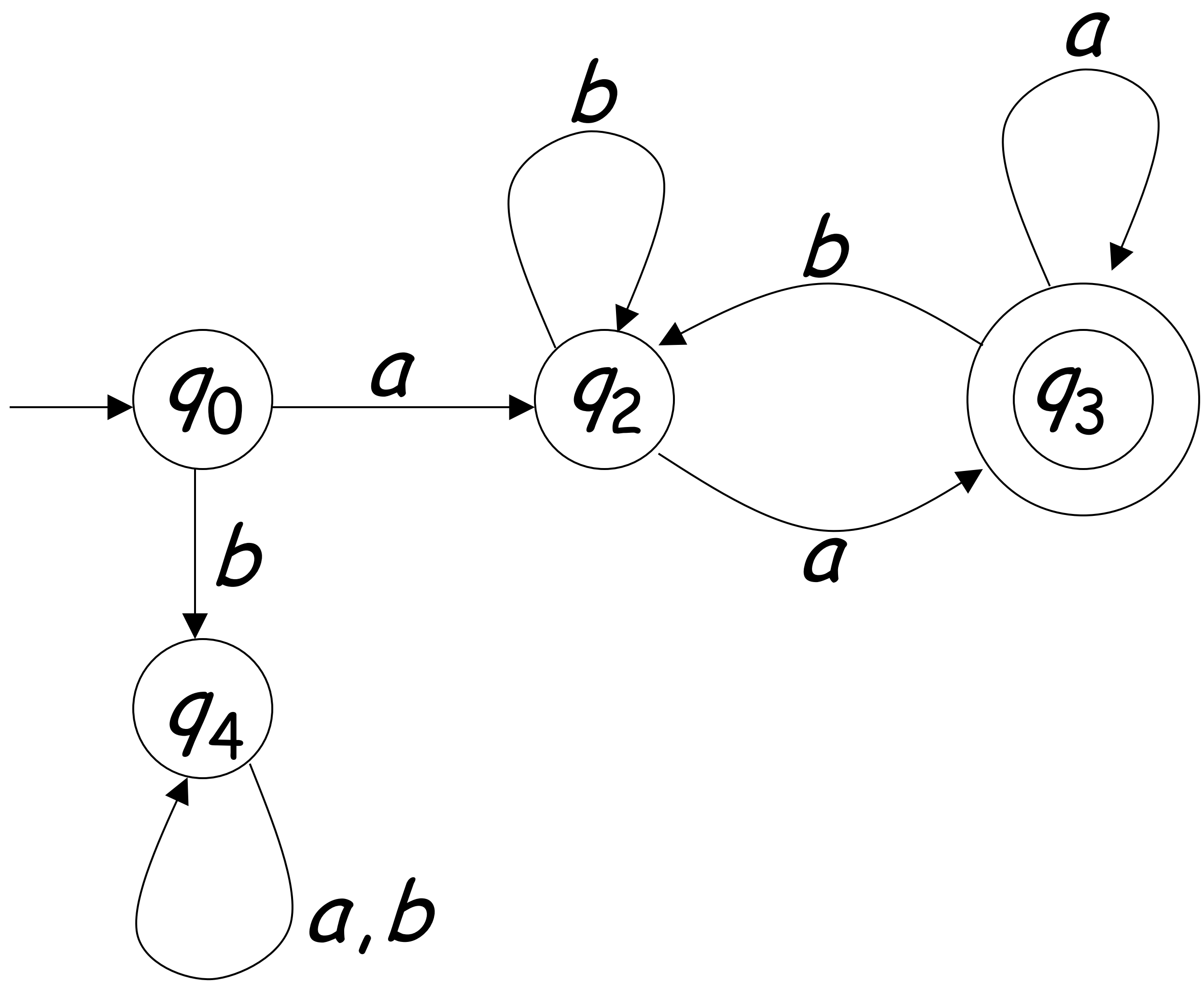
Example

$L(M)$ = { all strings without substring 001 }



Example

$$L(M) = \{awa : w \in \{a,b\}^*\}$$



Regular Languages

Definition:

A language L is regular if there is FA M such that $L = L(M)$

Observation:

All languages accepted by FAs form the family of regular languages

Examples of regular languages:

$\{abba\}$ $\{\lambda, ab, abba\}$

$\{awa : w \in \{a,b\}^*\}$ $\{a^n b : n \geq 0\}$

{ all strings with prefix ab }

{ all strings without substring 001 }

There exist automata that accept these Languages (see previous slides).

There exist languages which are <u>not</u> Regular:

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

There is no FA that accepts such a language

(we will prove this later in the class)

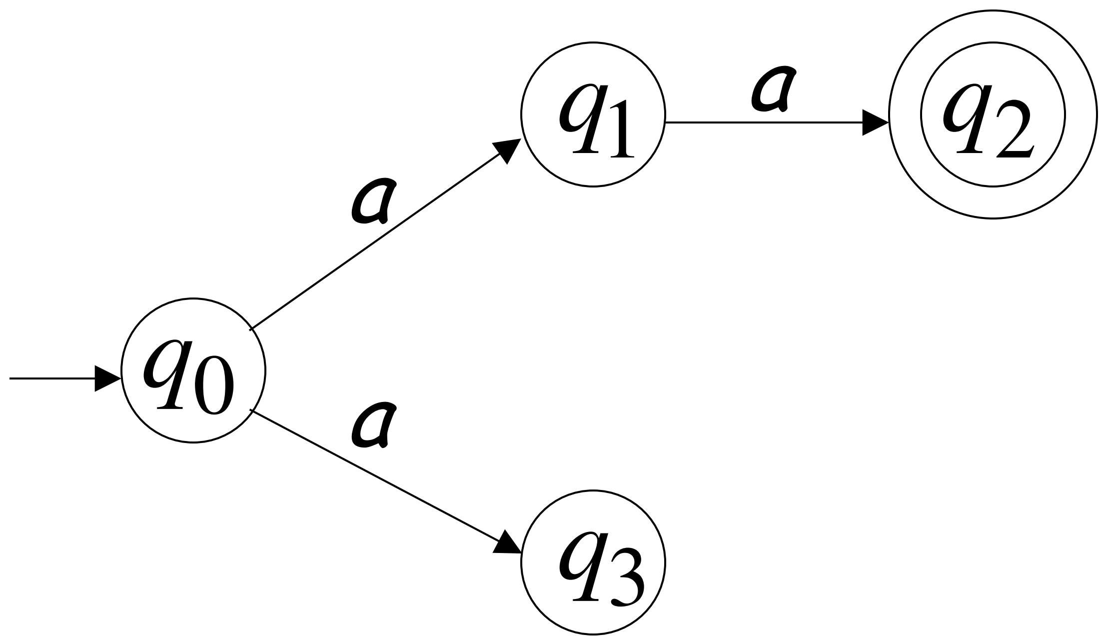
14B11CI171

Theory of Computation

Non-Deterministic Finite Automata

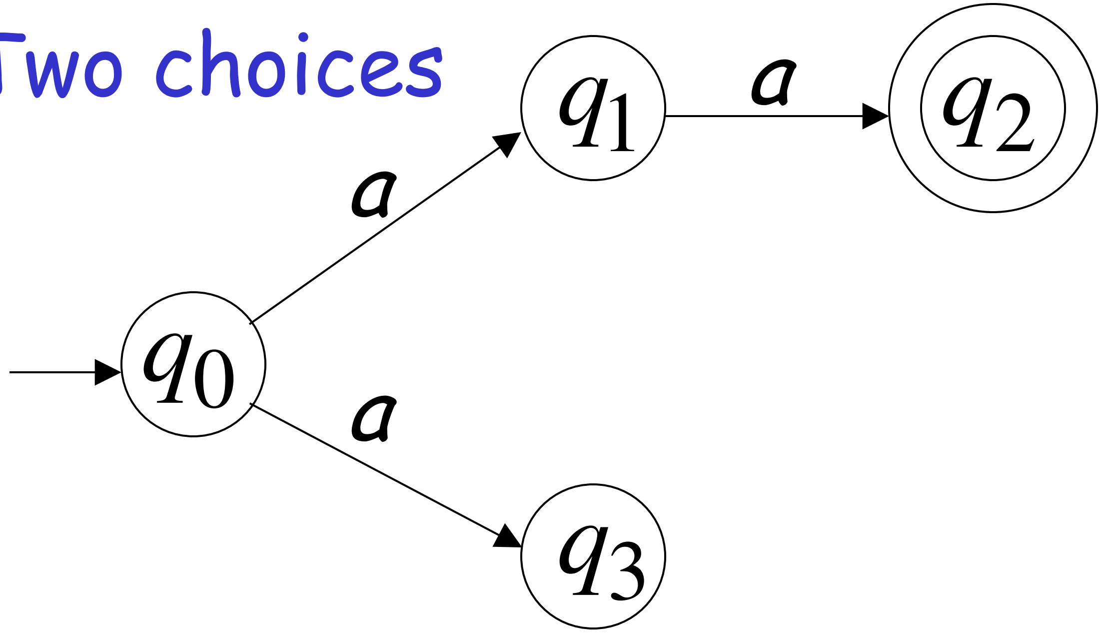
Nondeterministic Finite Automaton (NFA)

Alphabet = $\{a\}$

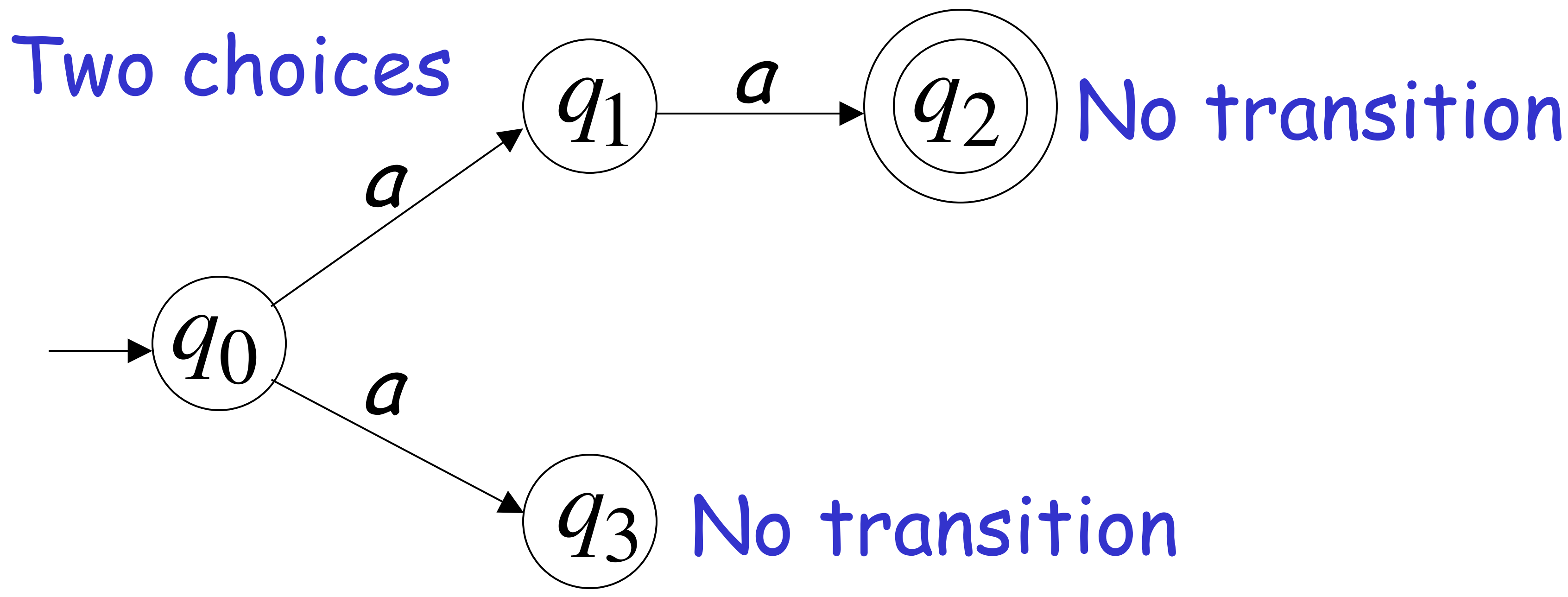


Alphabet = $\{a\}$

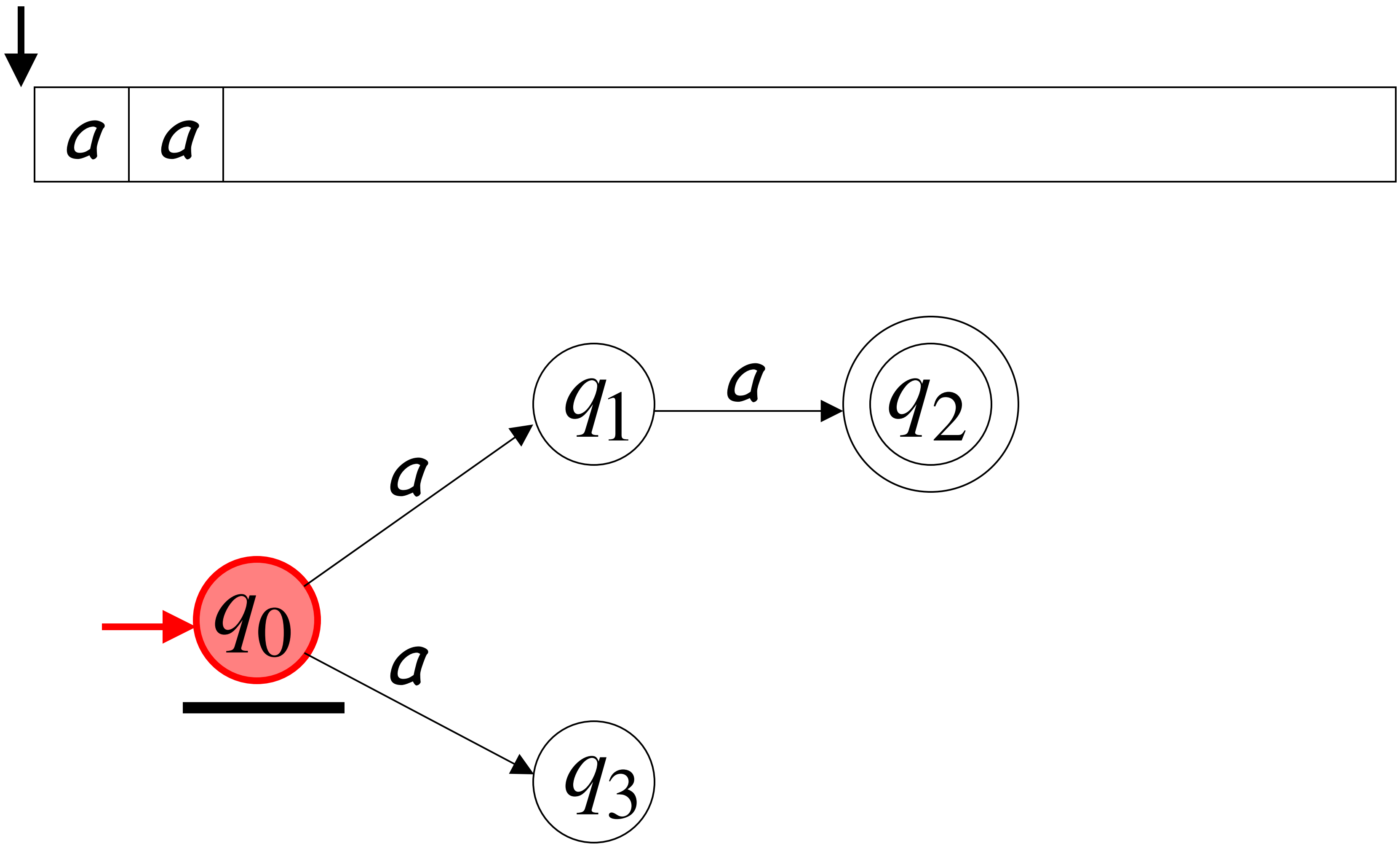
Two choices



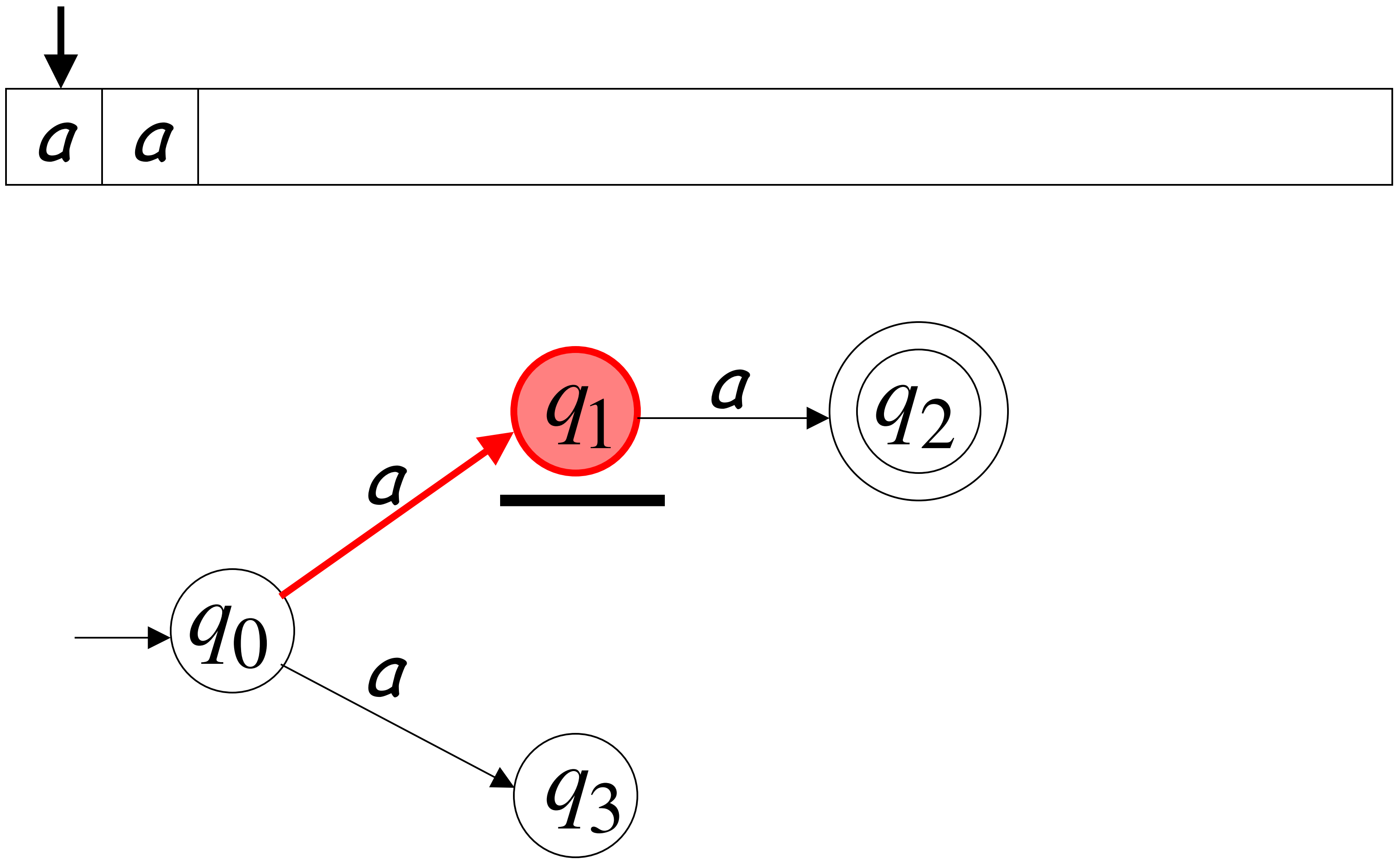
Alphabet = $\{a\}$



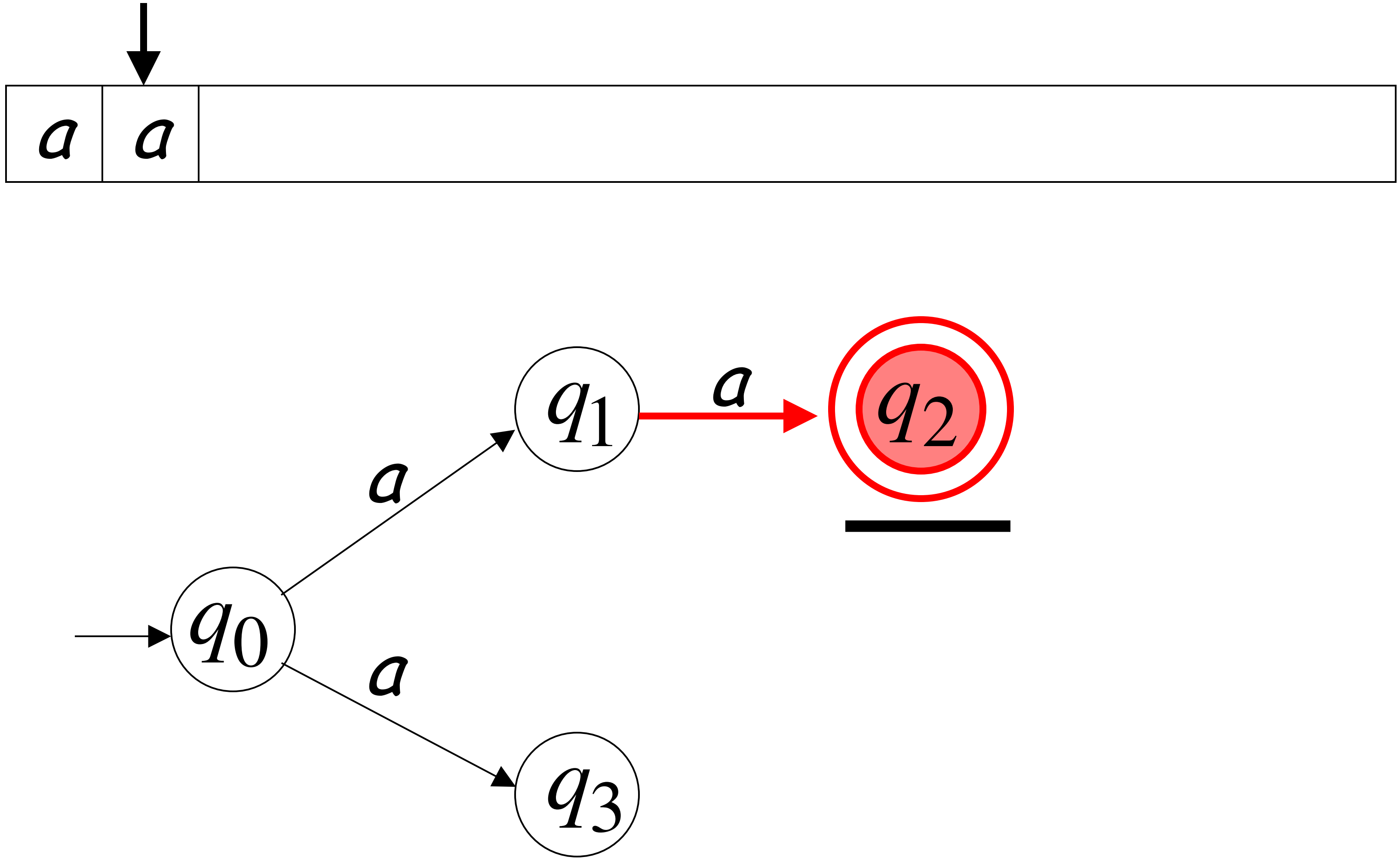
First Choice



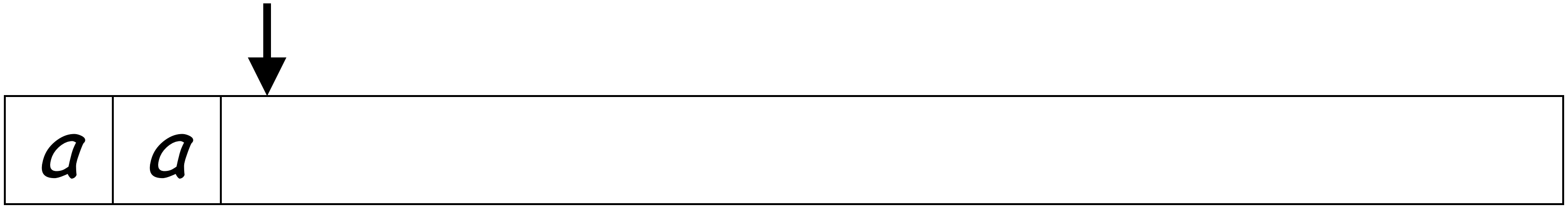
First Choice



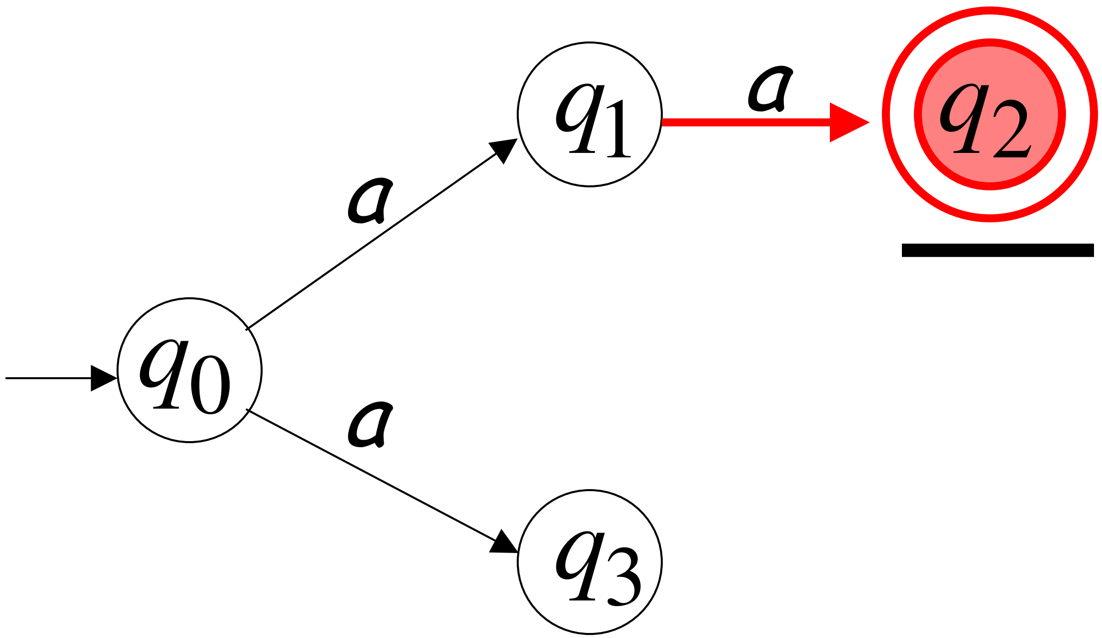
First Choice



First Choice

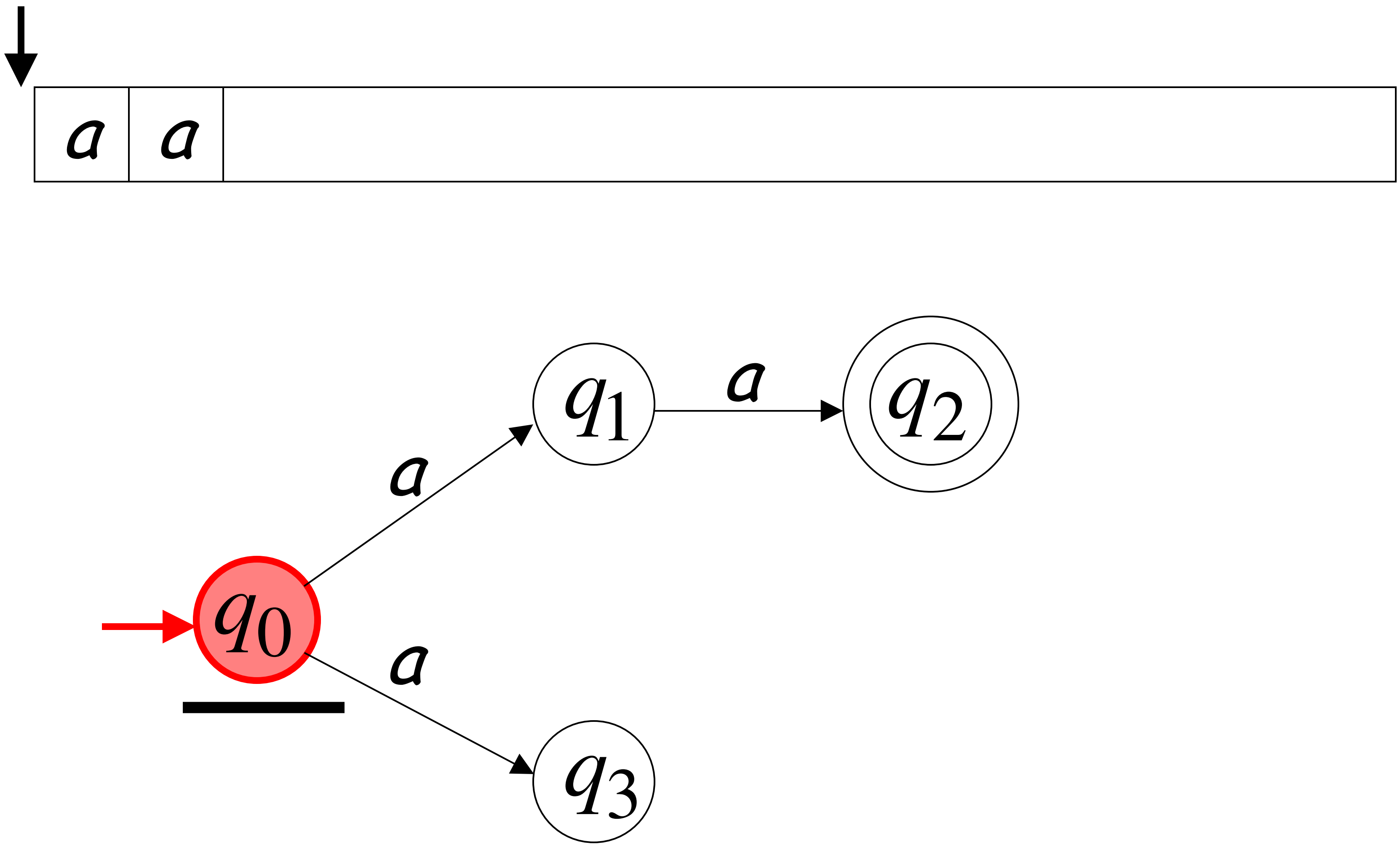


All input is consumed

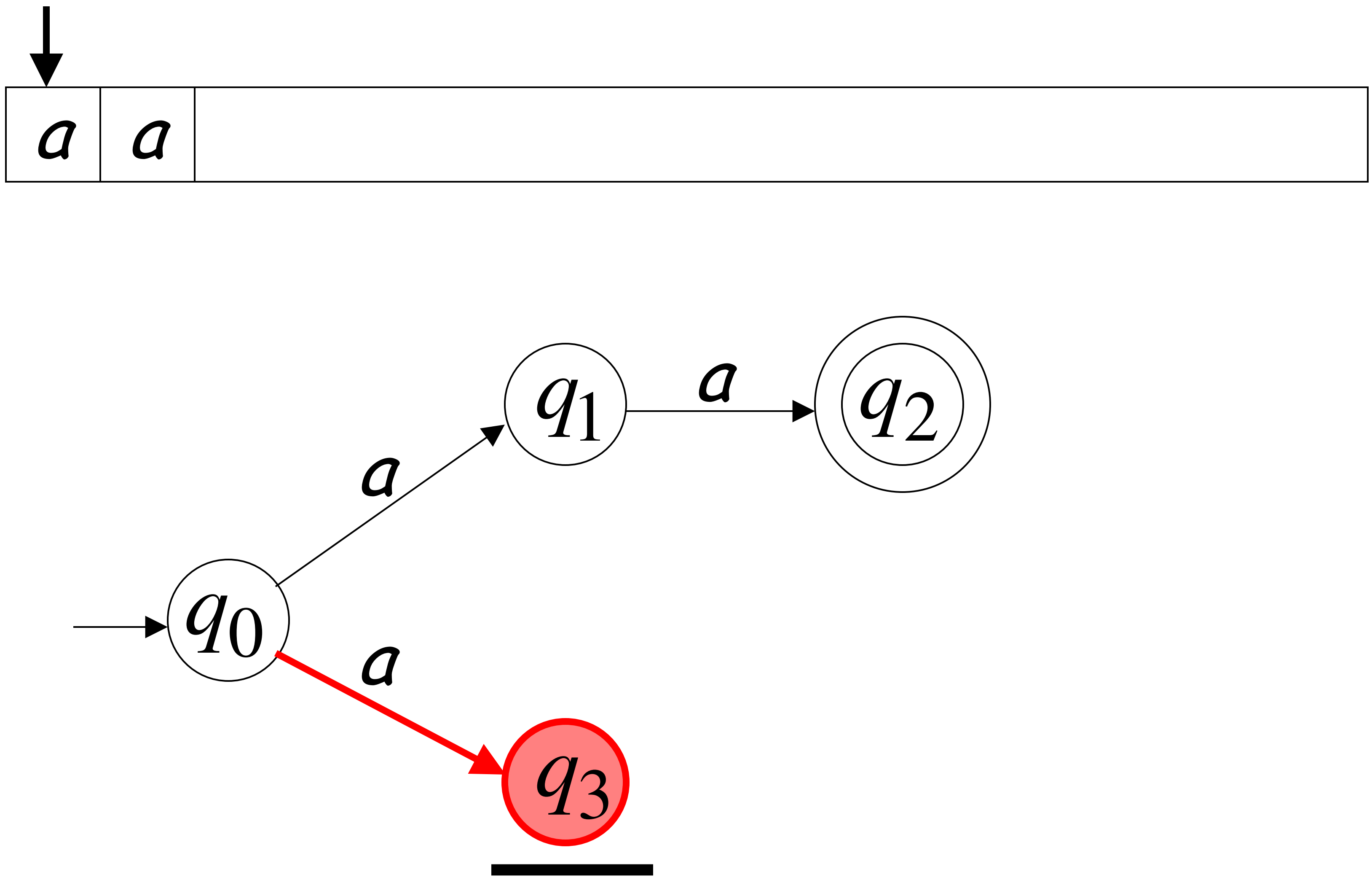


"accept"

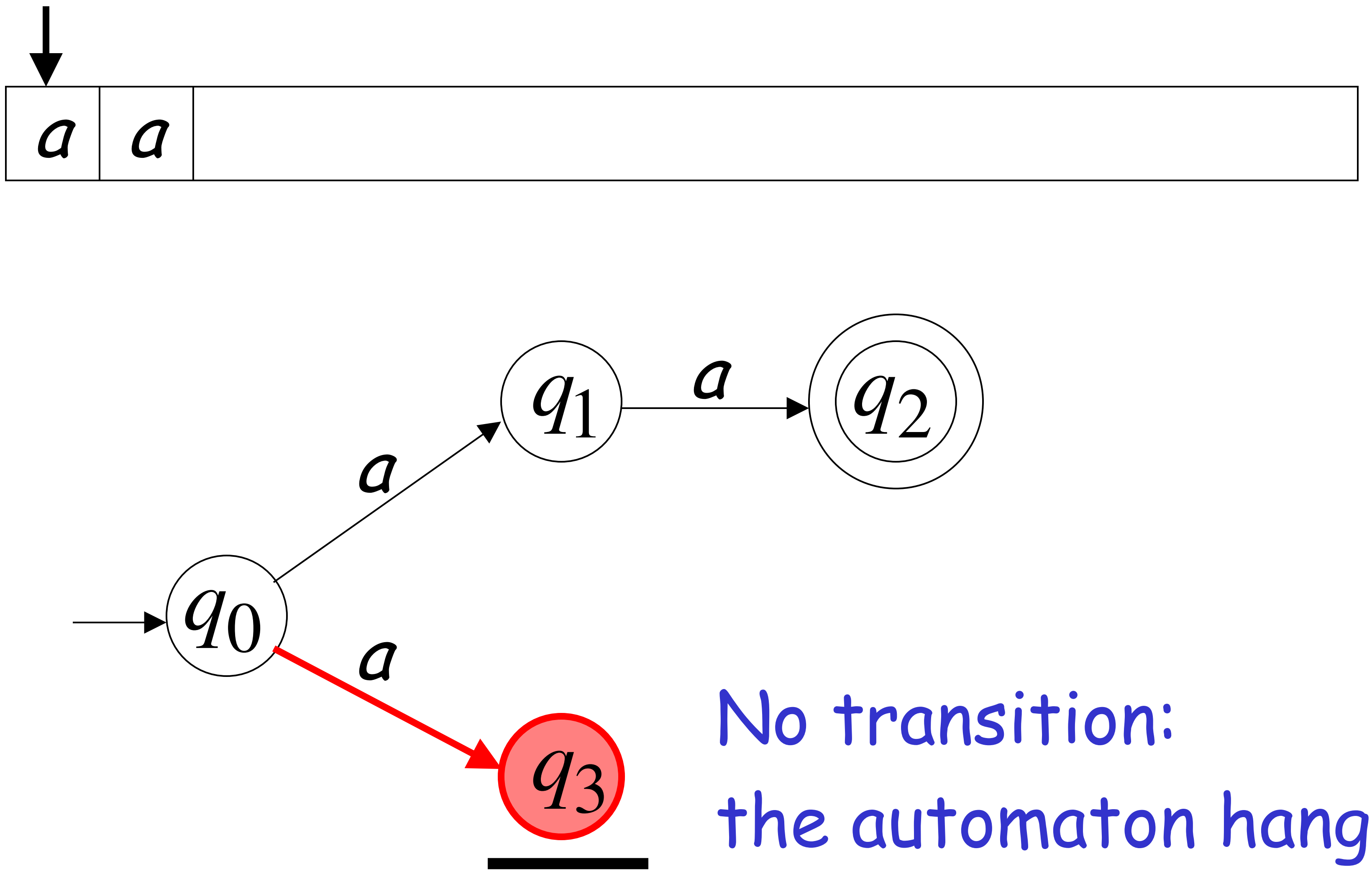
Second Choice



Second Choice

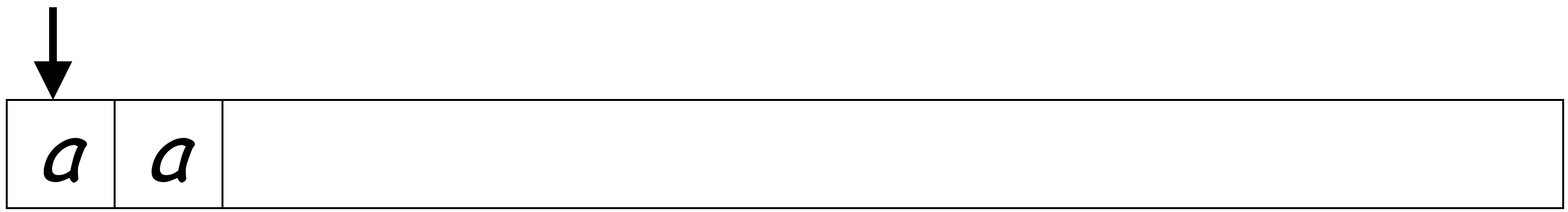


Second Choice

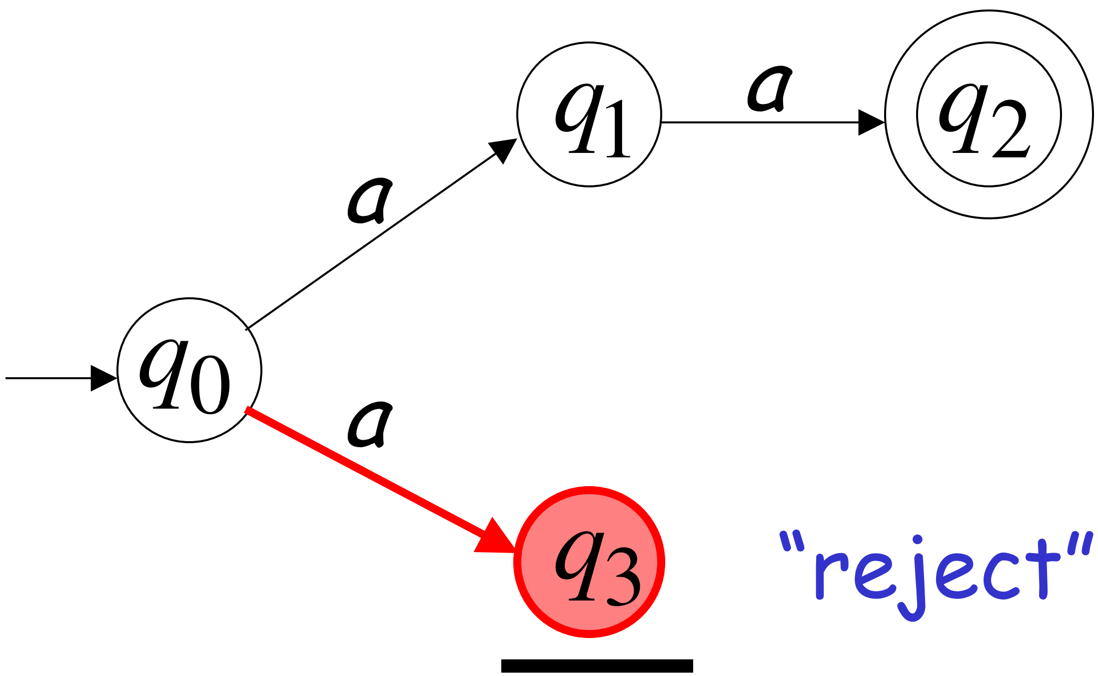


No transition:
the automaton hangs

Second Choice



Input cannot be consumed



"reject"

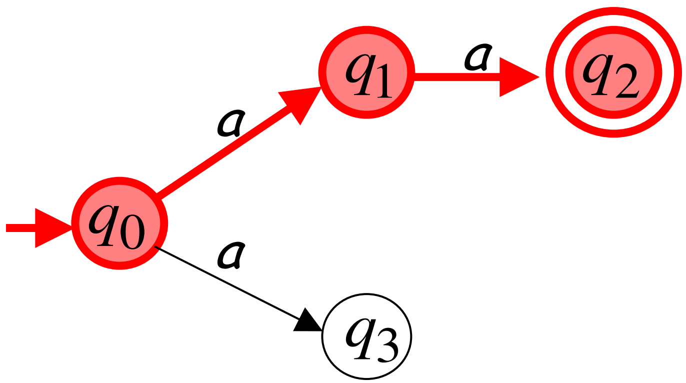
An NFA accepts a string:
when there is a computation of the NFA
that accepts the string

There is a computation:
all the input is consumed and the automaton
is in an accepting state

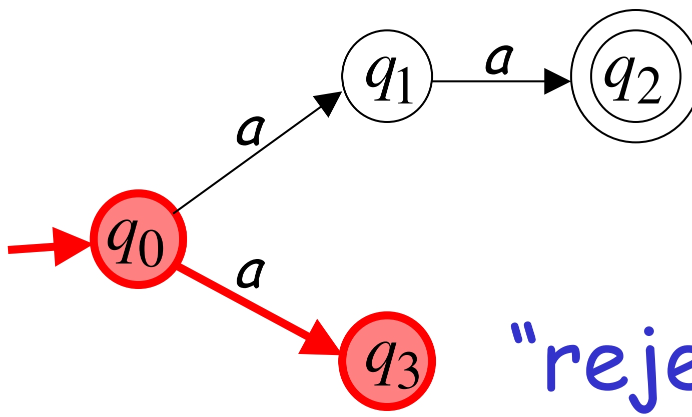
Example

aa is accepted by the NFA:

"accept"

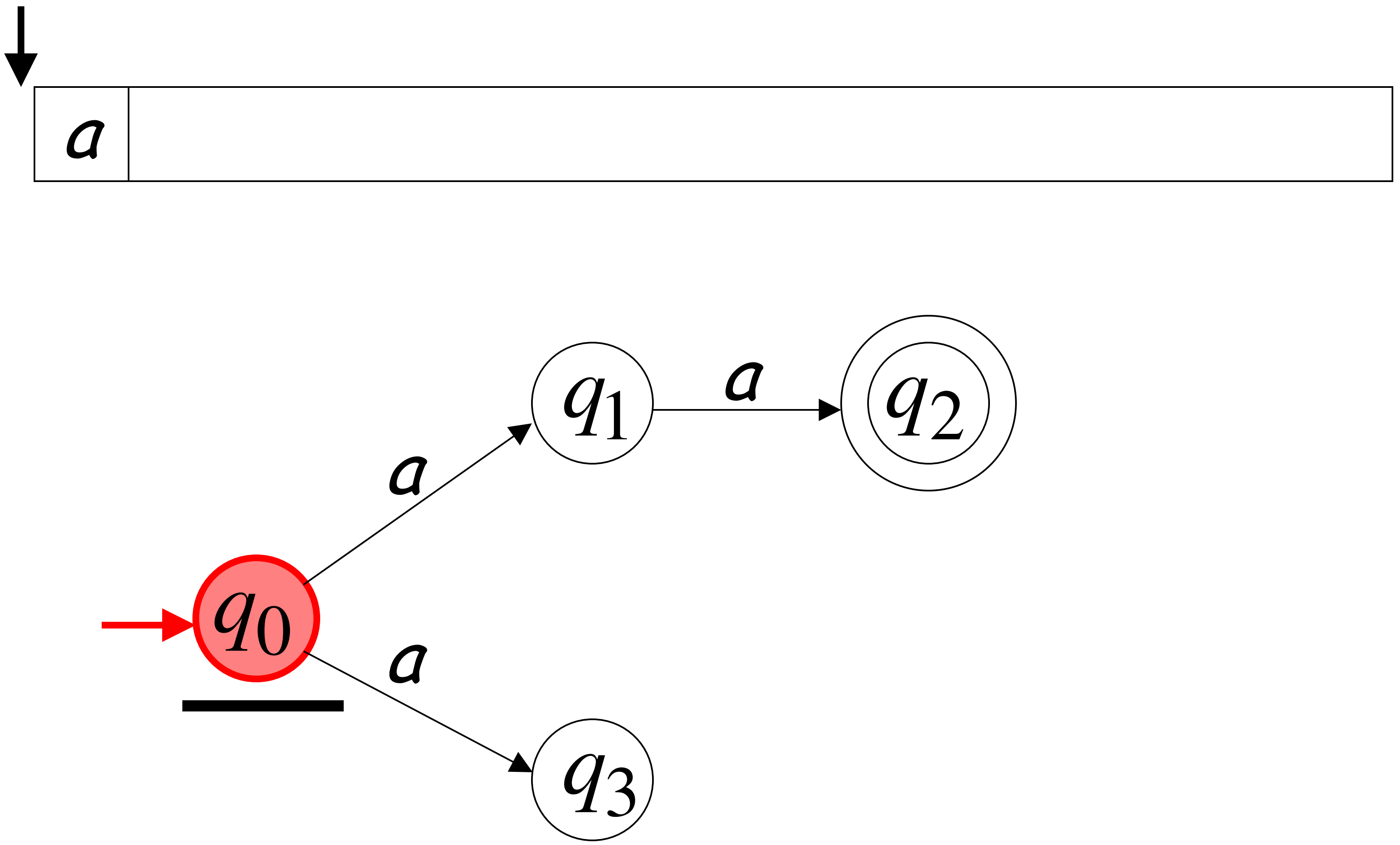


because this computation accepts aa

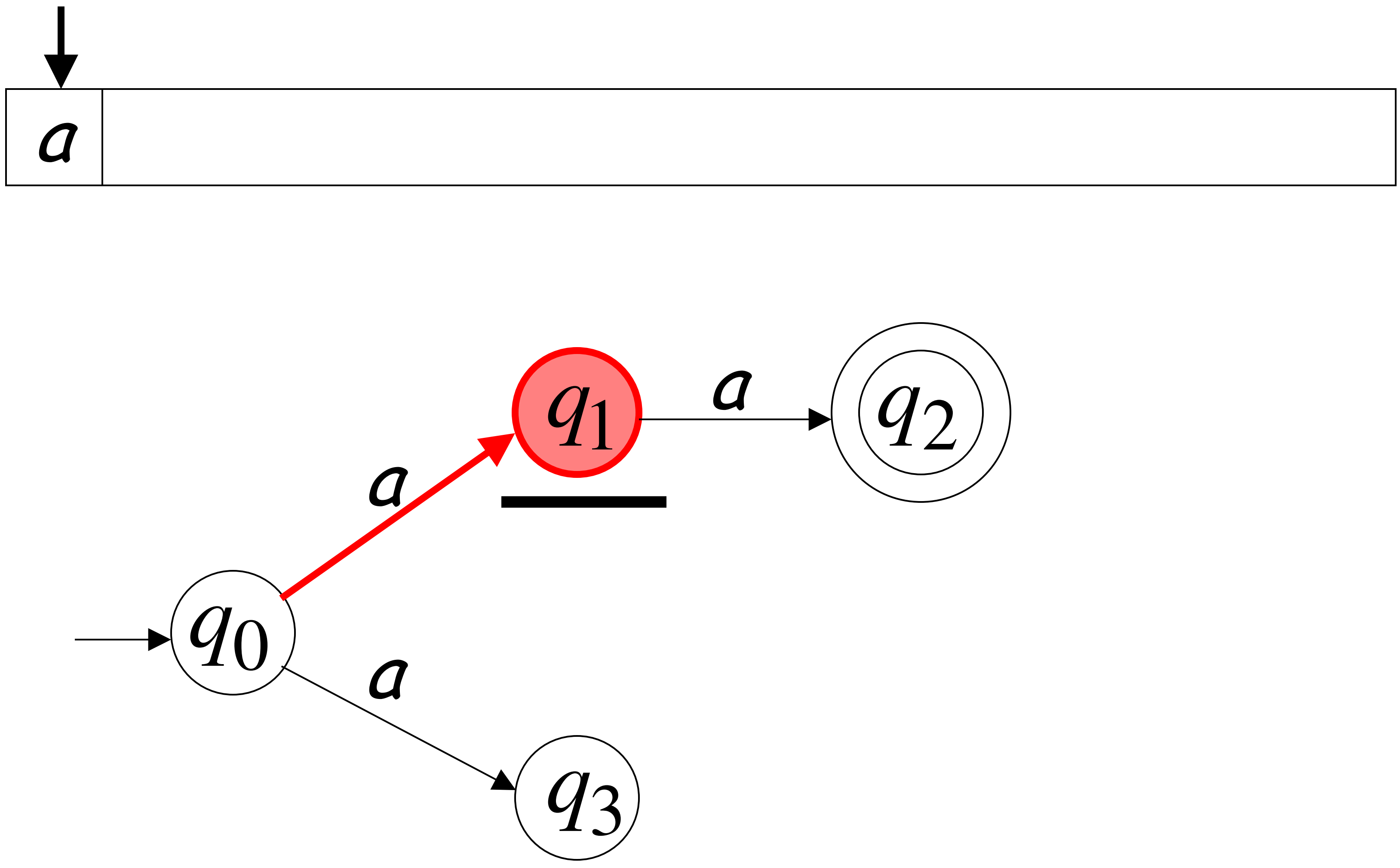


"reject"

