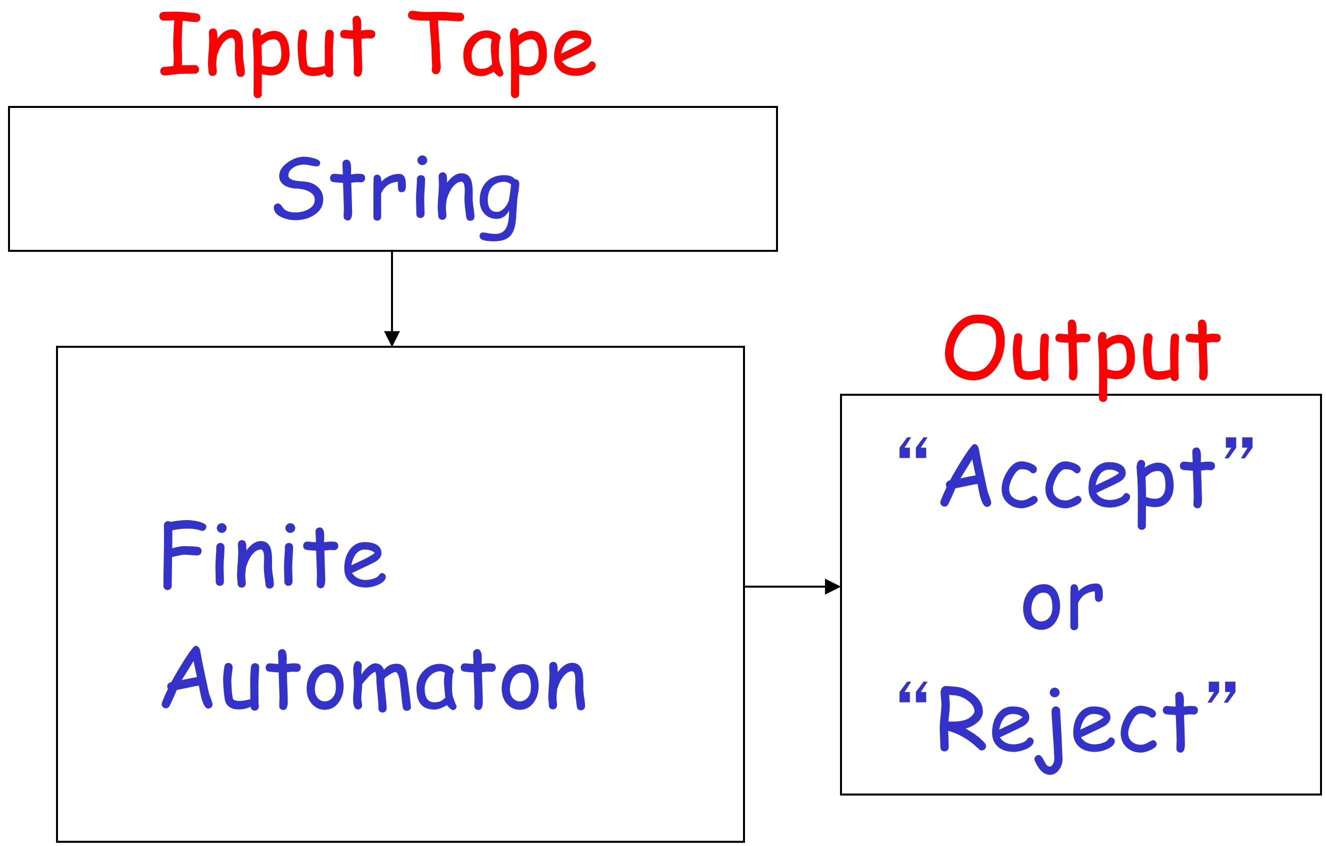
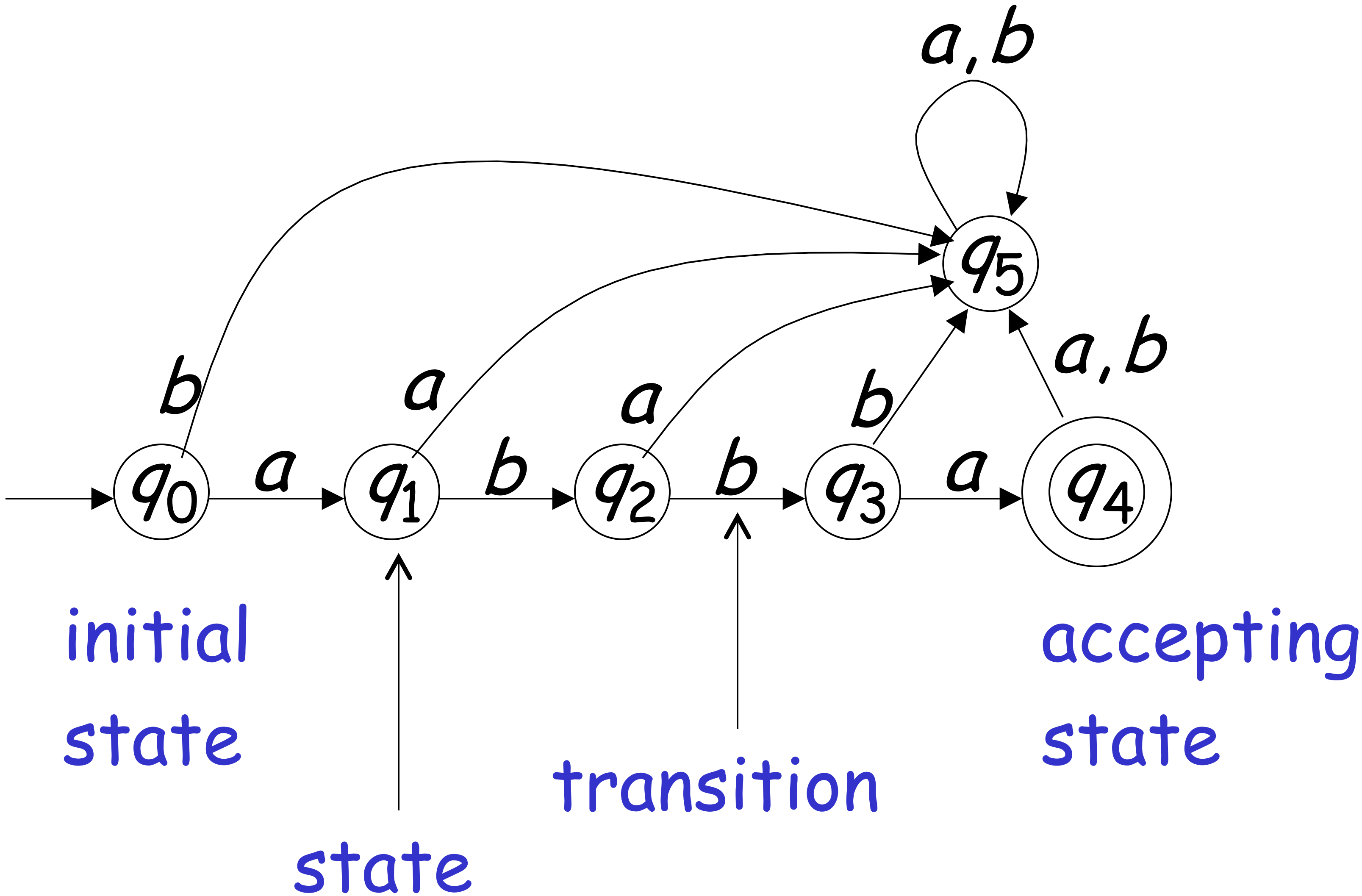
Deterministic Finite Automata And Regular Languages