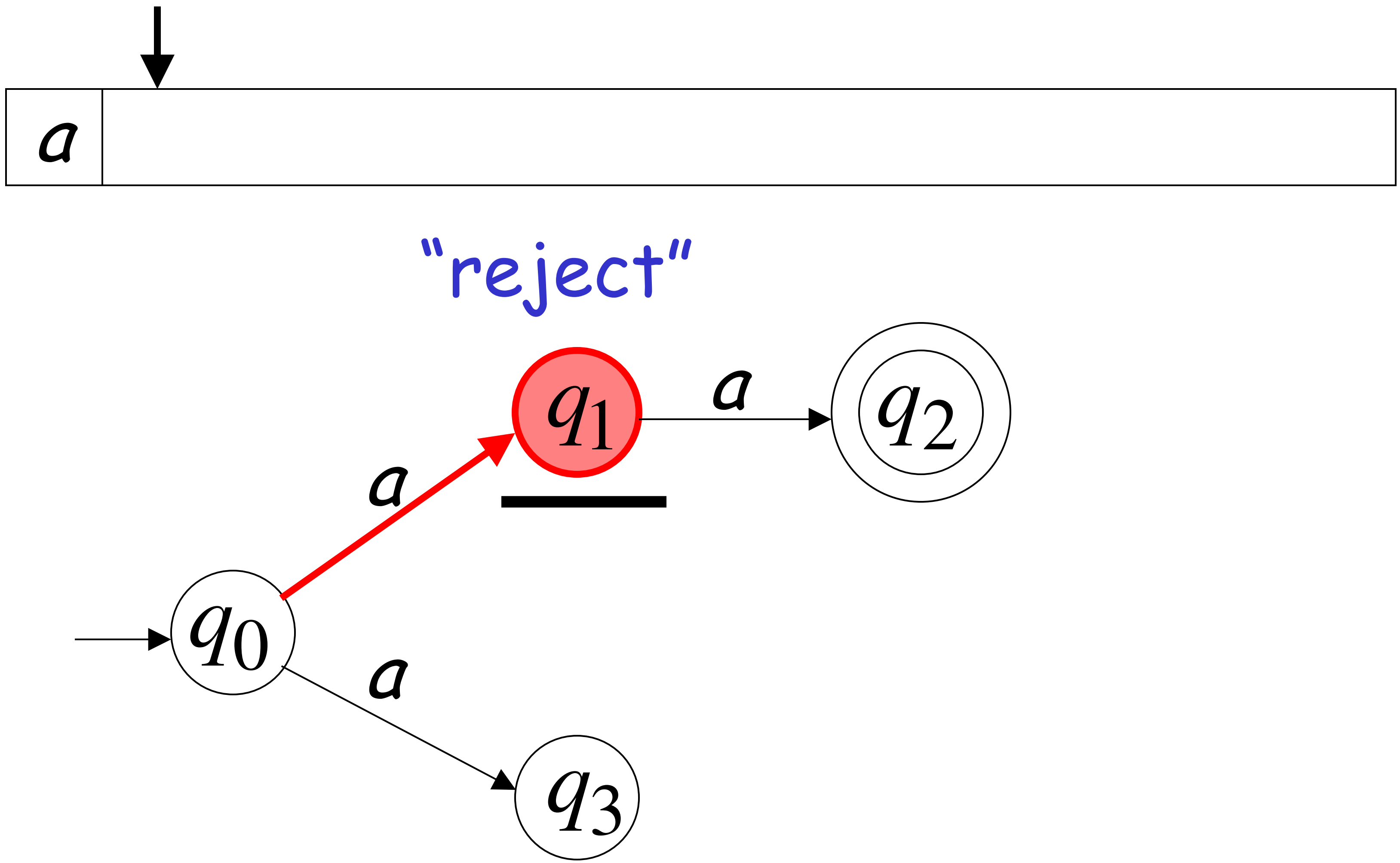
Rejection example



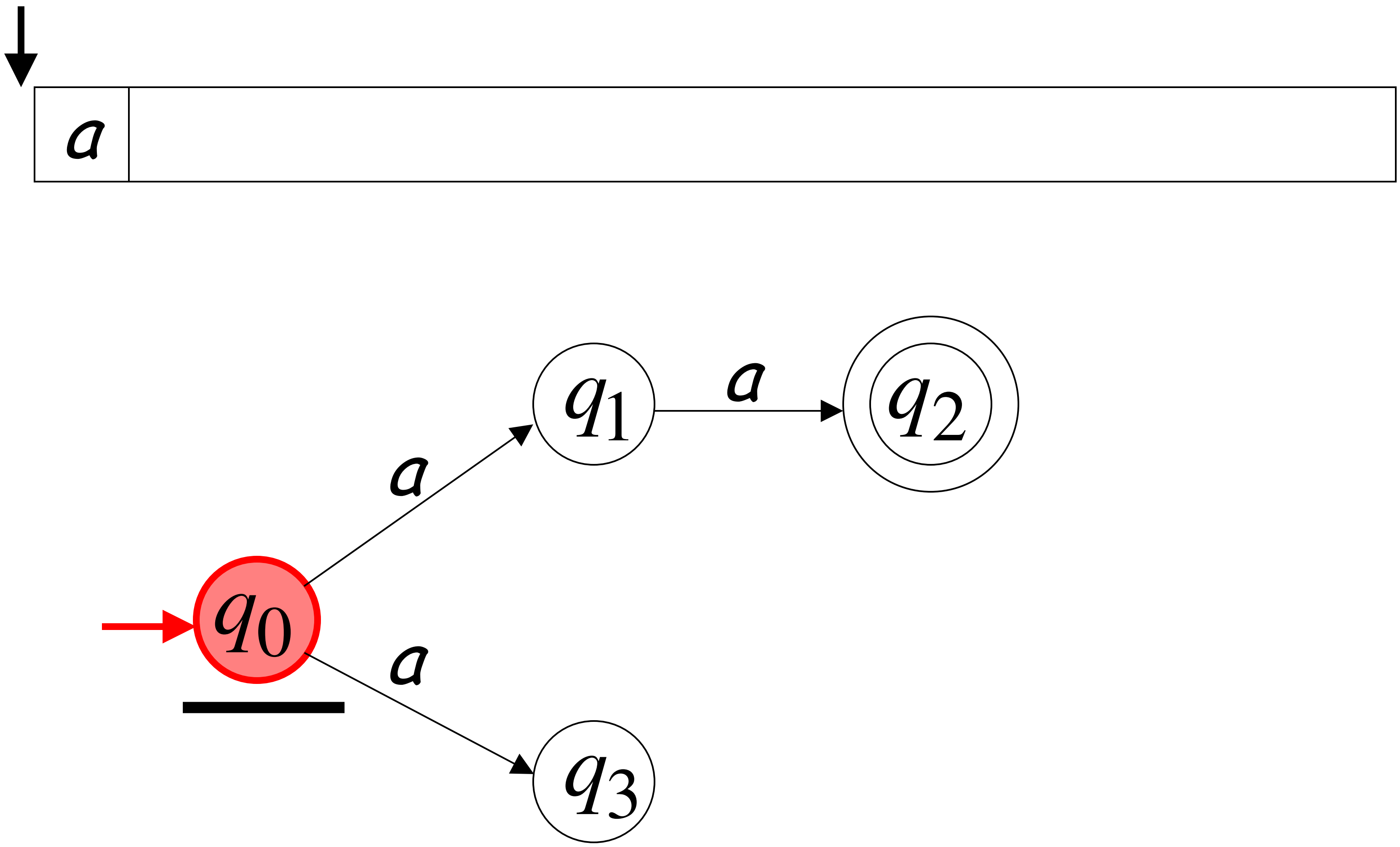
First Choice



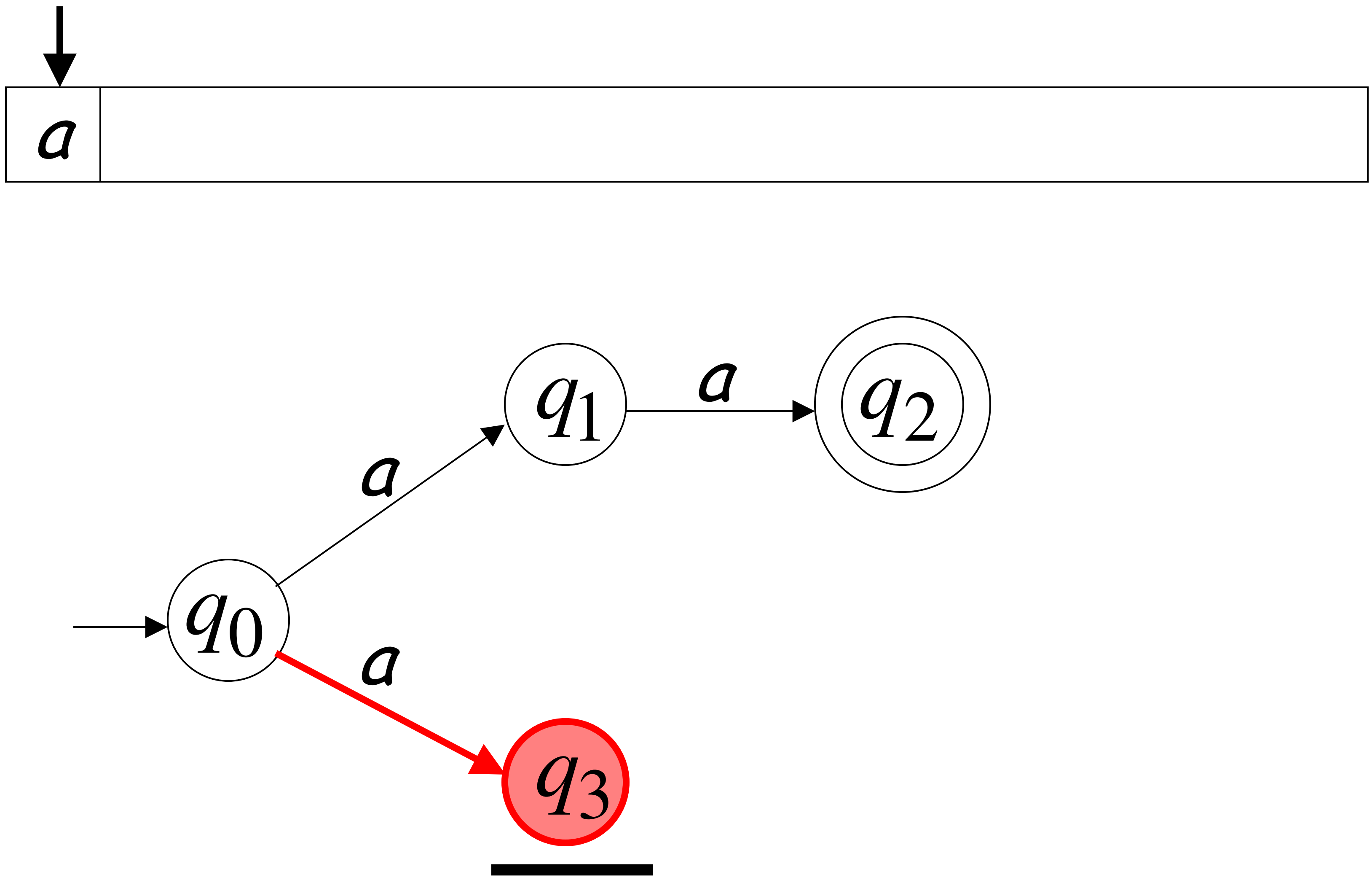
First Choice



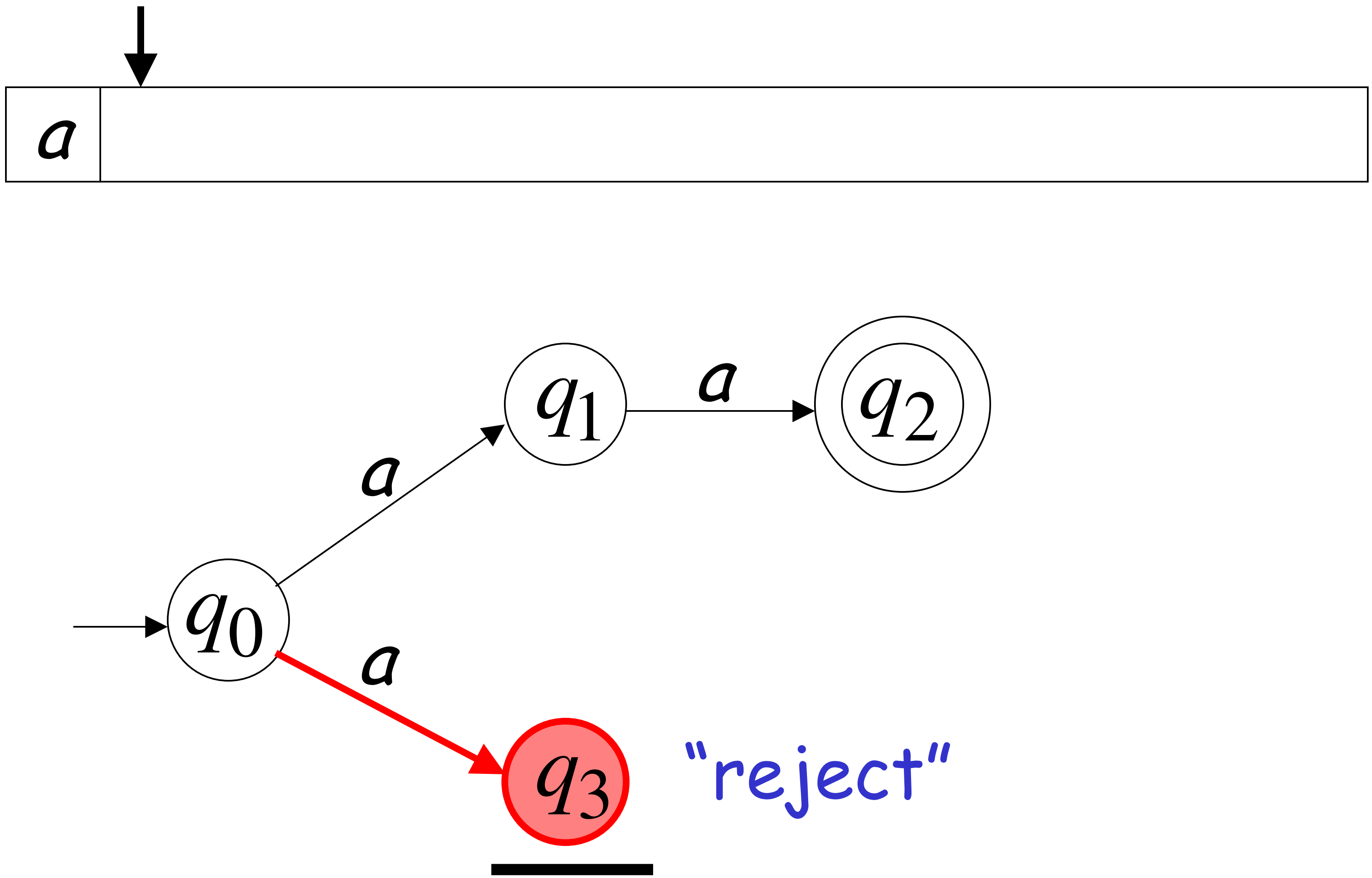
Second Choice



Second Choice



Second Choice



An NFA rejects a string:

when there is no computation of the NFA that accepts the string.

For each computation:

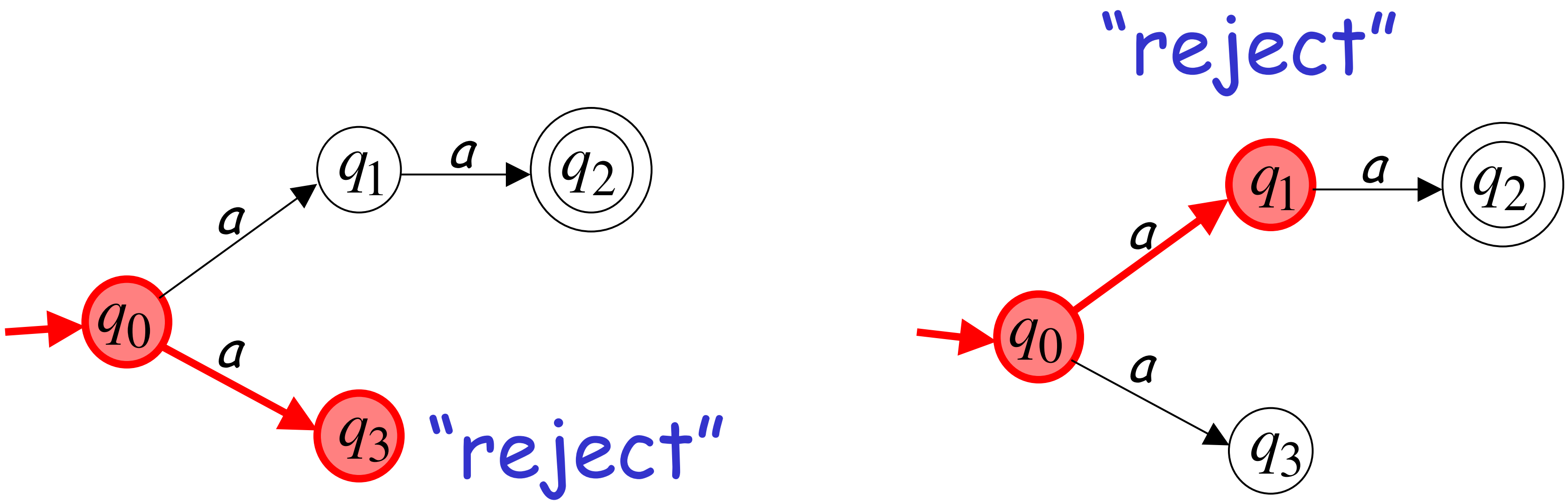
- All the input is consumed and the automaton is in a non final state

OR

- The input cannot be consumed

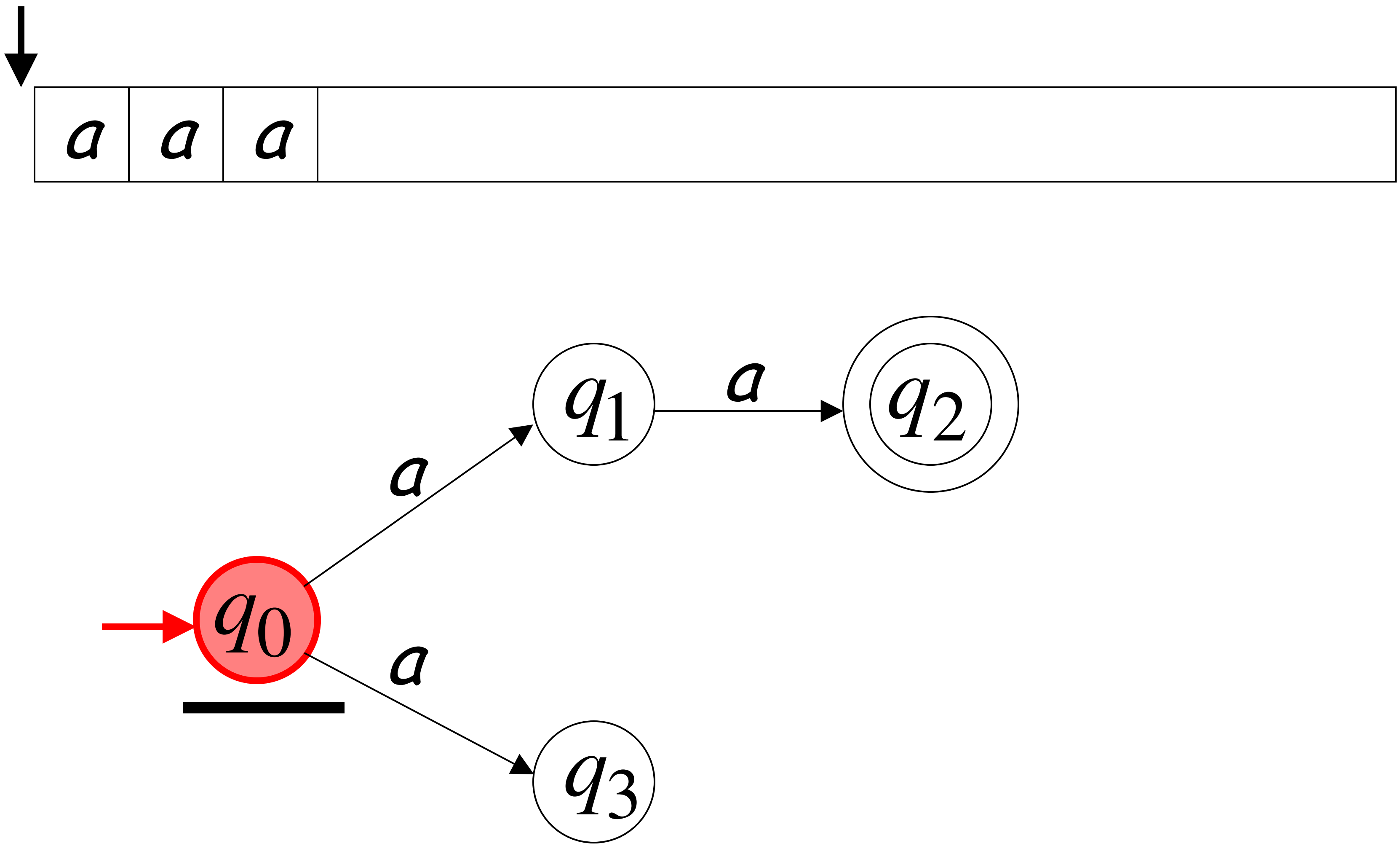
Example

a is rejected by the NFA:

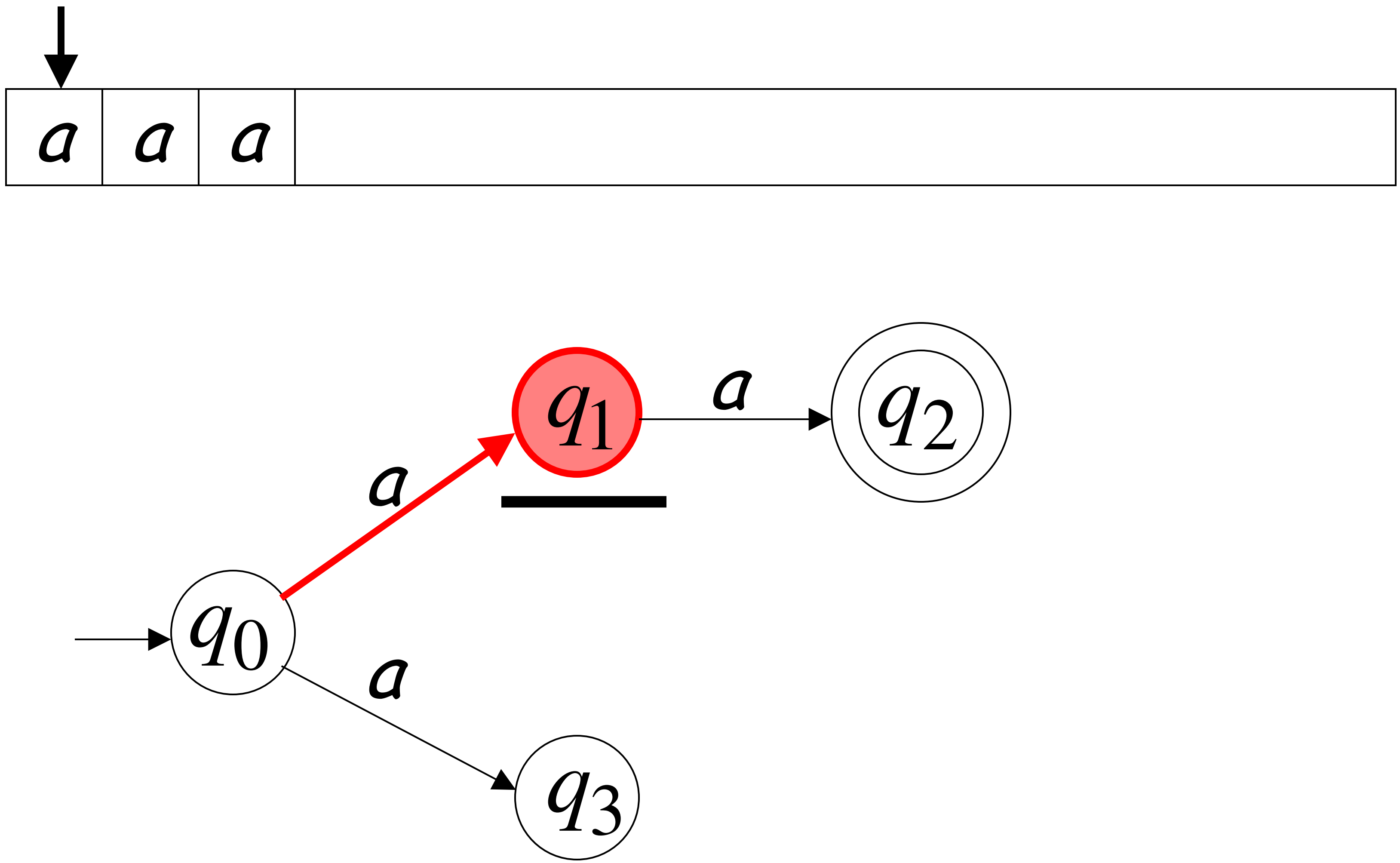


All possible computations lead to rejection

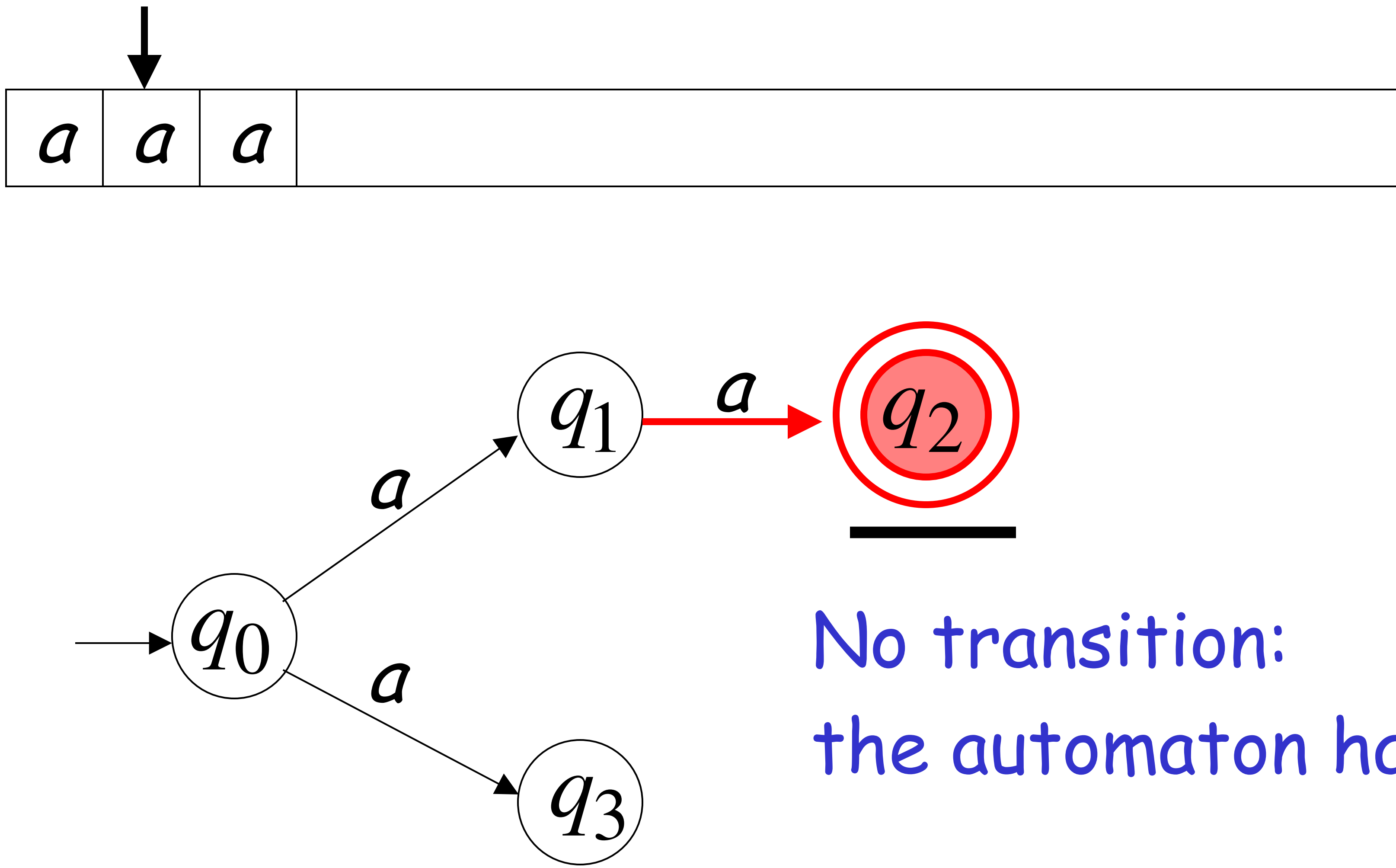
Rejection example



First Choice

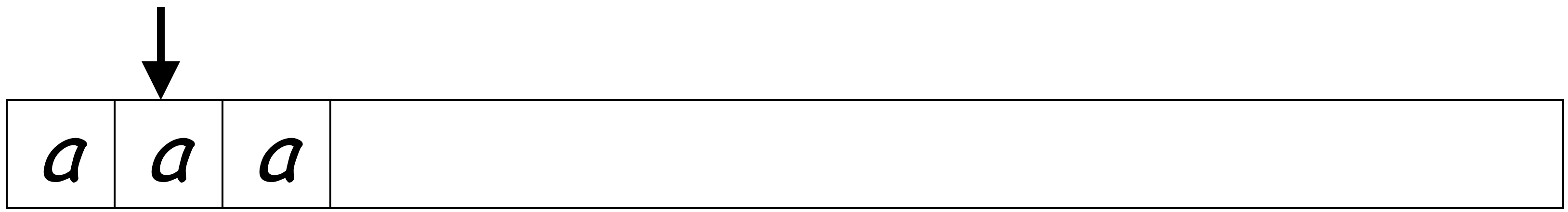


First Choice

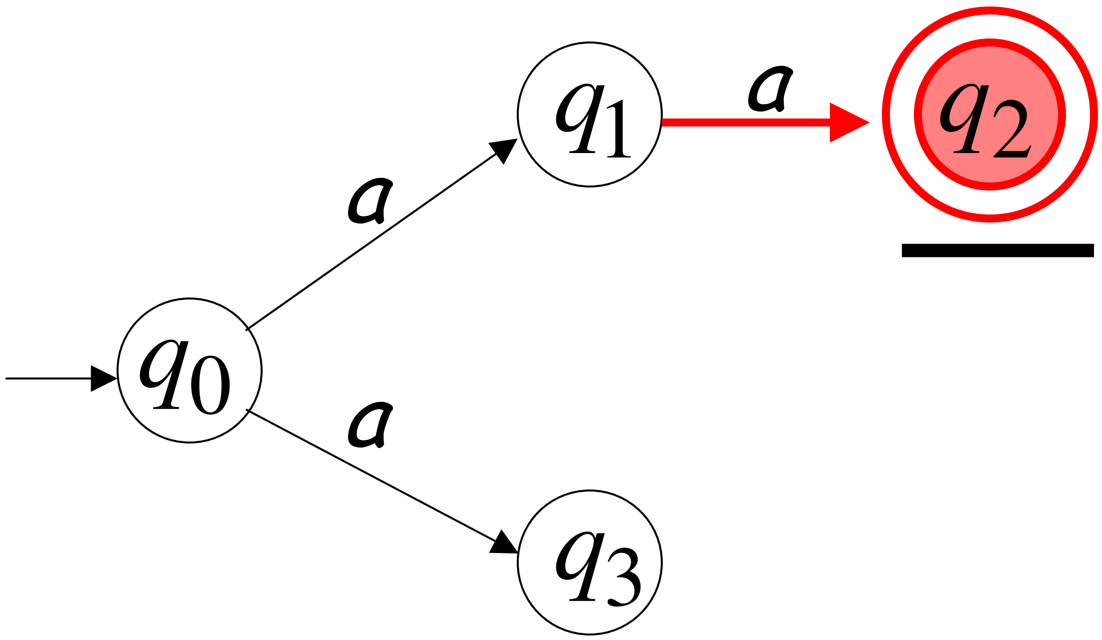


No transition:
the automaton hangs

First Choice

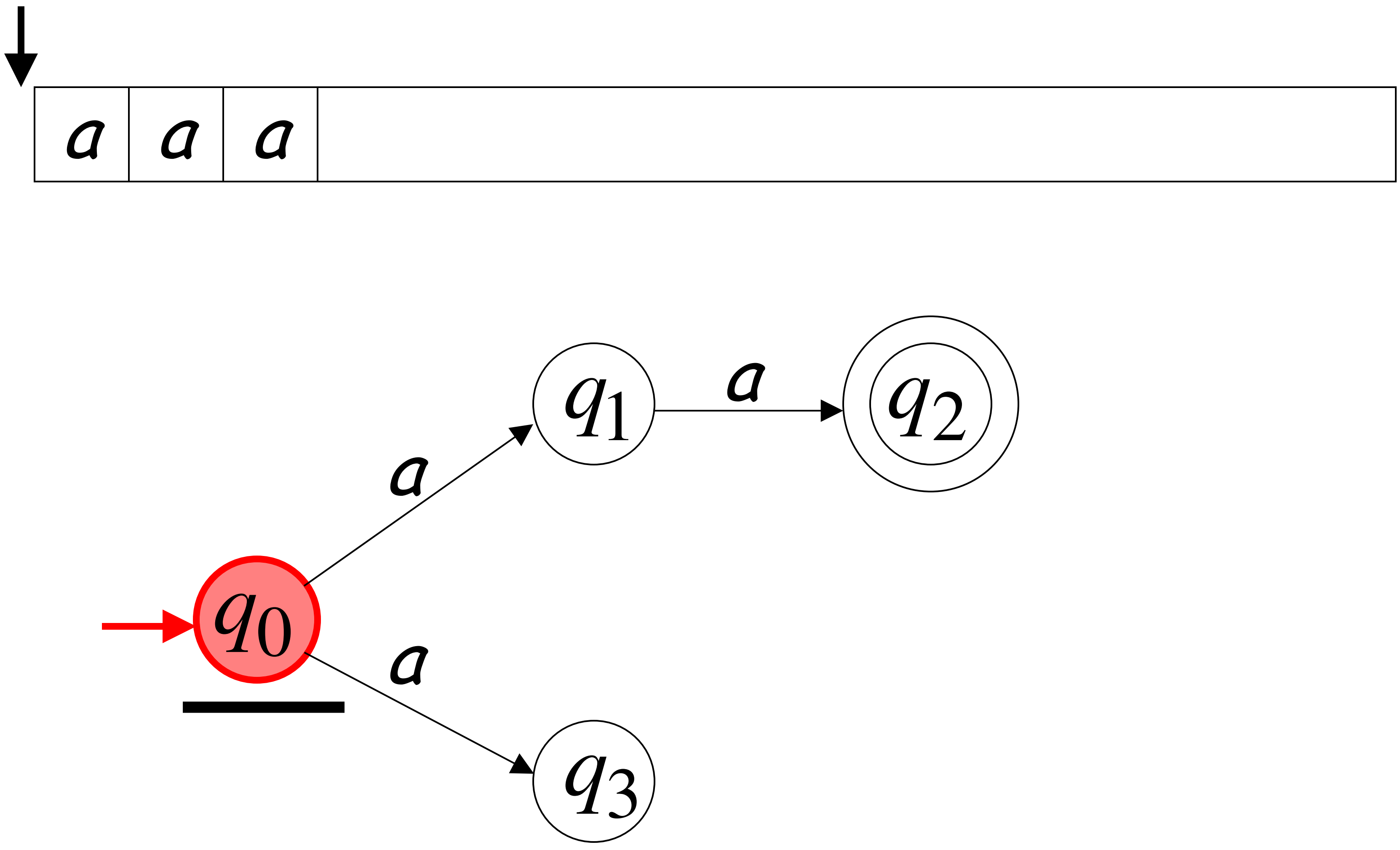


Input cannot be consumed

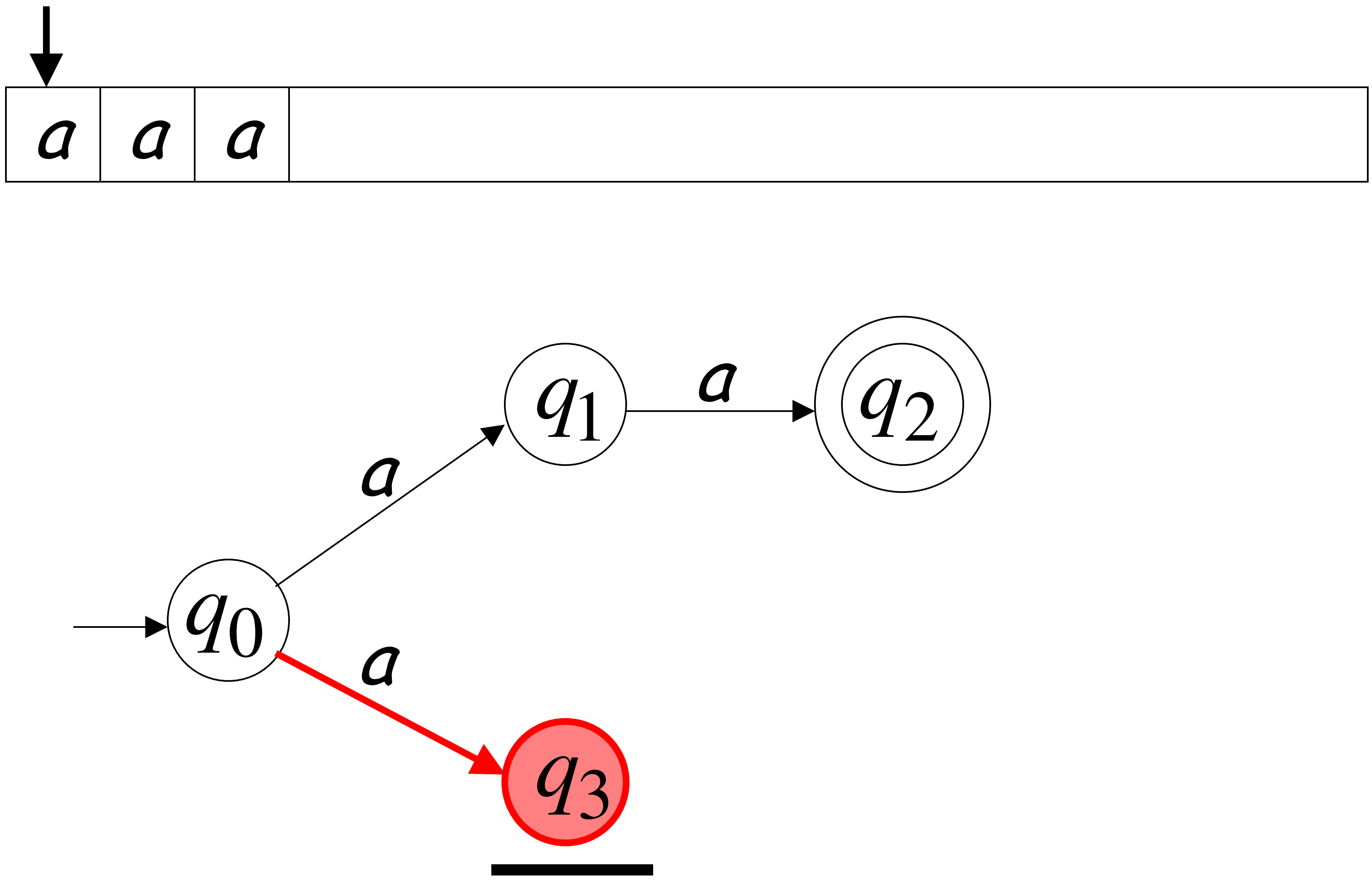


"reject"

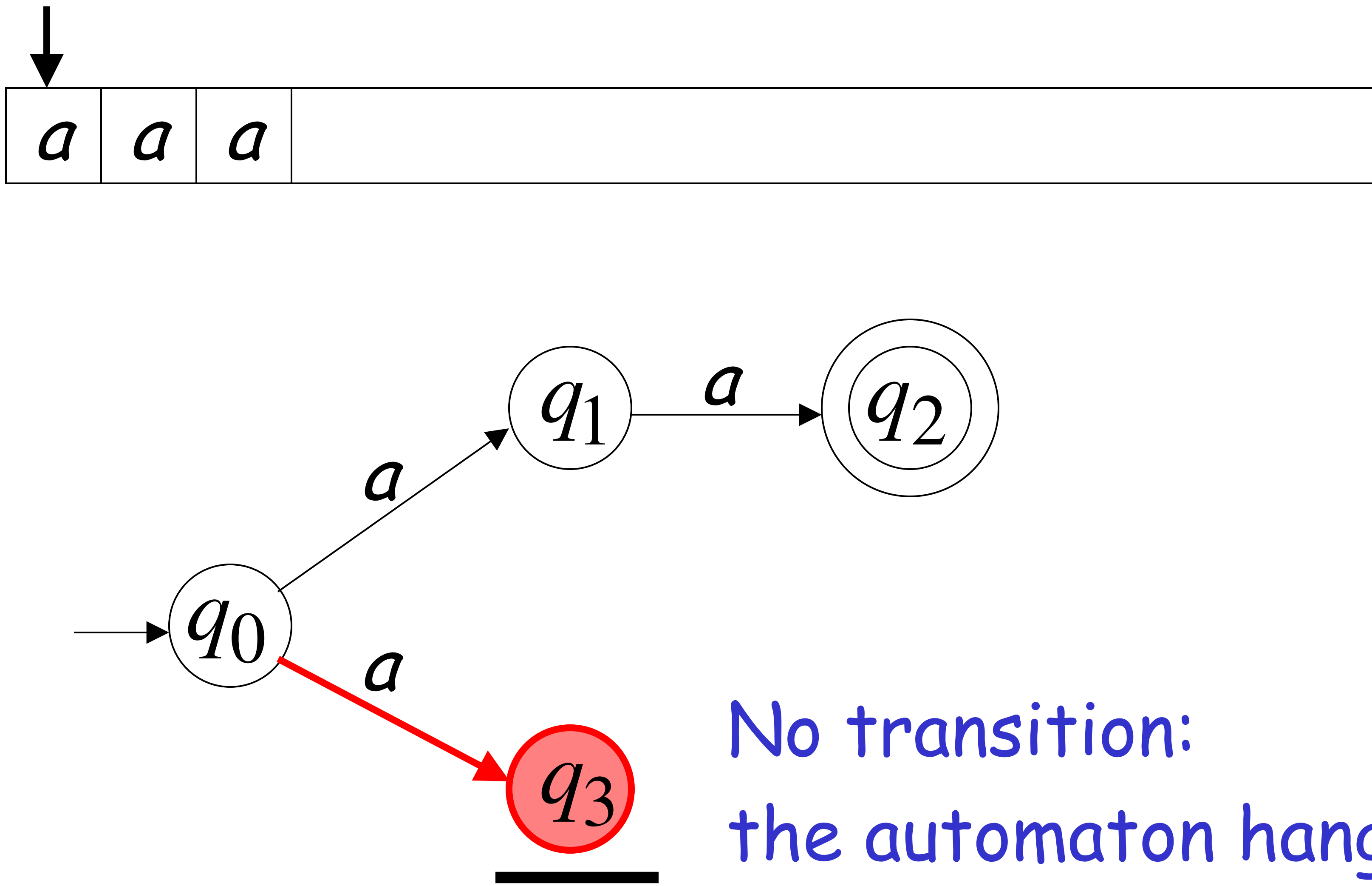
Second Choice



Second Choice

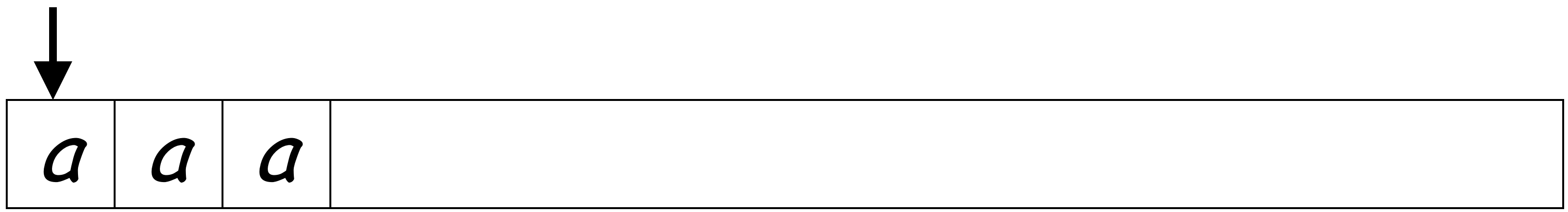


Second Choice

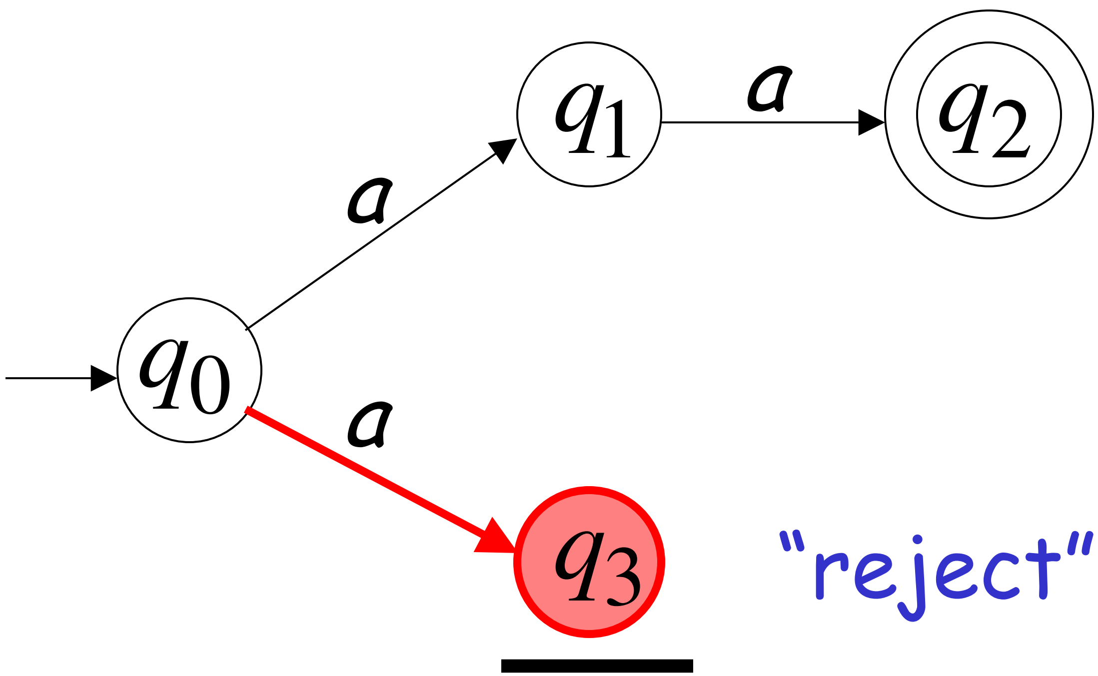


No transition:
the automaton hangs

Second Choice

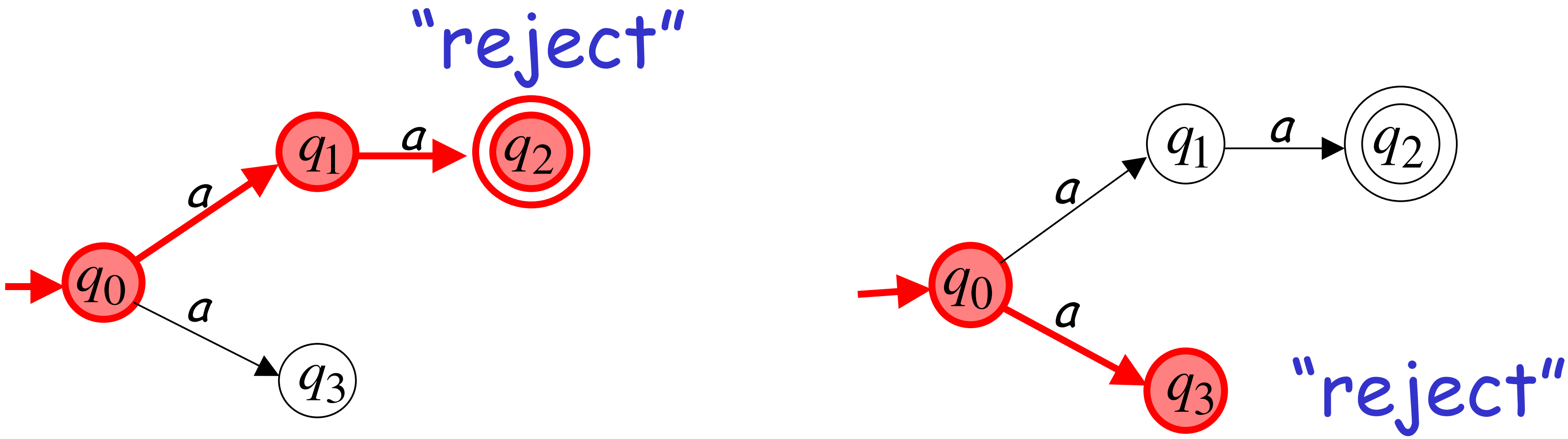


Input cannot be consumed



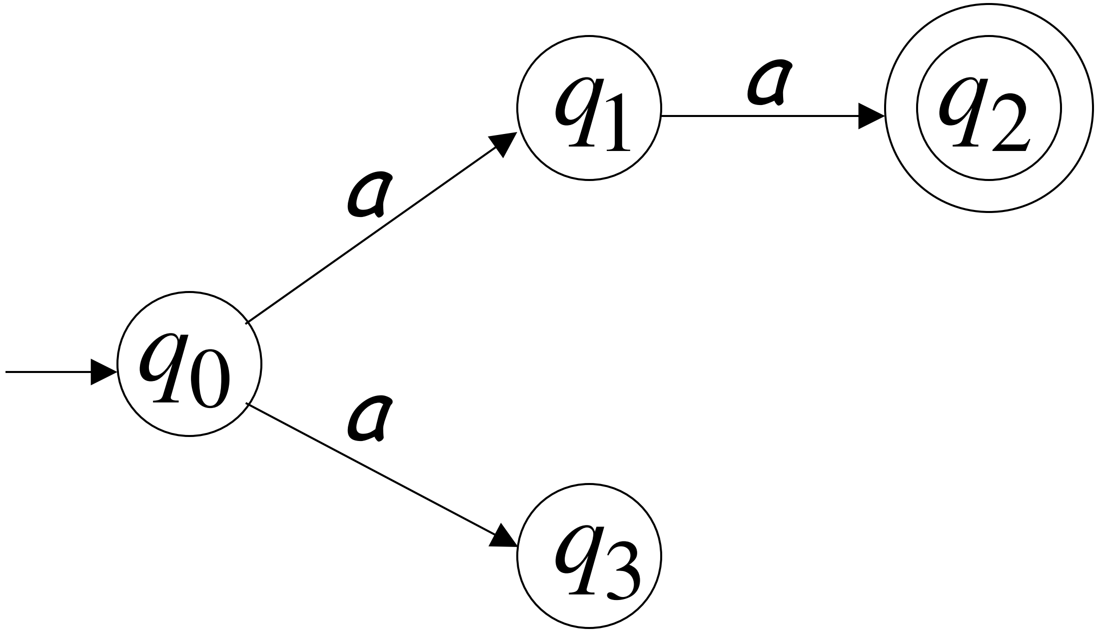
"reject"

aaa is rejected by the NFA:

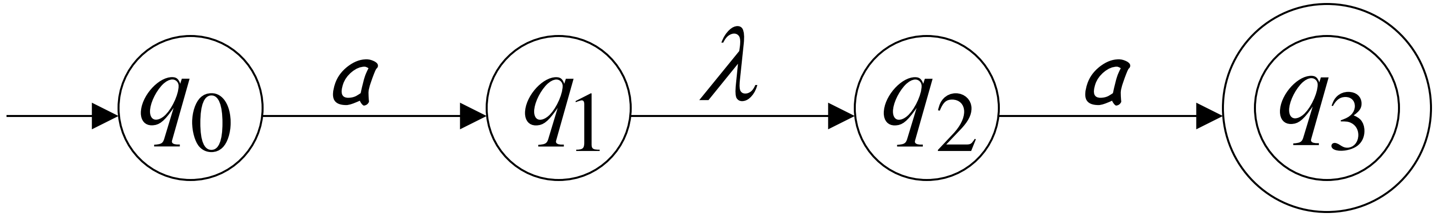


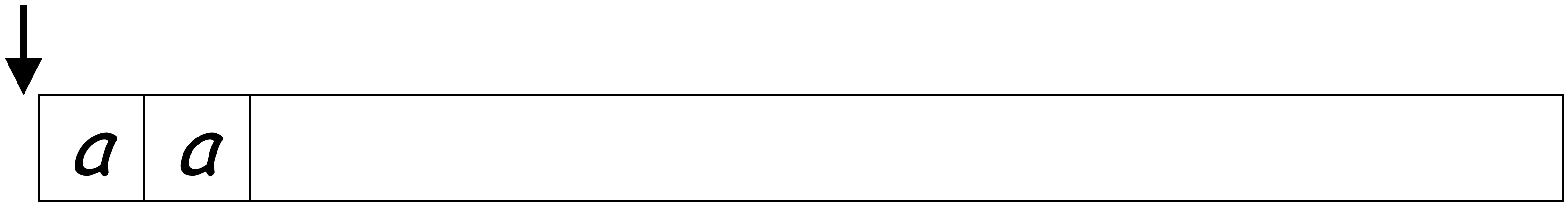
All possible computations lead to rejection

Language accepted: $L = \{aa\}$

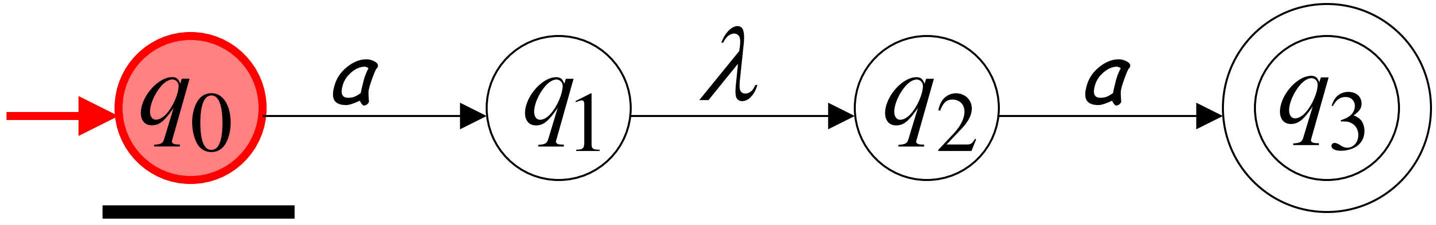


Lambda Transitions

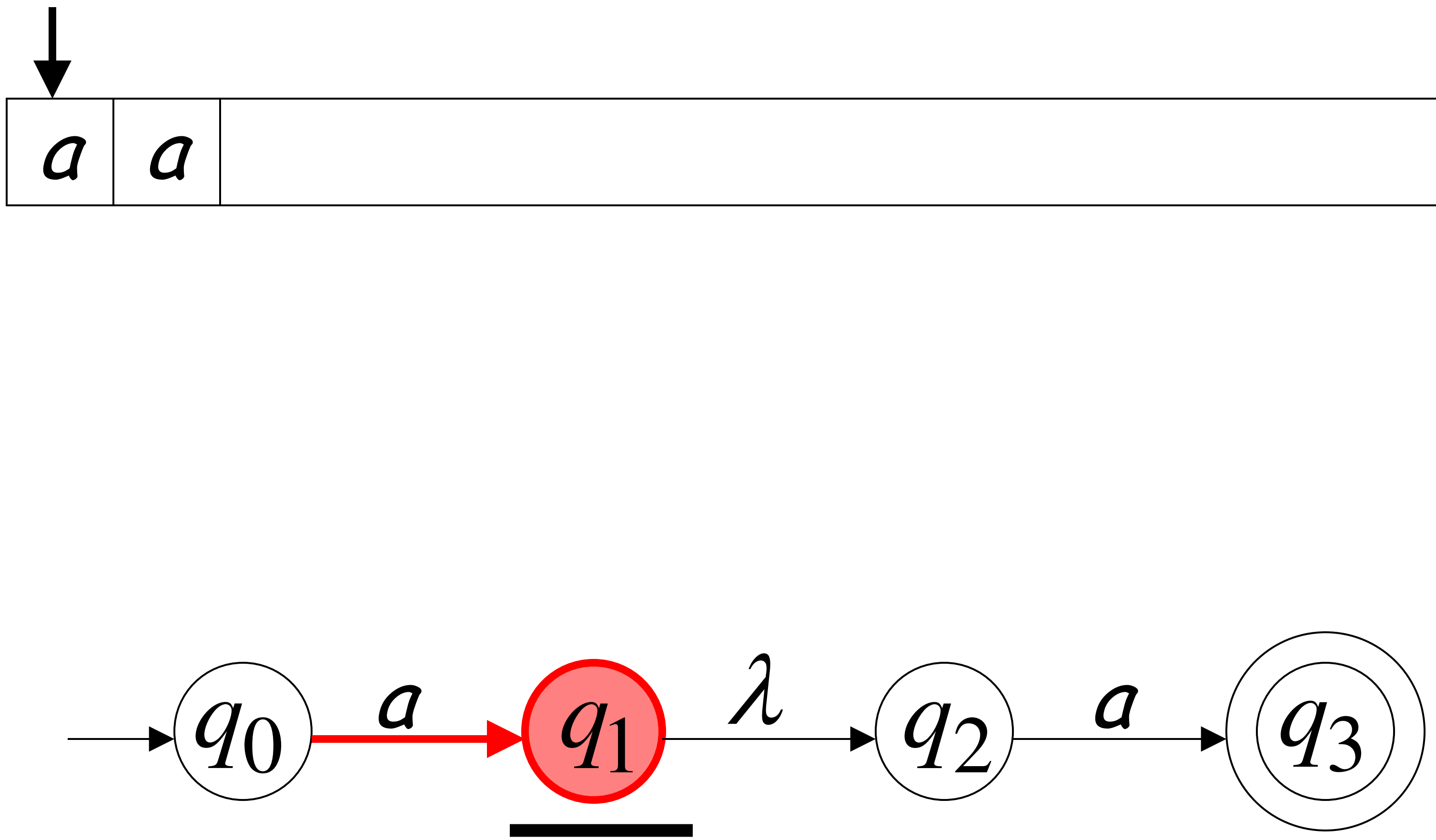




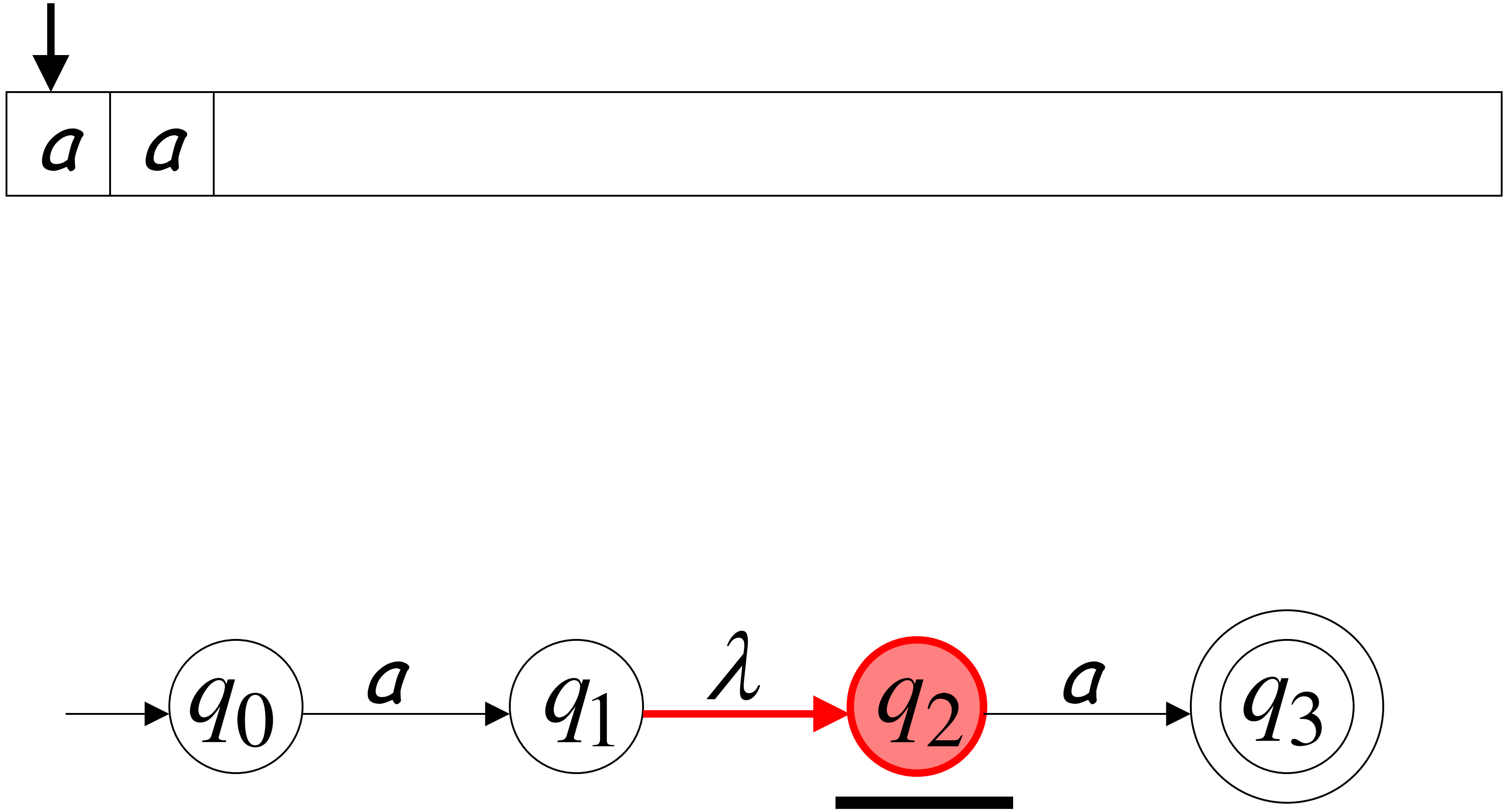
a
a

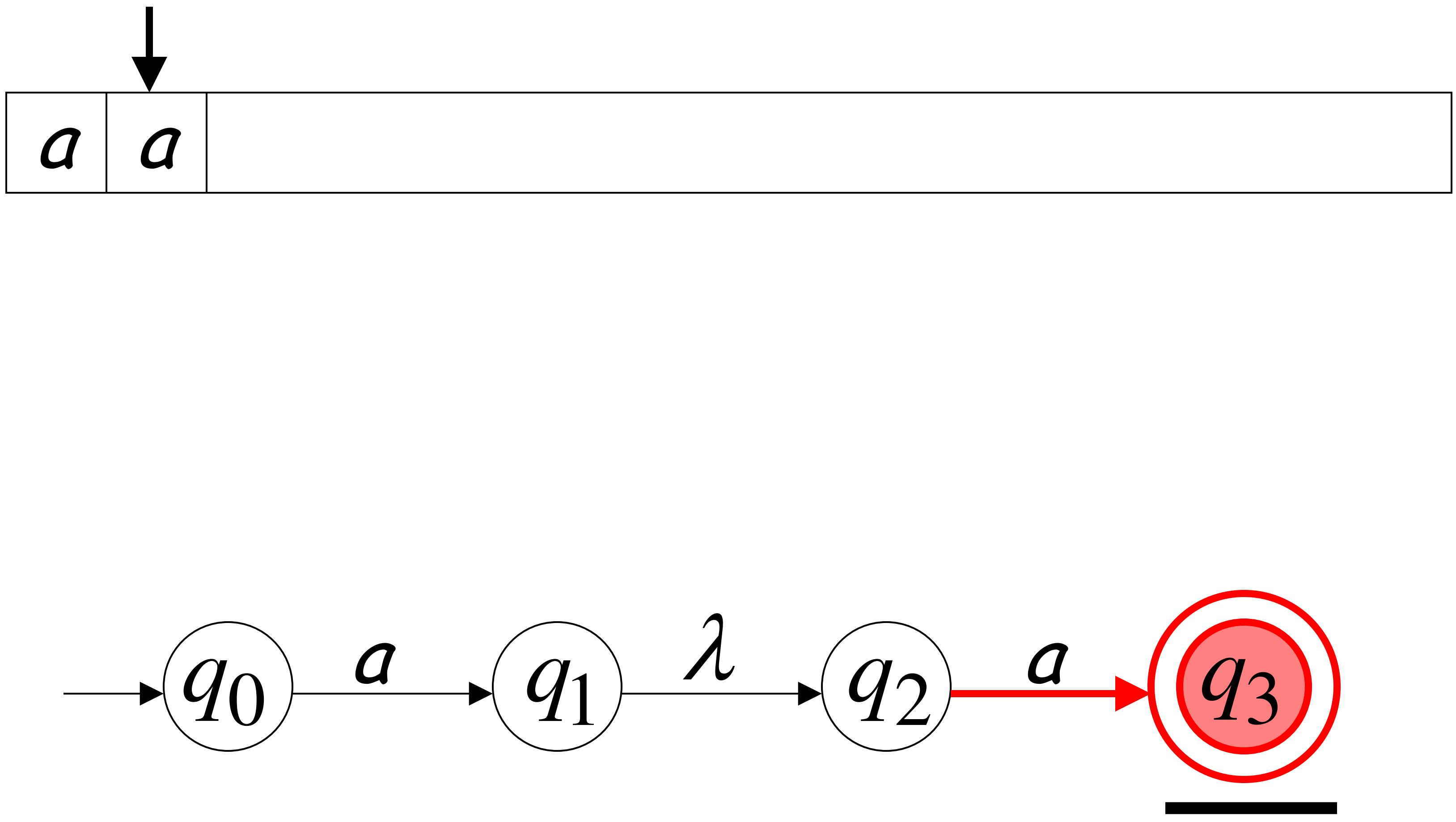


q_0
a
q_1
λ
q_2
a
q_3

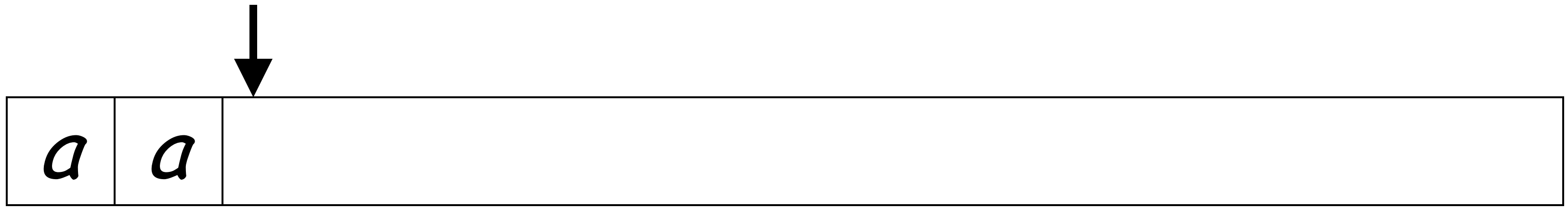

a a
a
λ
a
q_0
q_1
q_2
q_3

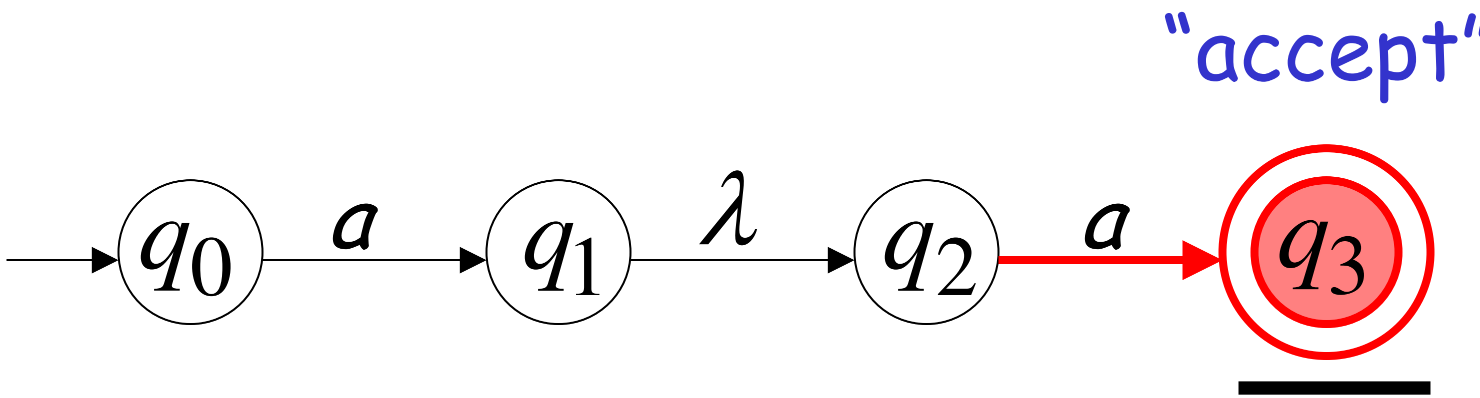
(read head does not move)




a a
a
λ
a
q0 q1 q2 q3

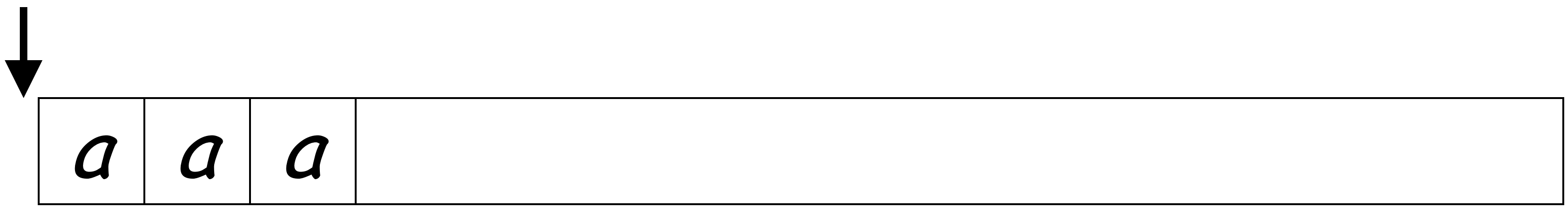
all input is consumed

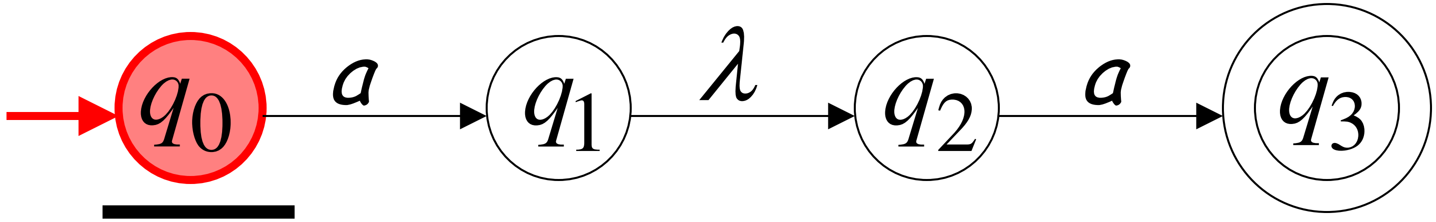


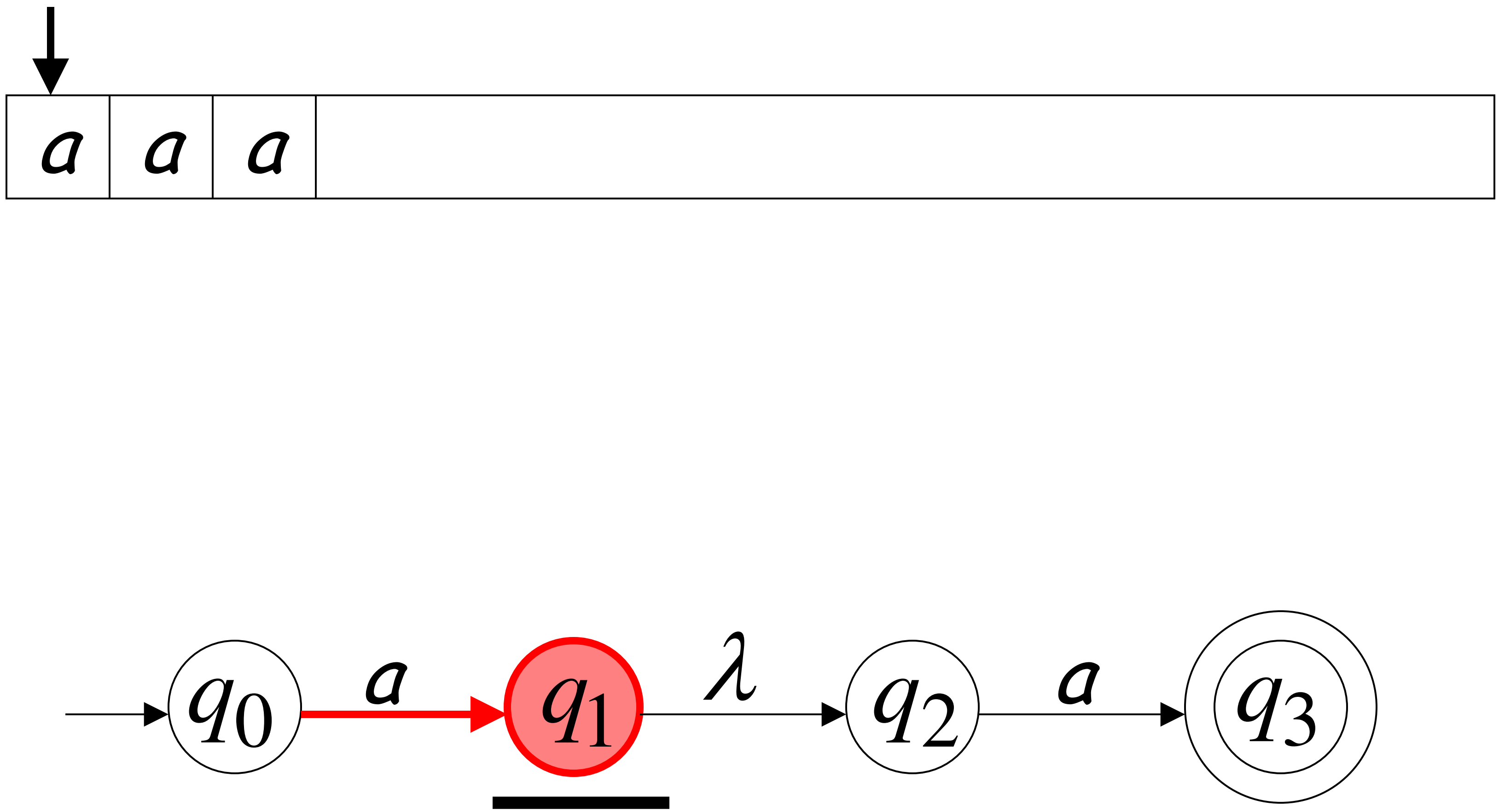


String aa is accepted

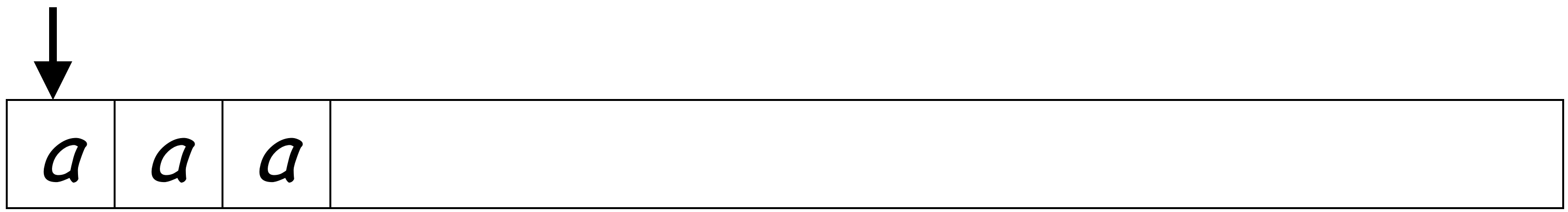
Rejection Example

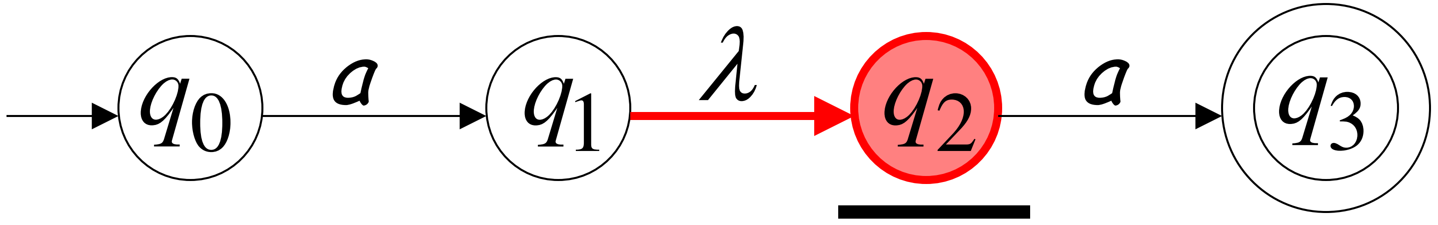


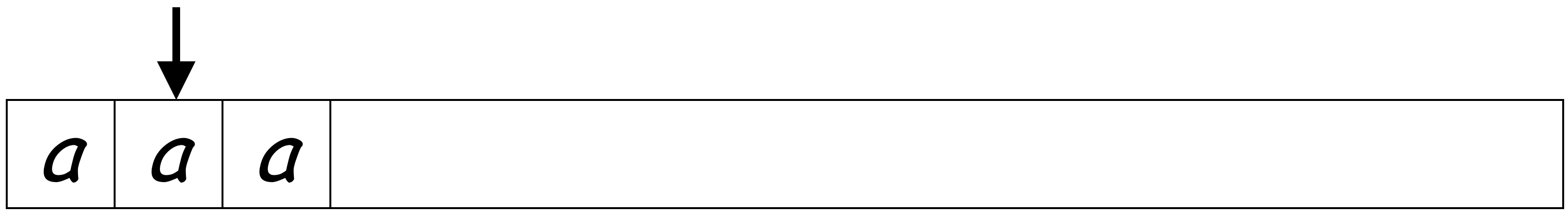


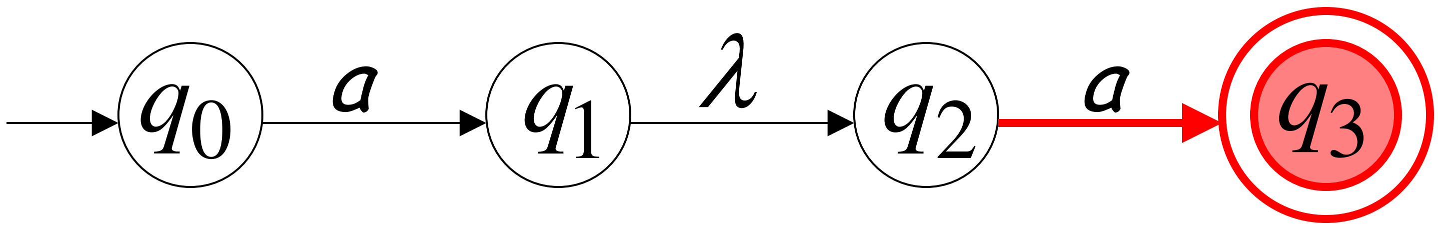

a a a
a
λ
a
q_0 q_1 q_2 q_3

(read head doesn't move)