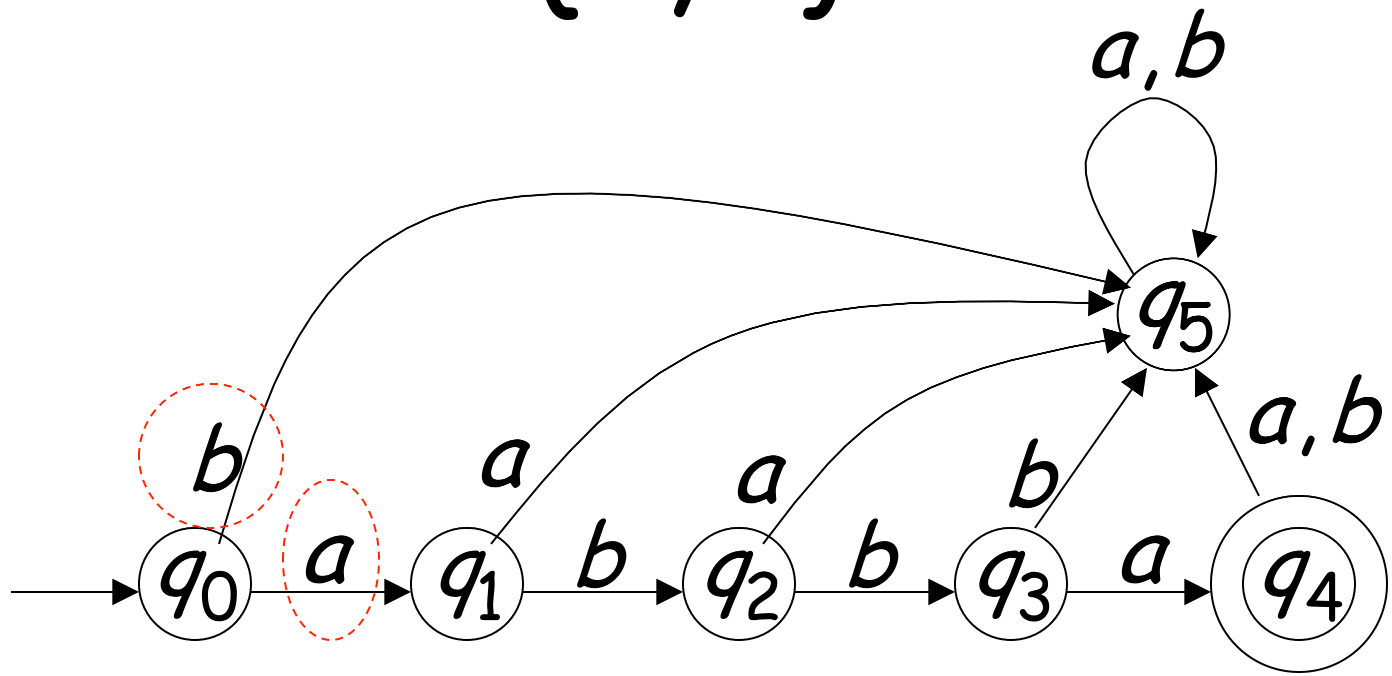
Deterministic Finite Automaton (DFA)