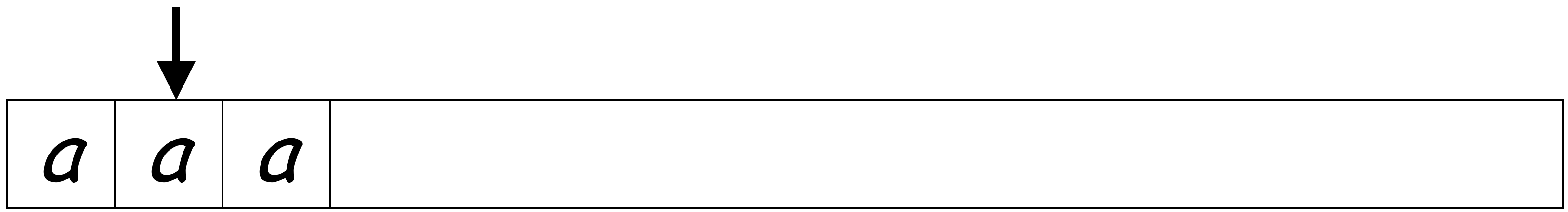


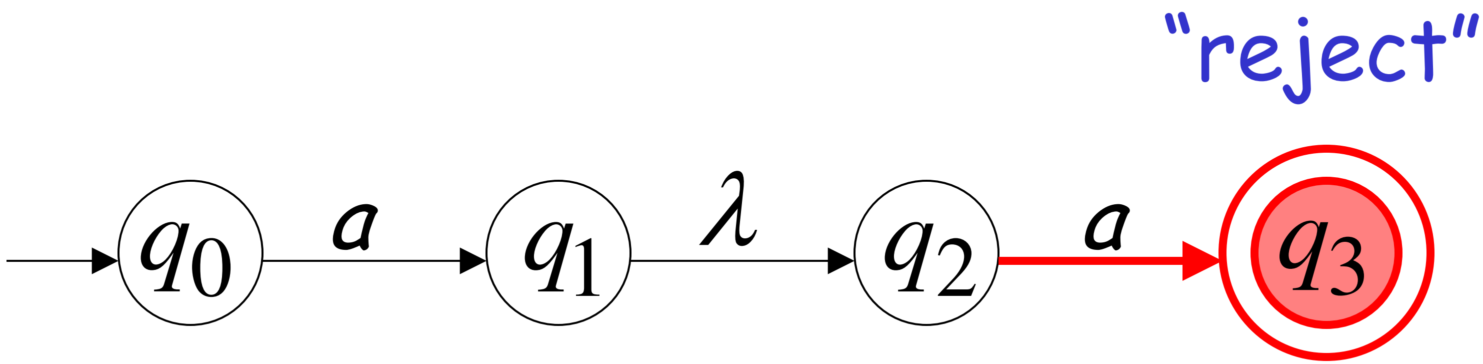




No transition:
the automaton hangs

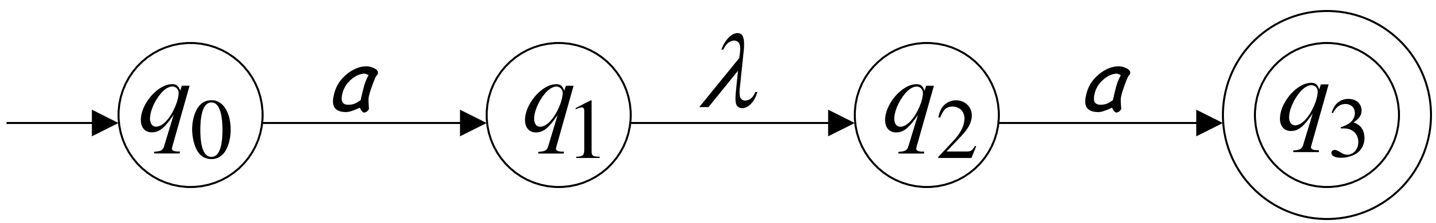
Input cannot be consumed



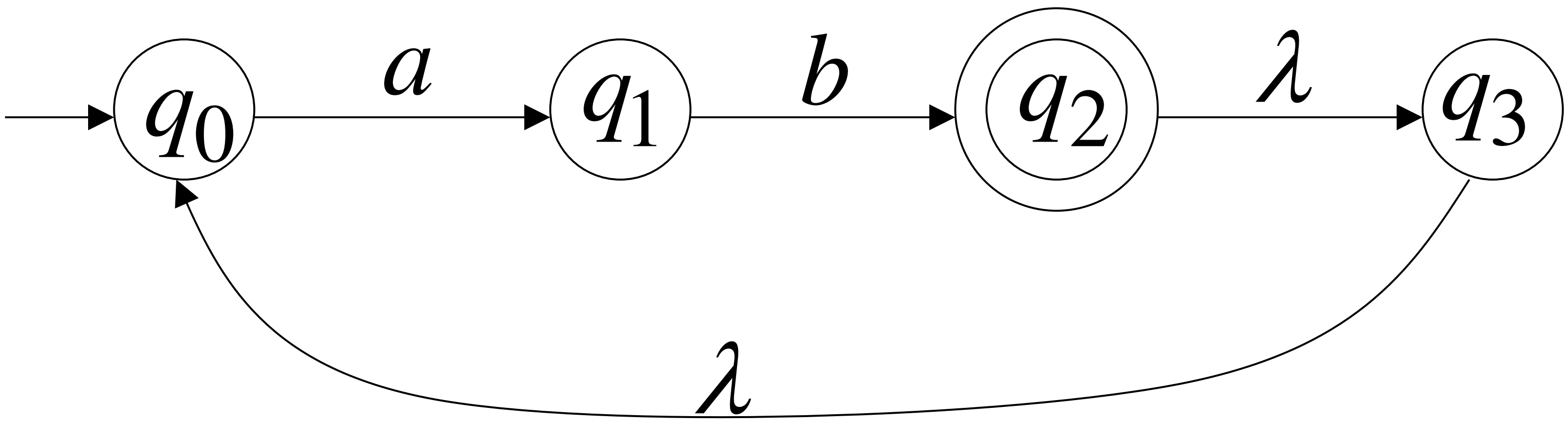


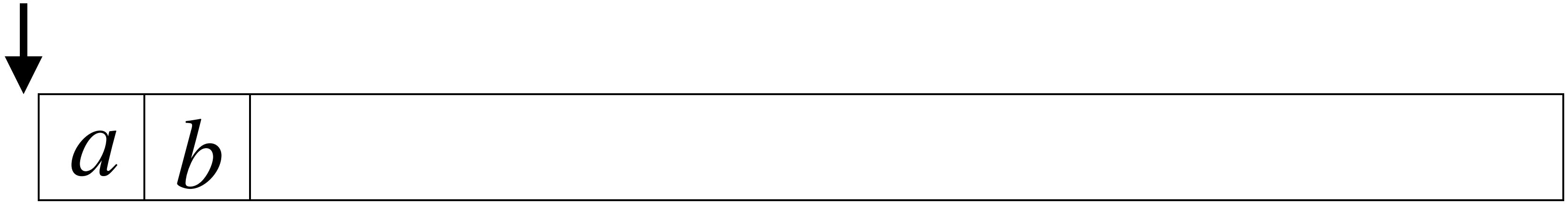
String aaa is rejected

Language accepted: $L = \{aa\}$

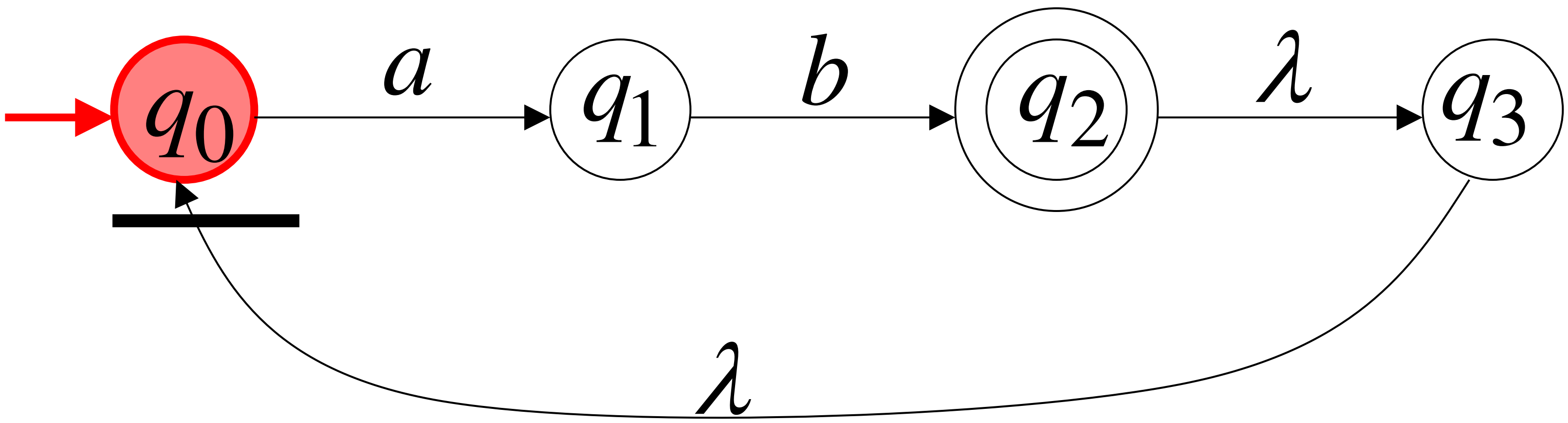


Another NFA Example

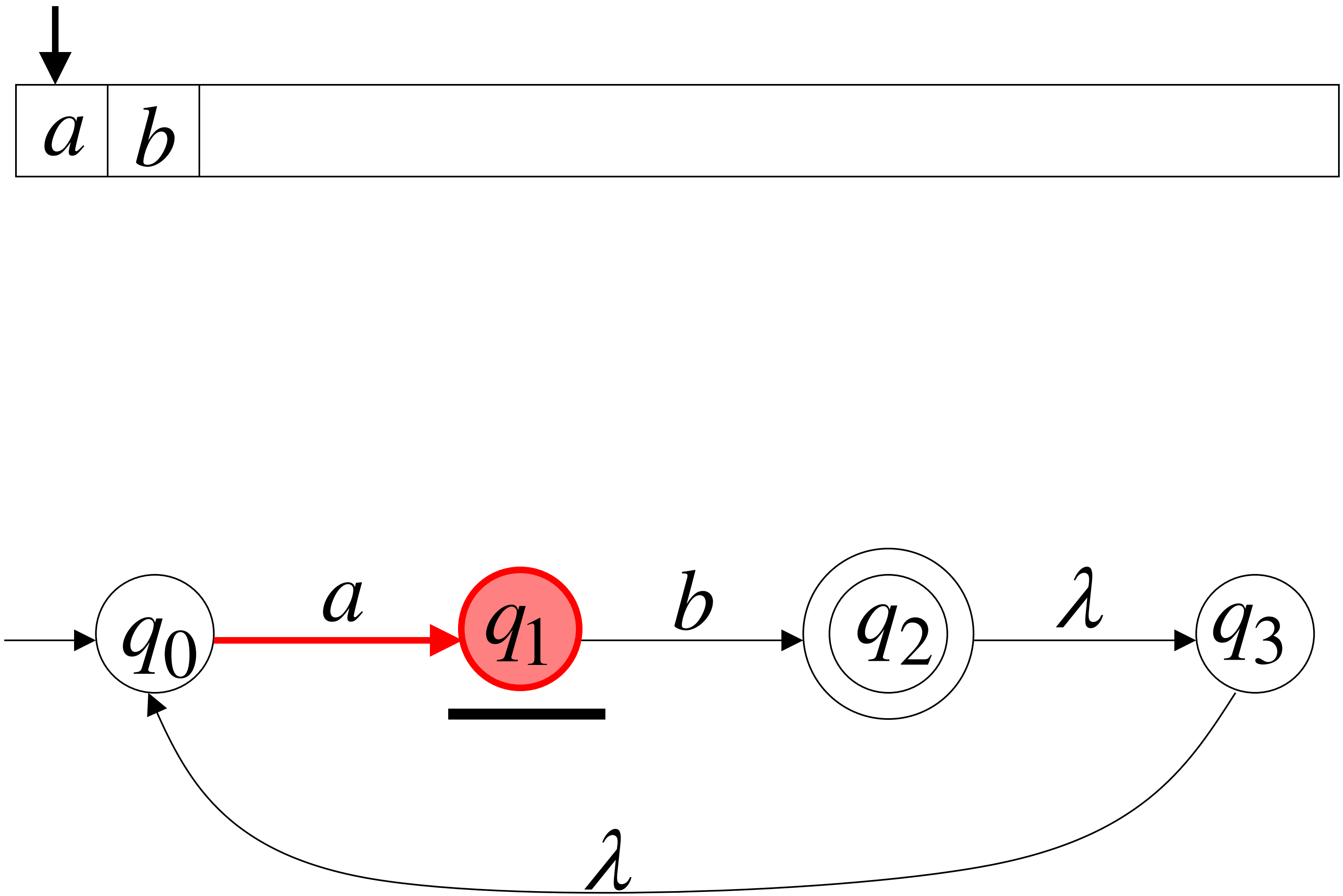


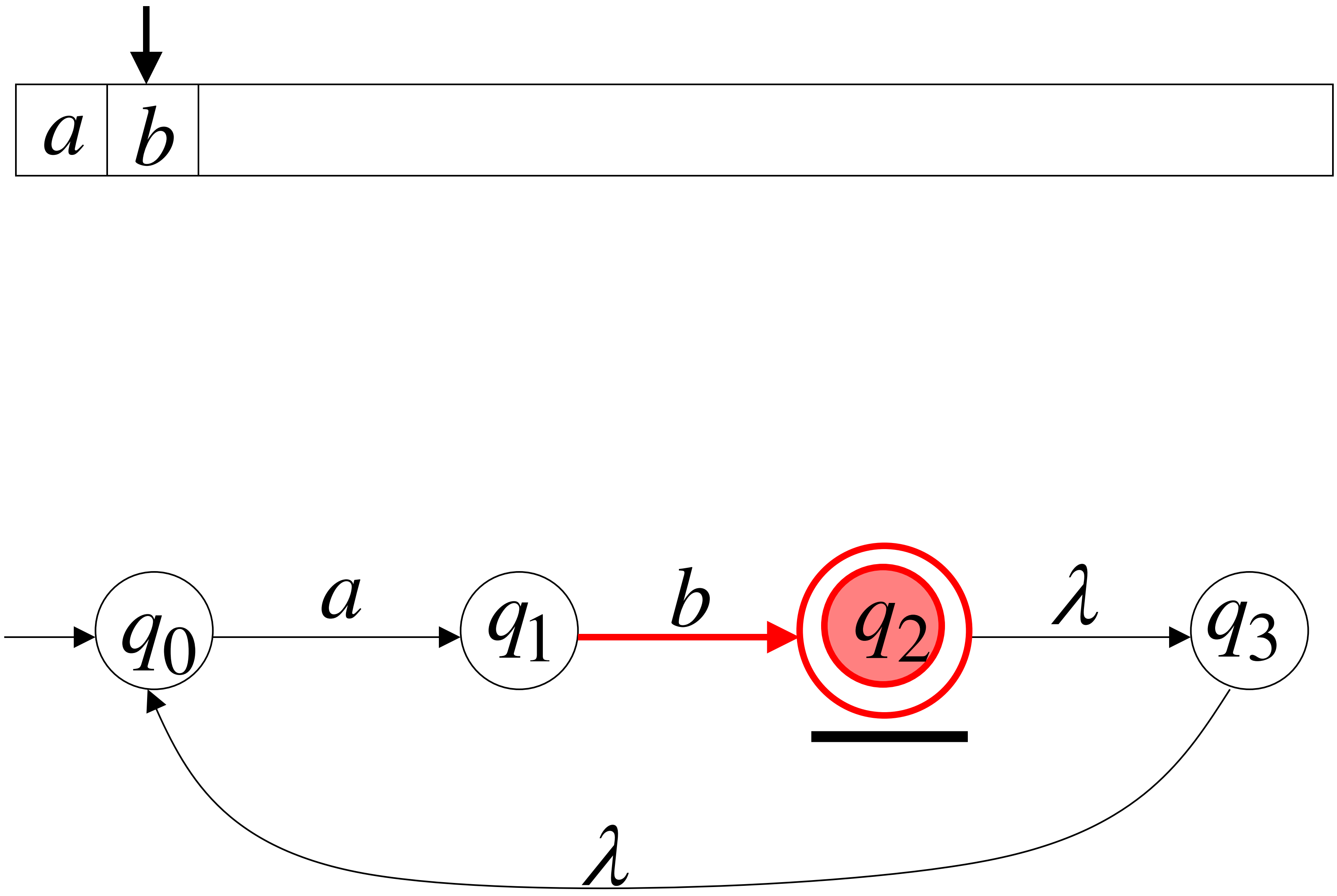


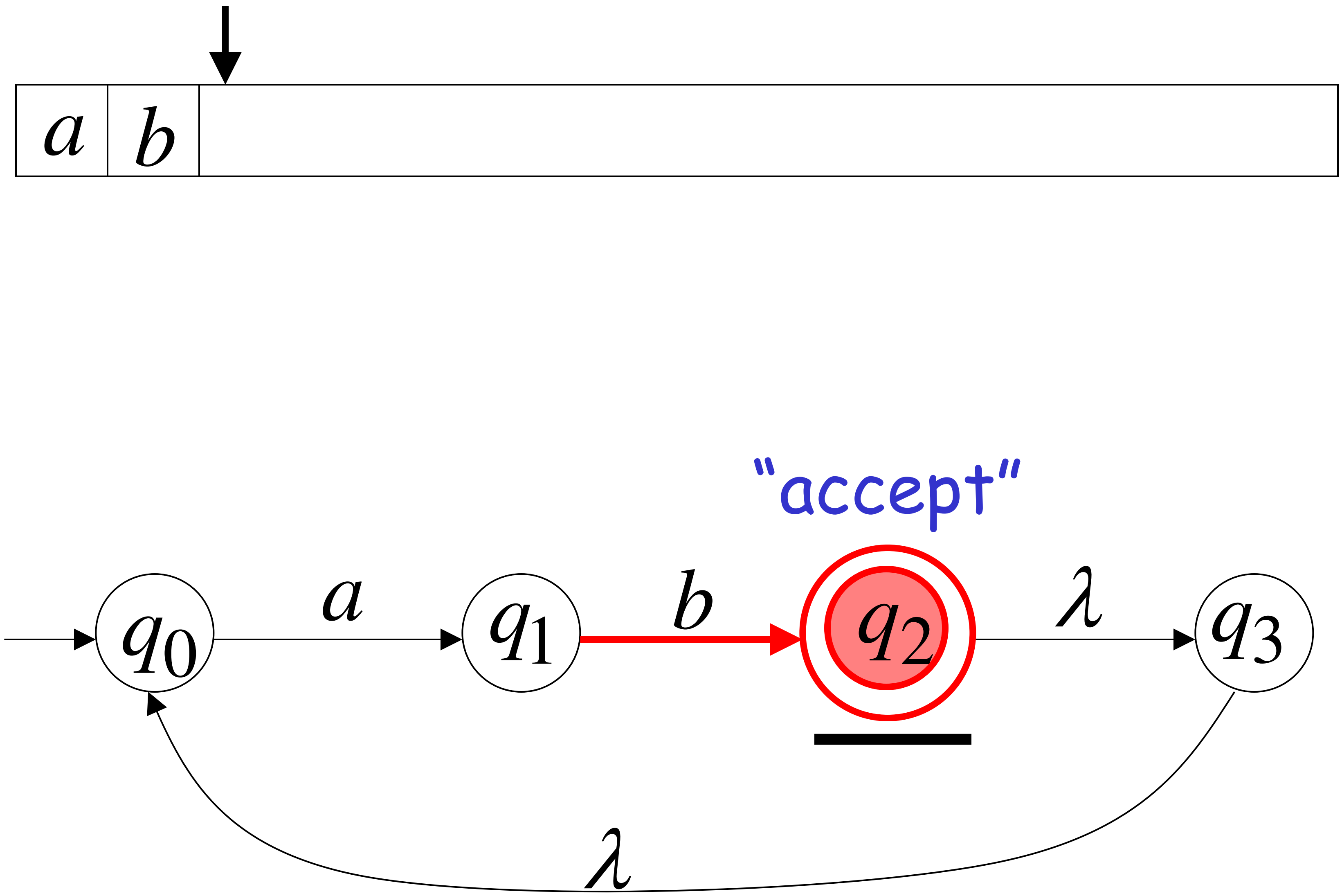
a
b



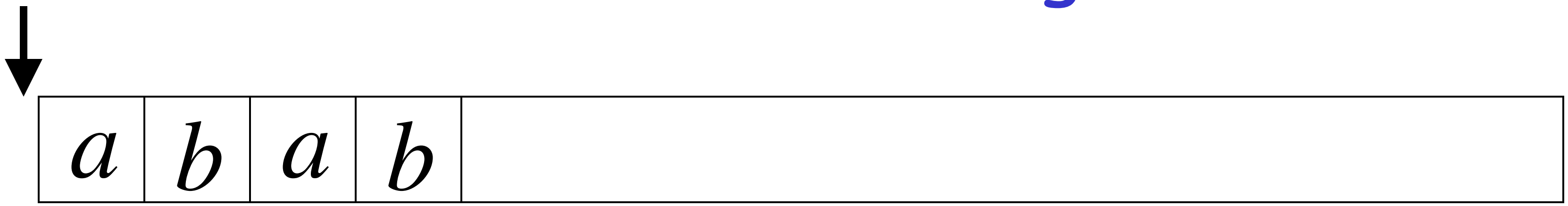
q_0
a
q_1
b
q_2
λ
q_3
λ

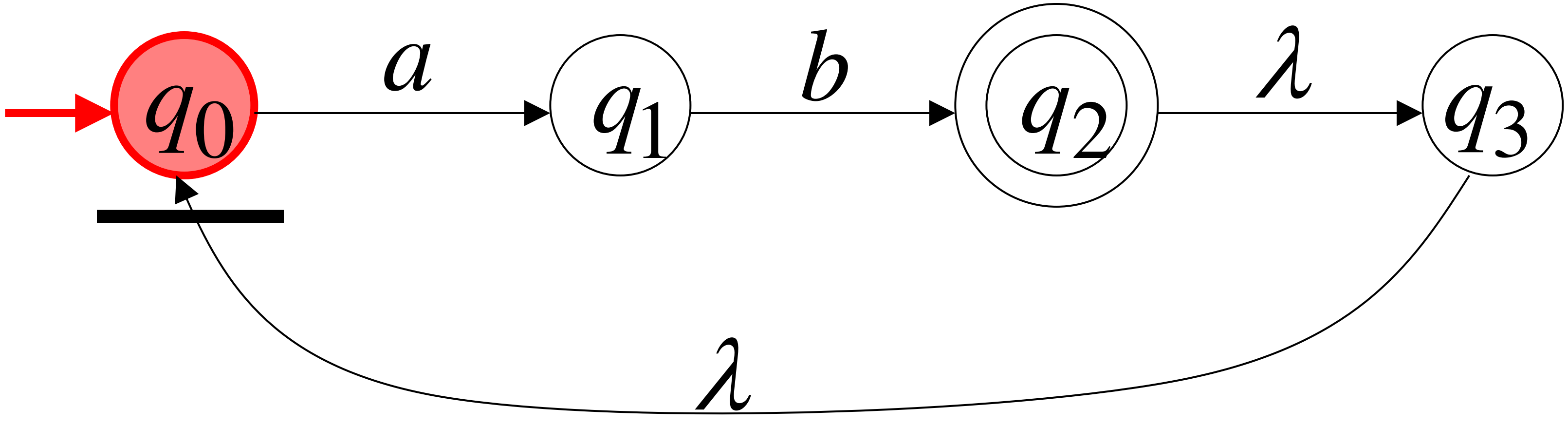

a
b
a
b
λ
λ
q_0
q_1
q_2
q_3

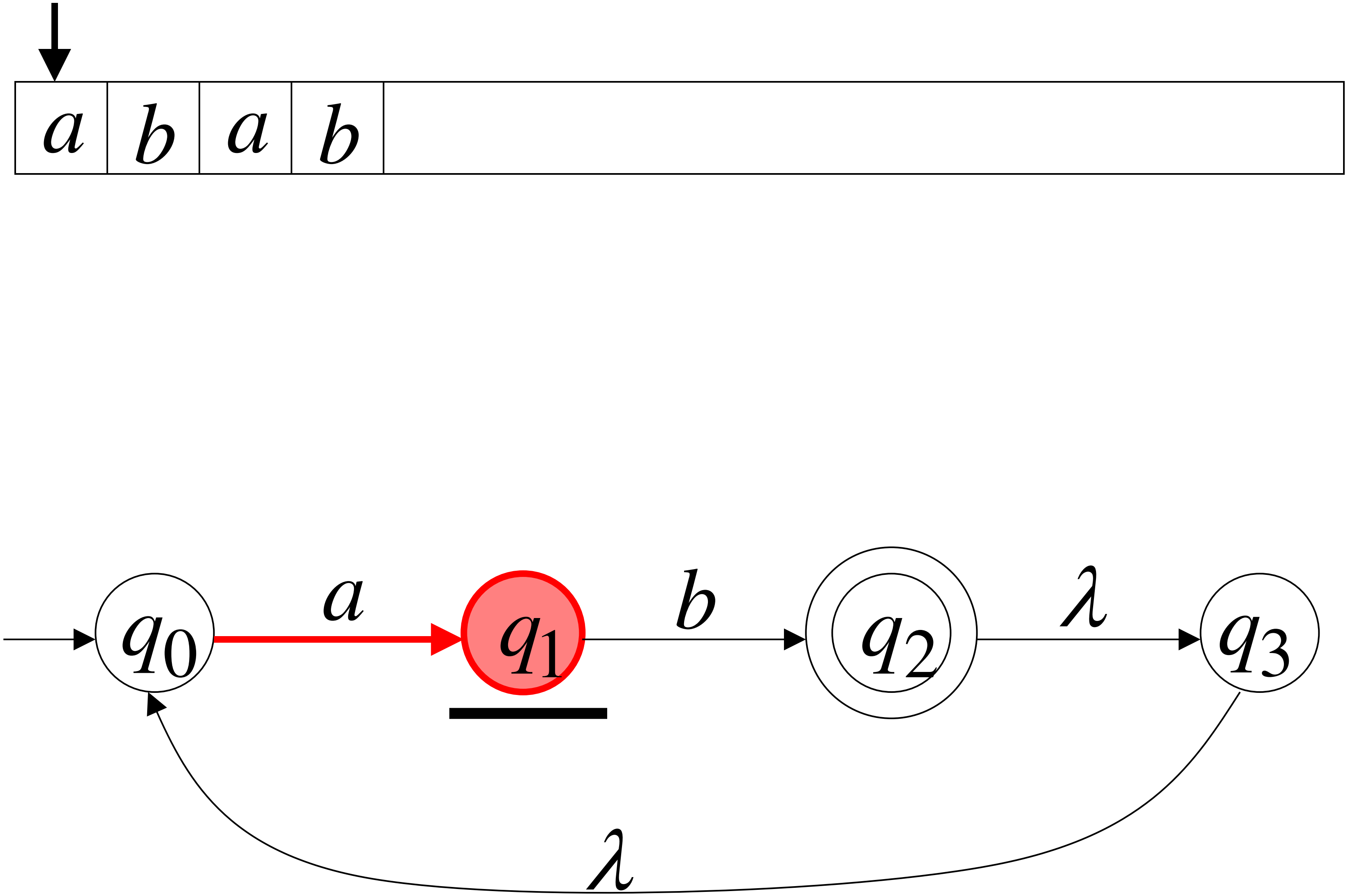

a
b
a
b
λ
q_0
q_1
q_2
q_3
λ

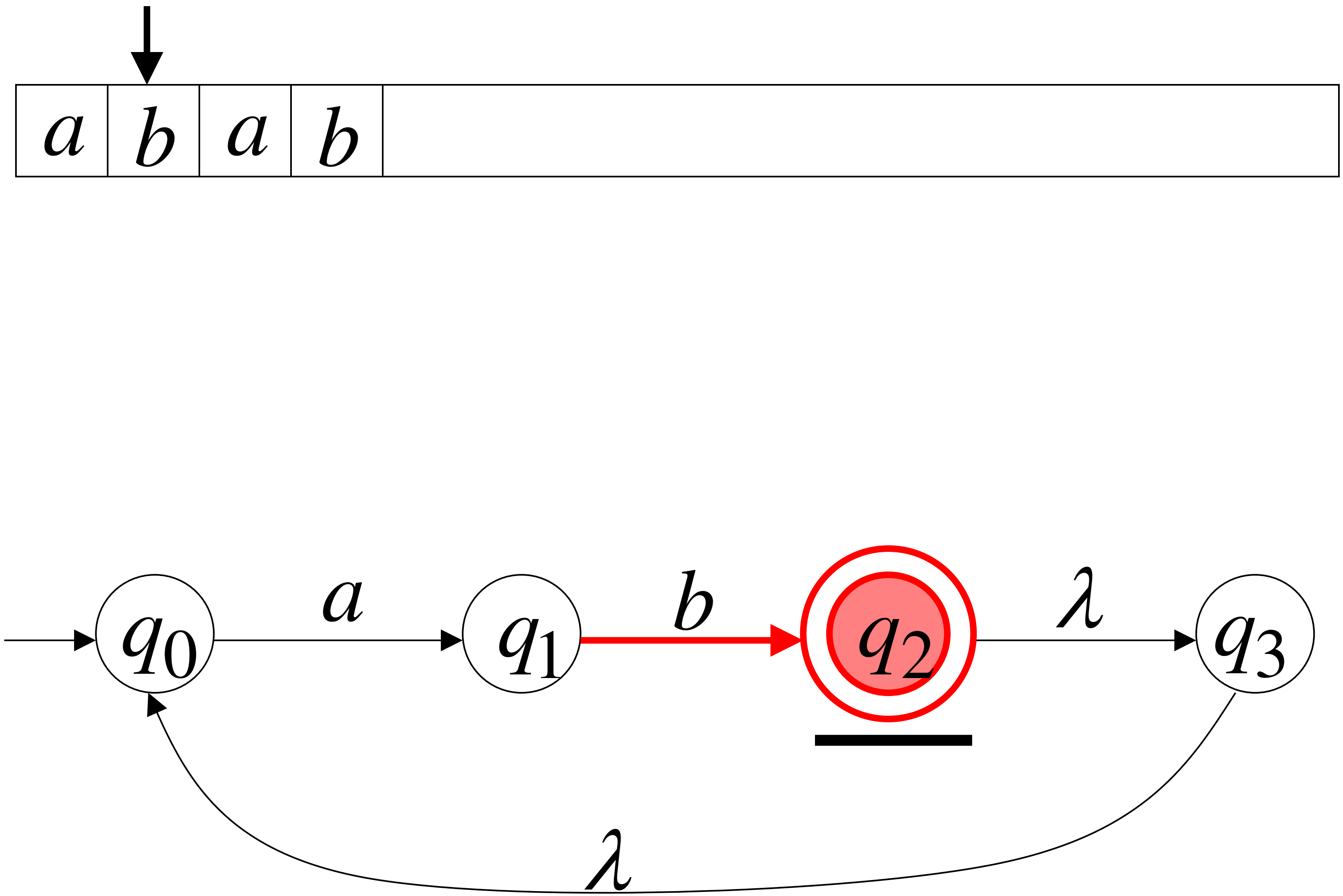

a
b
"accept"
a
b
λ
λ
q_0
q_1
q_2
q_3

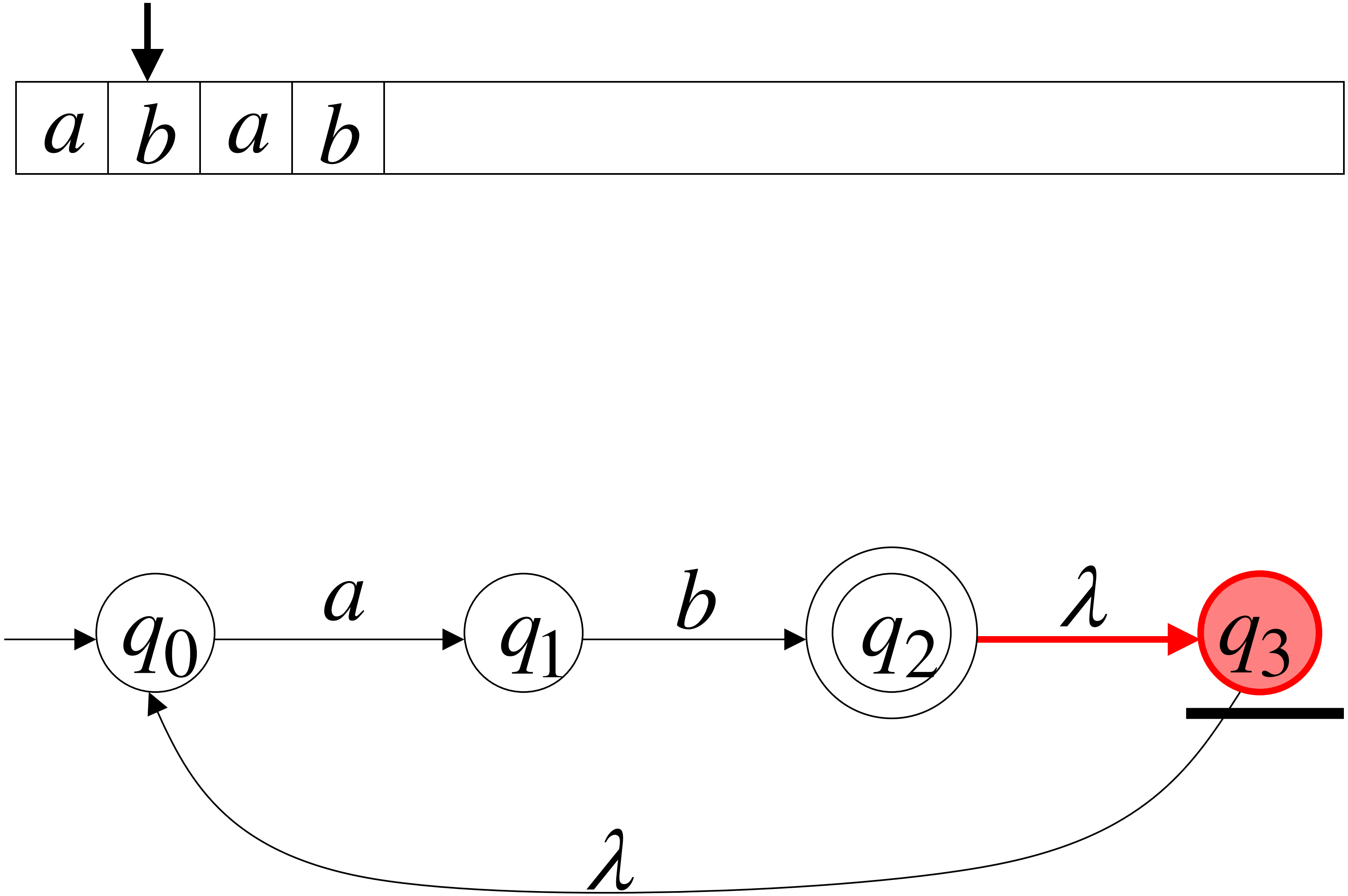
Another String



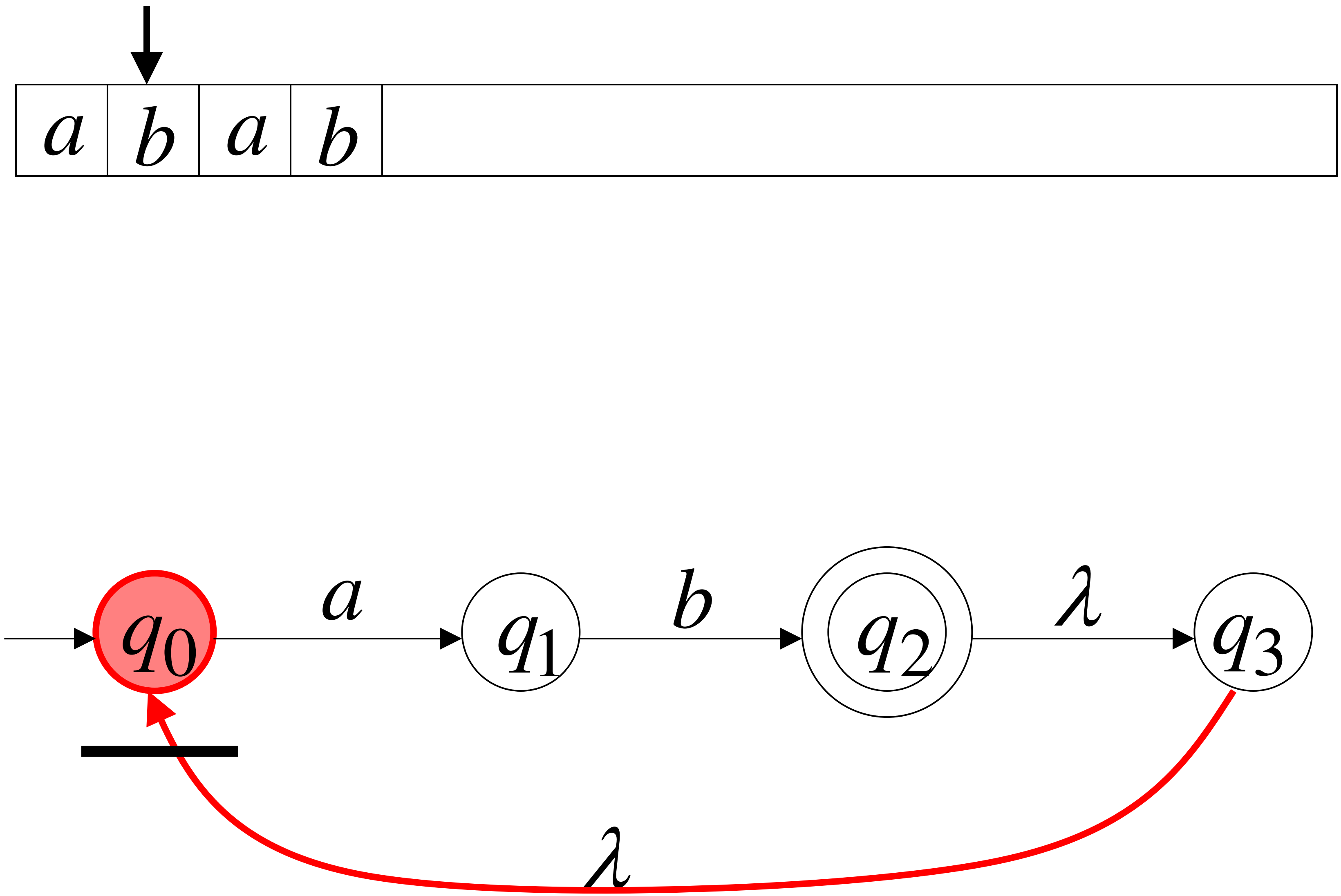


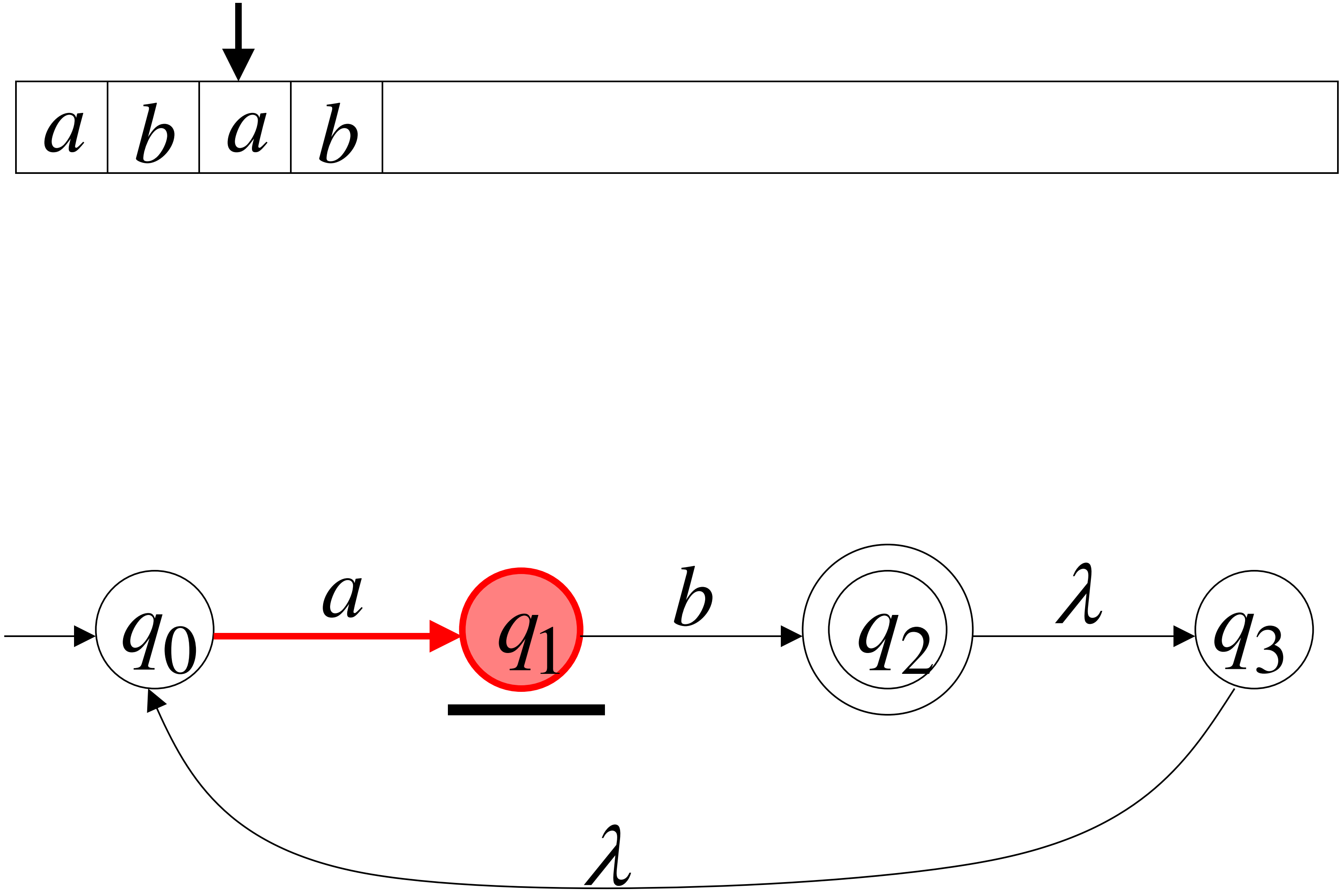

a
b
a
b
a
b
λ
λ
q_0
q_1
q_2
q_3


a
b
a
b
a
b
λ
λ
q_0
q_1
q_2
q_3

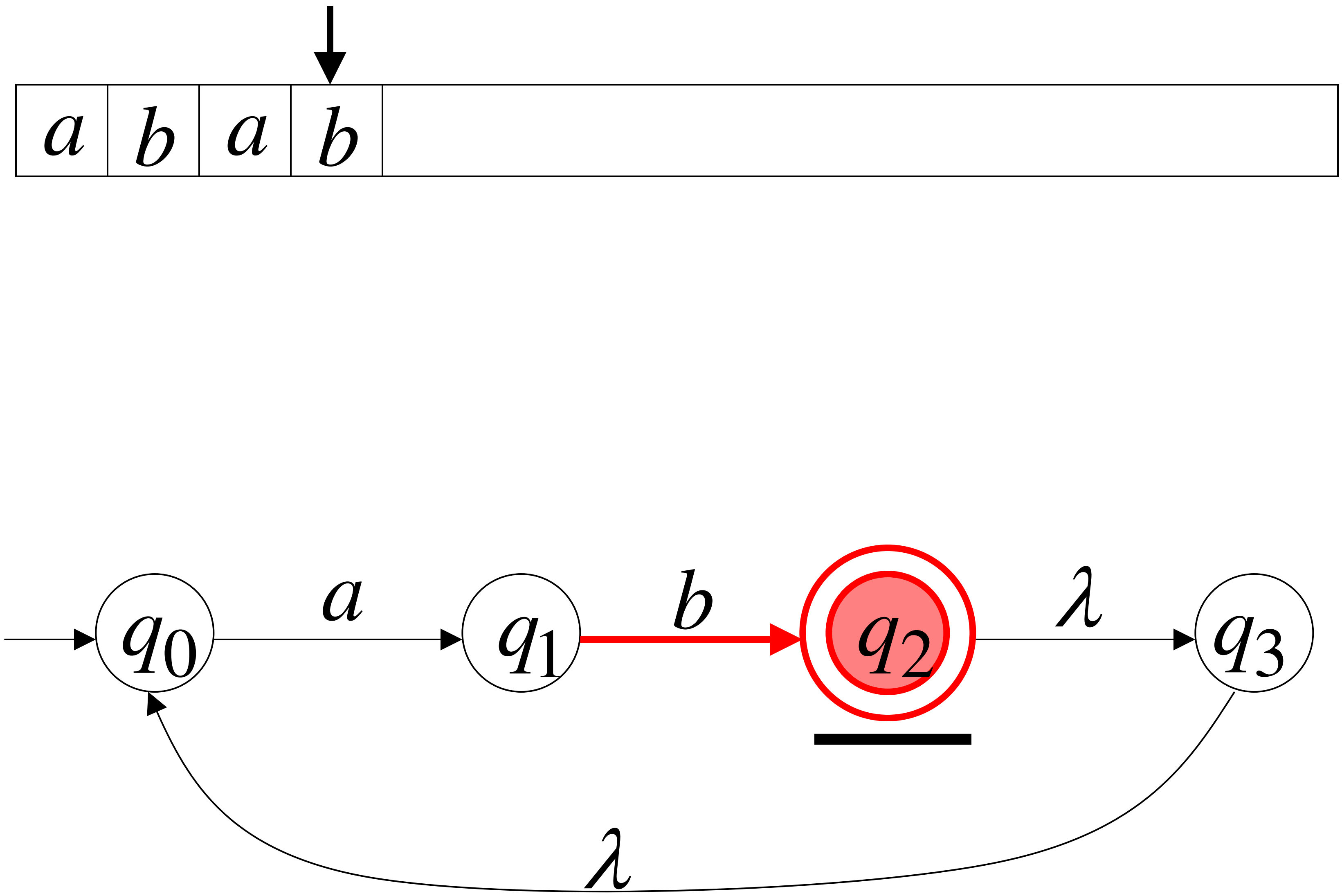


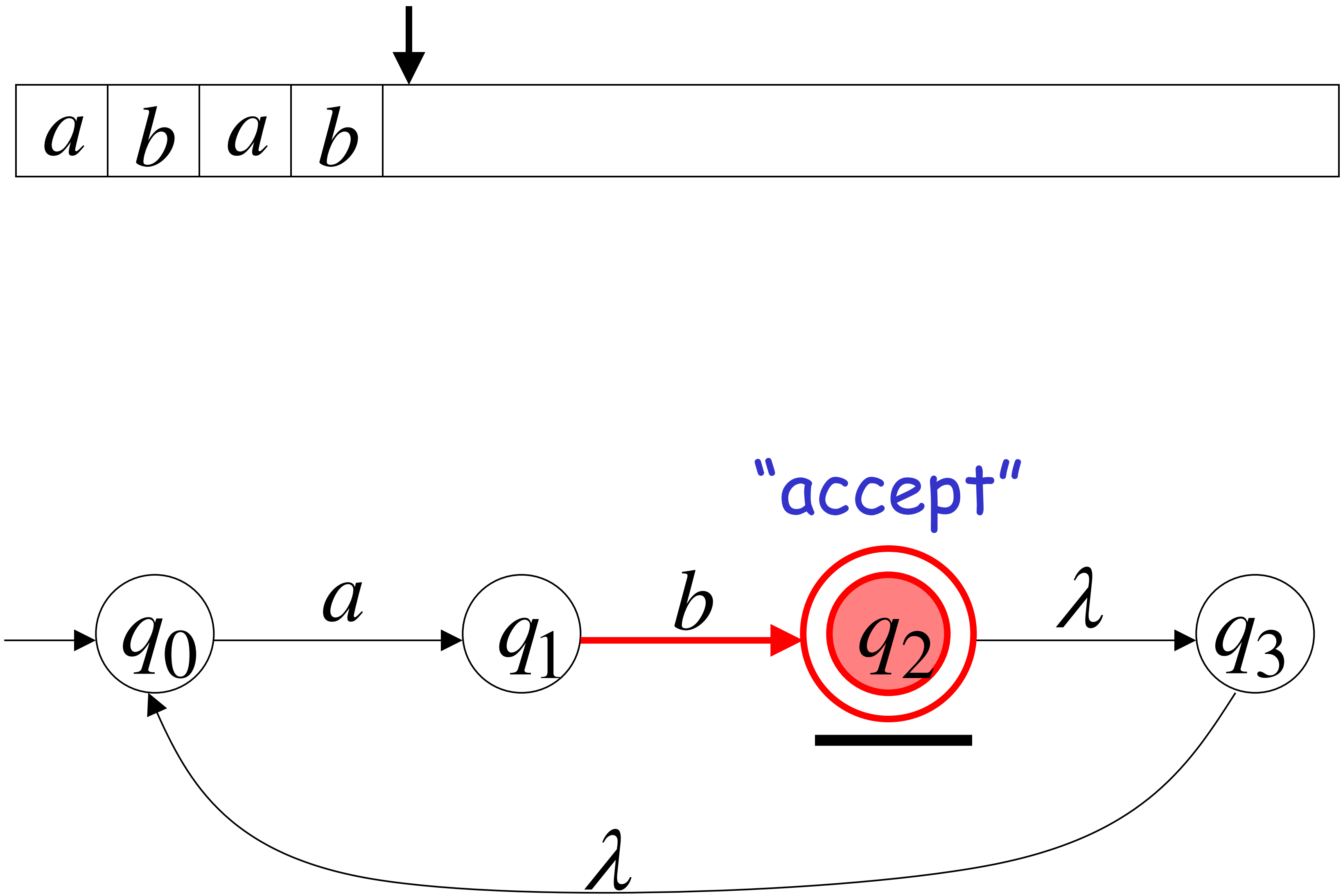
a
b
a
b
a
b
λ
λ
q_0
q_1
q_2
q_3


a
b
a
b
a
b
λ
q0
q1
q2
q3
λ



a
b
a
b
a
b
λ
λ
q_0
q_1
q_2
q_3

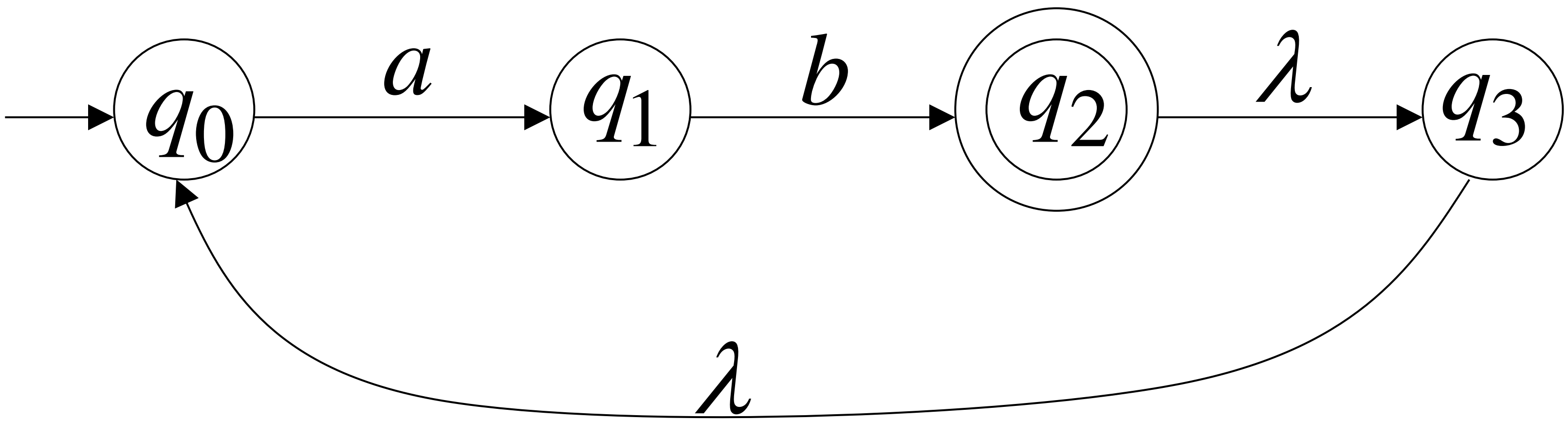

a
b
a
b
a
b
λ
q0
q1
q2
q3
λ


a
b
a
b
"accept"
a
b
λ
λ
q0
q1
q2
q3

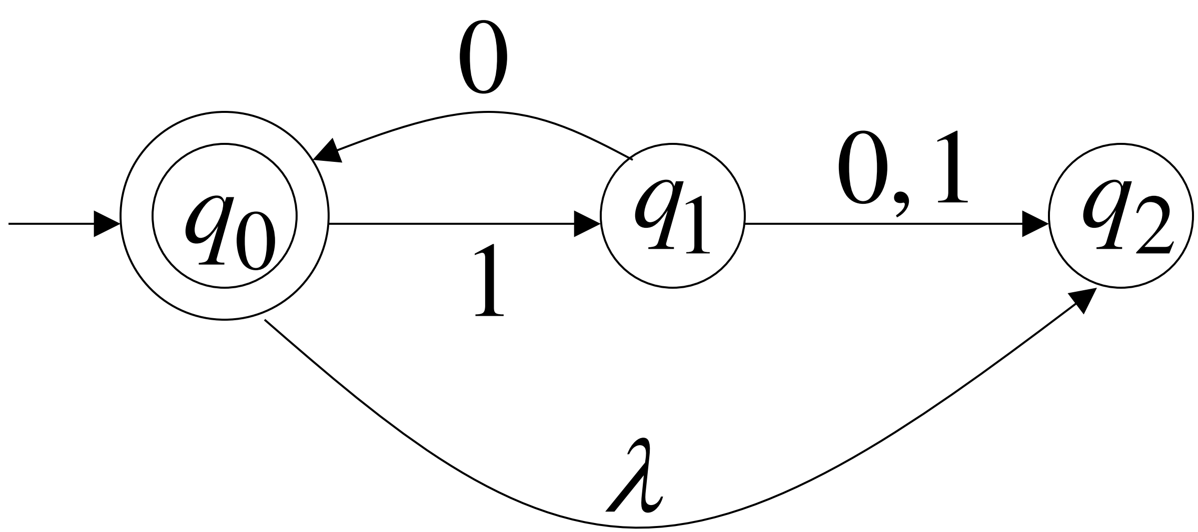
Language accepted

$$L = \{ab,\ abab,\ ababab,\ \ldots\}$$

$$= \{ab\}^{+}$$



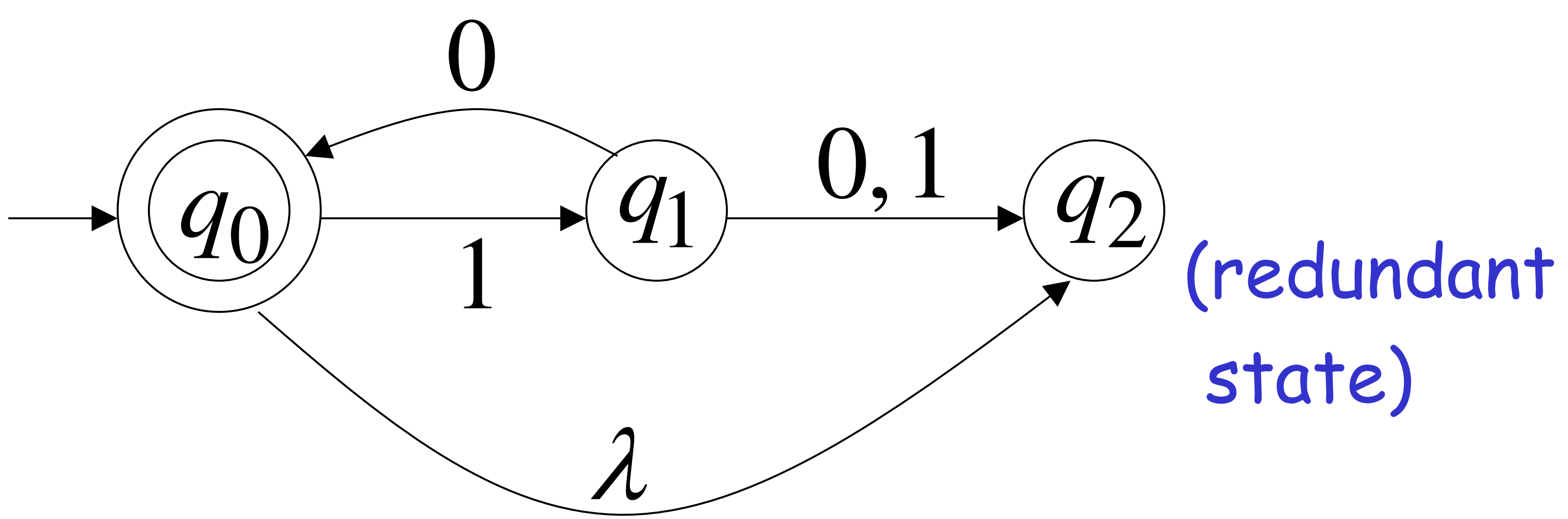
Another NFA Example



Language accepted

$$L(M) = \{\lambda,\ 10,\ 1010,\ 101010,\ \ldots\}$$

$$= \{10\}^*$$

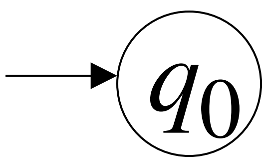


Remarks:

- The λ symbol never appears on the input tape

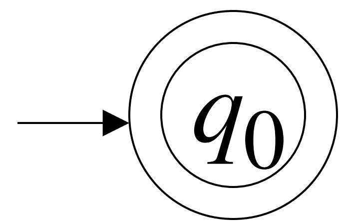
- Simple automata:

M_1



$L(M_1) = \{\ \}$

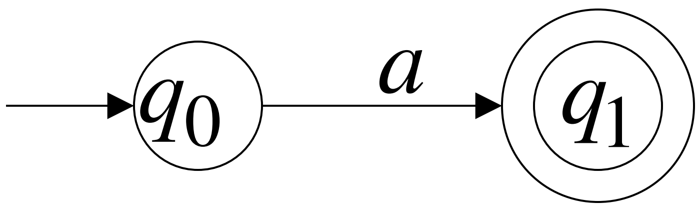
M_2



$L(M_2) = \{\lambda\}$

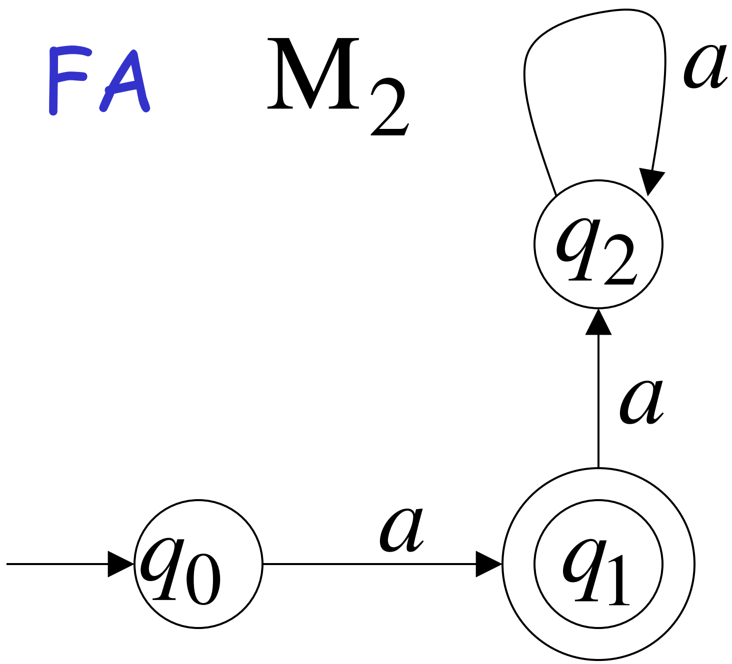
- NFAs are interesting because we can express languages easier than FAs