Transition Graph

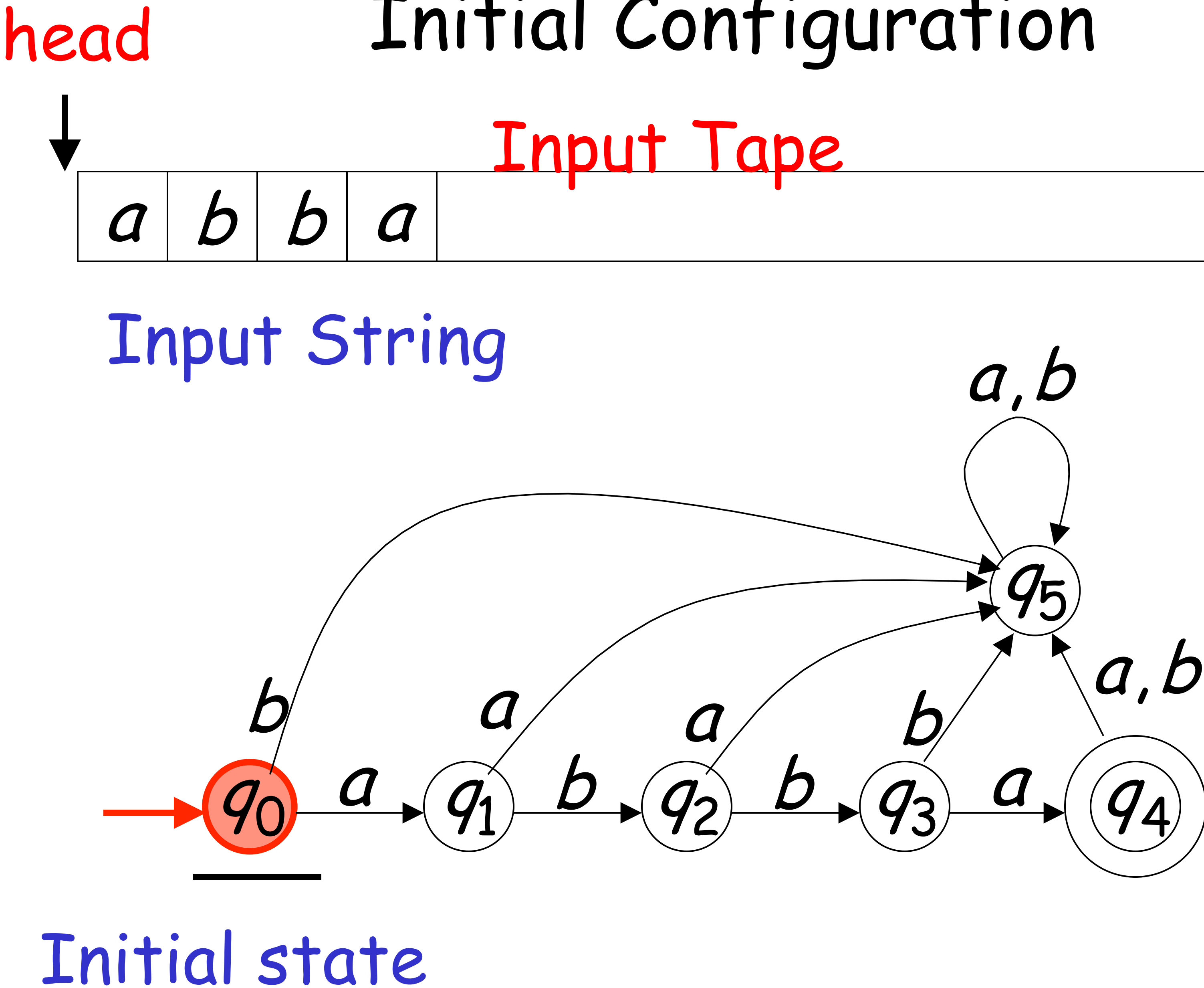


Alphabet $\Sigma = \{a, b\}$

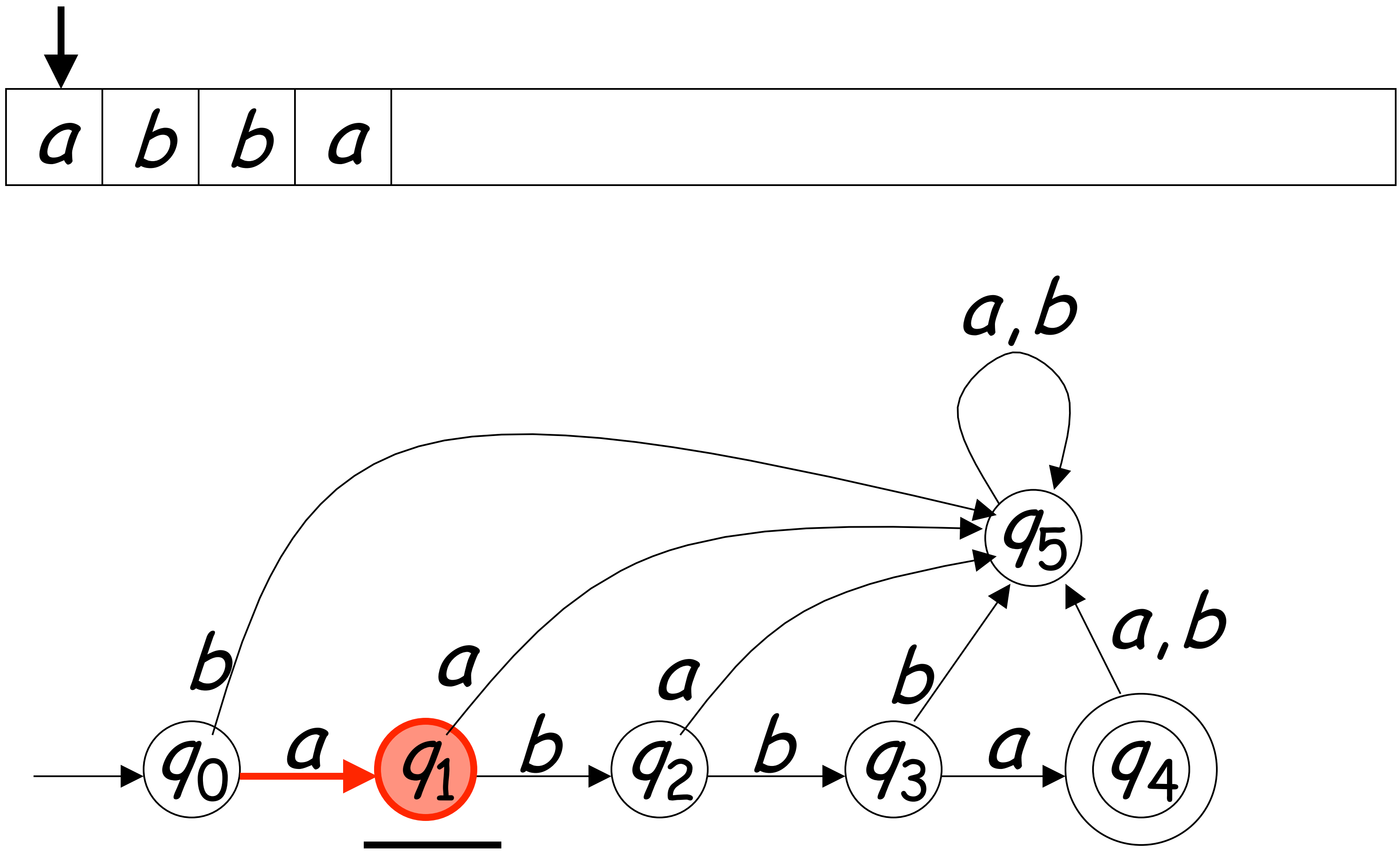