NFA M_1



$$L(M_1) = \{a\}$$

FA M_2



$$L(M_2) = \{a\}$$

Formal Definition of NFAs

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

$Q:$ Set of states, i.e. $\{q_0, q_1, q_2\}$

$\Sigma:$ Input aplhabet, i.e. $\{a, b\}$

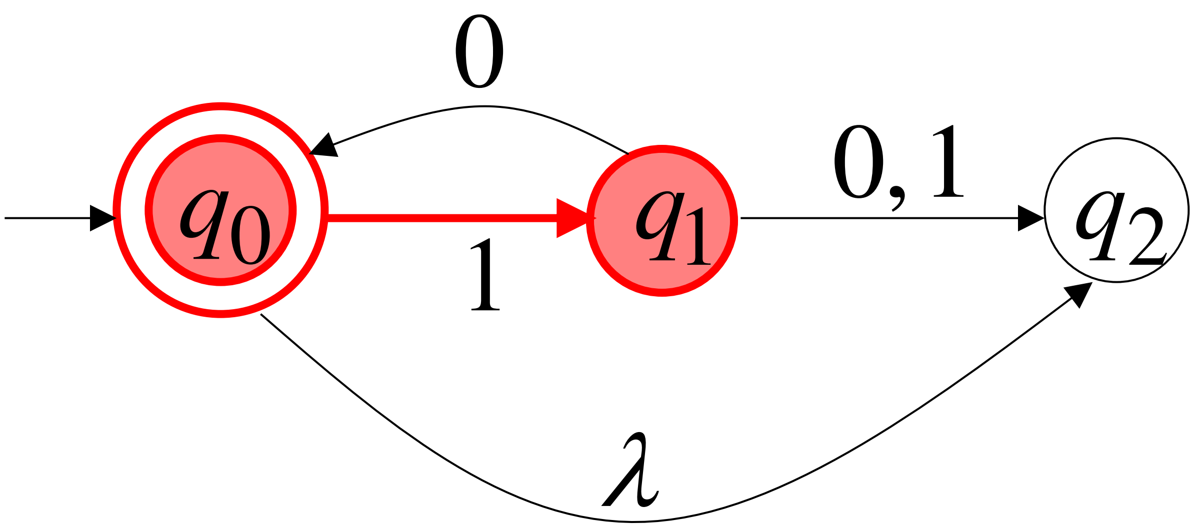
$\delta:$ Transition function

$q_0:$ Initial state

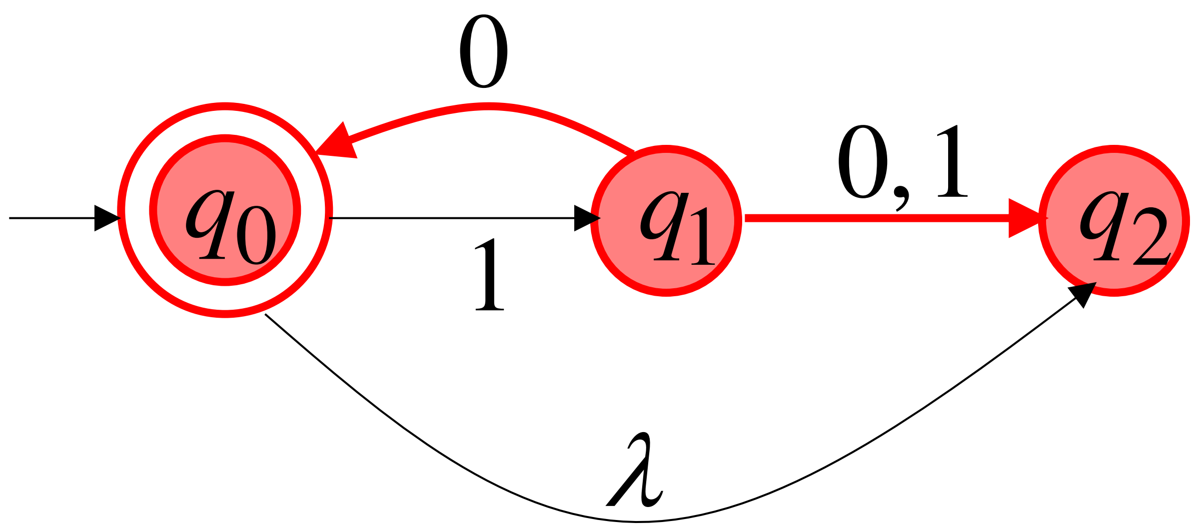
$F:$ Accepting states

Transition Function δ

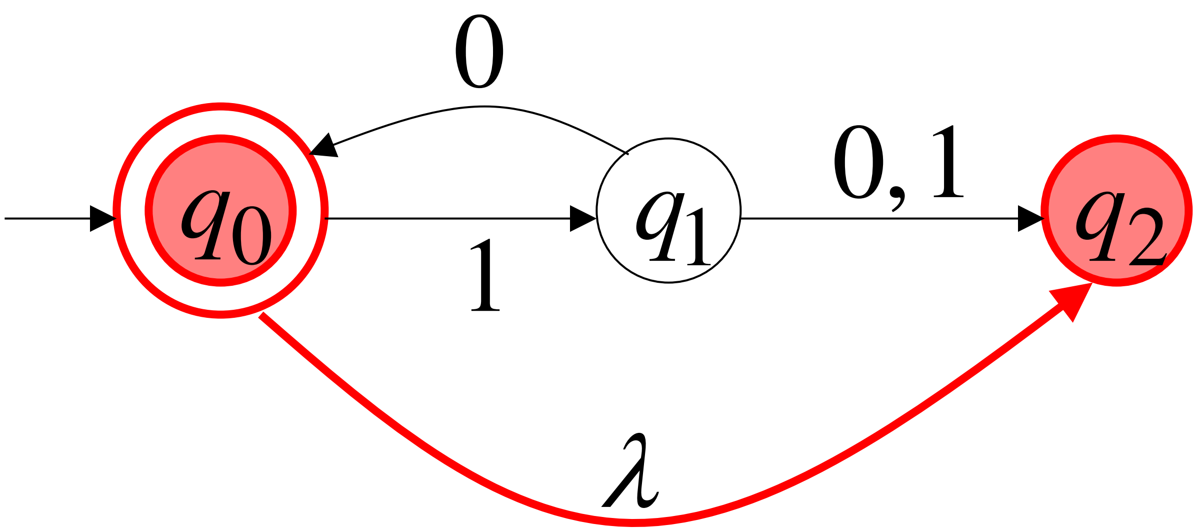
$$\delta(q_0, 1) = \{q_1\}$$



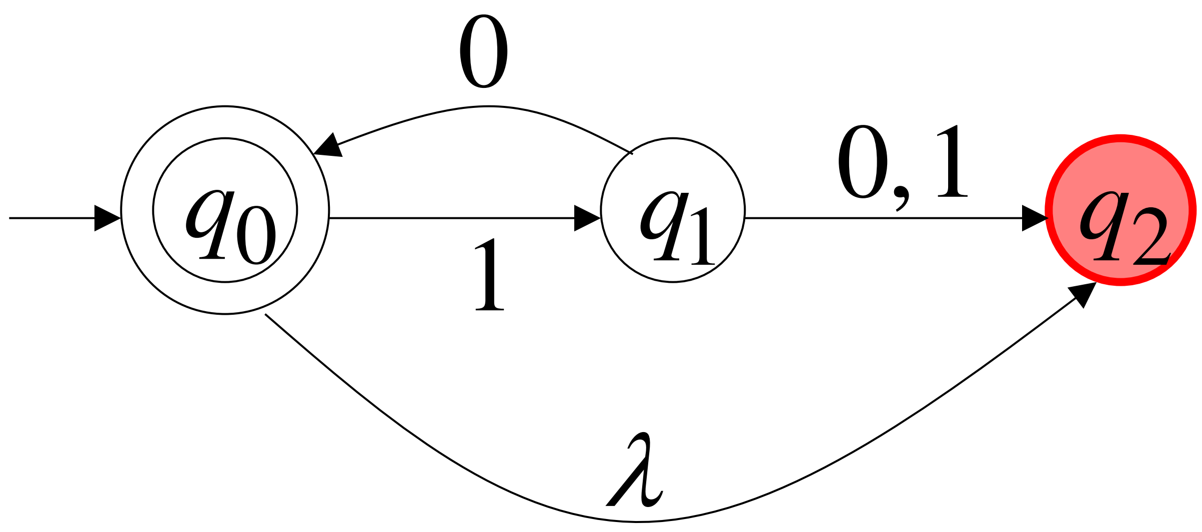
$$\delta(q_1, 0) = \{q_0, q_2\}$$



$$\delta(q_0, \lambda) = \{q_0, q_2\}$$

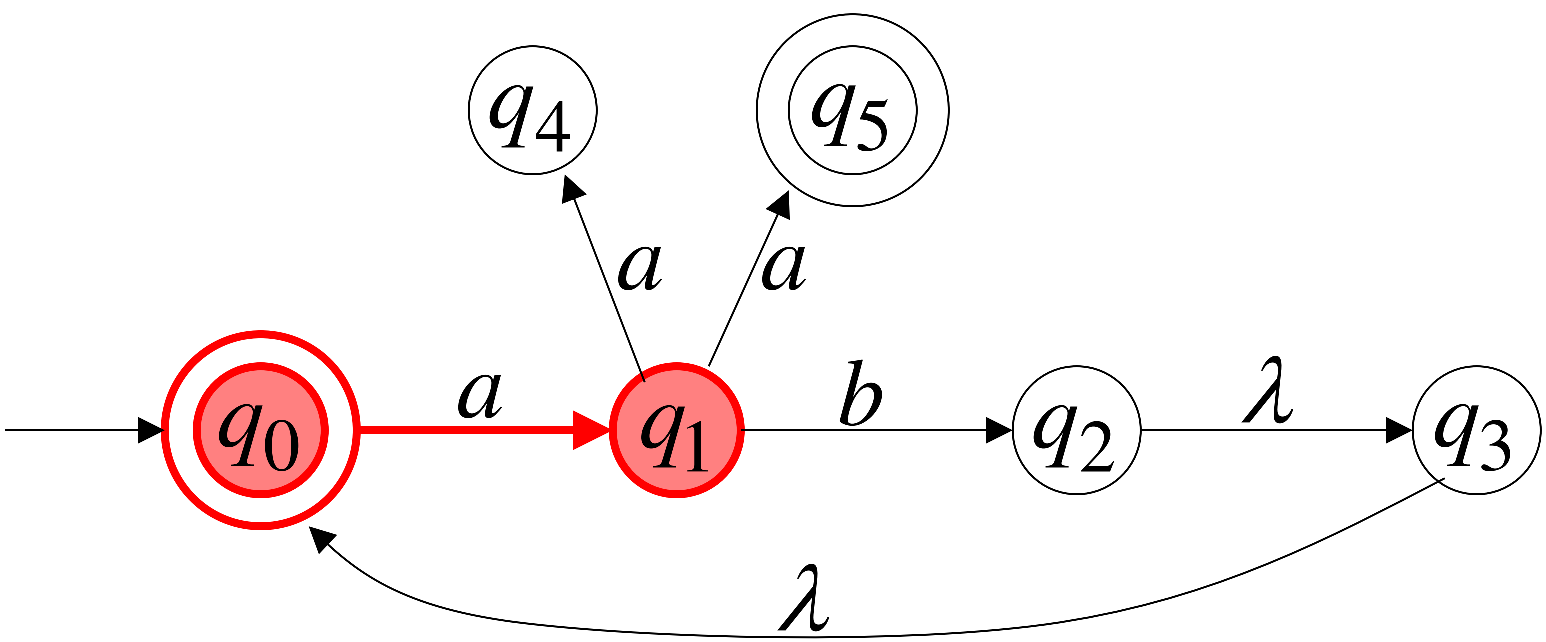


$$\delta(q_2, 1) = \varnothing$$

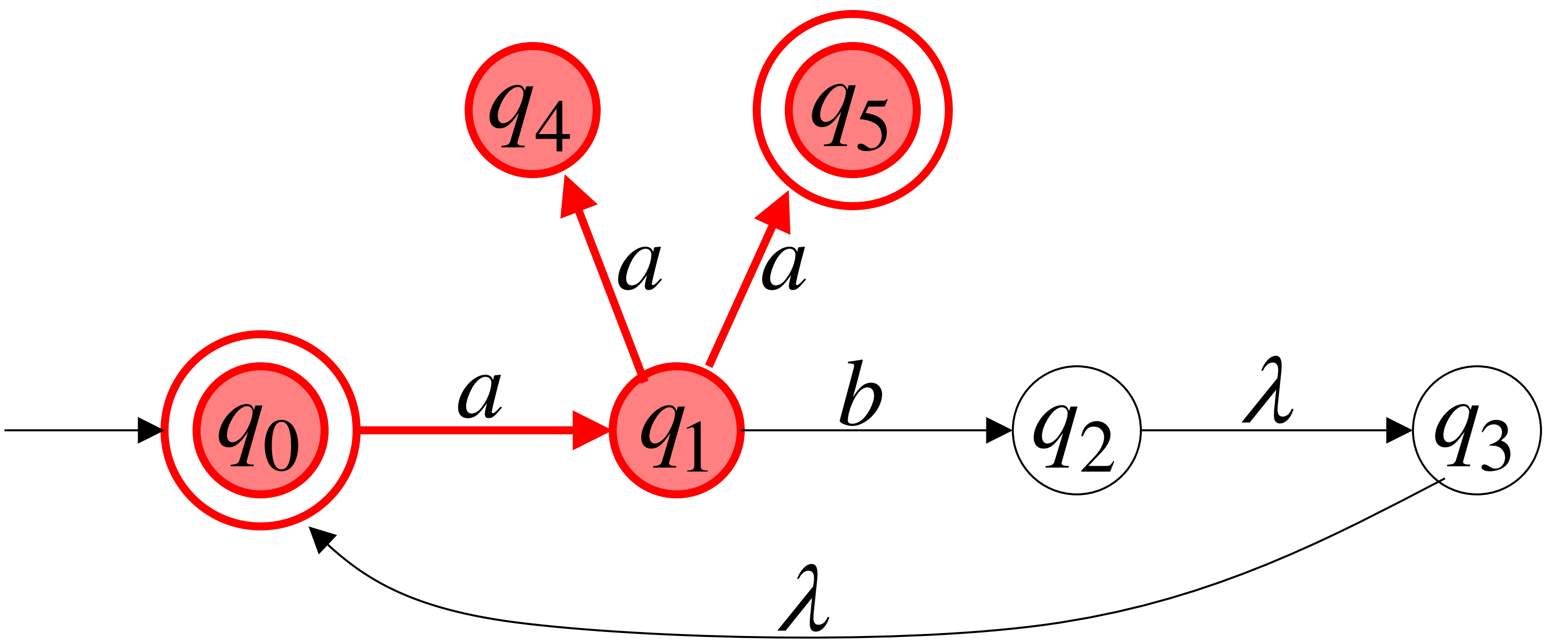


Extended Transition Function δ^*

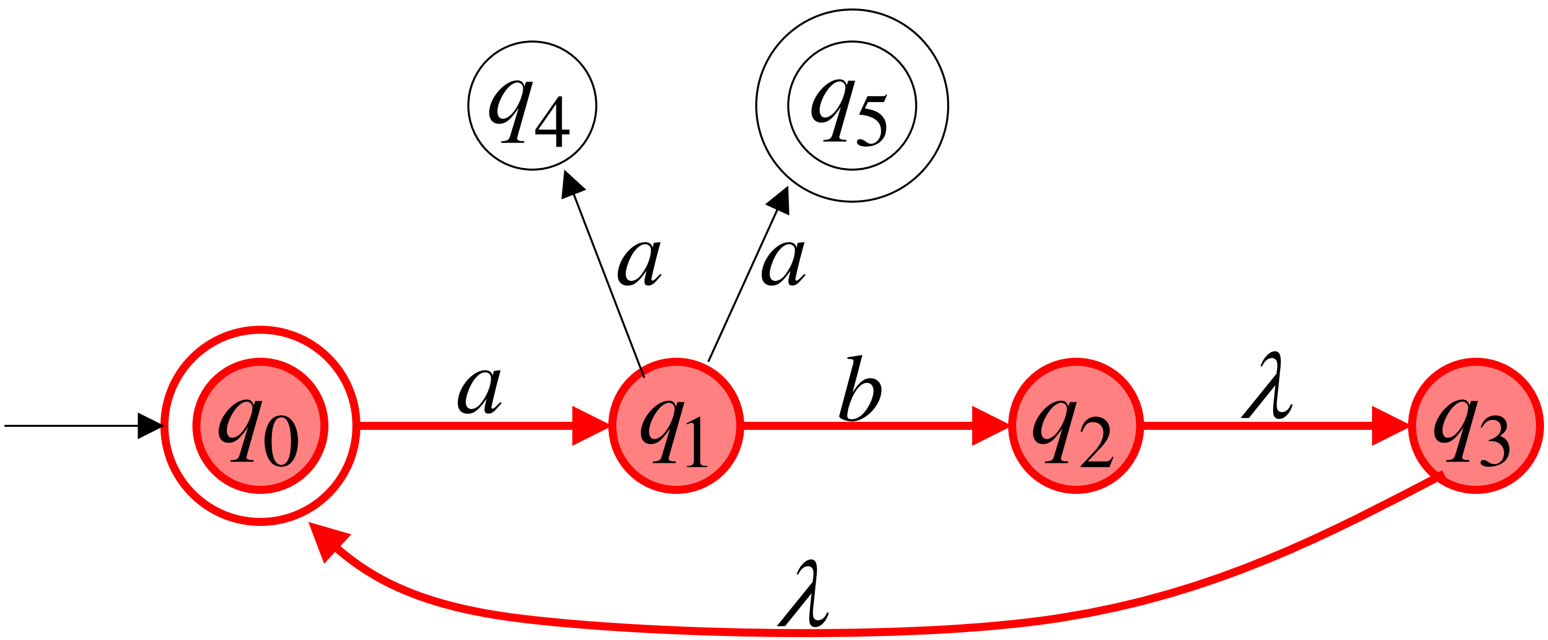
$$\delta^*(q_0, a) = \{q_1\}$$



$$\delta^*(q_0, aa) = \{q_4, q_5\}$$

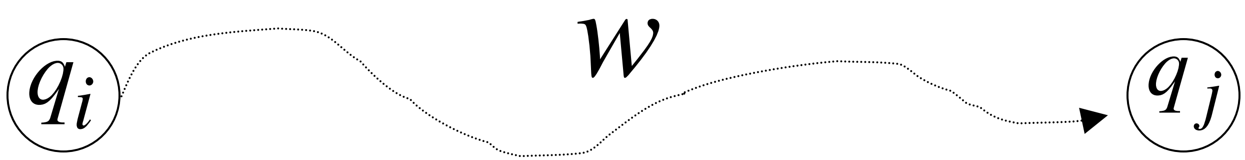


$$\delta^*(q_0, ab) = \{q_2, q_3, q_0\}$$

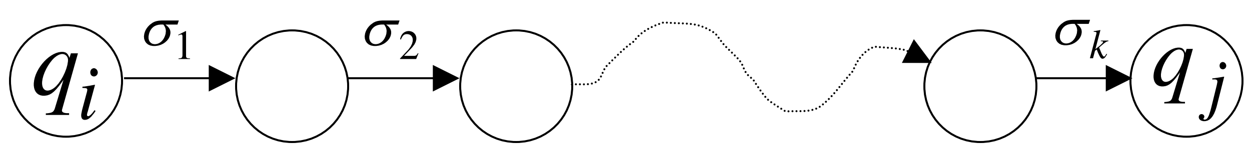


Formally

$q_j \in \delta^*(q_i, w)$: there is a walk from q_i to q_j with label w

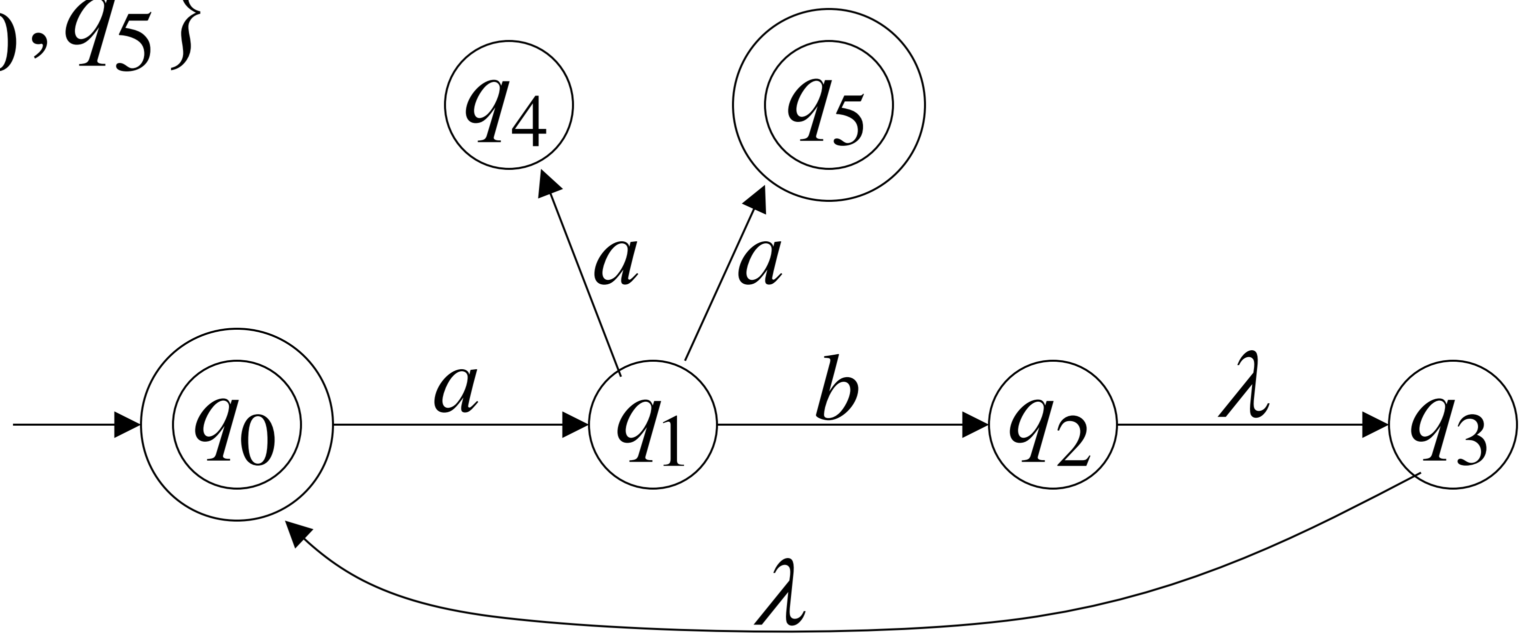


$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



The Language of an NFA M

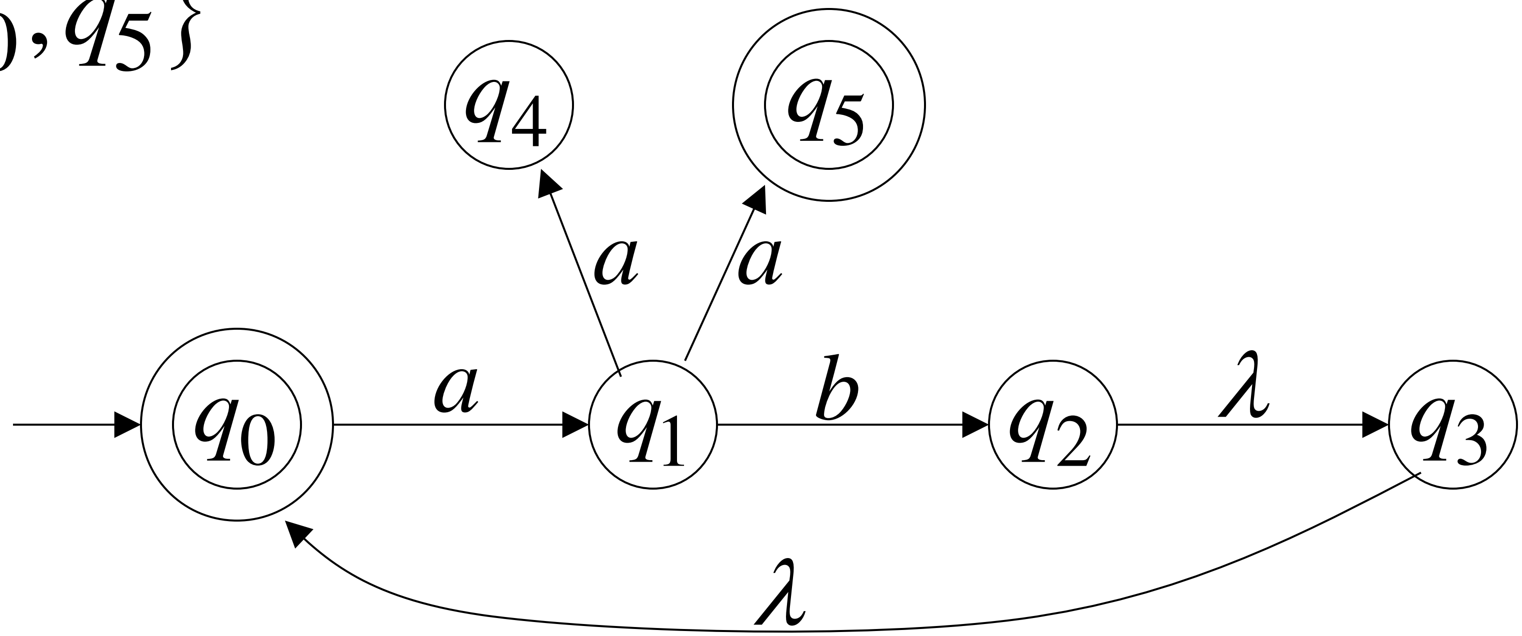
$F = \{q_0, q_5\}$



$$\delta^*(q_0, aa) = \{q_4, \underline{q_5}\} \qquad aa \in L(M)$$

$$\searrow \in F$$

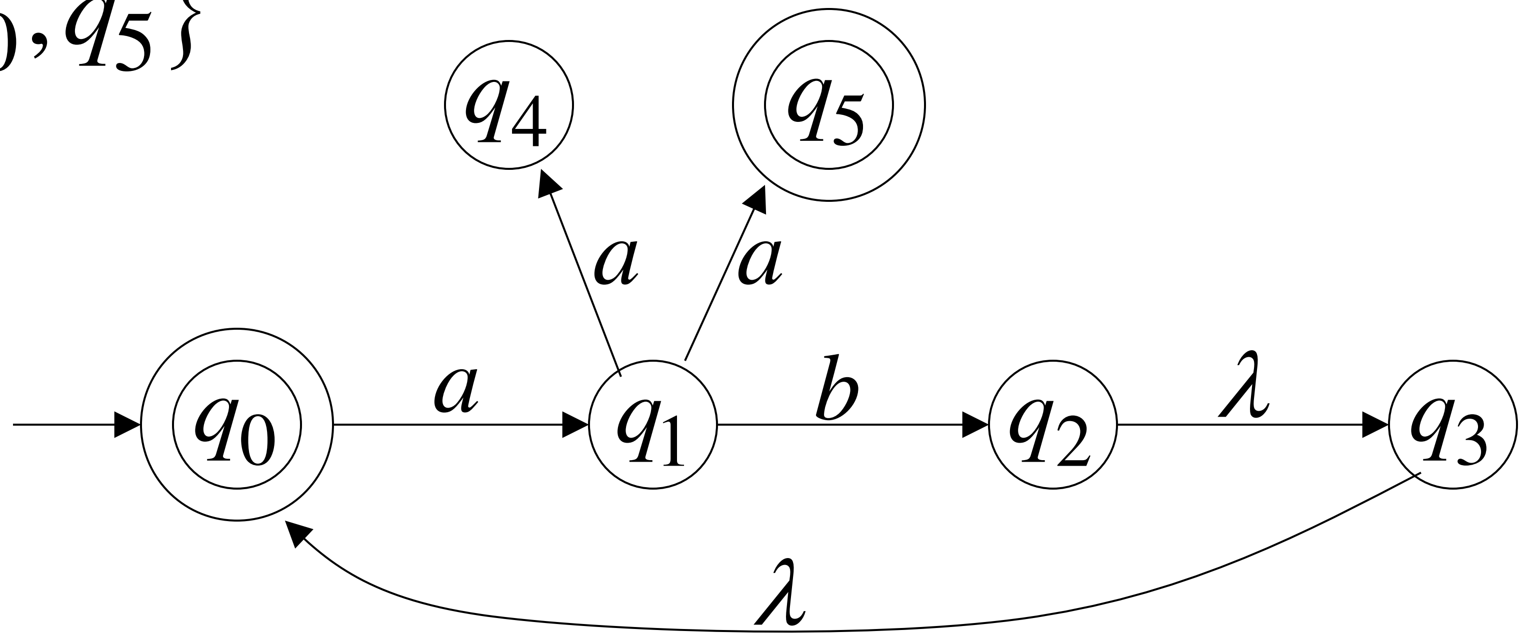
$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, ab) = \{q_2, q_3, \underline{q_0}\} \qquad ab \in L(M)$$

$$\in F$$

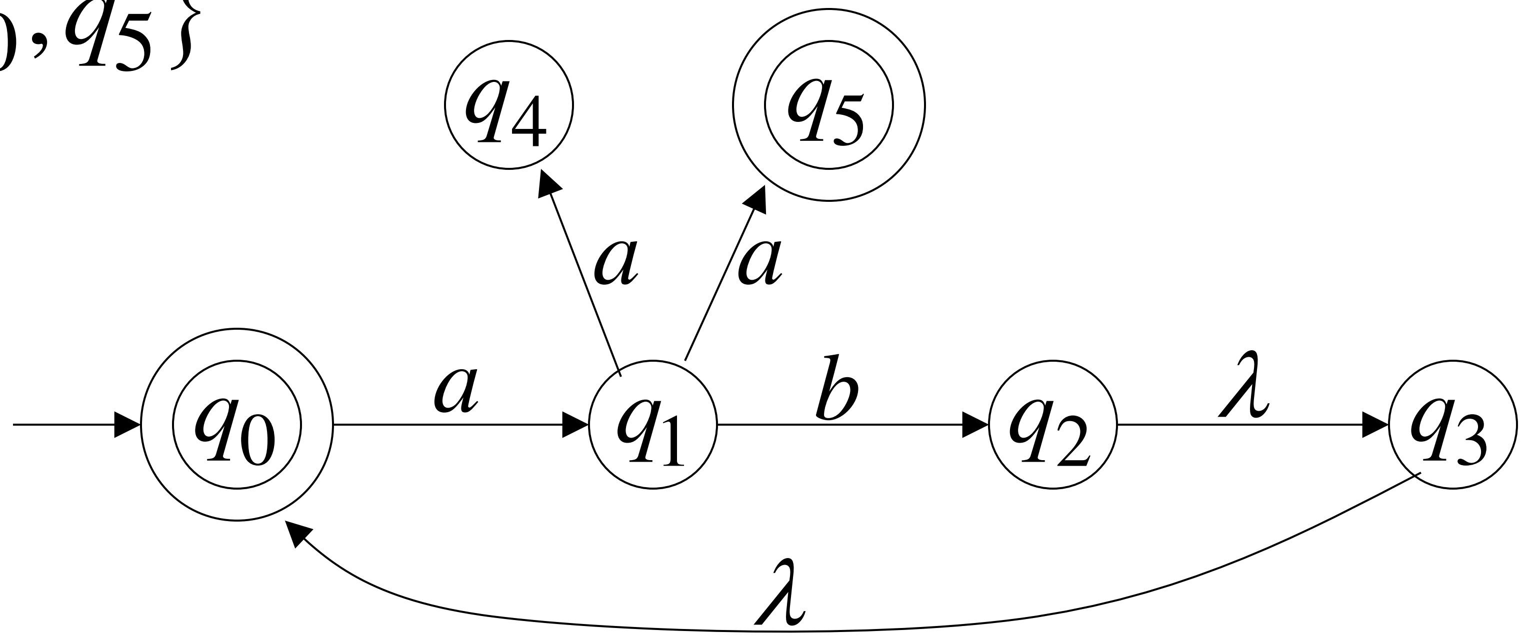
$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, abaa) = \{q_4, \underline{q_5}\} \qquad aaba \in L(M)$$

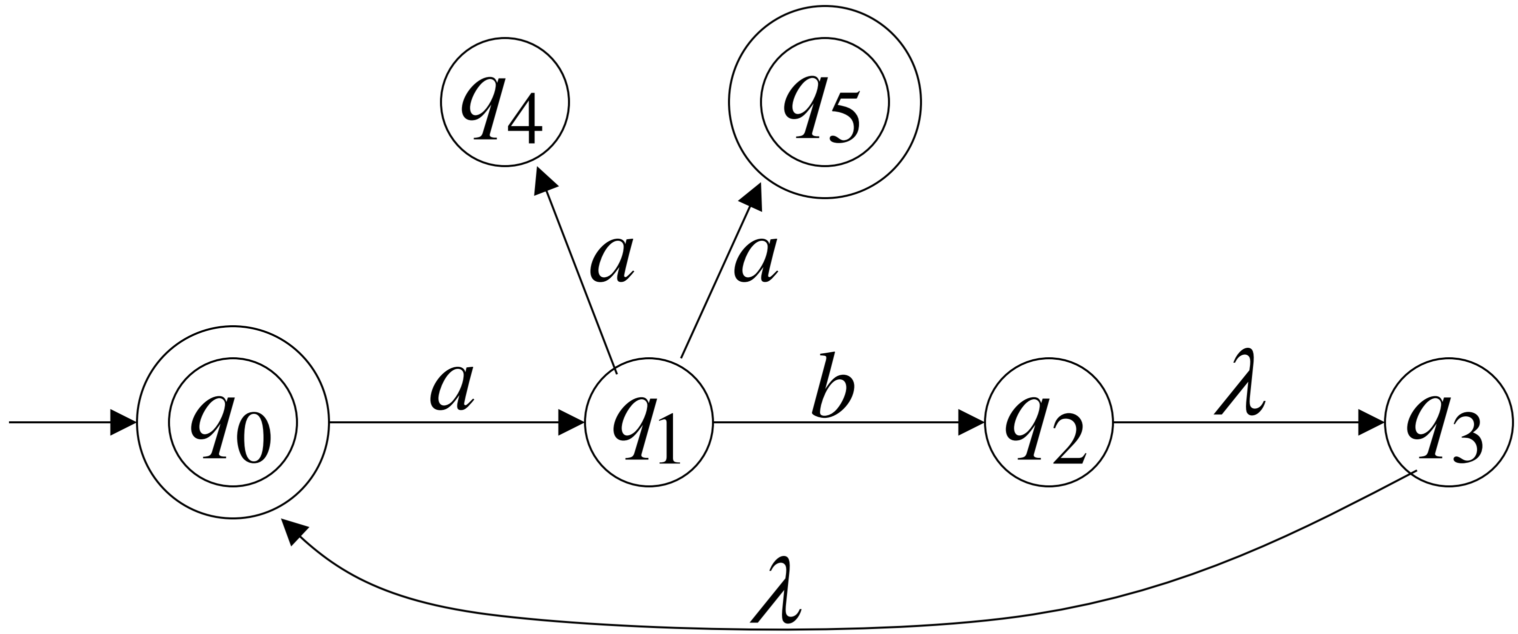
$$\in F$$

$F = \{q_0, q_5\}$



$\delta^*(q_0, aba) = \{q_1\}$ $\;\to\; \notin F$

$aba \notin L(M)$



$$L(M) = \{\lambda\} \cup \{ab\}^* \{aa\}$$

Formally

The language accepted by NFA M is:

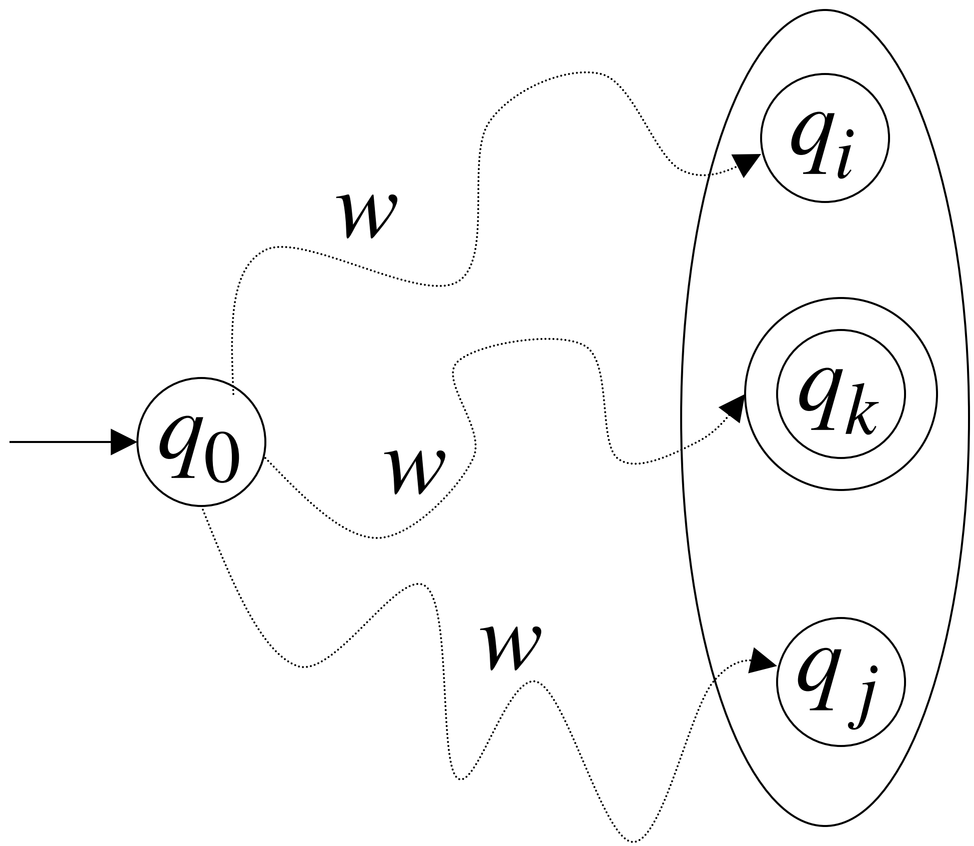
$$L(M) = \{w_1, w_2, w_3, \ldots\}$$

where $\delta^*(q_0, w_m) = \{q_i, q_j, \ldots, q_k, \ldots\}$

and there is some $q_k \in F$ (accepting state)

$w \in L(M)$

$\delta^*(q_0, w)$



$q_k \in F$

14B11CI171

Theory of Computation

NFAs accept the Regular Languages

Equivalence of Machines

Definition:

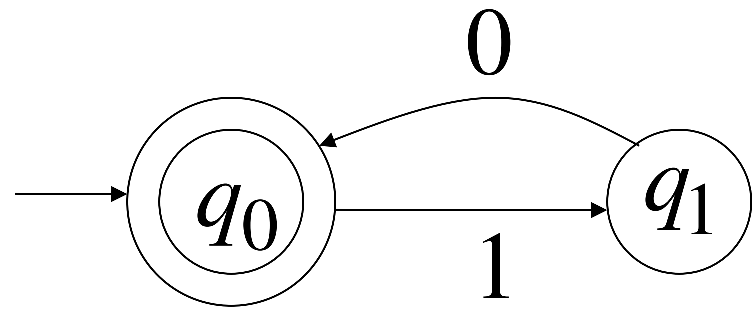
Machine M_1 is equivalent to machine M_2

if $L(M_1) = L(M_2)$

Example of equivalent machines

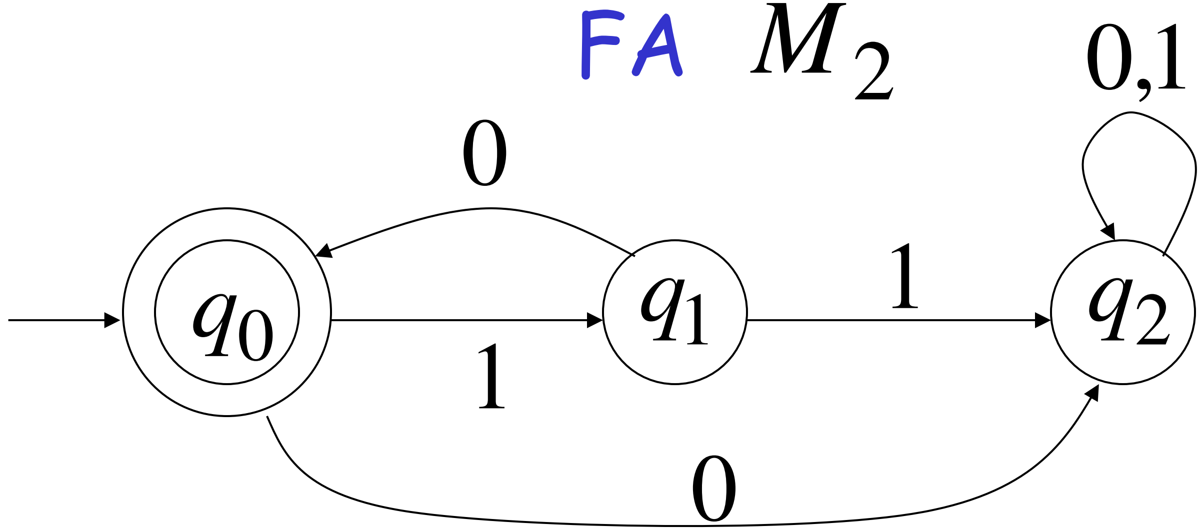
NFA M_1

$$L(M_1) = \{10\}^*$$



FA M_2

$$L(M_2) = \{10\}^*$$



We will prove:

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{accepted} \\ \text{by NFAs} \end{array} \right\} = \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Languages
accepted
by FAs

NFAs and FAs have the same computation power

We will show:

$$\left\{\begin{array}{l}\text{Languages}\\ \text{accepted}\\ \text{by NFAs}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

$$\left\{\begin{array}{l}\text{Languages}\\ \text{accepted}\\ \text{by NFAs}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Proof-Step 1

$$\left\{\begin{array}{l}\text{Languages} \\ \text{accepted} \\ \text{by NFAs}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular} \\ \text{Languages}\end{array}\right\}$$

Proof: Every FA is trivially an NFA

⇩

Any language L accepted by a FA is also accepted by an NFA

Proof-Step 2

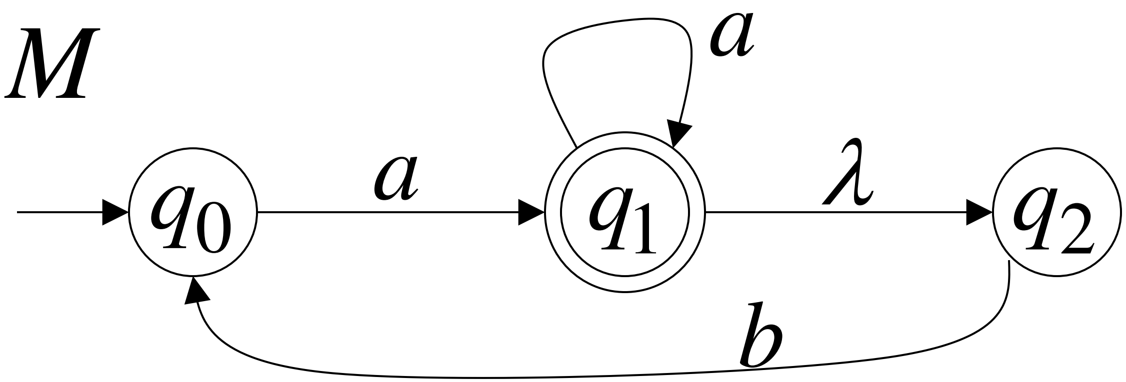
$$\left\{ \begin{array}{l} \text{Languages} \\ \text{accepted} \\ \text{by NFAs} \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Proof: Any NFA can be converted to an equivalent FA

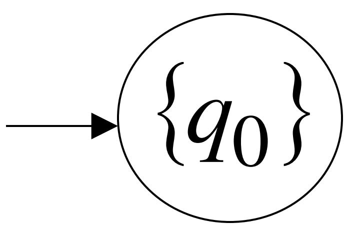
Any language L accepted by an NFA is also accepted by a FA

Convert NFA to FA

NFA M

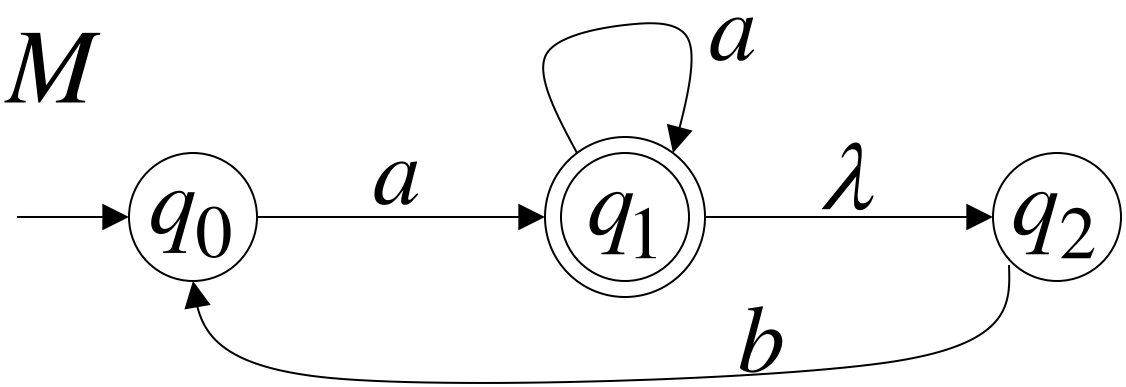


FA M'

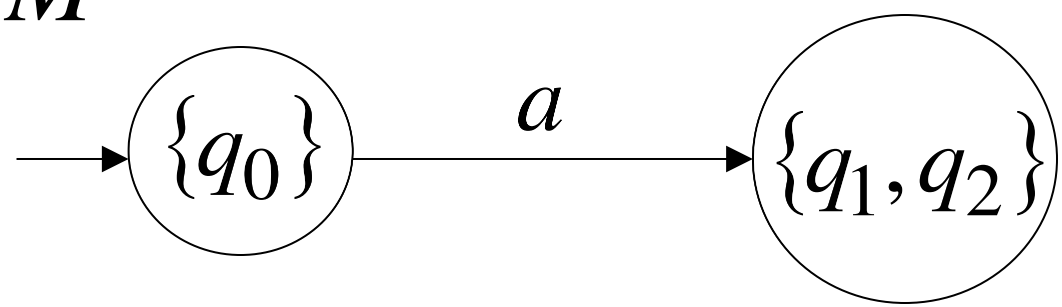


Convert NFA to FA

NFA M

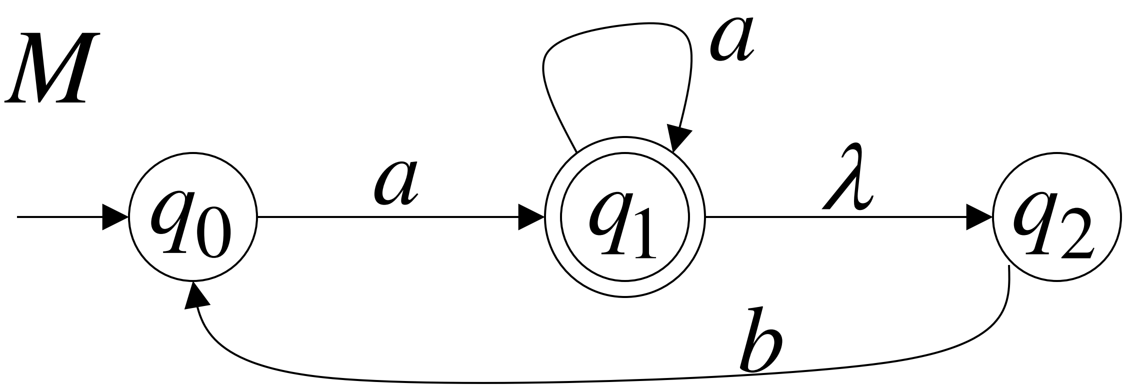


FA M'

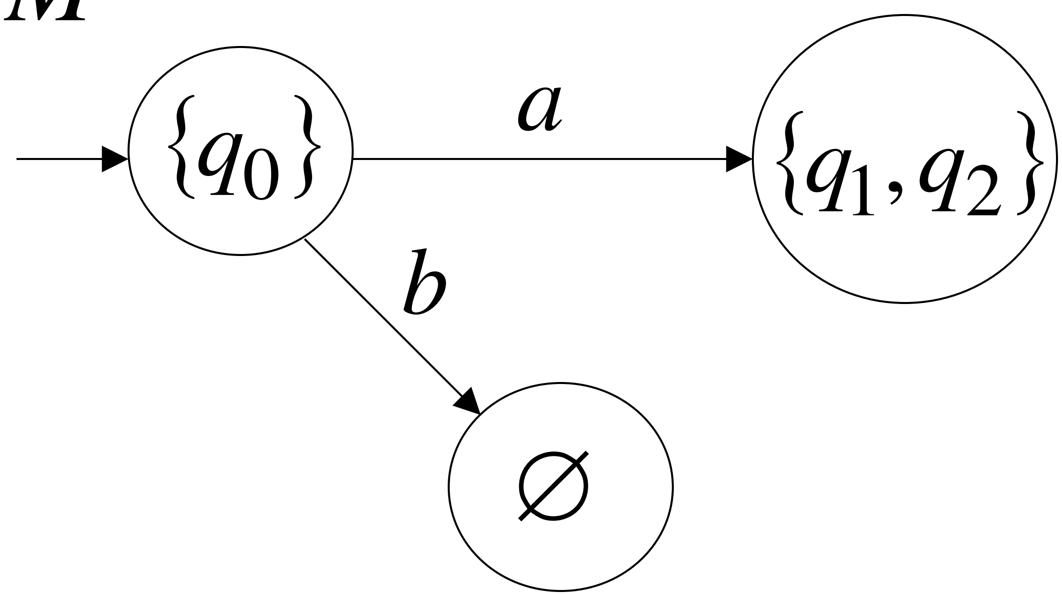


Convert NFA to FA

NFA M

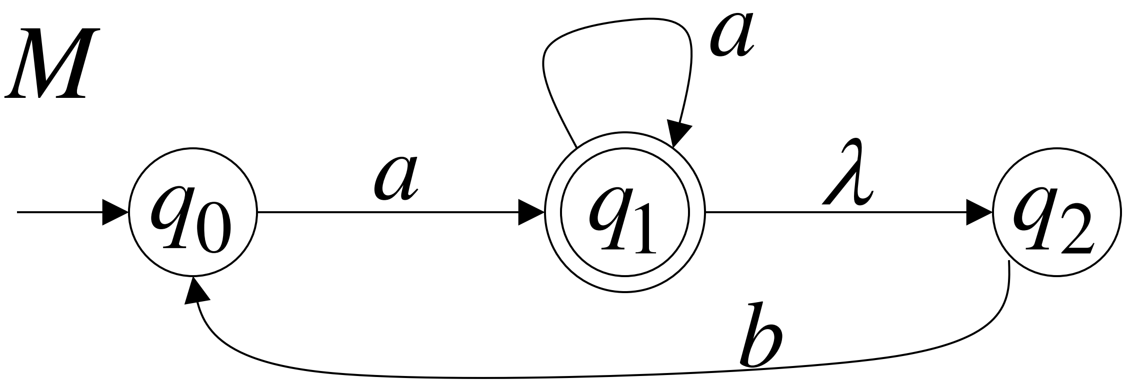


FA M'