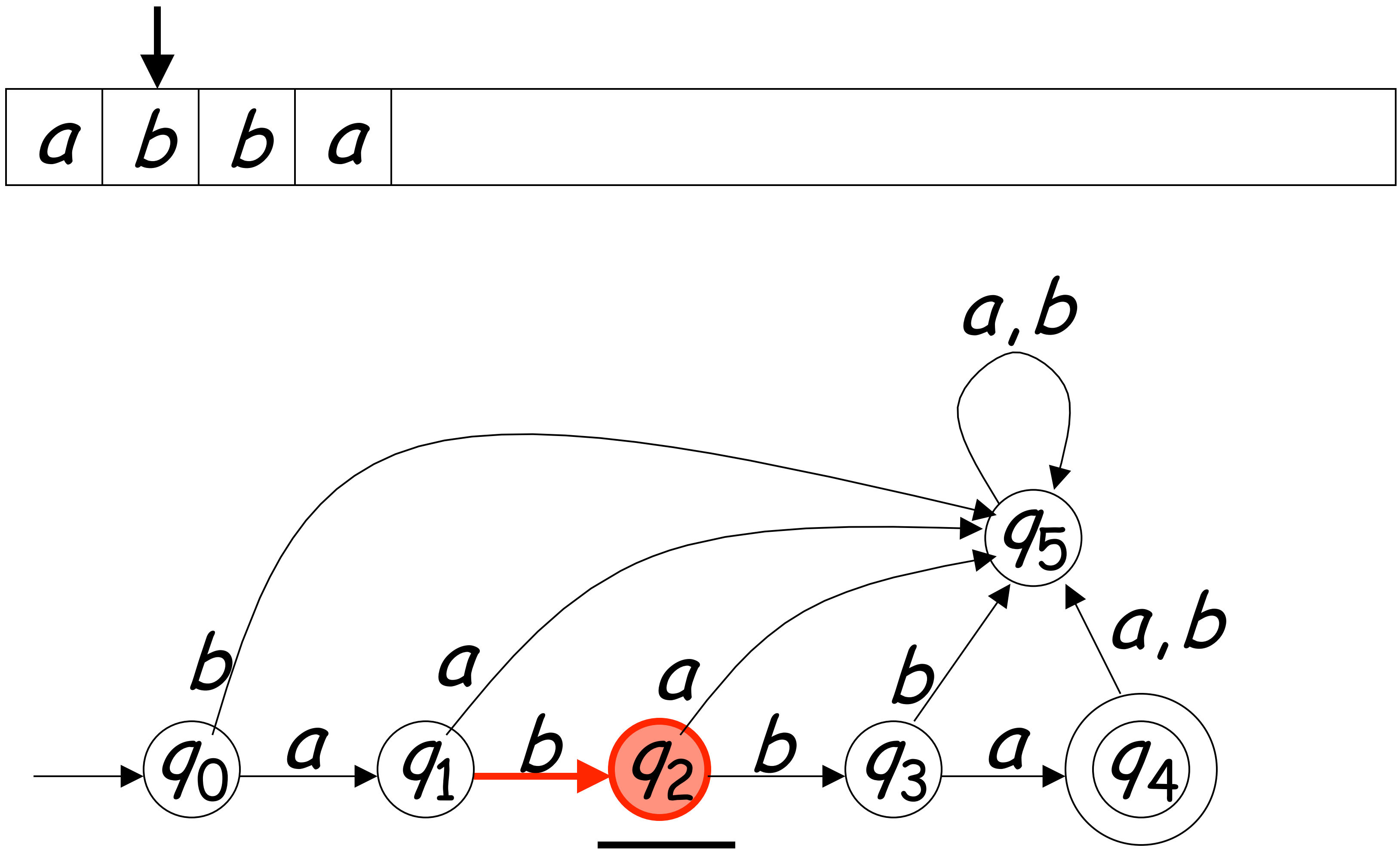
For <u>every</u> state, there is a transition
for <u>every</u> symbol in the alphabet