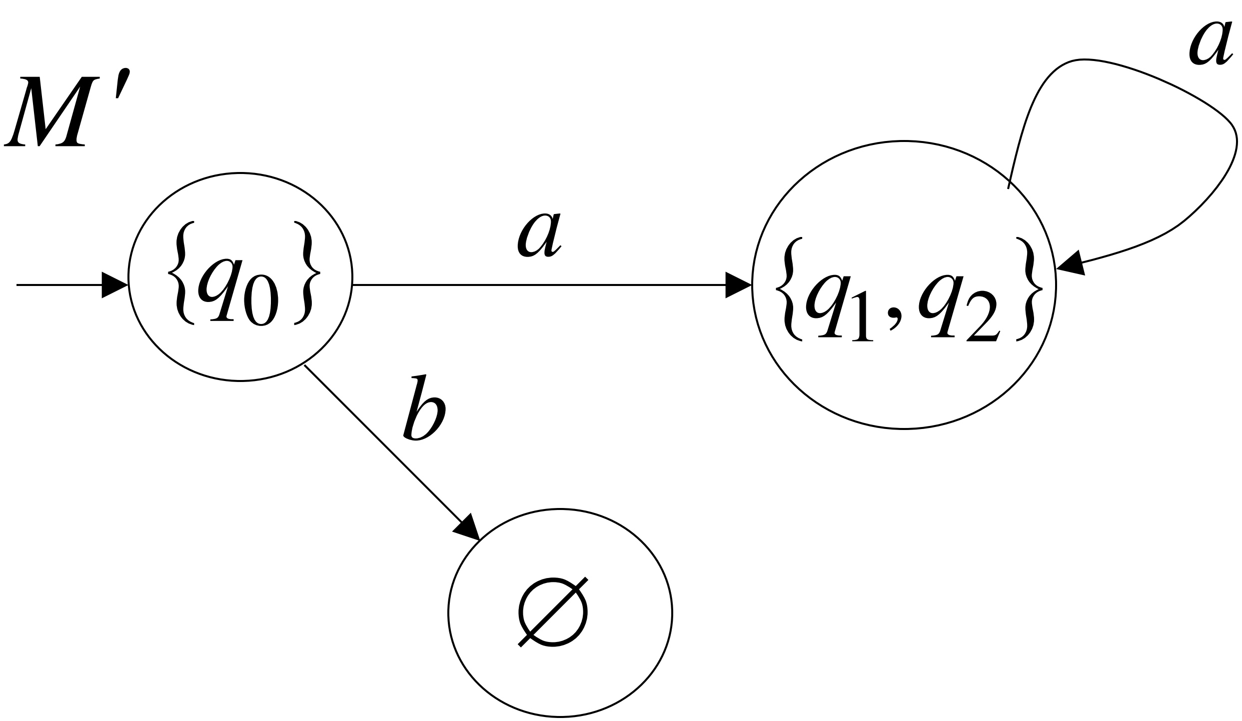


Convert NFA to FA

NFA M

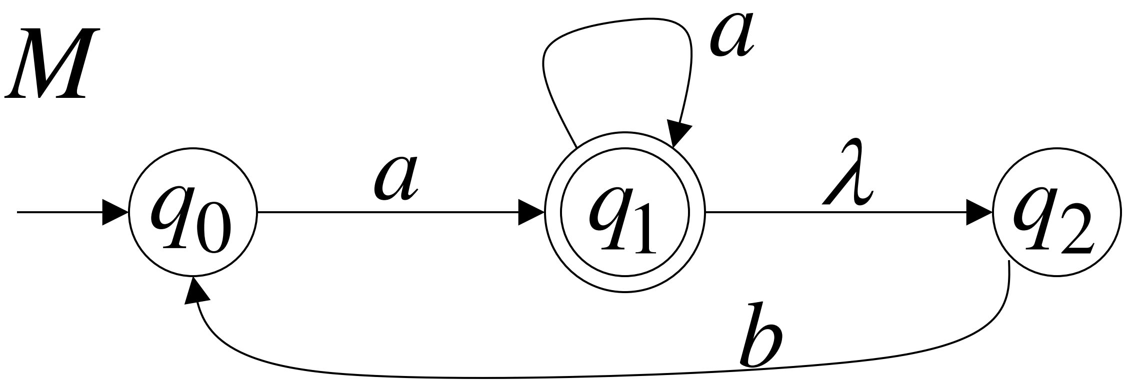


FA M'

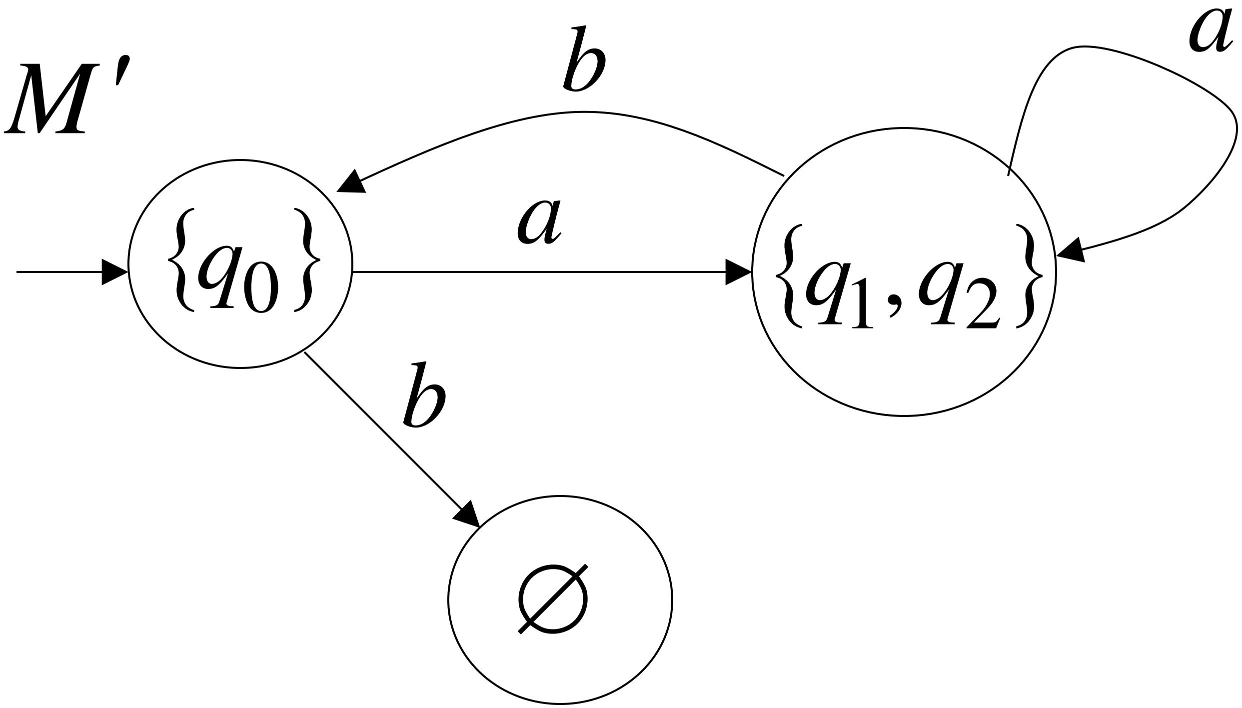


Convert NFA to FA

NFA M

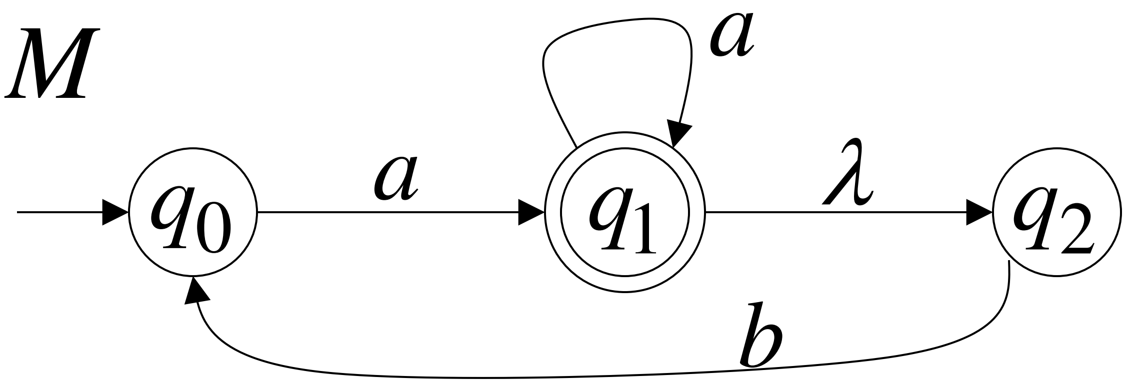


FA M'

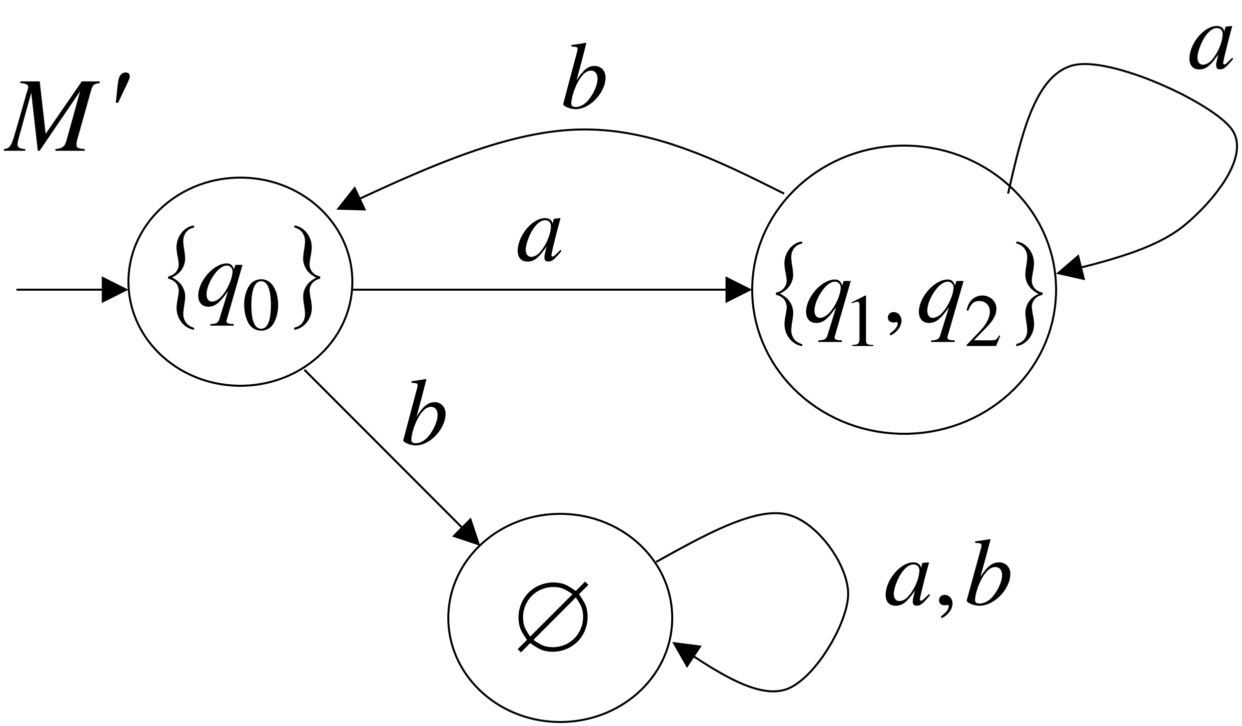


Convert NFA to FA

NFA M

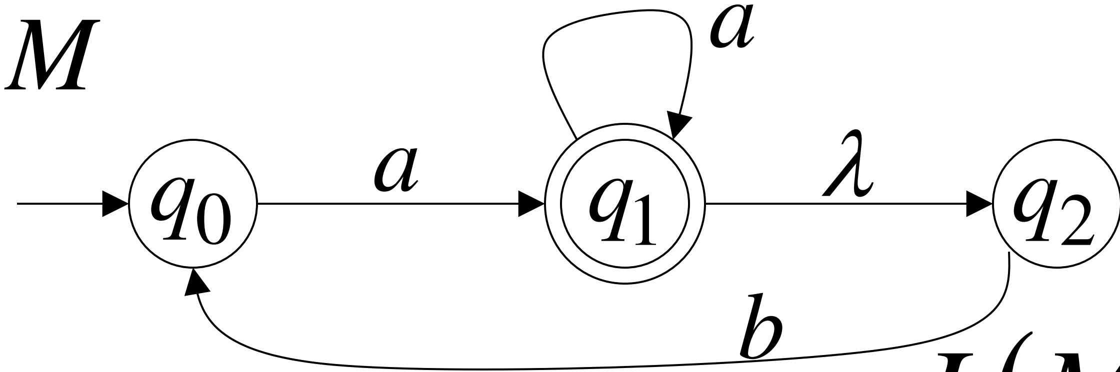


FA M'



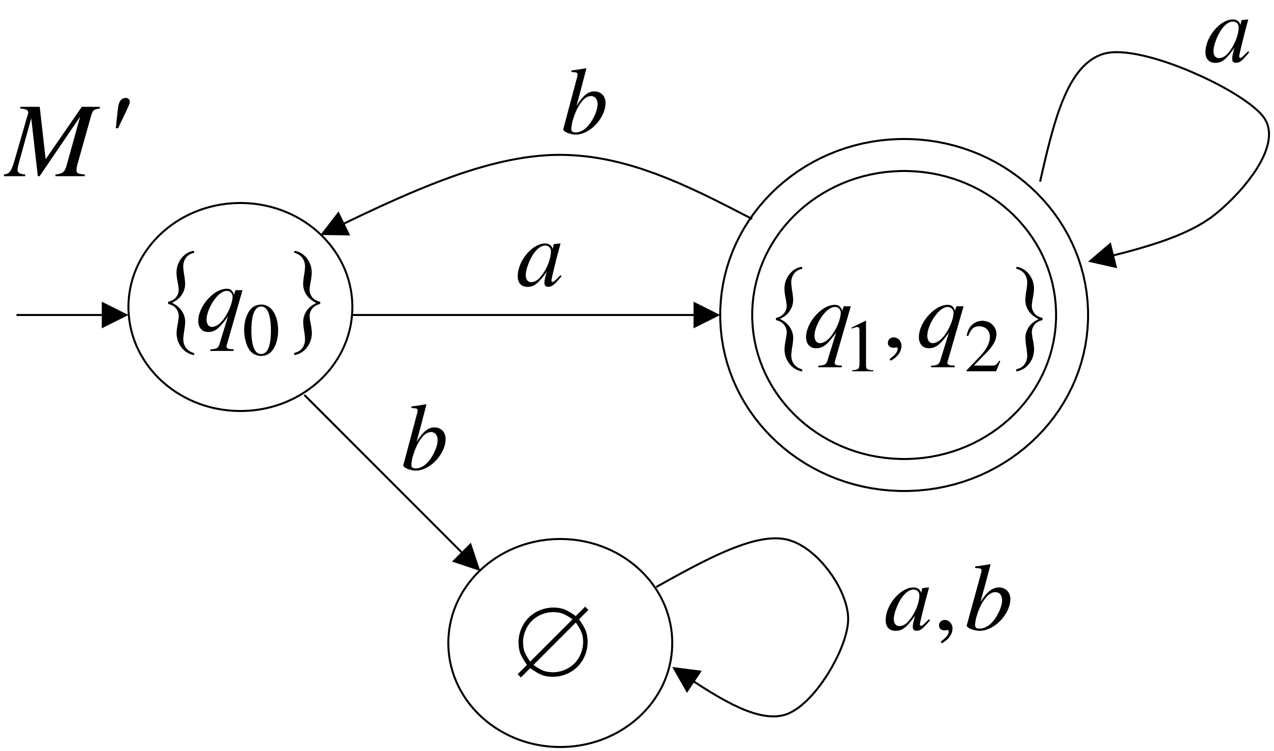
Convert NFA to FA

NFA M



$$L(M) = L(M')$$

FA M'



NFA to FA: Remarks

We are given an NFA M

We want to convert it
to an equivalent FA M'

With $L(M) = L(M')$

If the NFA has states

$$q_0, q_1, q_2, \ldots$$

the FA has states in the powerset

$$\emptyset, \{q_0\}, \{q_1\}, \{q_1, q_2\}, \{q_3, q_4, q_7\}, \ldots .$$

Procedure NFA to FA

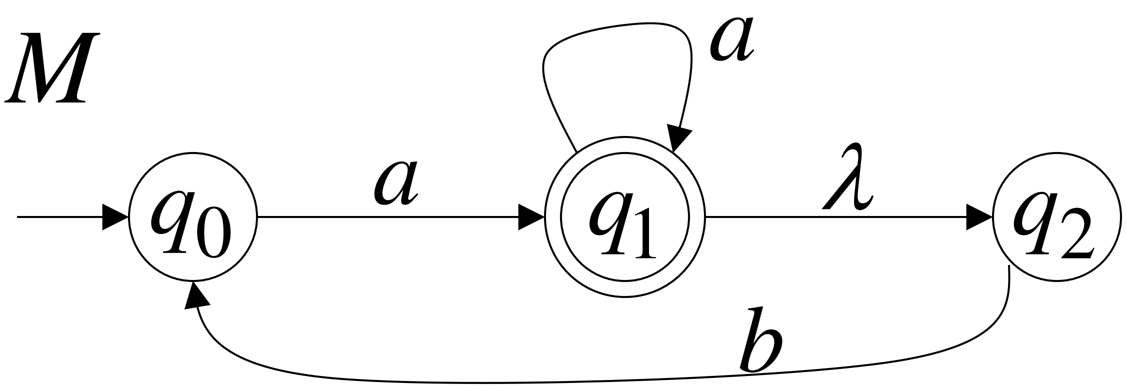
1. Initial state of NFA: q_0

$\Downarrow$

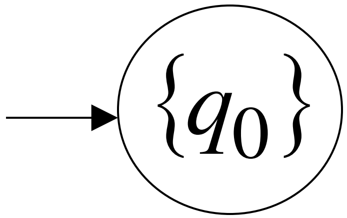
Initial state of FA: $\{q_0\}$

Example

NFA M



FA M'



Procedure NFA to FA

2. For every FA's state $\{q_i, q_j, \ldots, q_m\}$

Compute in the NFA

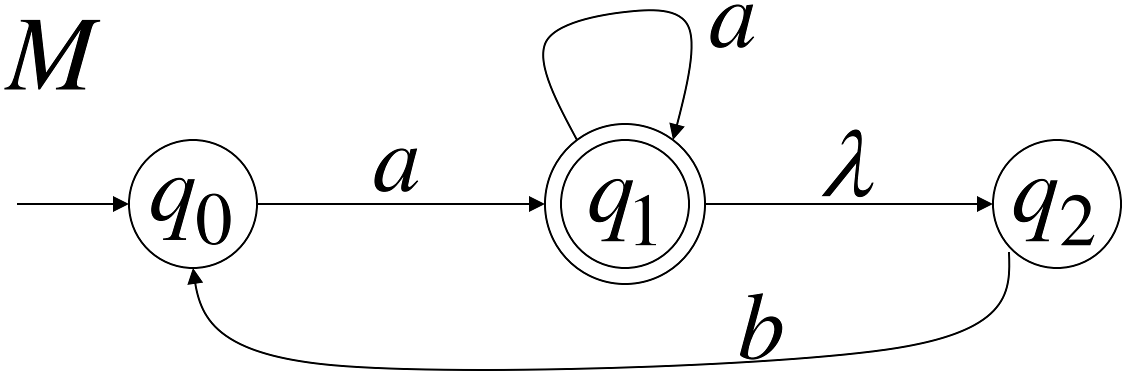
$$\left.\begin{array}{l} \delta^*(q_i, a), \\ \delta^*(q_j, a), \\ \ldots \end{array}\right\} = \{q'_i, q'_j, \ldots, q'_m\}$$

Add transition to FA

$$\delta(\{q_i, q_j, \ldots, q_m\},\ a) = \{q'_i, q'_j, \ldots, q'_m\}$$

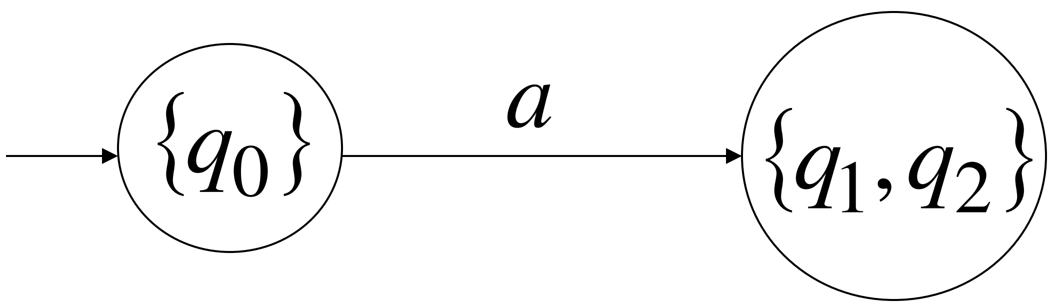
Exampe

NFA M



$$\delta^*(q_0, a) = \{q_1, q_2\}$$

FA M'



$$\delta(\{q_0\}, a) = \{q_1, q_2\}$$

Procedure NFA to FA

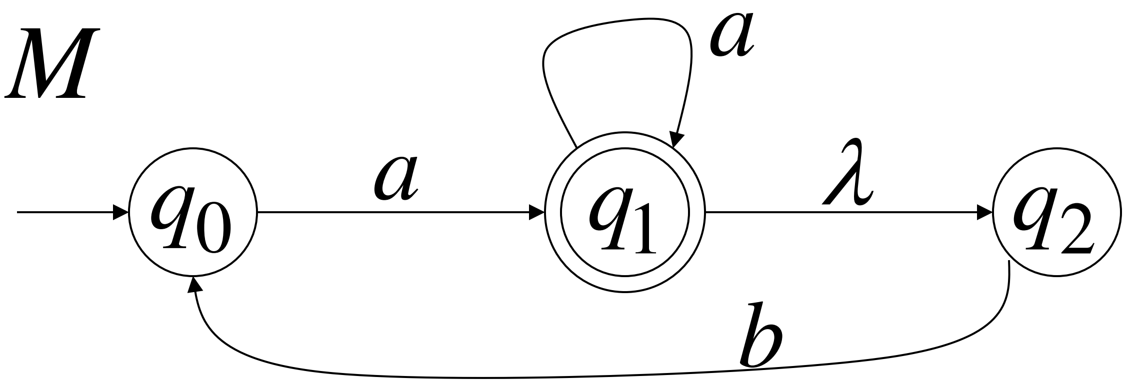
Repeat Step 2 for all letters in alphabet,

until

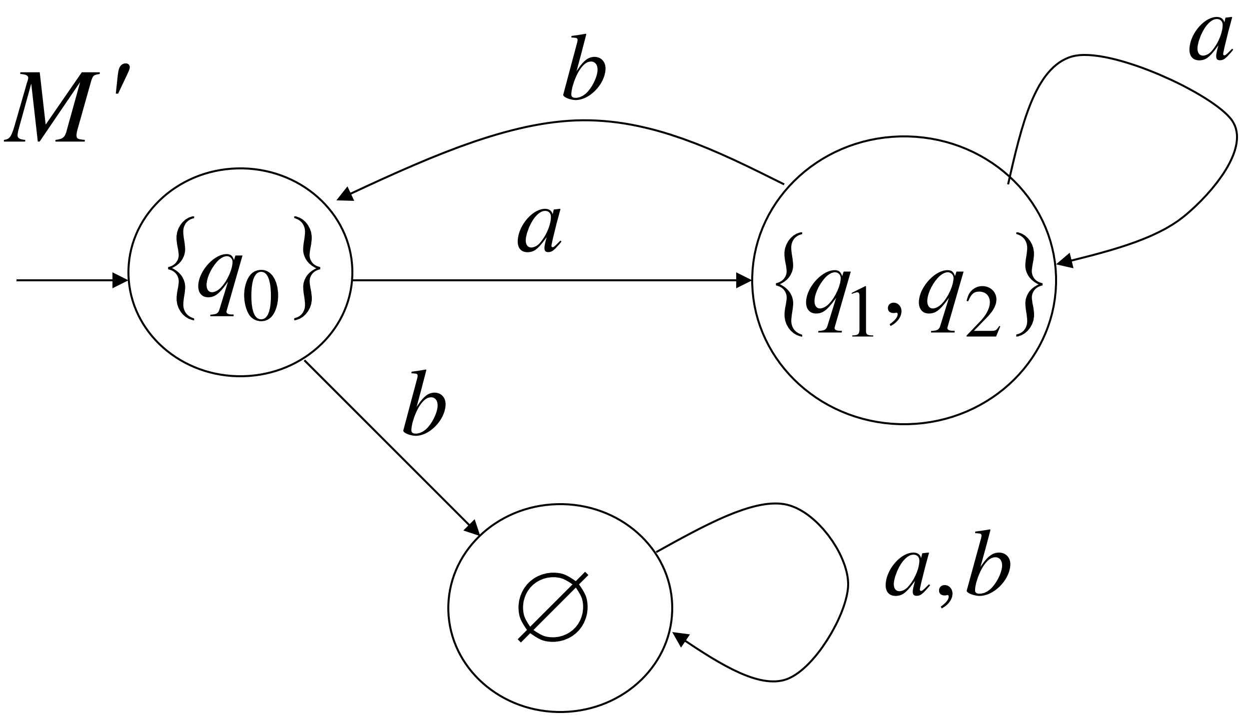
no more transitions can be added.

Example

NFA M



FA M'



Procedure NFA to FA

3. For any FA state $\{q_i, q_j, \ldots, q_m\}$

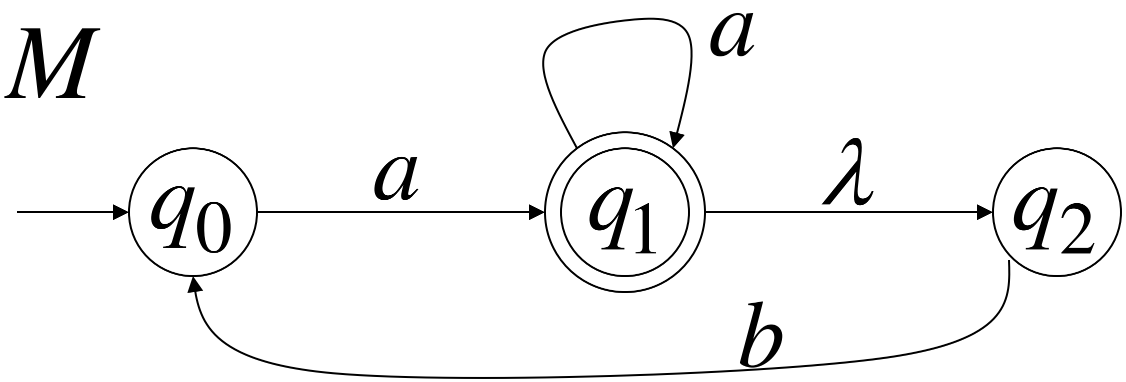
If q_j is accepting state in NFA

Then, $\{q_i, q_j, \ldots, q_m\}$

is accepting state in FA

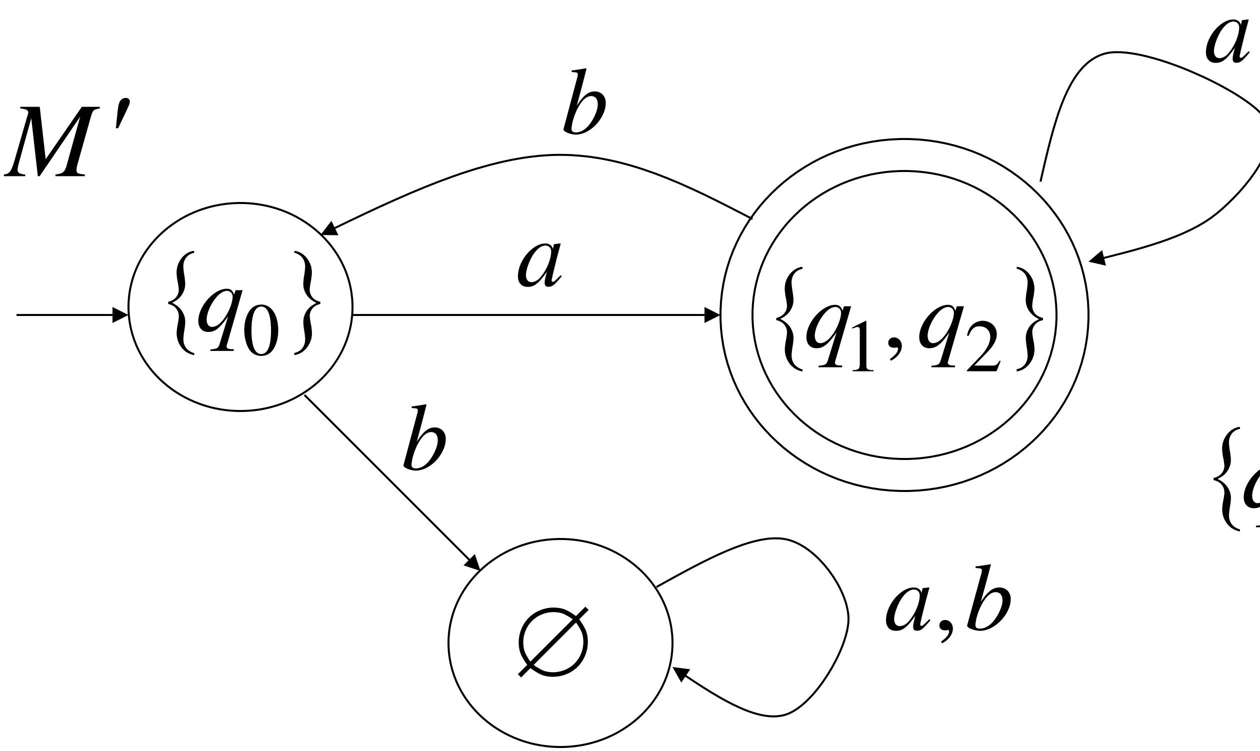
Example

NFA M



$q_1 \in F$

FA M'



$\{q_1, q_2\} \in F'$

Theorem

Take NFA M

Apply procedure to obtain FA M'

Then M and M' are equivalent :

$$L(M) = L(M')$$

Proof

$$L(M) = L(M')$$

$$\Updownarrow$$

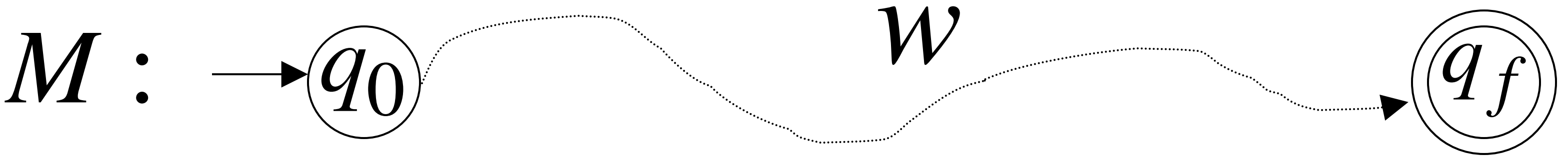
$$L(M) \subseteq L(M') \quad \text{AND} \quad L(M) \supseteq L(M')$$

First we show: $L(M) \subseteq L(M')$

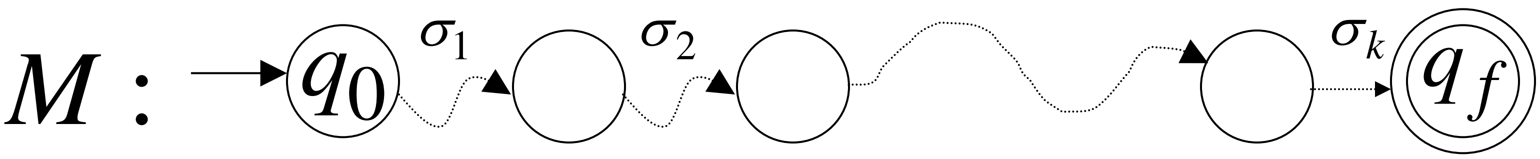
Take arbitrary: $w \in L(M)$

We will prove: $w \in L(M')$

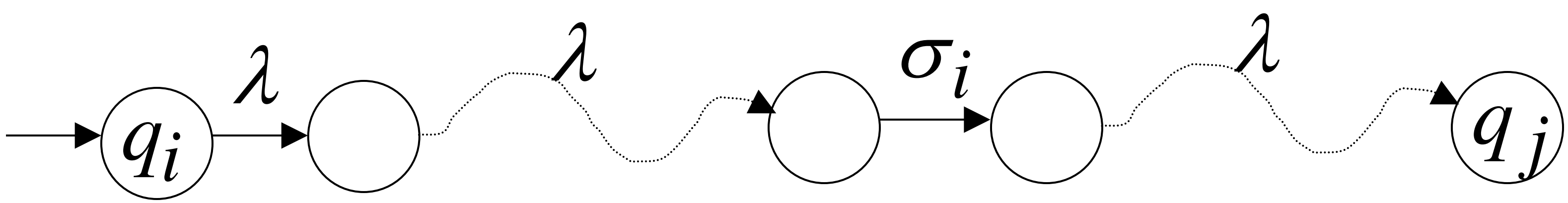
$$w \in L(M)$$



$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$

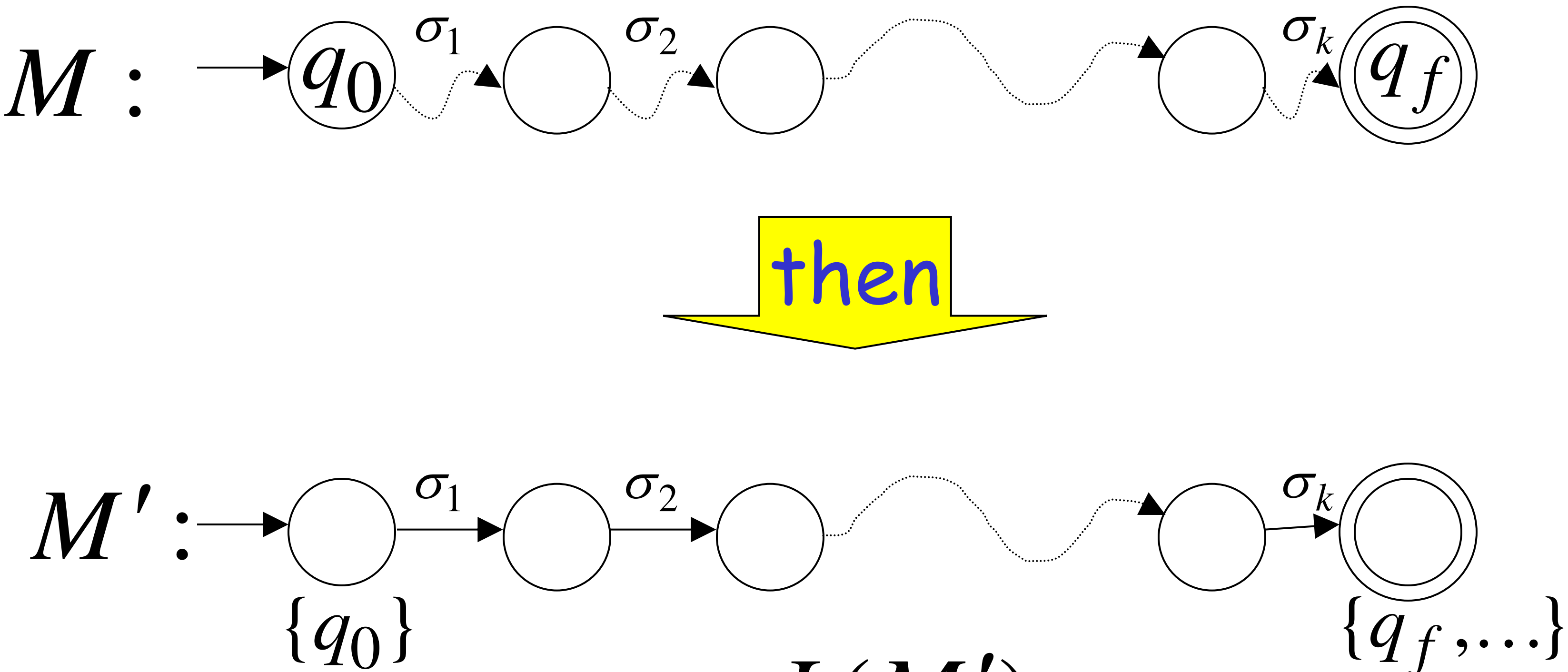


denotes



We will show that if $w \in L(M)$

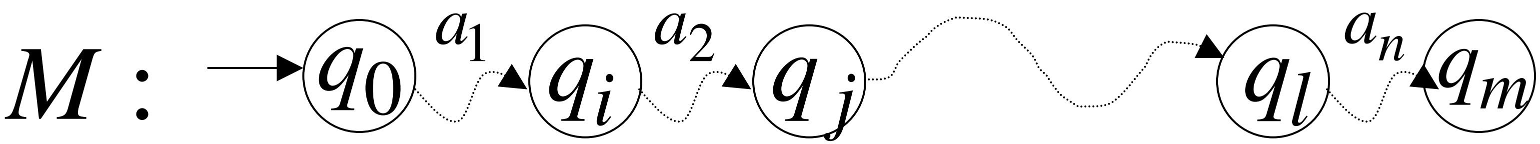
$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



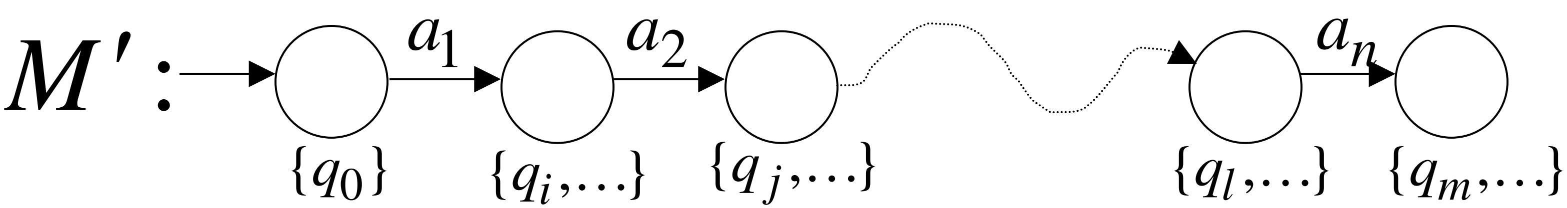
$$w \in L(M')$$

More generally, we will show that if in M:

(arbitrary string) $v = a_1 a_2 \cdots a_n$



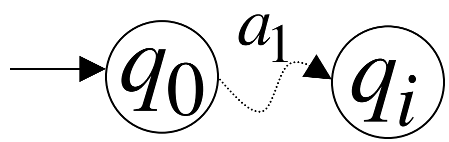
then



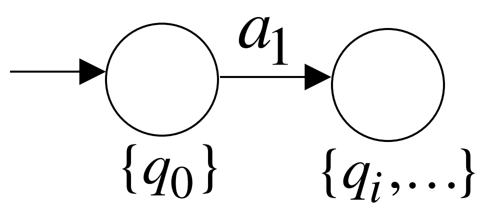
Proof by induction on $|v|$

Induction Basis: $v = a_1$

$M:$



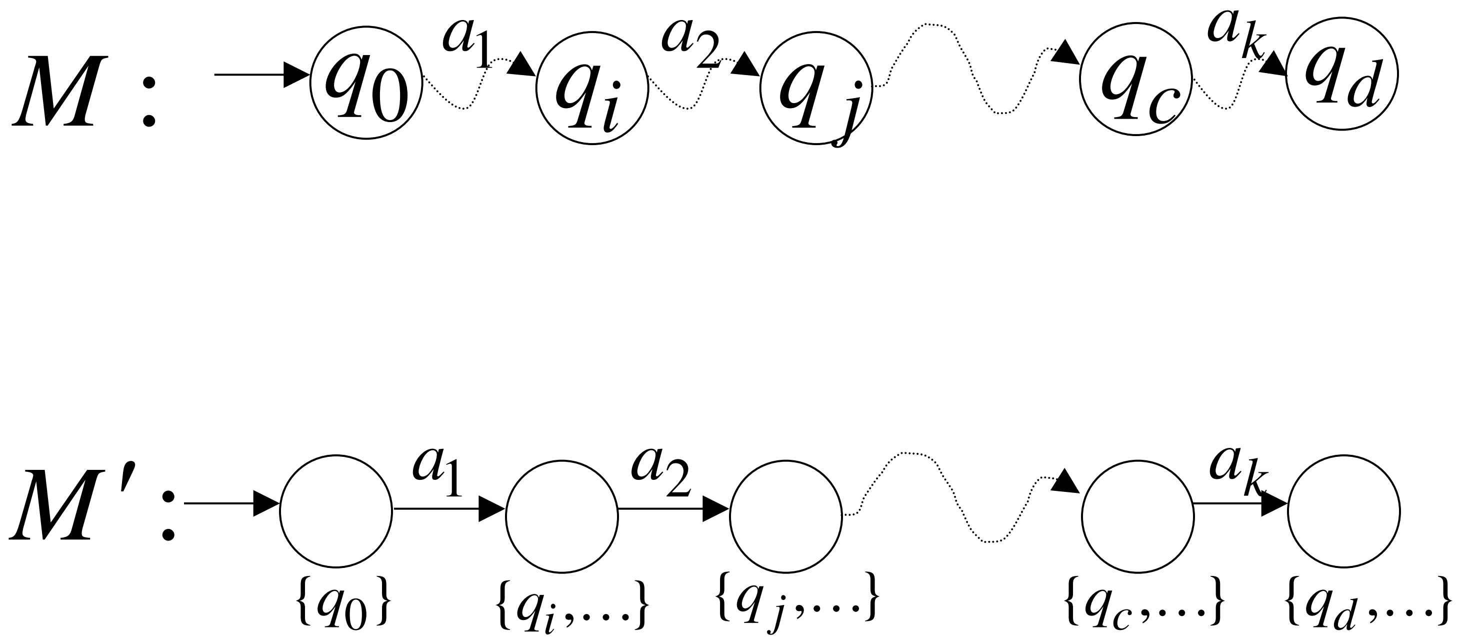
$M':$



Is true by construction of M':

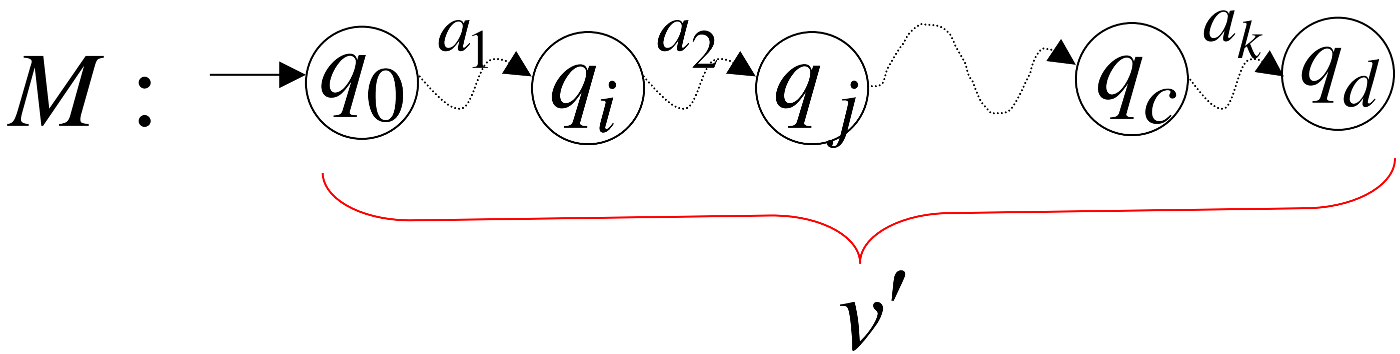
Induction hypothesis: $1 \leq |v| \leq k$

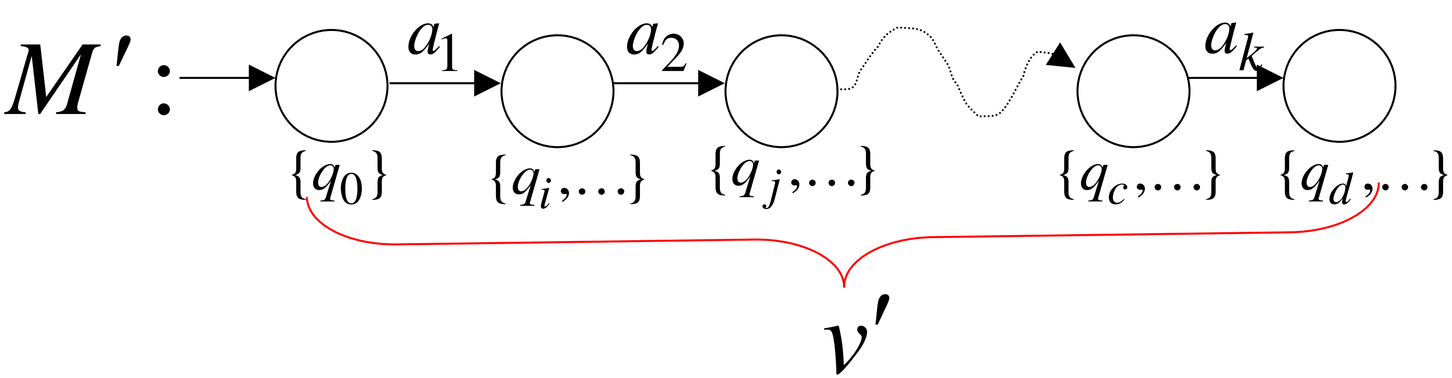
$$v = a_1 a_2 \cdots a_k$$



Induction Step: $|v| = k+1$

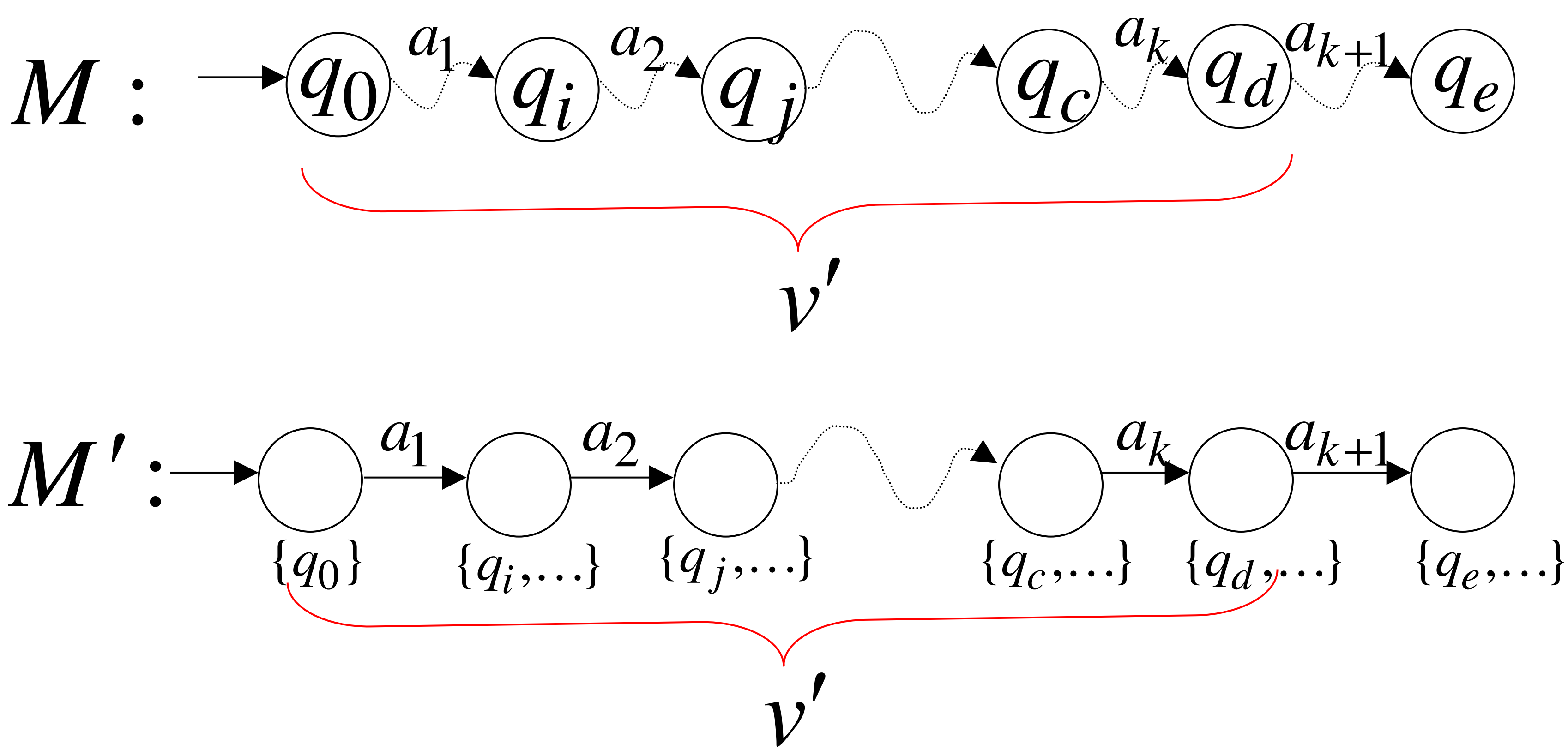
$$v = \underbrace{a_1 a_2 \cdots a_k}_{v'} a_{k+1} = v' a_{k+1}$$