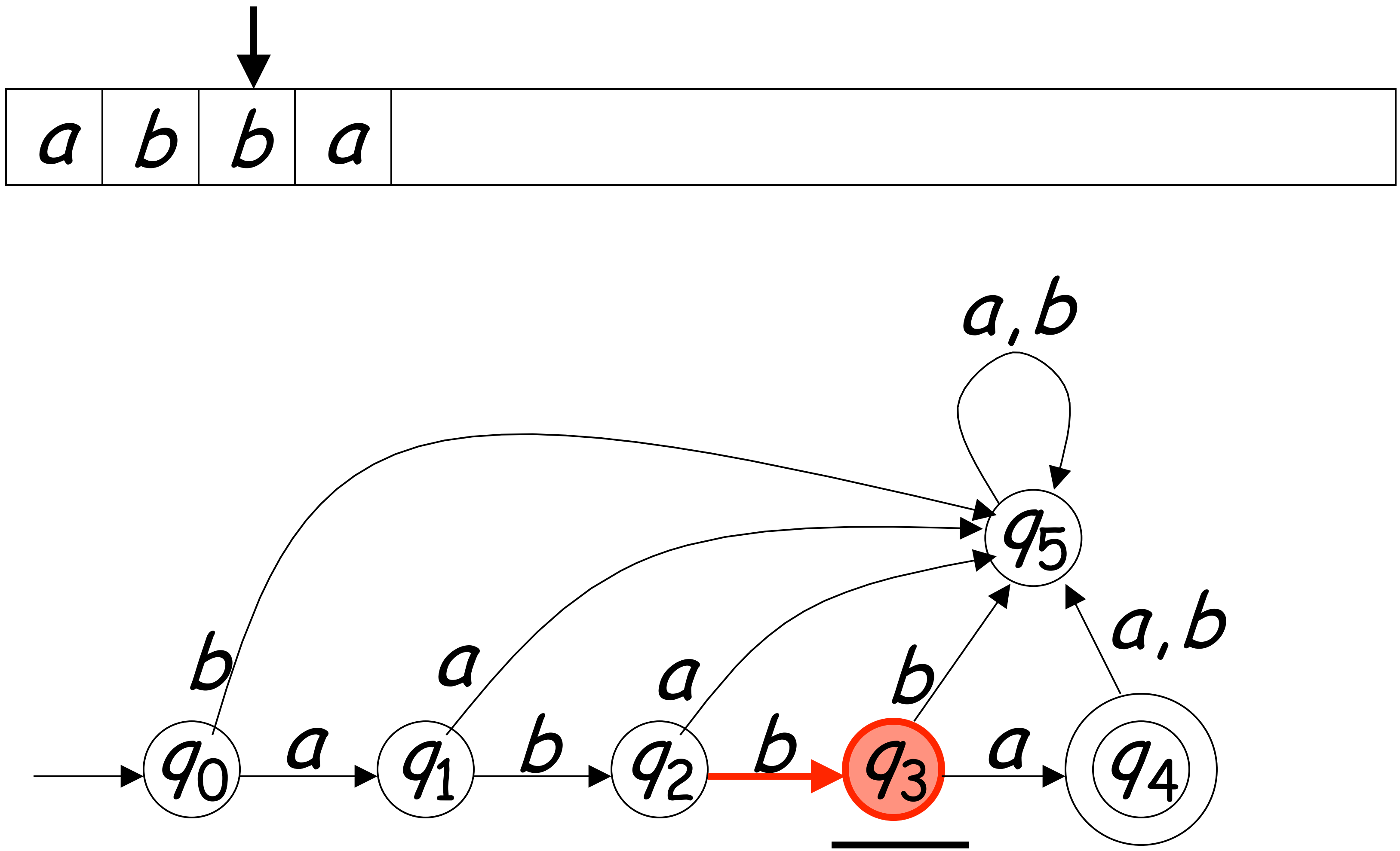
Initial Configuration



Scanning the Input

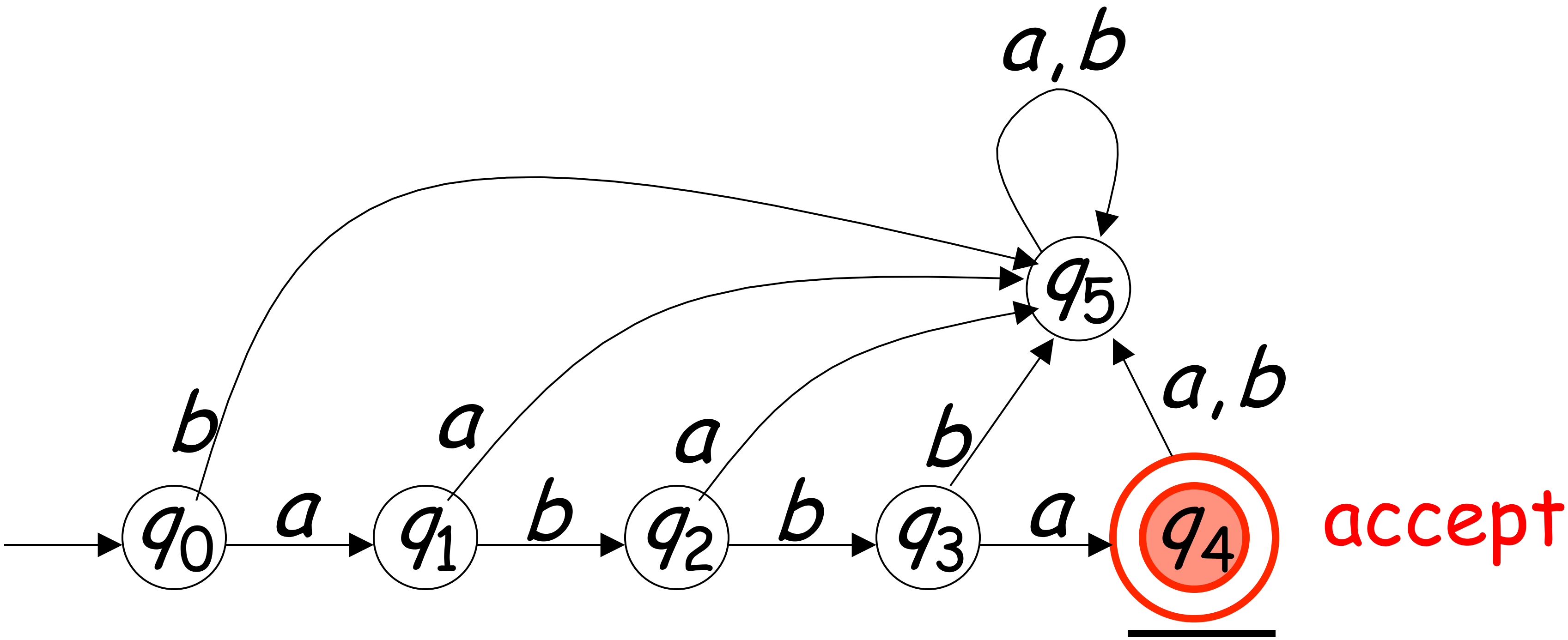



a
b
b
a
a, b
b
a
a
b
a, b
a
b
b
a