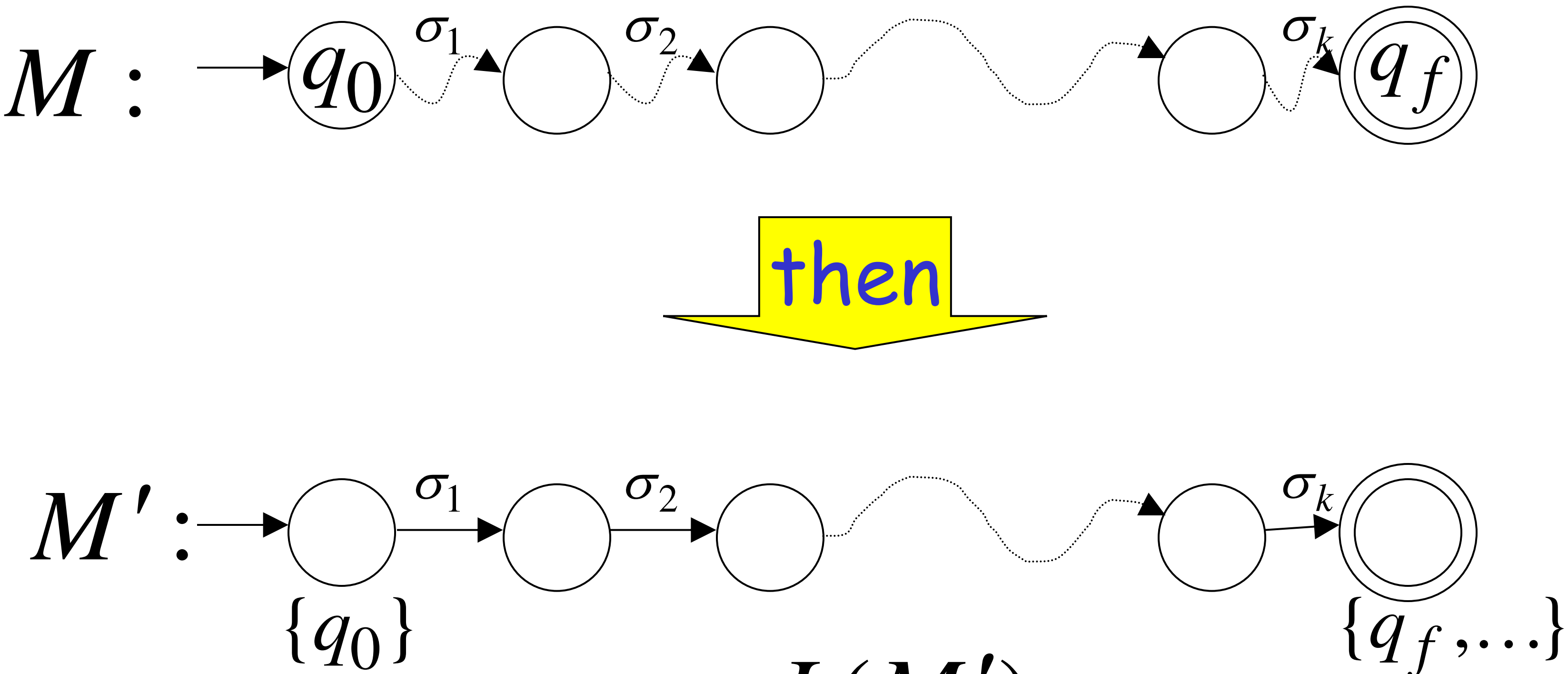
Induction Step: $|v| = k+1$

$$v = \underbrace{a_1 a_2 \cdots a_k}_{v'} a_{k+1} = v' a_{k+1}$$



Therefore if $w \in L(M)$

$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



$$w \in L(M')$$

We have shown: $L(M) \subseteq L(M')$

We also need to show: $L(M) \supseteq L(M')$

(proof is similar)

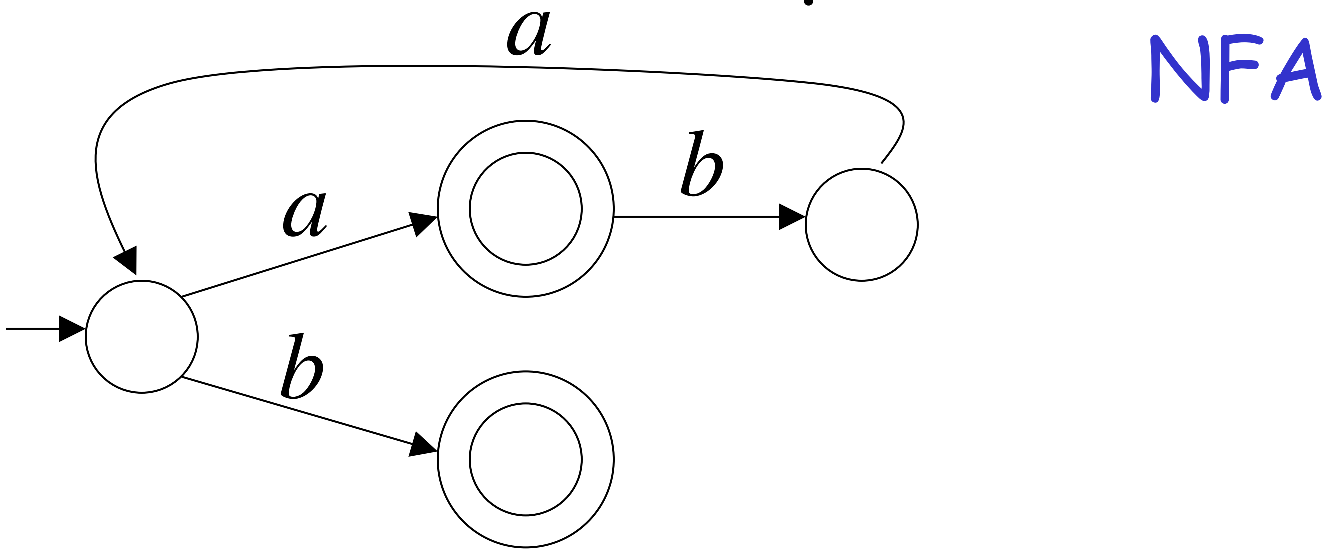
Single Accepting State for NFAs

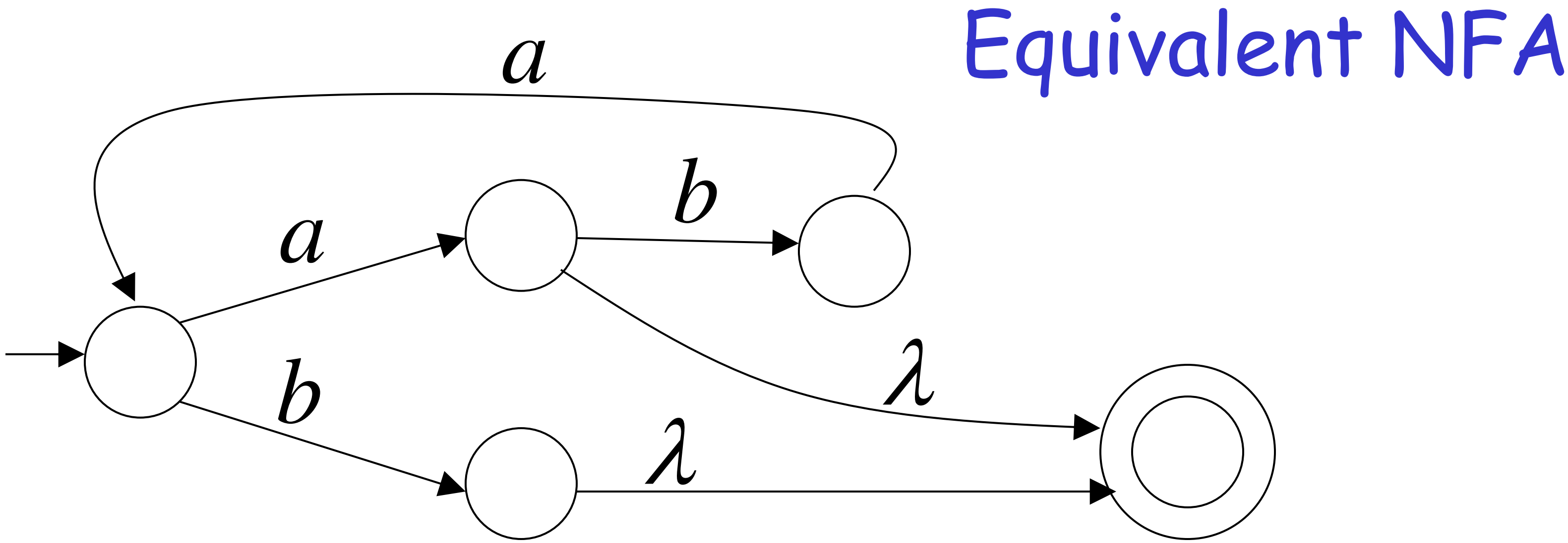
Any NFA can be converted

to an equivalent NFA

with a single accepting state

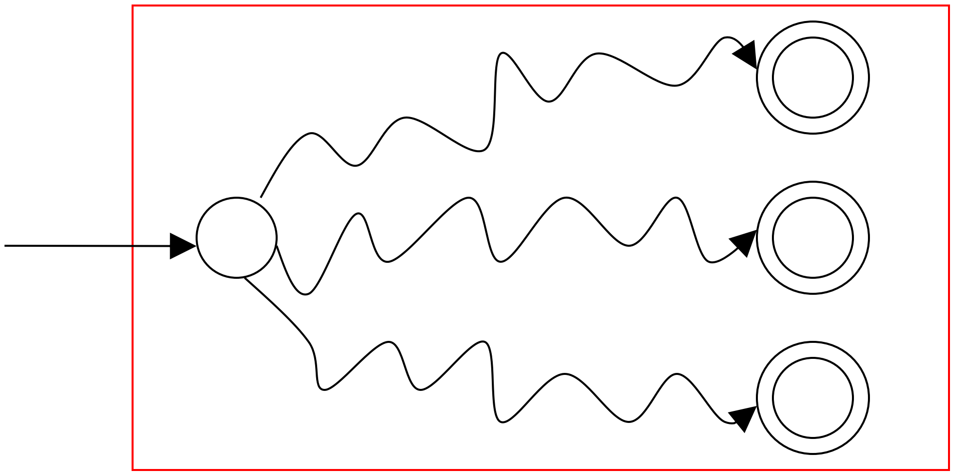
Example



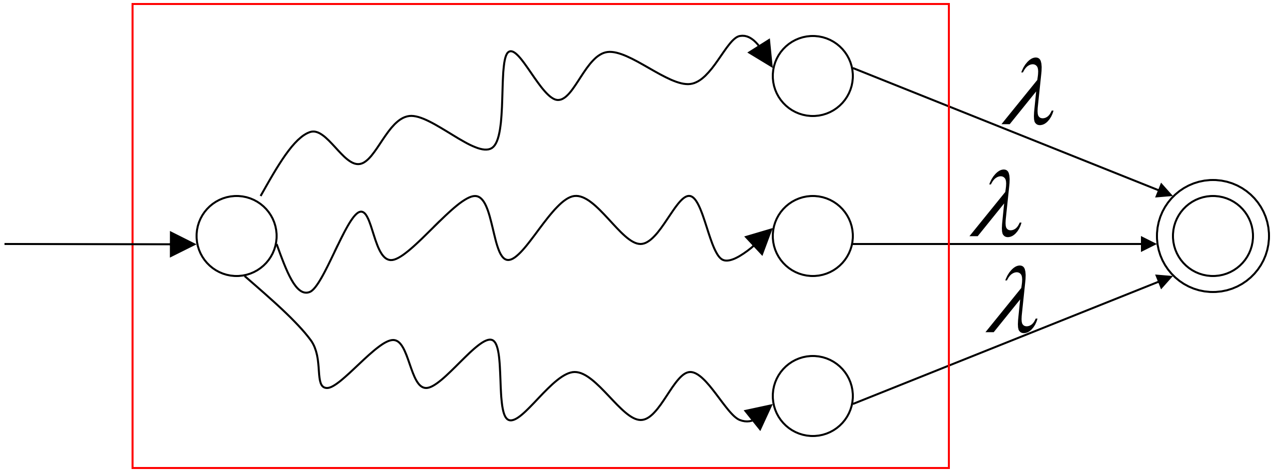


In General

NFA



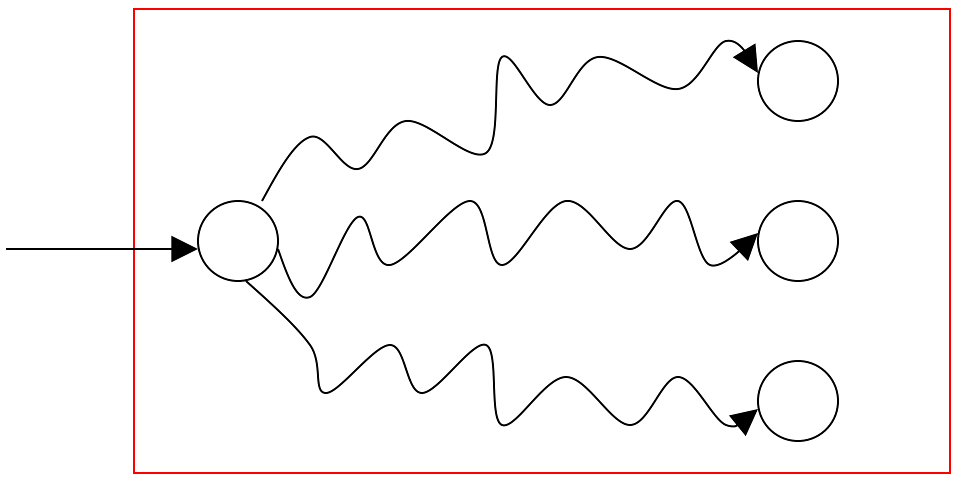
Equivalent NFA

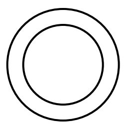


Single accepting state

Extreme Case

NFA without accepting state





Add an accepting state
without transitions

Properties of Regular Languages

For regular languages L_1 and L_2 we will prove that:

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1 *$

Reversal: L_1^R

Complement: $\overline{L_1}$

Intersection: $L_1 \cap L_2$

Are regular Languages

We say: Regular languages are **closed under**

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1{}^*$

Reversal: $L_1{}^R$

Complement: $\overline{L_1}$

Intersection: $L_1 \cap L_2$

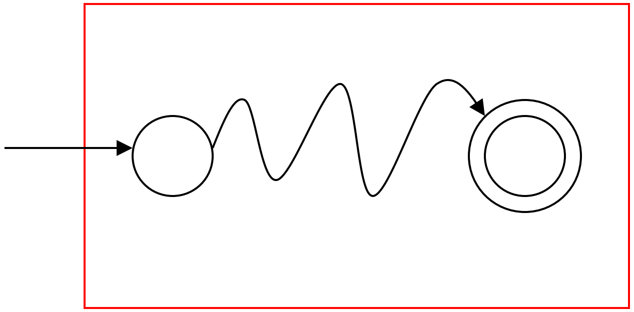
Regular language L_1

Regular language L_2

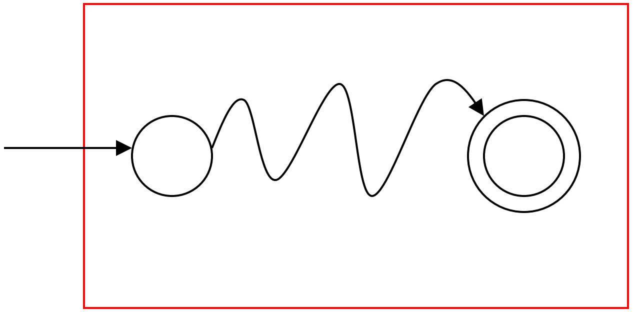
$$L(M_1) = L_1$$

$$L(M_2) = L_2$$

NFA M_1



NFA M_2



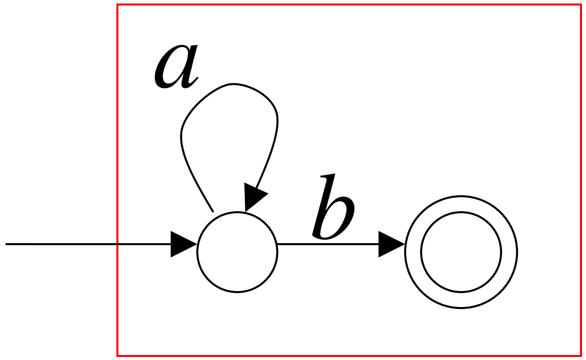
Single accepting state

Single accepting state

Example

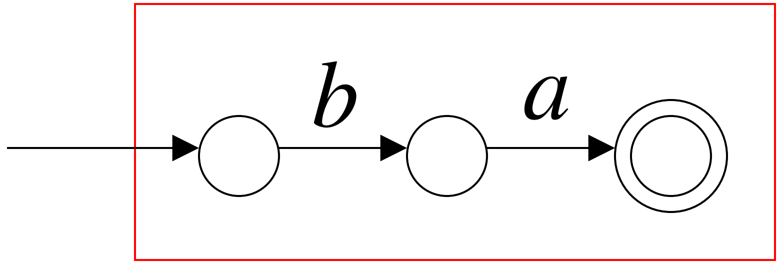
$$L_1 = \{a^n b\} \quad n \geq 0$$

M_1



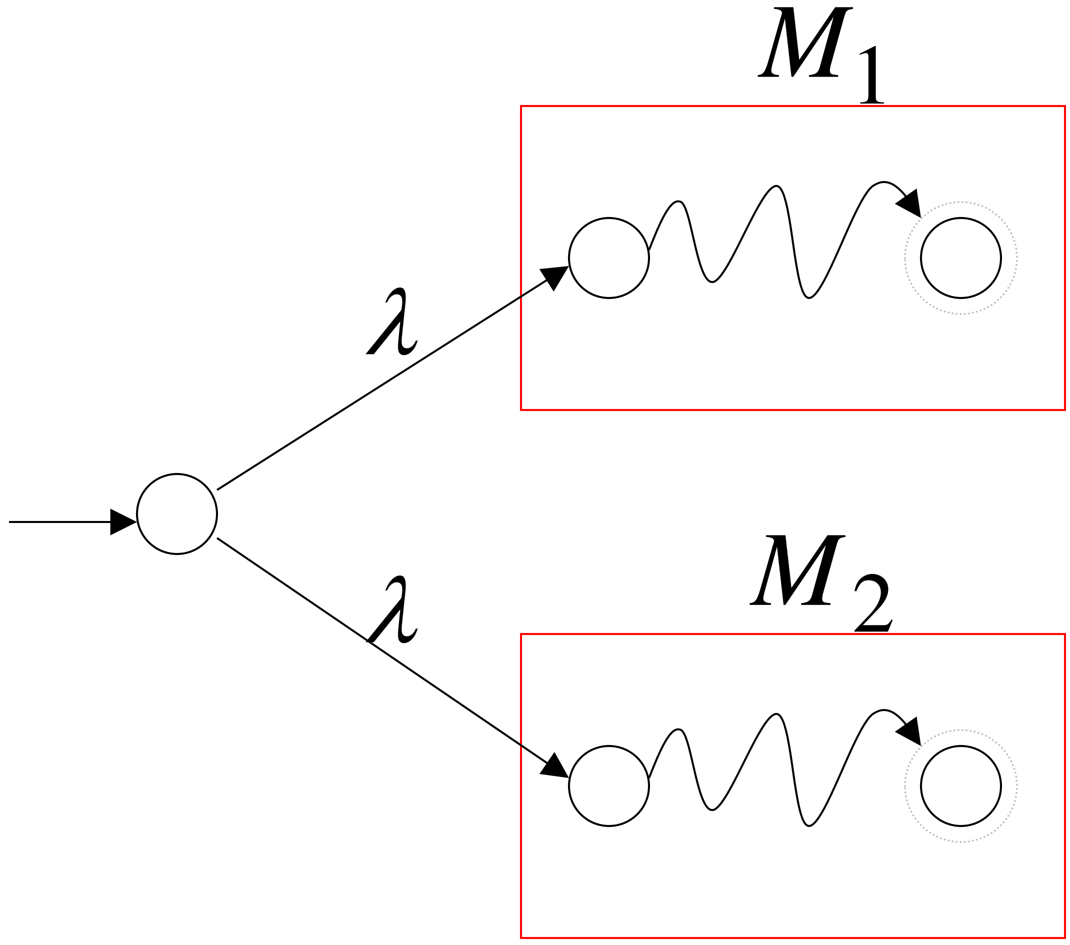
$$L_2 = \{ba\}$$

M_2



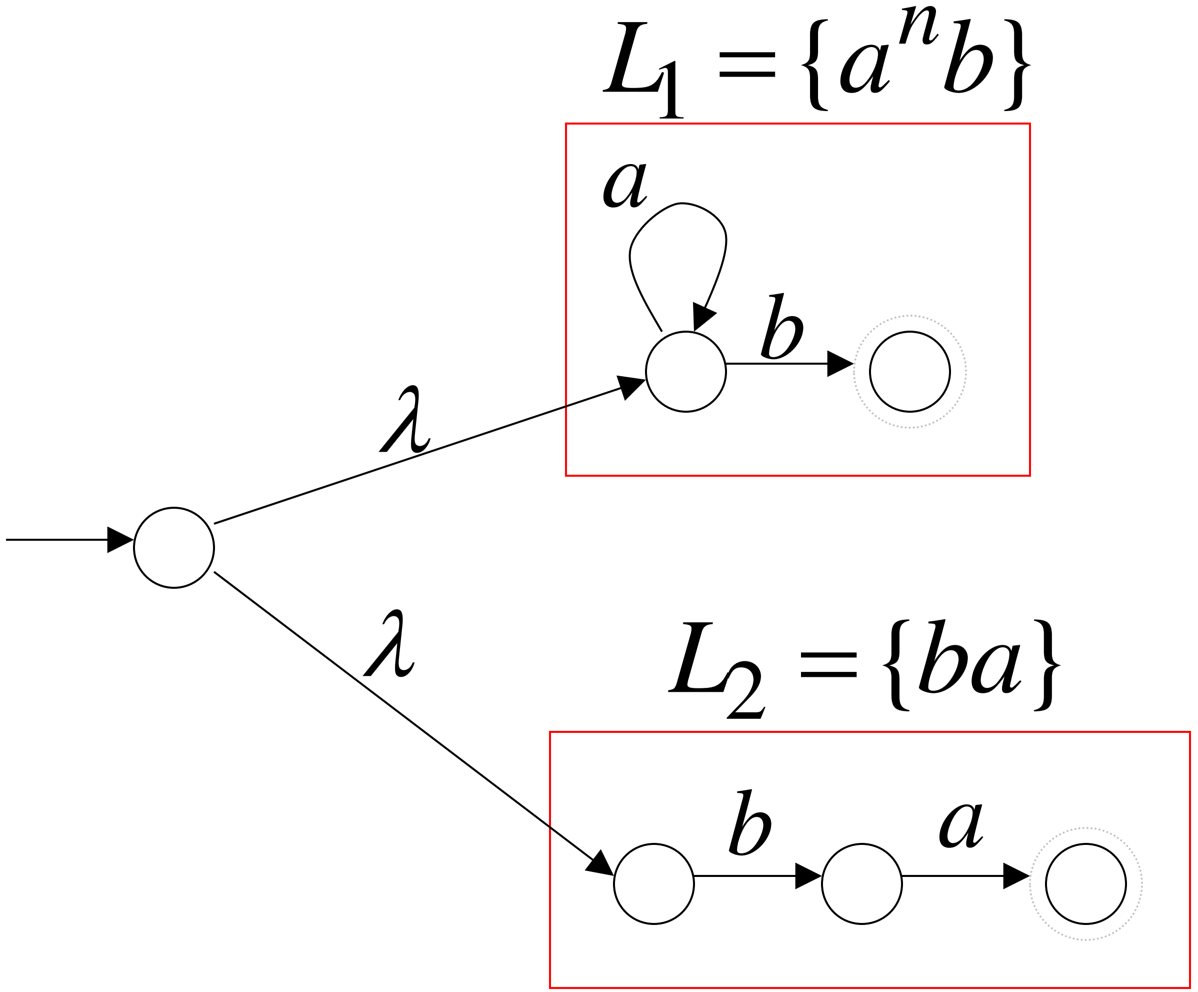
Union

NFA for $L_1 \cup L_2$



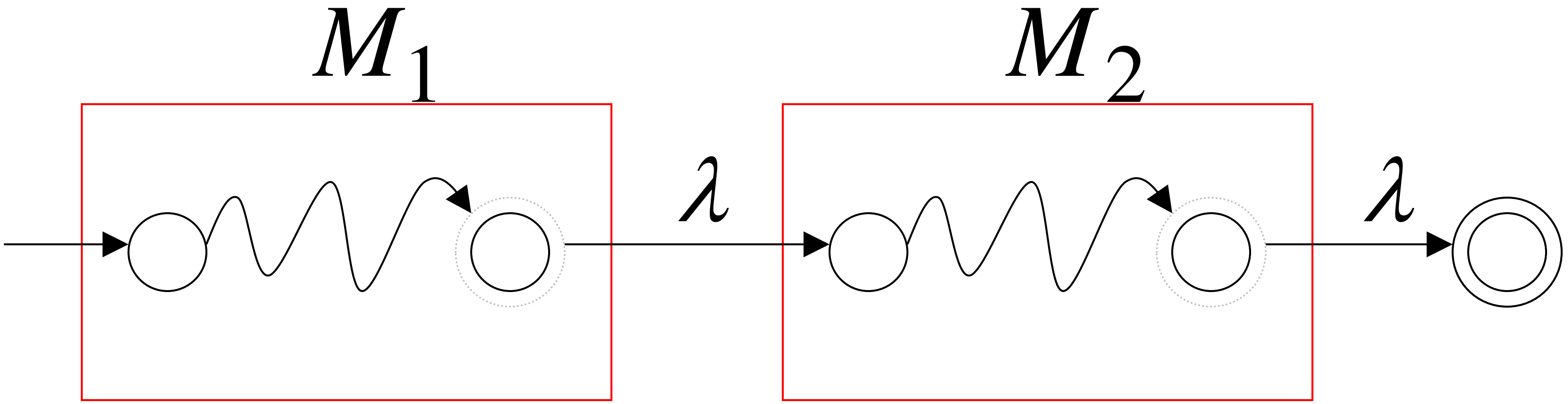
Example

NFA for $L_1 \cup L_2 = \{a^n b\} \cup \{ba\}$



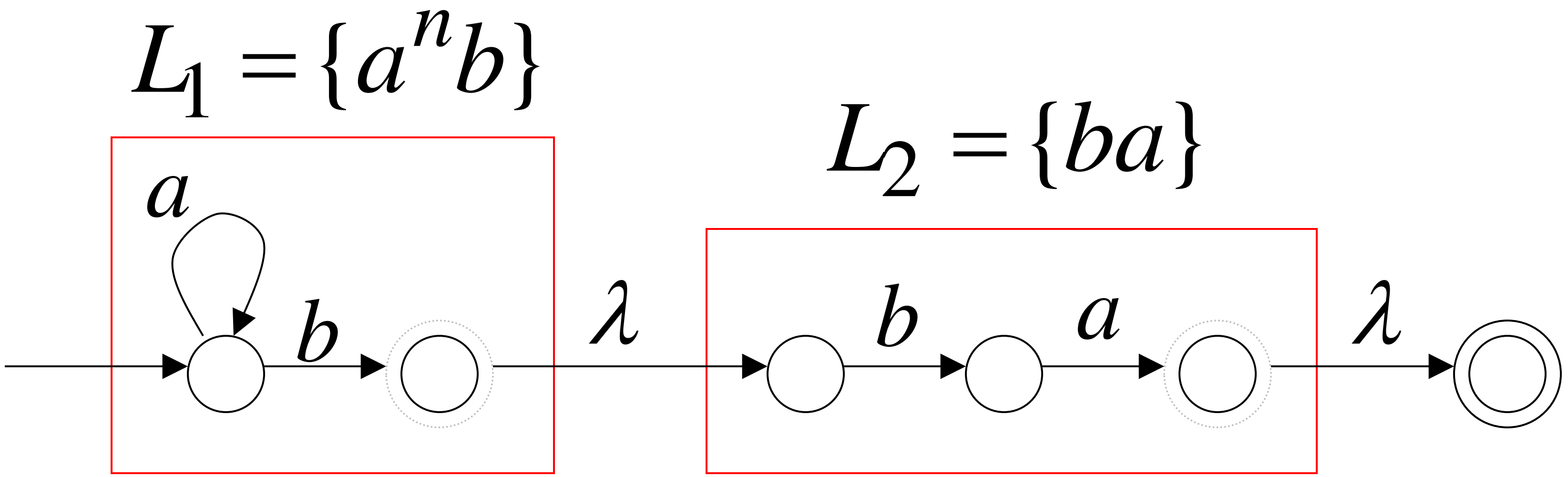
Concatenation

NFA for $L_1 L_2$



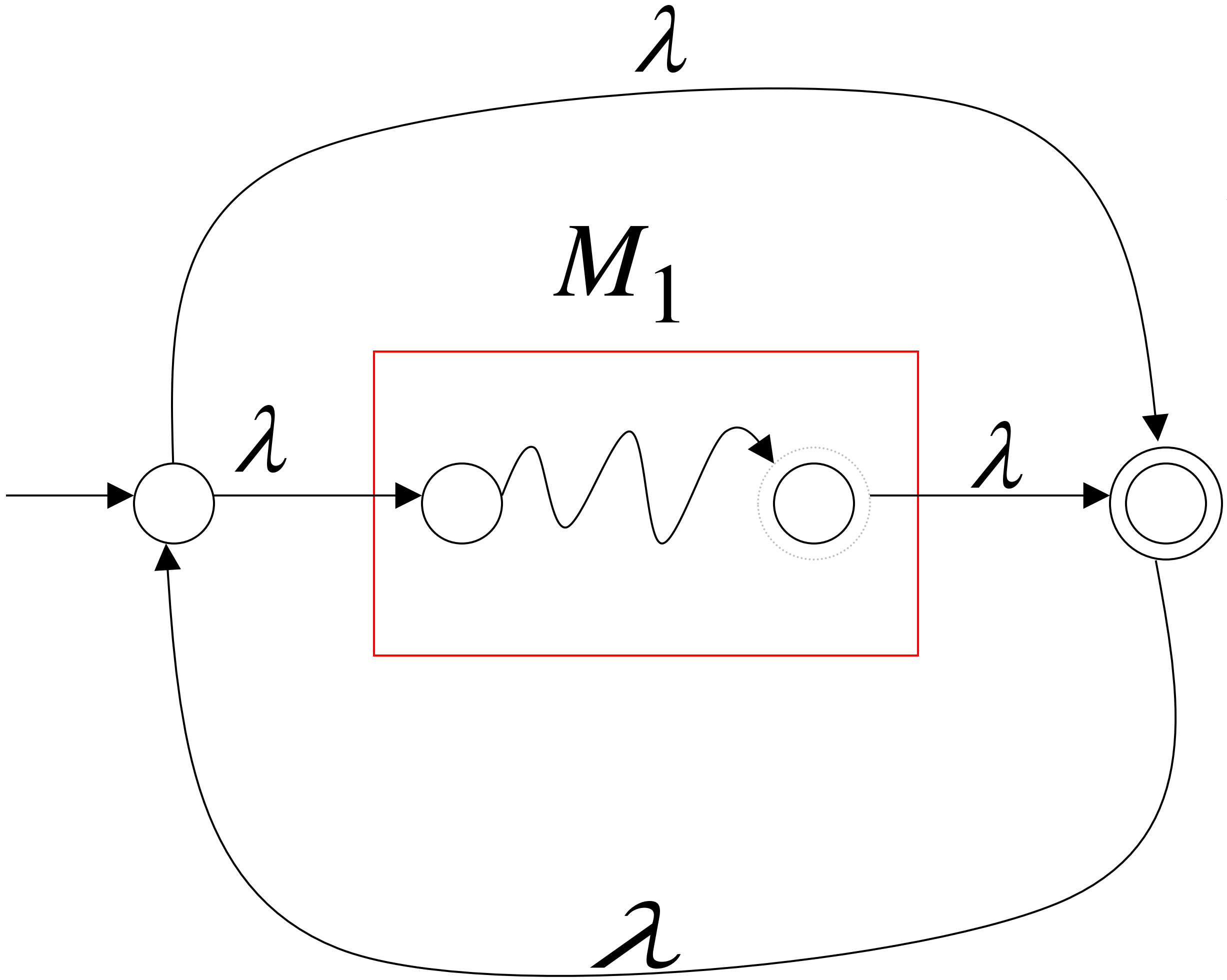
Example

NFA for $L_1 L_2 = \{a^n b\}\{ba\} = \{a^n bba\}$



Star Operation

NFA for $L_1{}^*$



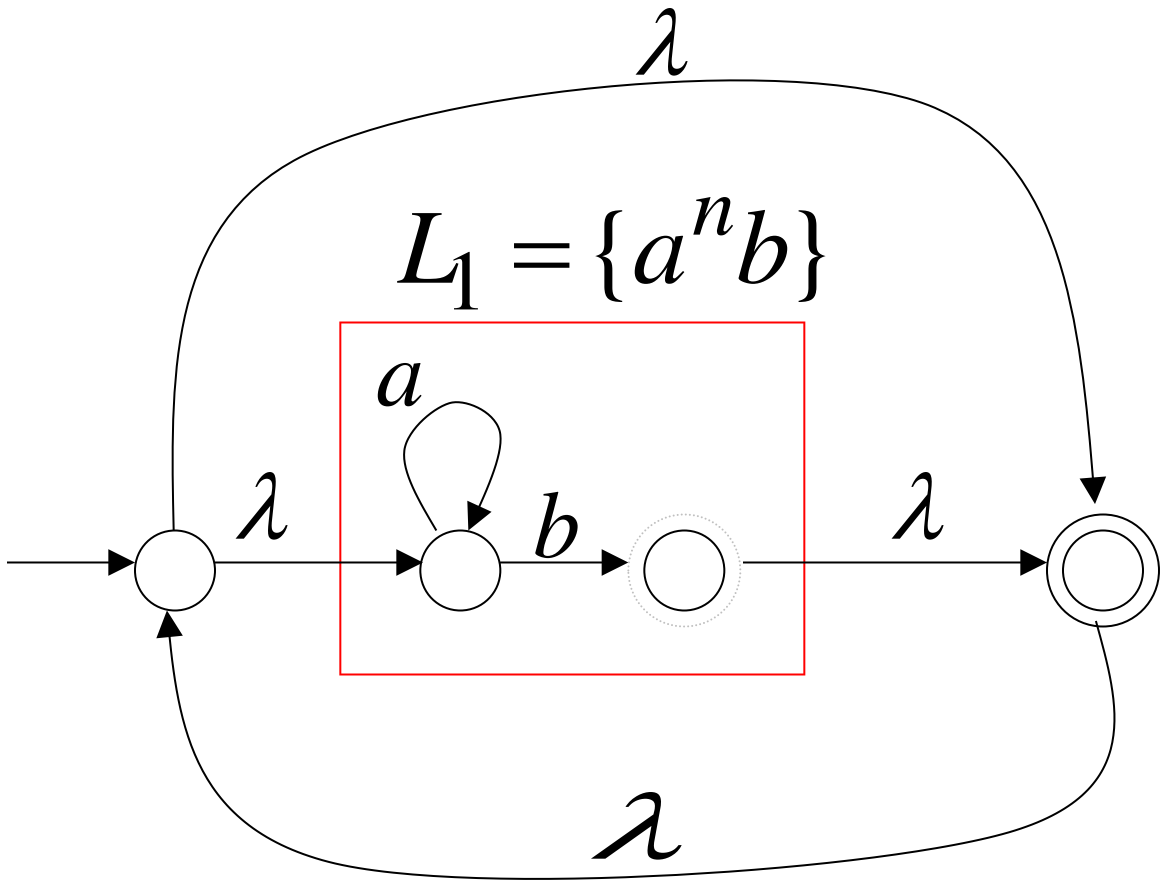
$\lambda \in L_1{}^*$

Example

NFA for $L_1^* = \{a^n b\}^*$

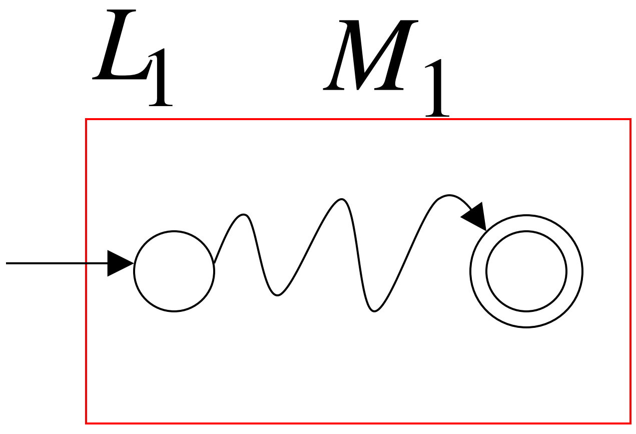
$w = w_1 w_2 \cdots w_k$

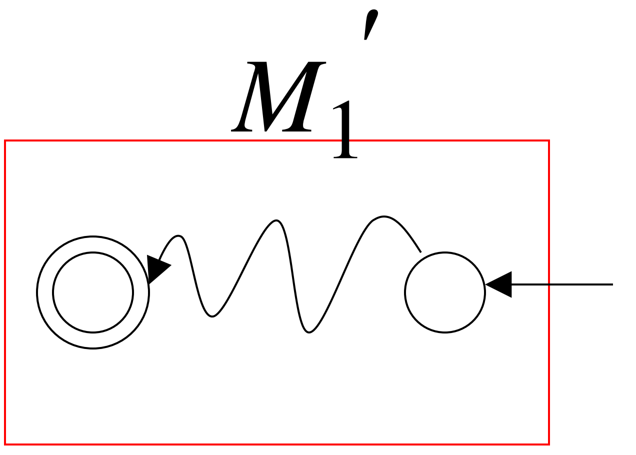
$w_i \in L_1$



Reverse

NFA for L_1^R





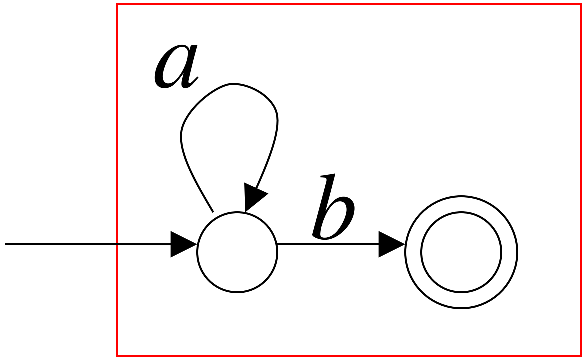
1. Reverse all transitions

2. Make initial state accepting state and vice versa

Example

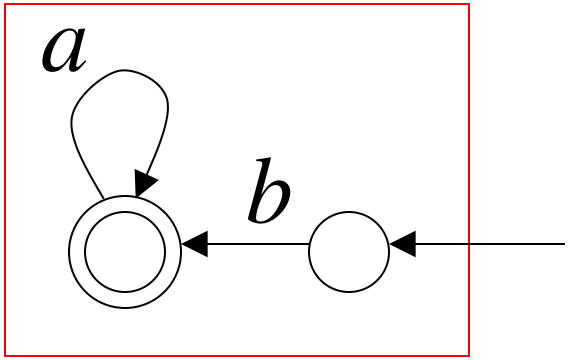
M_1

$L_1 = \{a^n b\}$

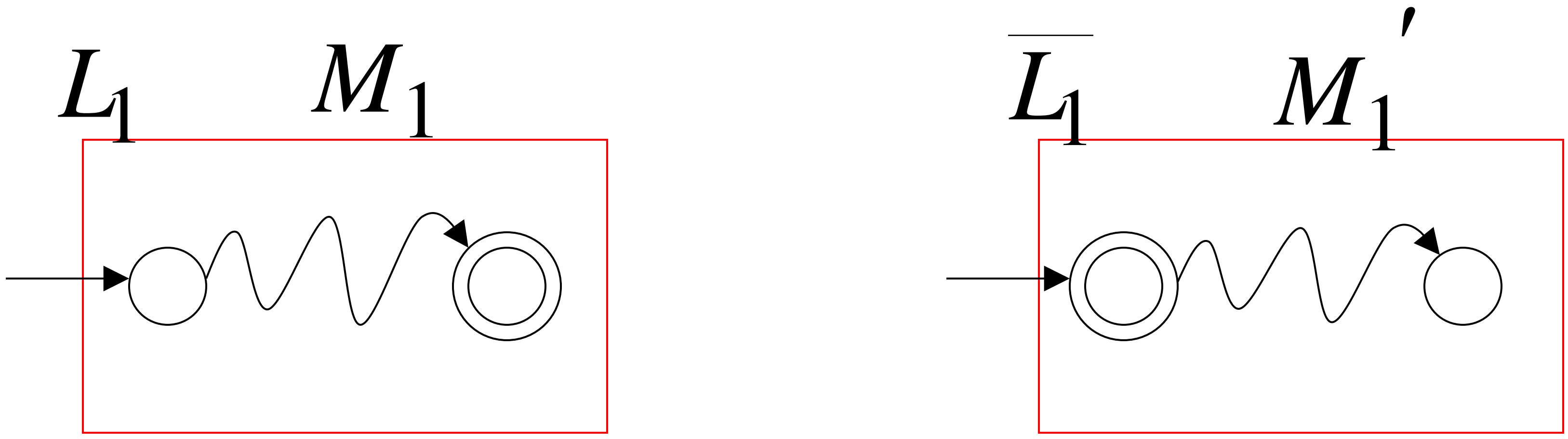


M_1'

$L_1^R = \{ba^n\}$



Complement

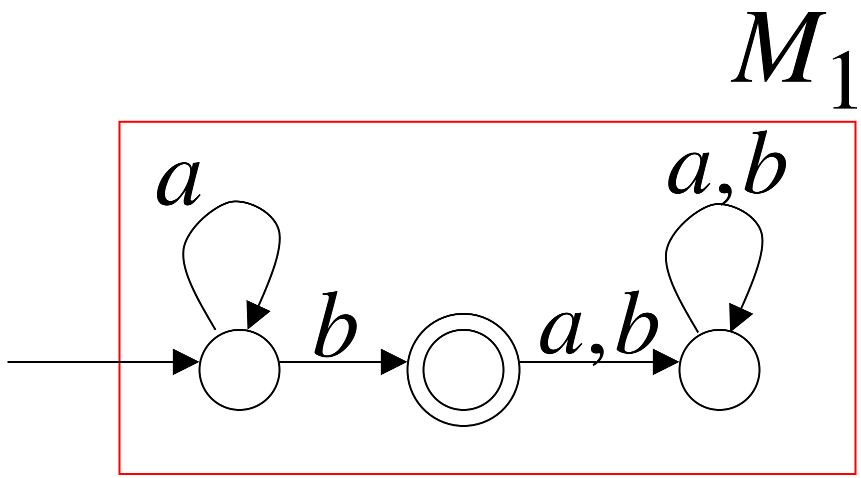


1. Take the **FA** that accepts L_1

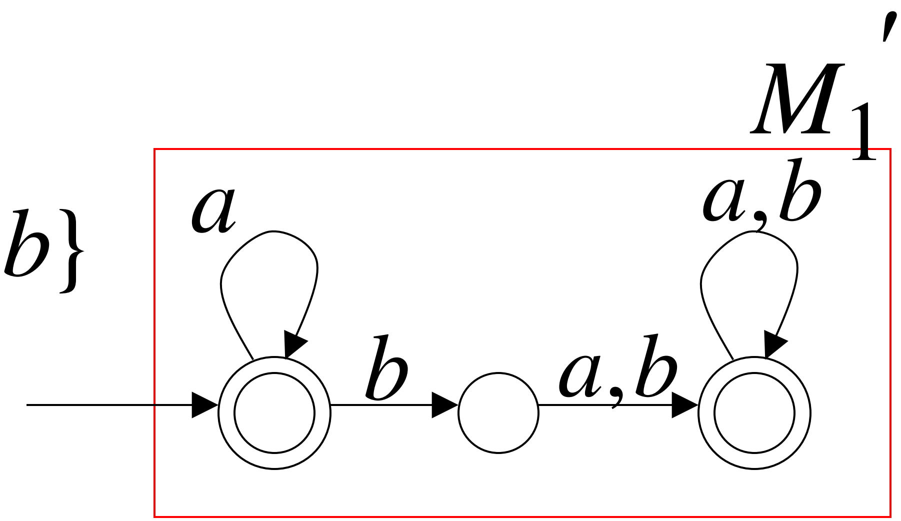
2. Make final states non-final, and vice-versa

Example

$$L_1 = \{a^n b\}$$



$$\overline{L_1} = \{a,b\}^* - \{a^n b\}$$



Intersection

L_1 regular

L_2 regular

We show

$L_1 \cap L_2$

regular

DeMorgan's Law: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

$L_1 \,,\; L_2$ regular

$\Rightarrow$ $\overline{L_1} \,,\; \overline{L_2}$ regular

$\Rightarrow$ $\overline{L_1} \cup \overline{L_2}$ regular

$\Rightarrow$ $\overline{\overline{L_1} \cup \overline{L_2}}$ regular

$\Rightarrow$ $L_1 \cap L_2$ regular

Example

$L_1 = \{a^n b\}$ regular

$L_2 = \{ab, ba\}$ regular

$\Rightarrow L_1 \cap L_2 = \{ab\}$

regular

Another Proof for Intersection Closure

Machine M_1

FA for L_1

Machine M_2

FA for L_2

Construct a new FA M that accepts $L_1 \cap L_2$

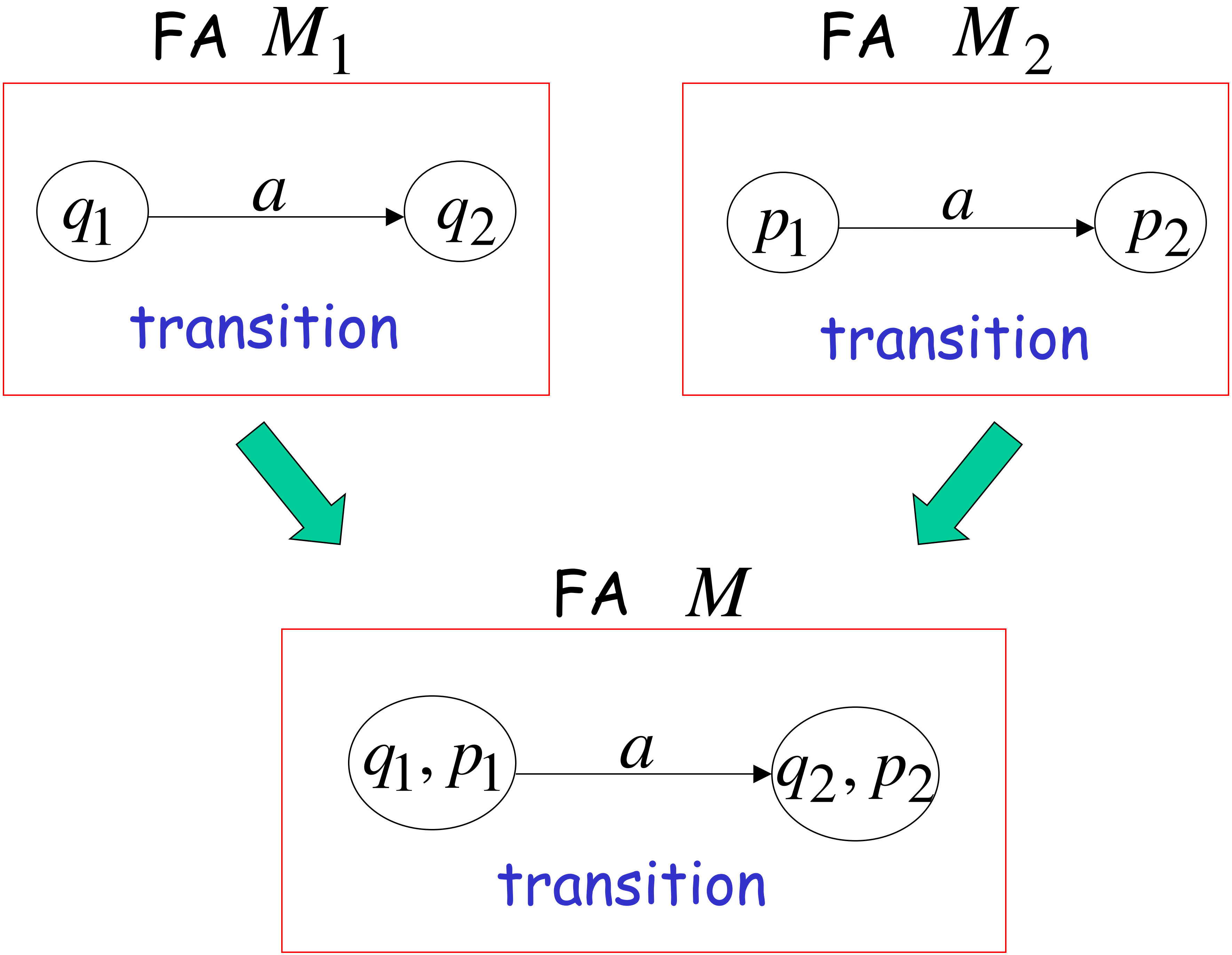
M simulates in parallel M_1 and M_2

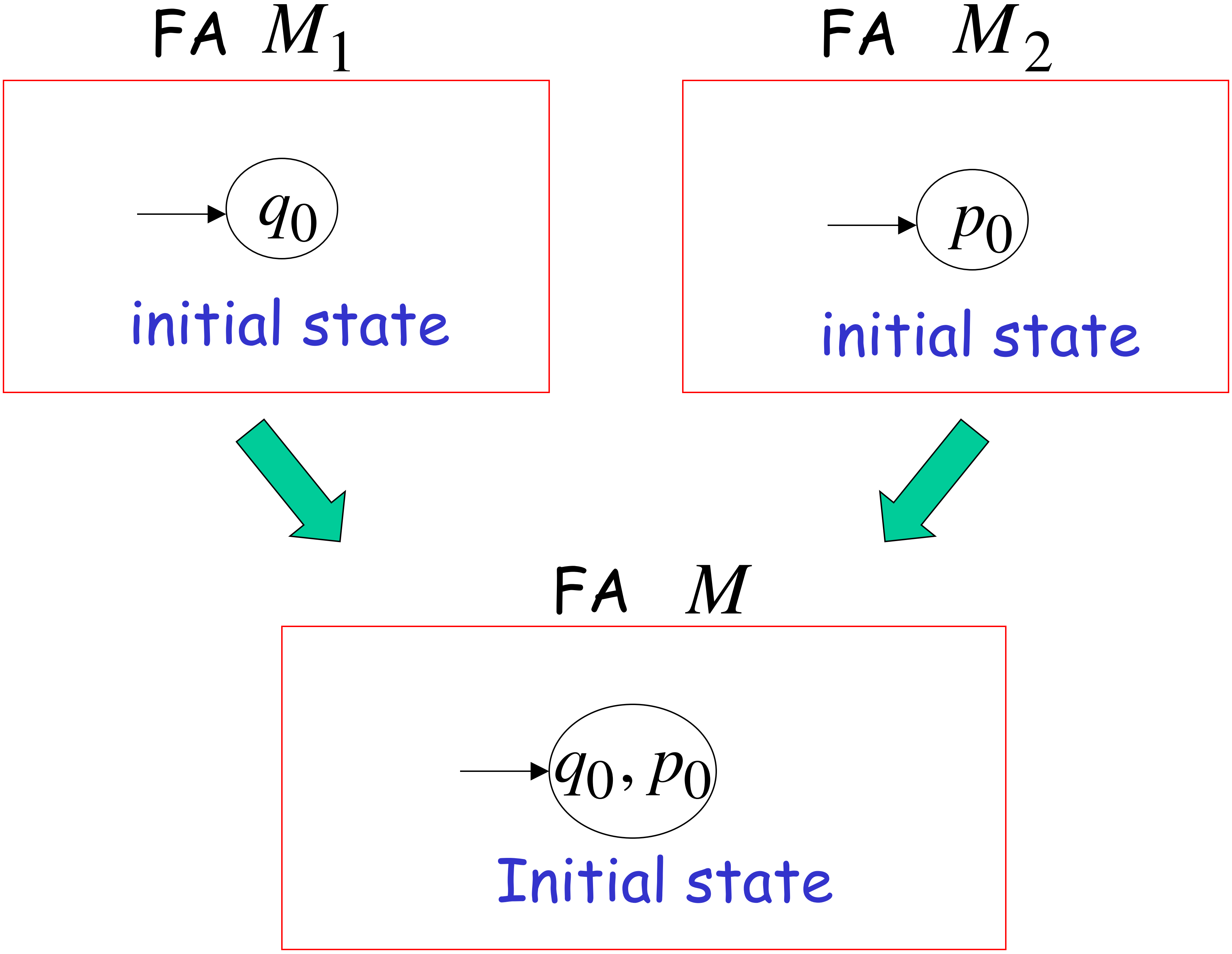
States in M

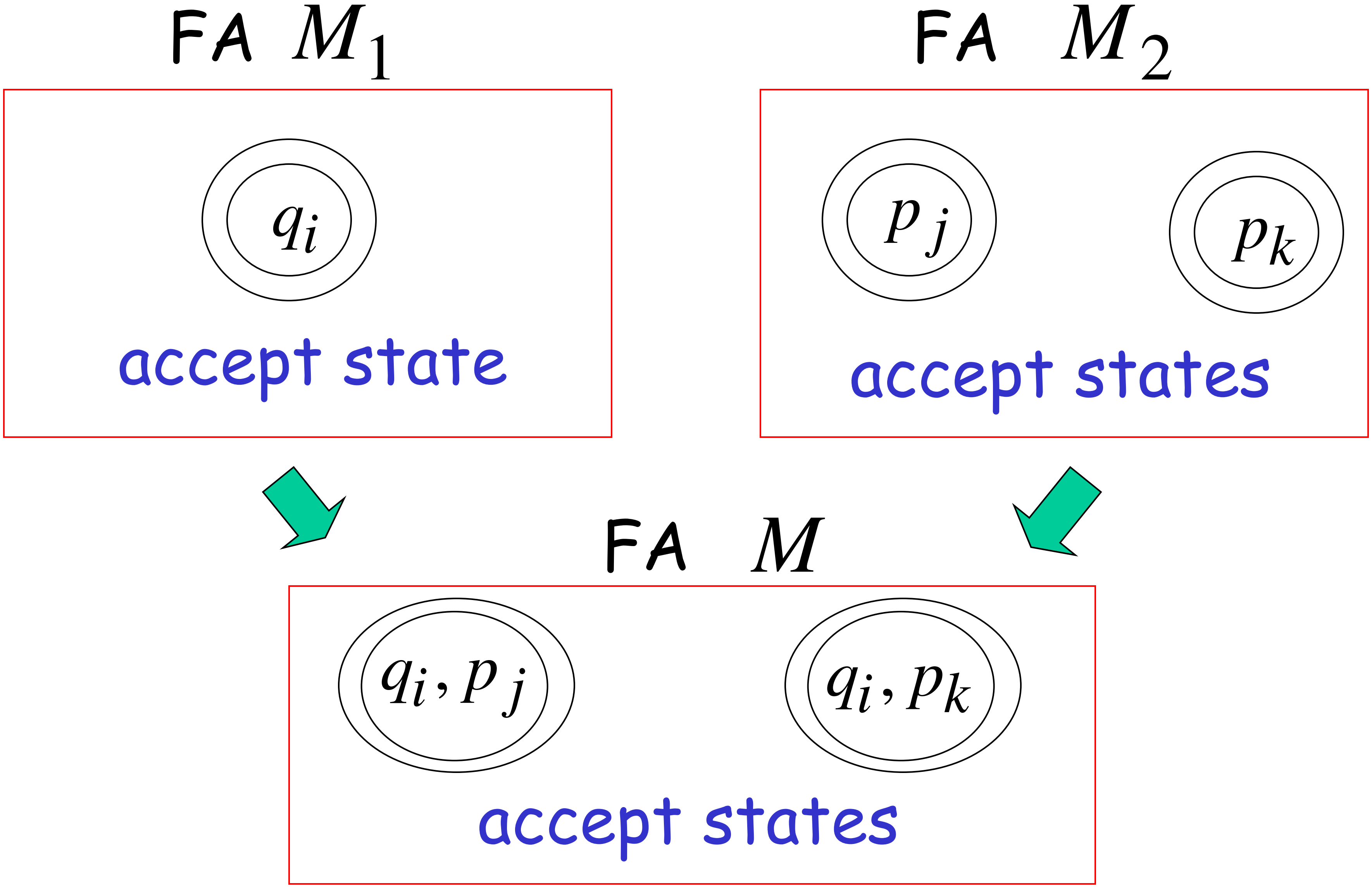
q_i, p_j

State in M_1

State in M_2


FA M_1
q_1
a
q_2
transition
FA M_2
p_1
a
p_2
transition
FA M
q_1, p_1
a
q_2, p_2
transition


FA M_1
q_0
initial state
FA M_2
p_0
initial state
FA M
q_0, p_0
Initial state



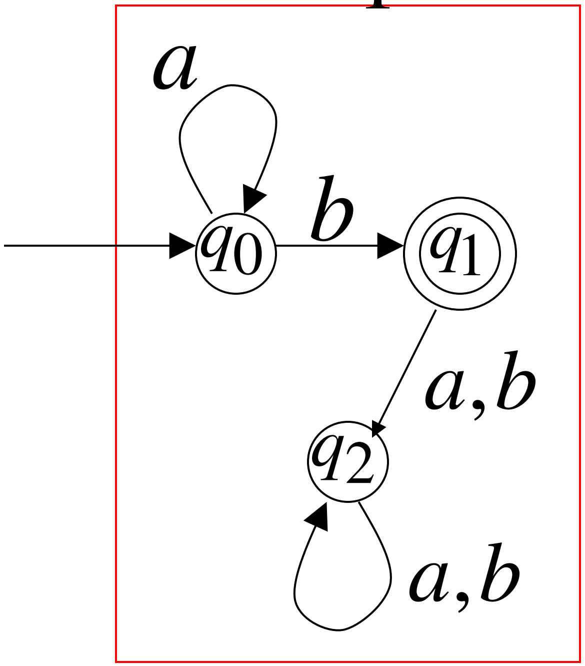
Both constituents must be accepting states

Example:

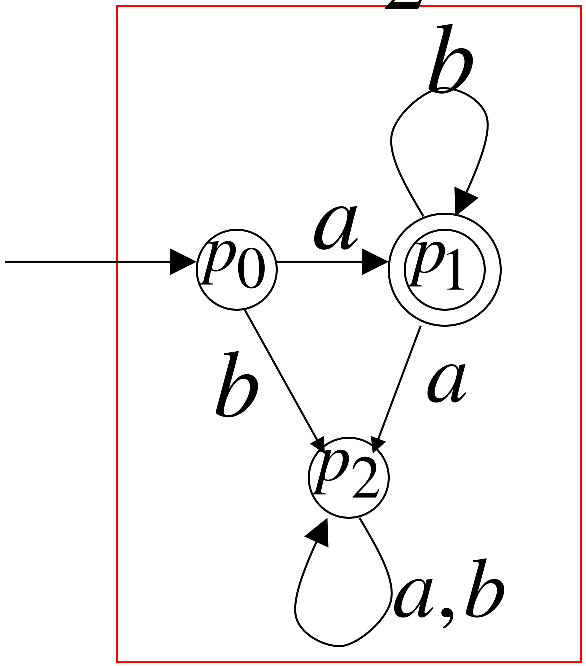
$$L_1 = \{a^n b\} \quad n \geq 0$$

$$L_2 = \{ab^m\} \quad m \geq 0$$

M_1

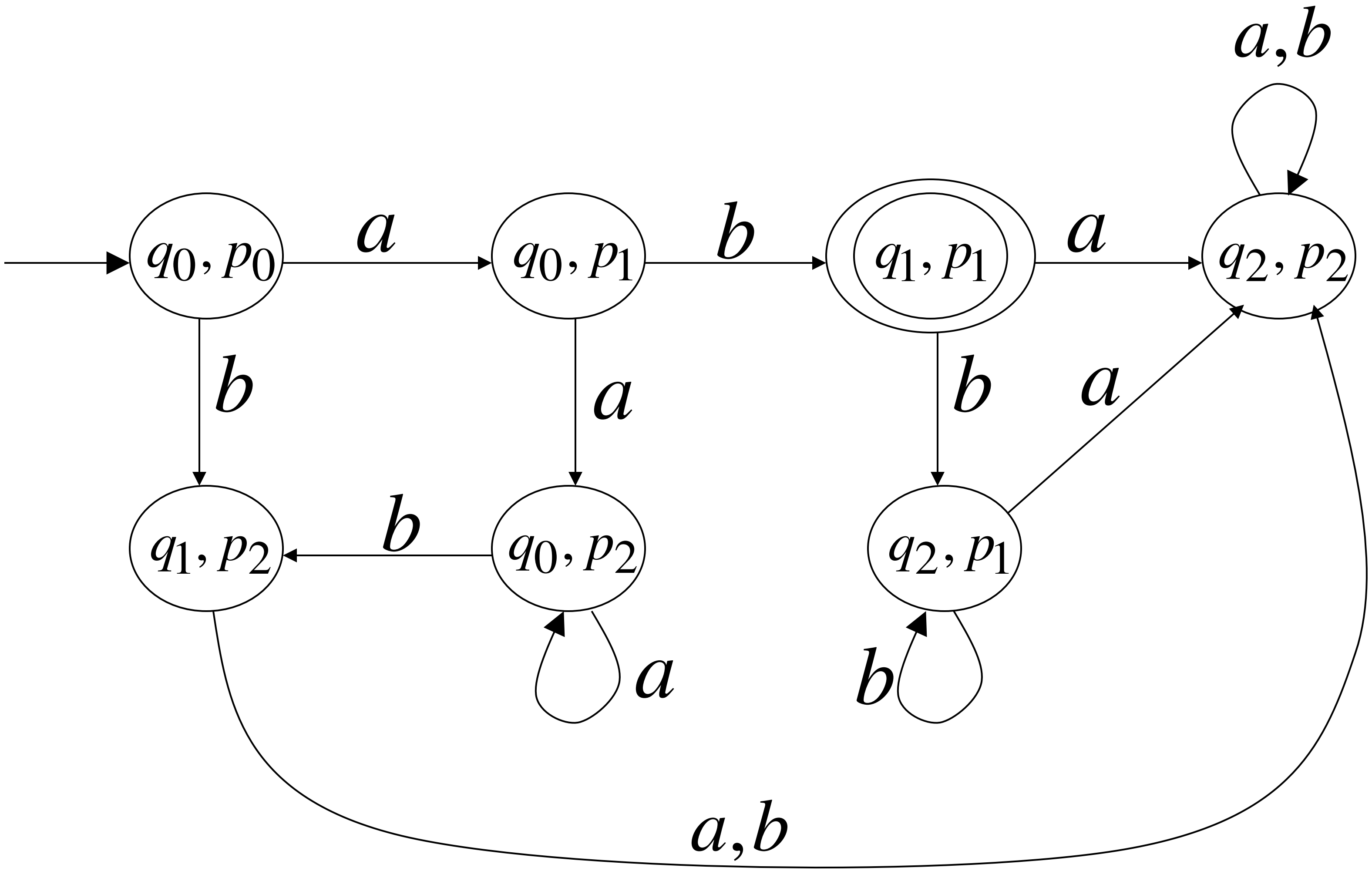


M_2



Automaton for intersection

$$L = \{a^n b\} \cap \{ab^n\} = \{ab\}$$



M simulates in parallel M_1 and M_2

M accepts string w if and only if

M_1 accepts string w and

M_2 accepts string w

$$L(M) = L(M_1) \cap L(M_2)$$

14B11CI171

Theory of Computation

Regular Expressions

Regular Expressions

Regular expressions describe regular languages

Example: $(a+b\cdot c)^*$

describes the language

$$\{a,bc\}^* = \{\lambda, a, bc, aa, abc, bca, \ldots\}$$

Recursive Definition

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

Given regular expressions r_1 and r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \cdot r_2 \\ r_1{}^* \\ (r_1) \end{array}\right\} \text{Are regular expressions}$$

Examples

A regular expression: $(a+b \cdot c)^* \cdot (c+\varnothing)$

Not a regular expression: $(a+b+)$

Languages of Regular Expressions

$L(r)$: language of regular expression r

Example

$$L((a+b\cdot c)^*)=\{\lambda,a,bc,aa,abc,bca,...\}$$

Definition

For primitive regular expressions:

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

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

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

Definition (continued)

For regular expressions r_1 and r_2

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

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

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

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

Example

Regular expression: $(a+b)\cdot a^*$

$$\begin{aligned}
L((a+b)\cdot a^*) &= L((a+b))\,L(a^*) \\
&= L(a+b)\,L(a^*) \\
&= (L(a)\cup L(b))\,(L(a))^* \\
&= (\{a\}\cup\{b\})\,(\{a\})^* \\
&= \{a,b\}\,\{\lambda,a,aa,aaa,\ldots\} \\
&= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}
\end{aligned}$$

Example

Regular expression $r = (a+b)^*(a+bb)$

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}$$

Example

Regular expression $r = (aa)^*(bb)^*b$

$$L(r) = \{a^{2n}b^{2m}b: \quad n, m \geq 0\}$$

Example

Regular expression $r = (0+1)^*00(0+1)^*$

$L(r)$ = { all strings with at least two consecutive 0 }

Example

Regular expression $r = (1 + 01)^*(0 + \lambda)$

$L(r)$ = { all strings without two consecutive 0 }

Equivalent Regular Expressions

Definition:

Regular expressions r_1 and r_2

are **equivalent** if $L(r_1) = L(r_2)$

Example

L = { all strings without two consecutive 0 }

$$r_1 = (1 + 01)^*(0 + \lambda)$$

$$r_2 = (1^*011^*)^*(0 + \lambda) + 1^*(0 + \lambda)$$

$$L(r_1) = L(r_2) = L \quad \Rightarrow$$

r_1 and r_2 are equivalent regular expr.

Regular Expressions and Regular Languages

Theorem

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} = \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

We will show:

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Proof - Part 1

$$\left\{\begin{array}{l}\text{Languages} \\ \text{Generated by} \\ \text{Regular Expressions}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular} \\ \text{Languages}\end{array}\right\}$$

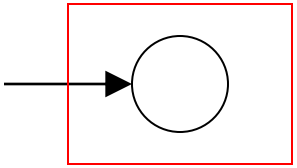
For any regular expression r
the language $L(r)$ is regular

Proof by induction on the size of r

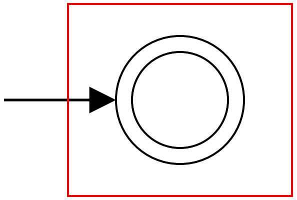
Induction Basis

Primitive Regular Expressions: $\emptyset, \quad \lambda, \quad \alpha$

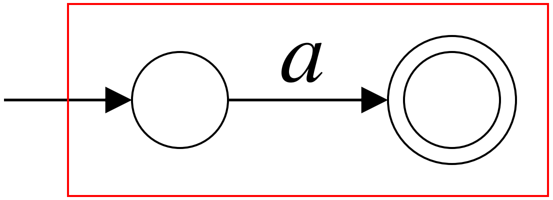
NFAs



$$L(M_1) = \emptyset = L(\emptyset)$$



$$L(M_2) = \{\lambda\} = L(\lambda)$$



$$L(M_3) = \{a\} = L(a)$$

regular languages

Inductive Hypothesis

Assume

for regular expressions r_1 and r_2

that

$L(r_1)$ and $L(r_2)$ are regular languages

Inductive Step

We will prove:

$$L(r_1 + r_2)$$

$$L(r_1 \cdot r_2)$$

$$L(r_1 *)$$

$$L((r_1))$$

Are regular Languages

By definition of regular expressions:

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

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

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

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

By inductive hypothesis we know:

$L(r_1)$ and $L(r_2)$ are regular languages

We also know:

Regular languages are closed under:

Union	$L(r_1) \cup L(r_2)$
Concatenation	$L(r_1)\,L(r_2)$
Star	$(L(r_1))^*$

Therefore:

$$\left.\begin{array}{l} L(r_1 + r_2) = L(r_1) \cup L(r_2) \\ \\ L(r_1 \cdot r_2) = L(r_1)\, L(r_2) \\ \\ L(r_1 *) = (L(r_1))* \end{array}\right\} \text{Are regular languages}$$

And trivially:

$L((r_1))$ is a regular language

Proof - Part 2

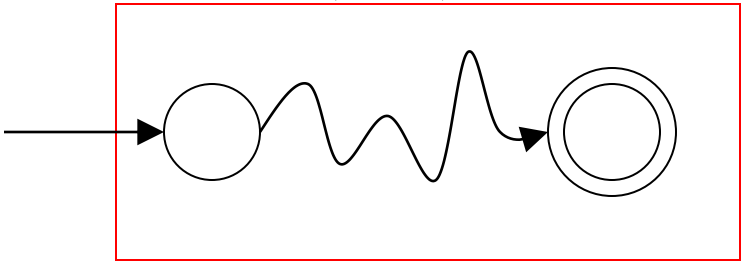
$$\left\{\begin{array}{l}\text{Languages} \\ \text{Generated by} \\ \text{Regular Expressions}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular} \\ \text{Languages}\end{array}\right\}$$

For any regular language L there is a regular expression r with $L(r) = L$

Proof by construction of regular expression

Since L is regular take the NFA M that accepts it

$$L(M) = L$$



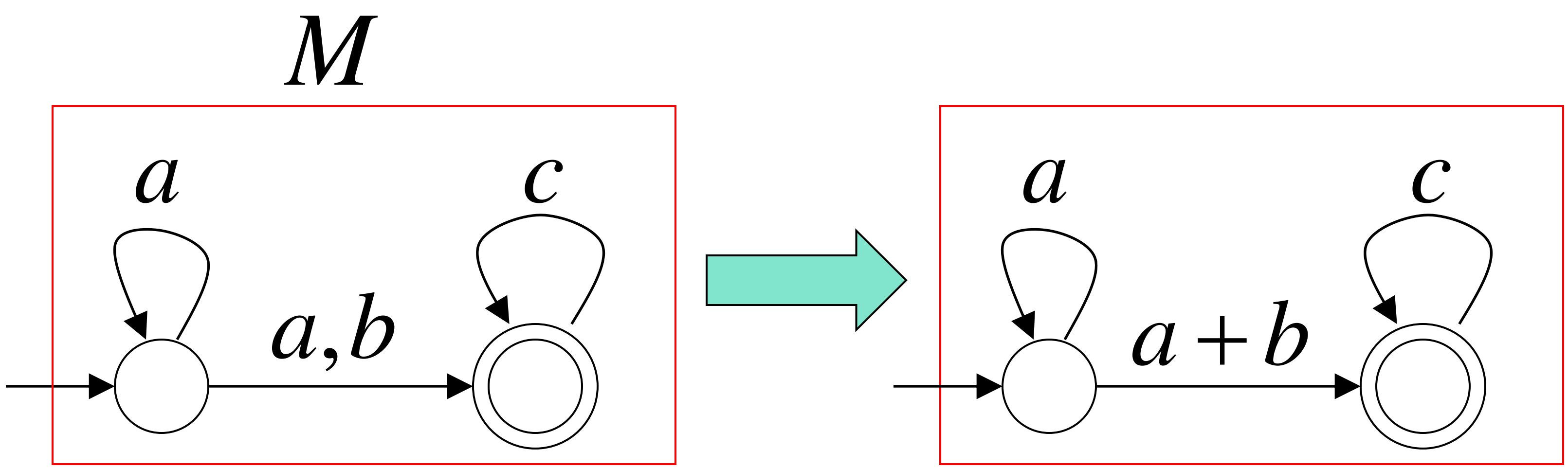
Single final state

From M construct the equivalent

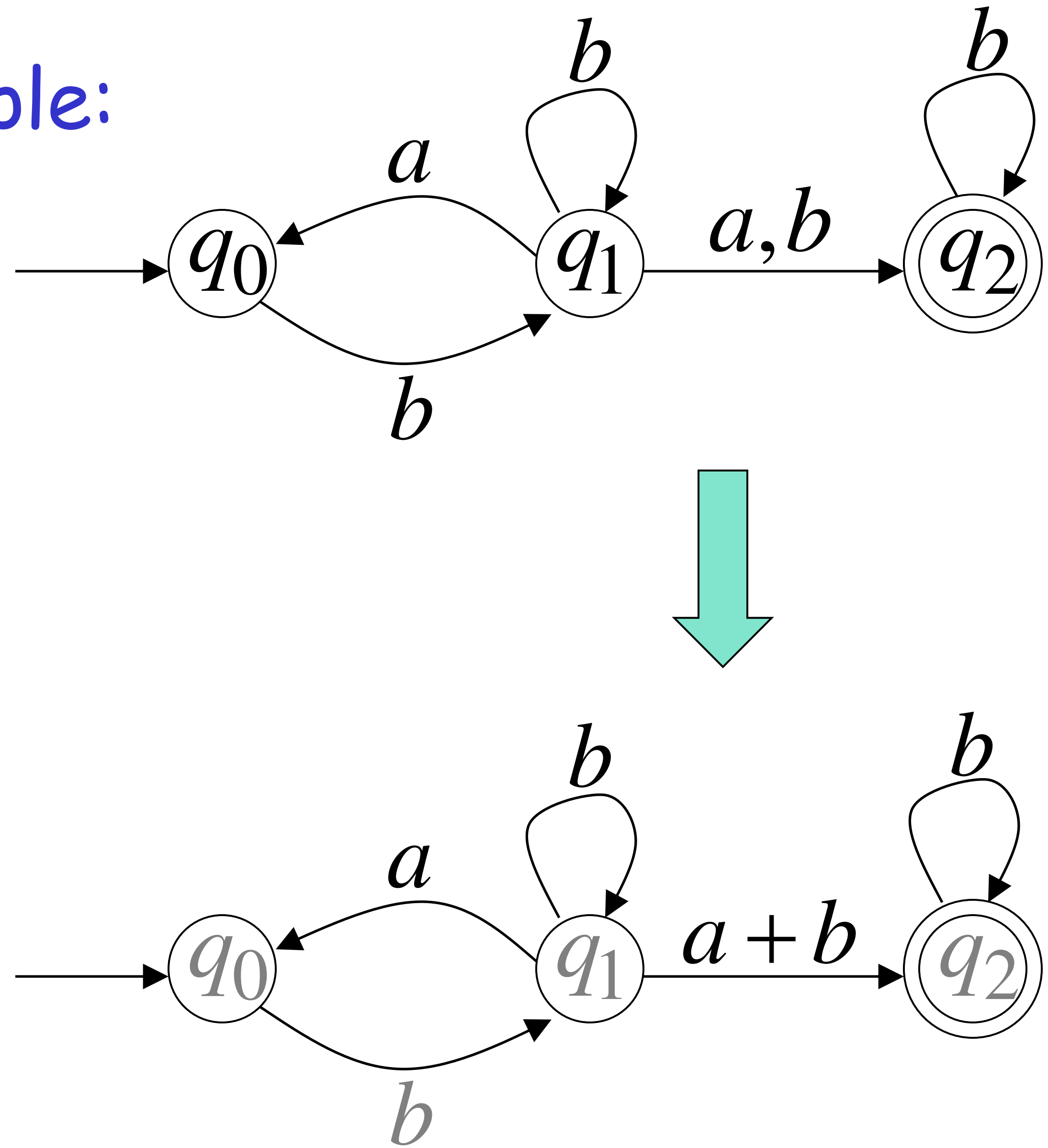
Generalized Transition Graph

in which transition labels are regular expressions

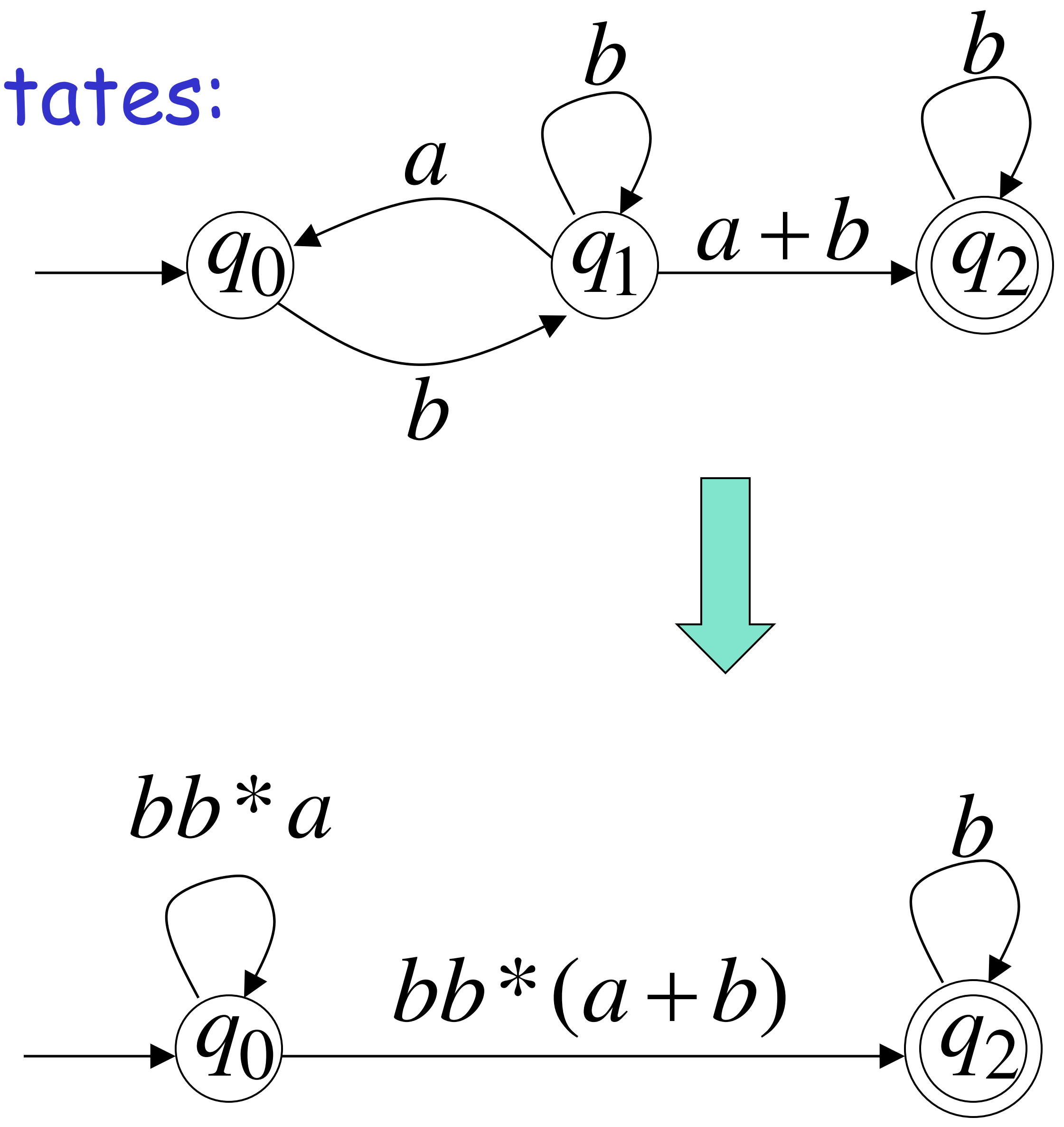
Example:



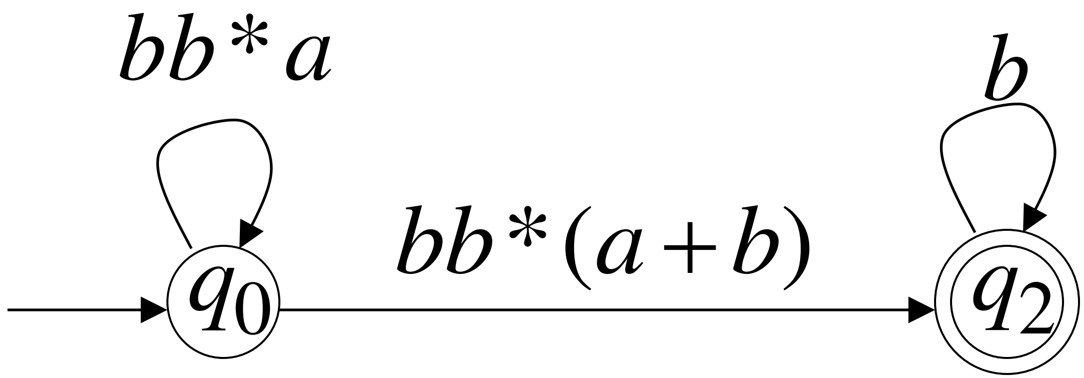
Another Example:



Reducing the states:



Resulting Regular Expression:

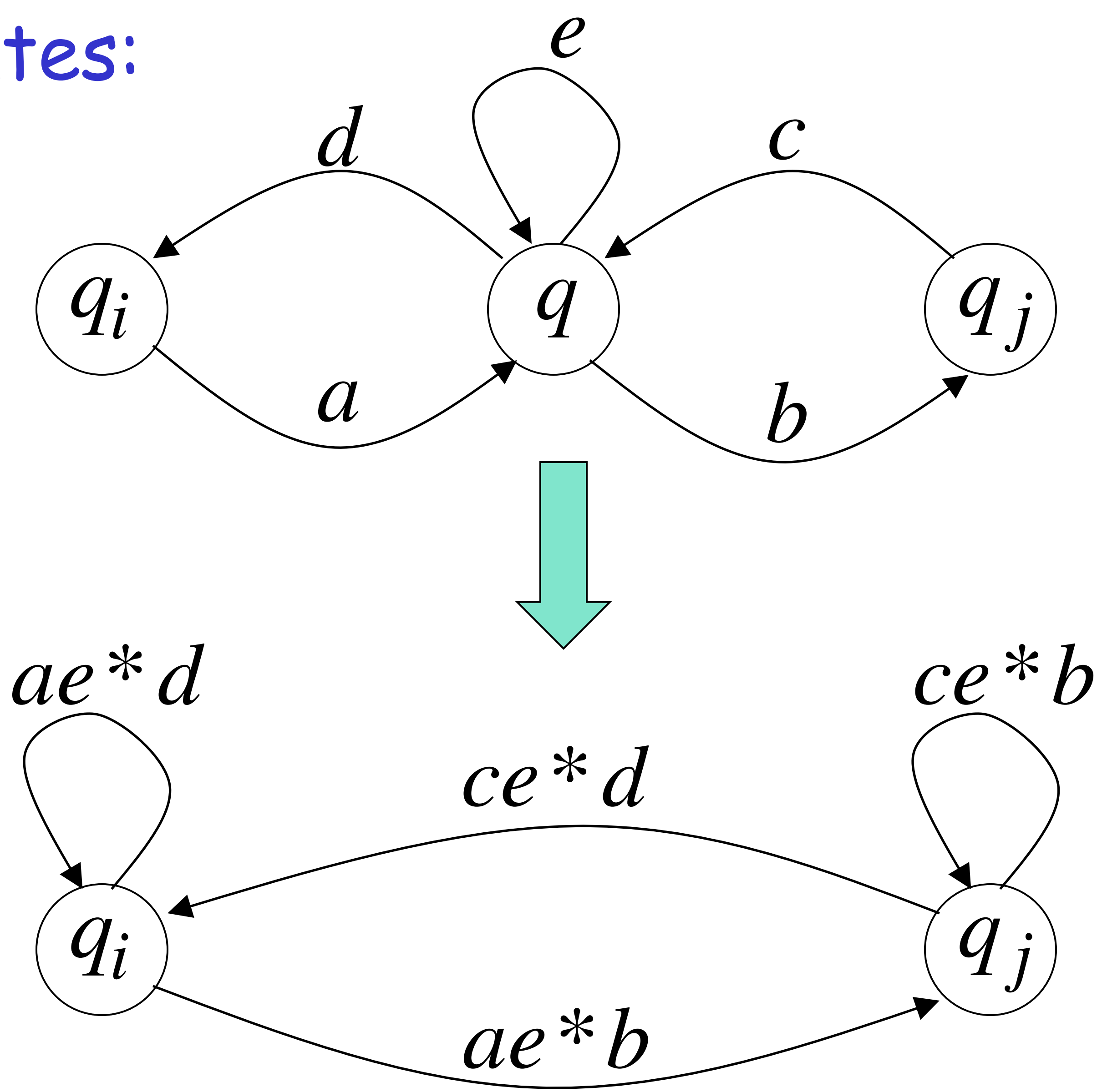


$$r = (bb^*a)^*bb^*(a+b)b^*$$

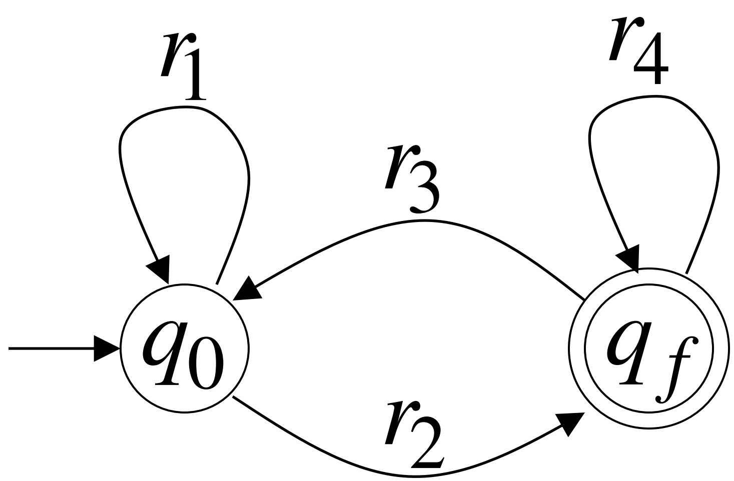
$$L(r) = L(M) = L$$

In General

Removing states:



The final transition graph:

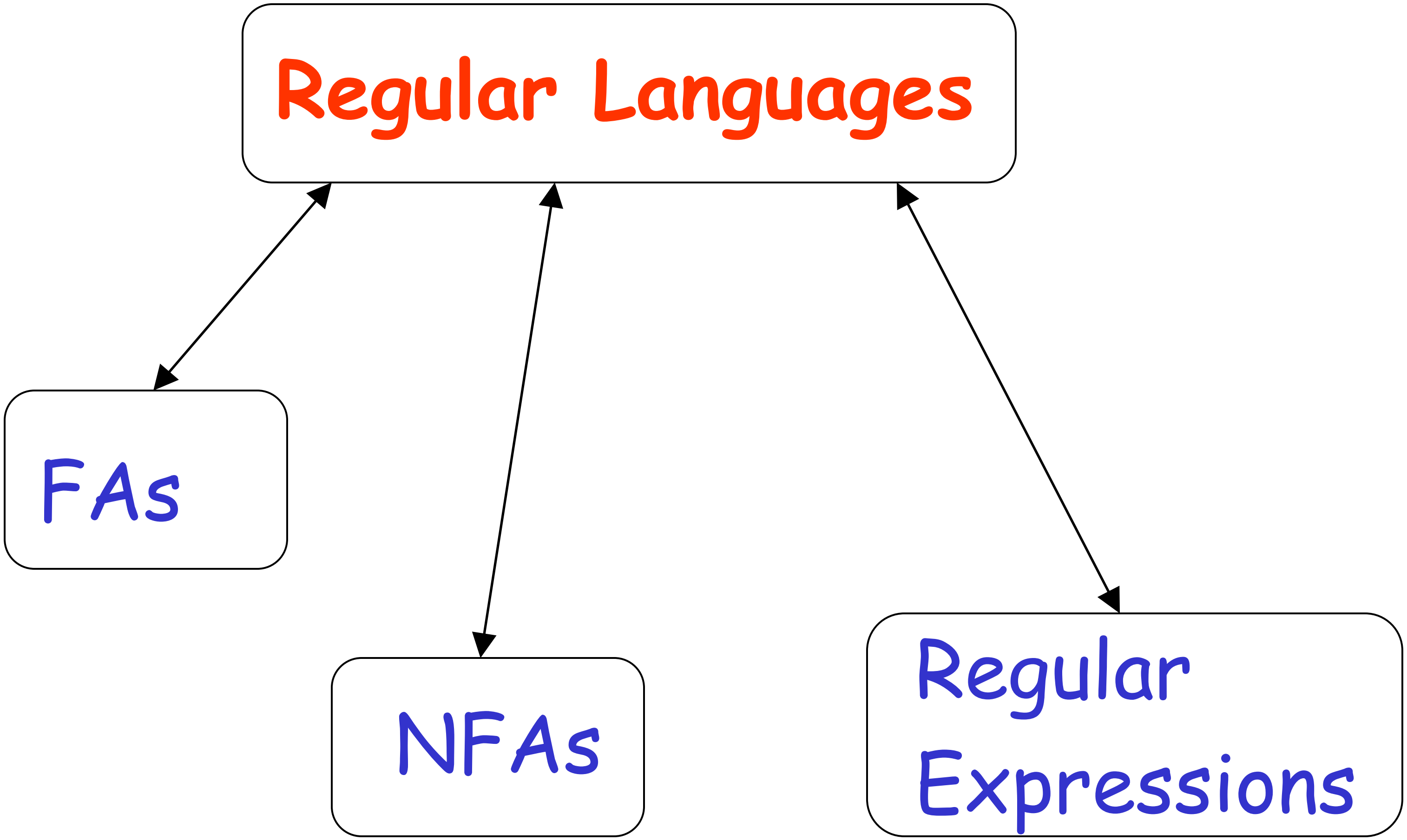


The resulting regular expression:

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*$$

$$L(r) = L(M) = L$$

Standard Representations of Regular Languages



When we say: We are given a Regular Language L

We mean: Language L is in a standard representation

Elementary Questions

about

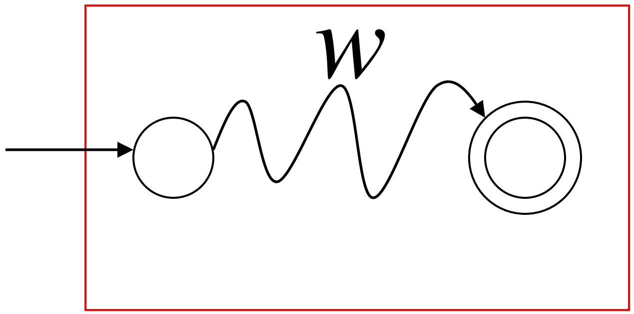
Regular Languages

Membership Question

Question: Given regular language L
and string w
how can we check if $w \in L$?

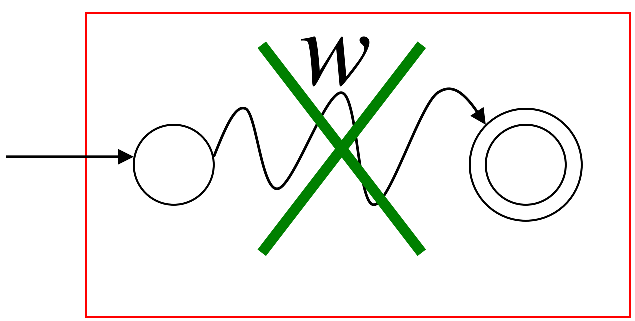
Answer: Take the DFA that accepts L
and check if w is accepted

DFA



$w \in L$

DFA



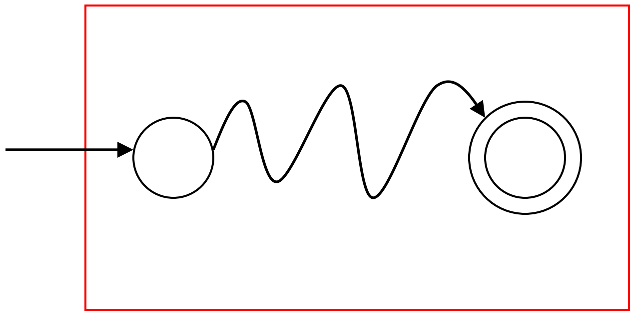
$w \notin L$

Question: Given regular language L how can we check if L is empty: $(L = \emptyset)$?

Answer: Take the DFA that accepts L

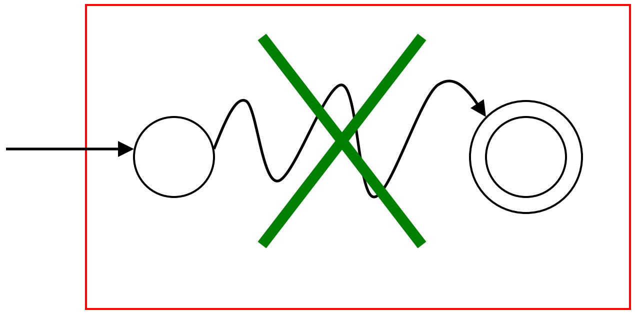
Check if there is any path from the initial state to a final state

DFA



$$L \neq \emptyset$$

DFA



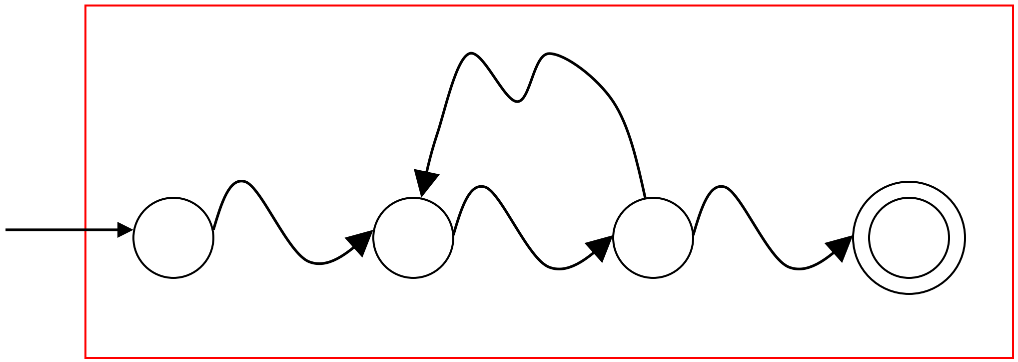
$$L = \emptyset$$

Question: Given regular language L how can we check if L is finite?

Answer: Take the DFA that accepts L

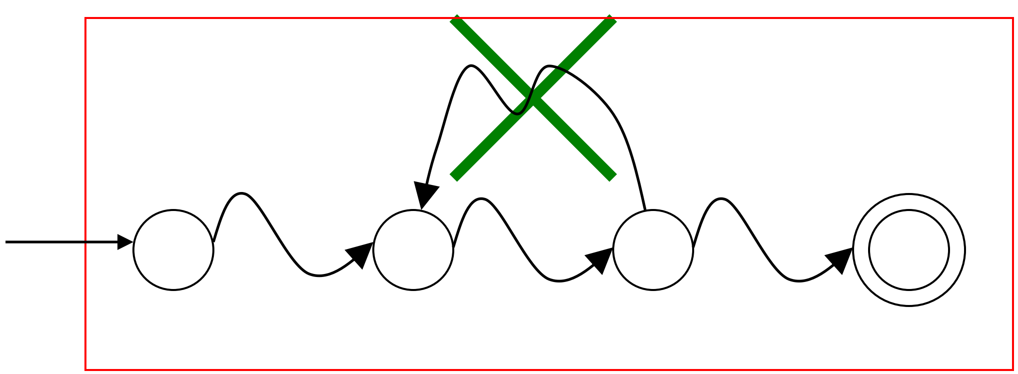
Check if there is a walk with cycle from the initial state to a final state

DFA



L is infinite

DFA



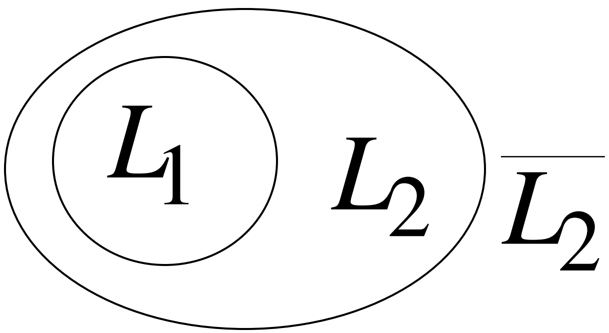
L is finite

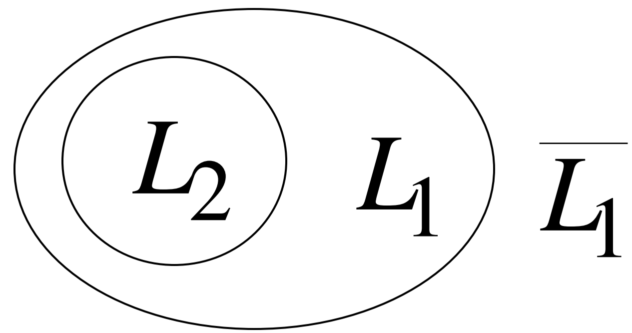
Question: Given regular languages L_1 and L_2 how can we check if $L_1 = L_2$?

Answer: Find if $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$

$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$$

$L_1 \cap \overline{L_2} = \emptyset$ **and** $\overline{L_1} \cap L_2 = \emptyset$





$L_1 \subseteq L_2$ $\qquad$ $L_2 \subseteq L_1$

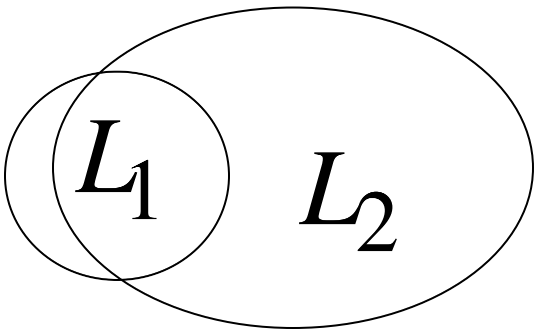
$$L_1 = L_2$$

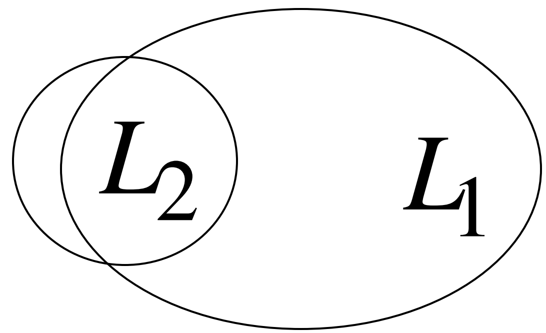
$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) \neq \emptyset$$

$$L_1 \cap \overline{L_2} \neq \emptyset$$

or

$$\overline{L_1} \cap L_2 \neq \emptyset$$





$$L_1 \not\subset L_2$$

$$L_2 \not\subset L_1$$

$$L_1 \neq L_2$$