q_0
q_1
q_2
q_3
q_4
q_5


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

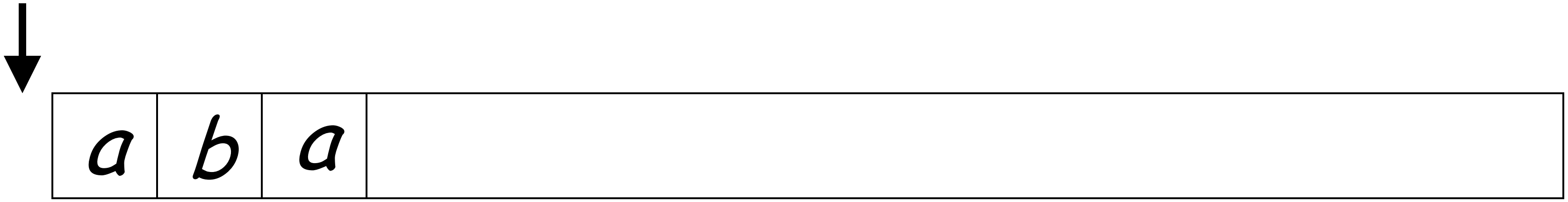
Input finished

a	b	b	a	

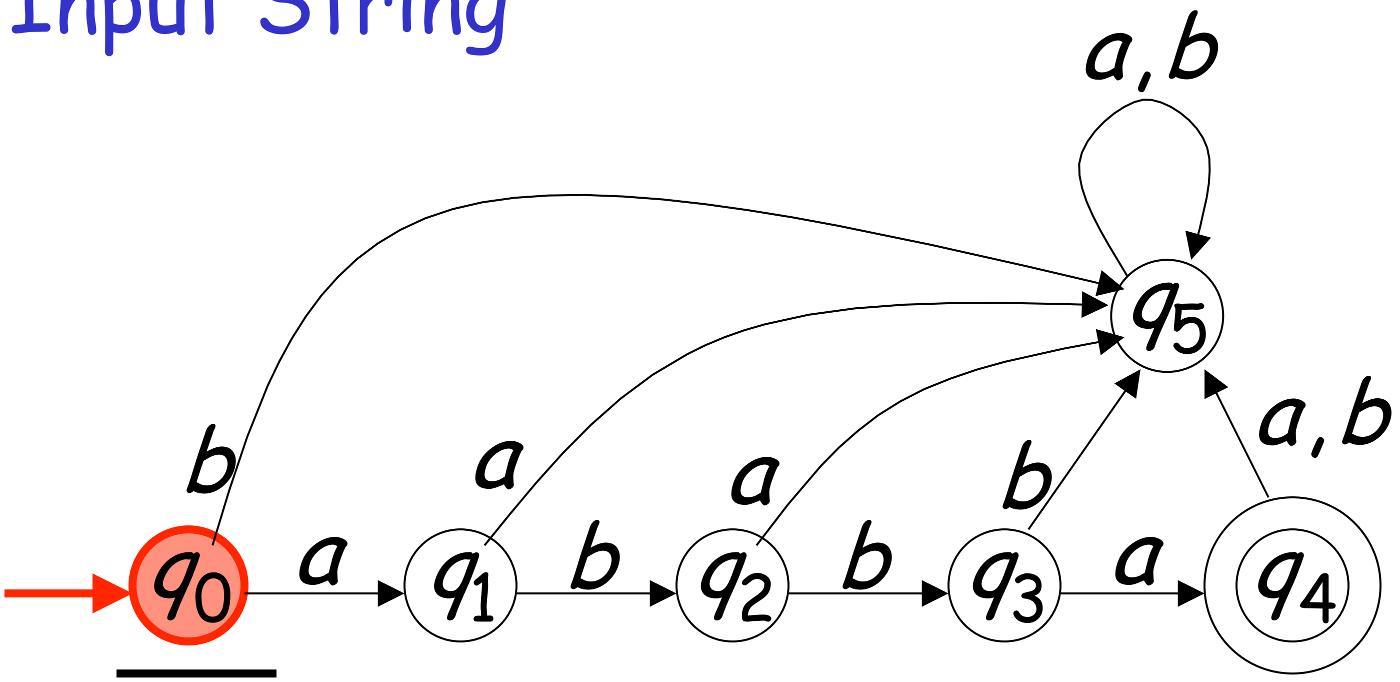


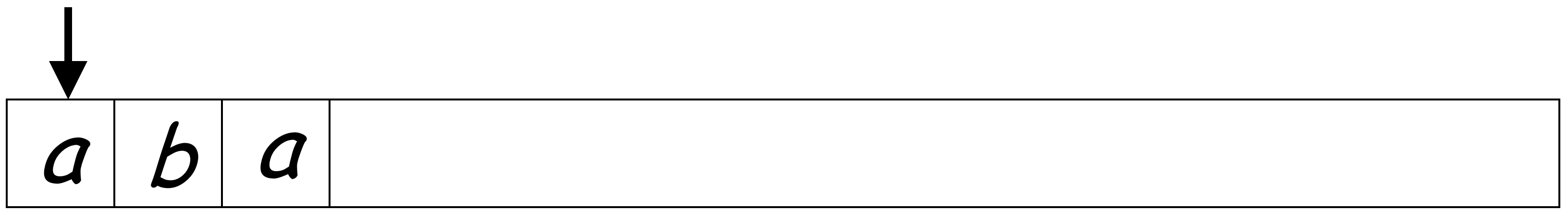
Last state determines the outcome

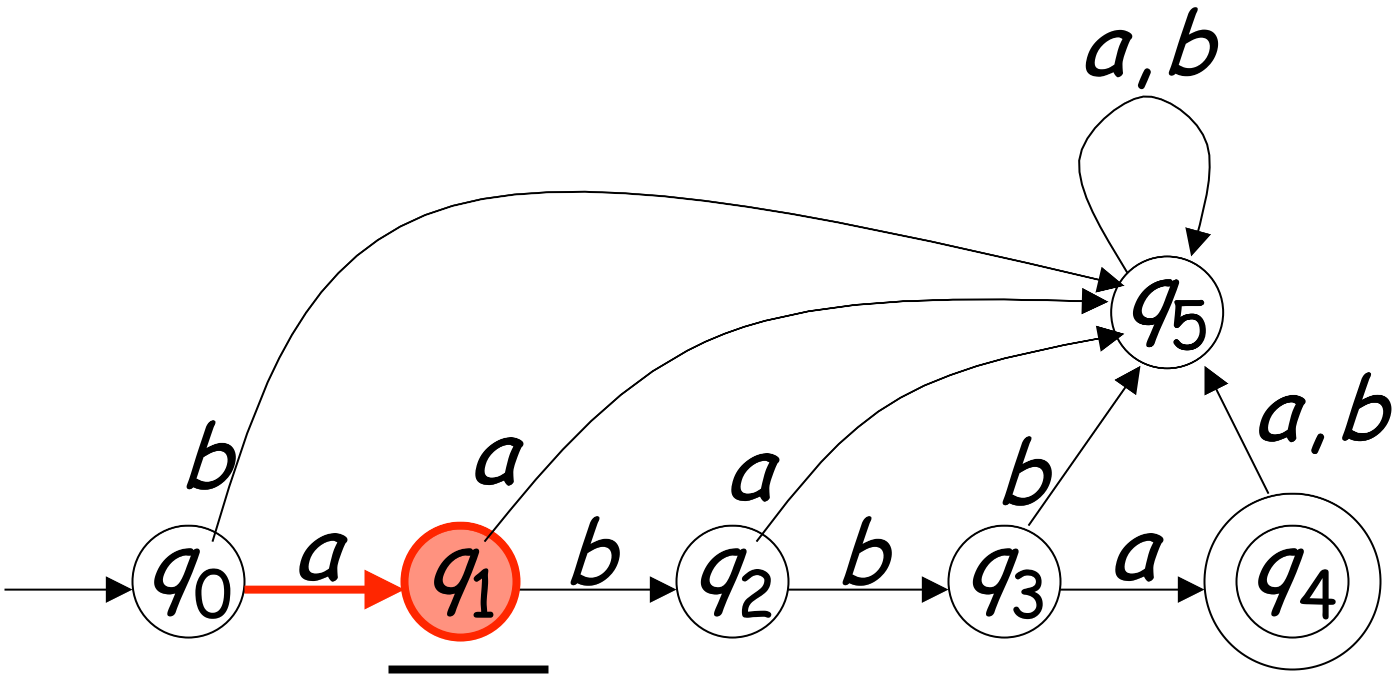
A Rejection Case

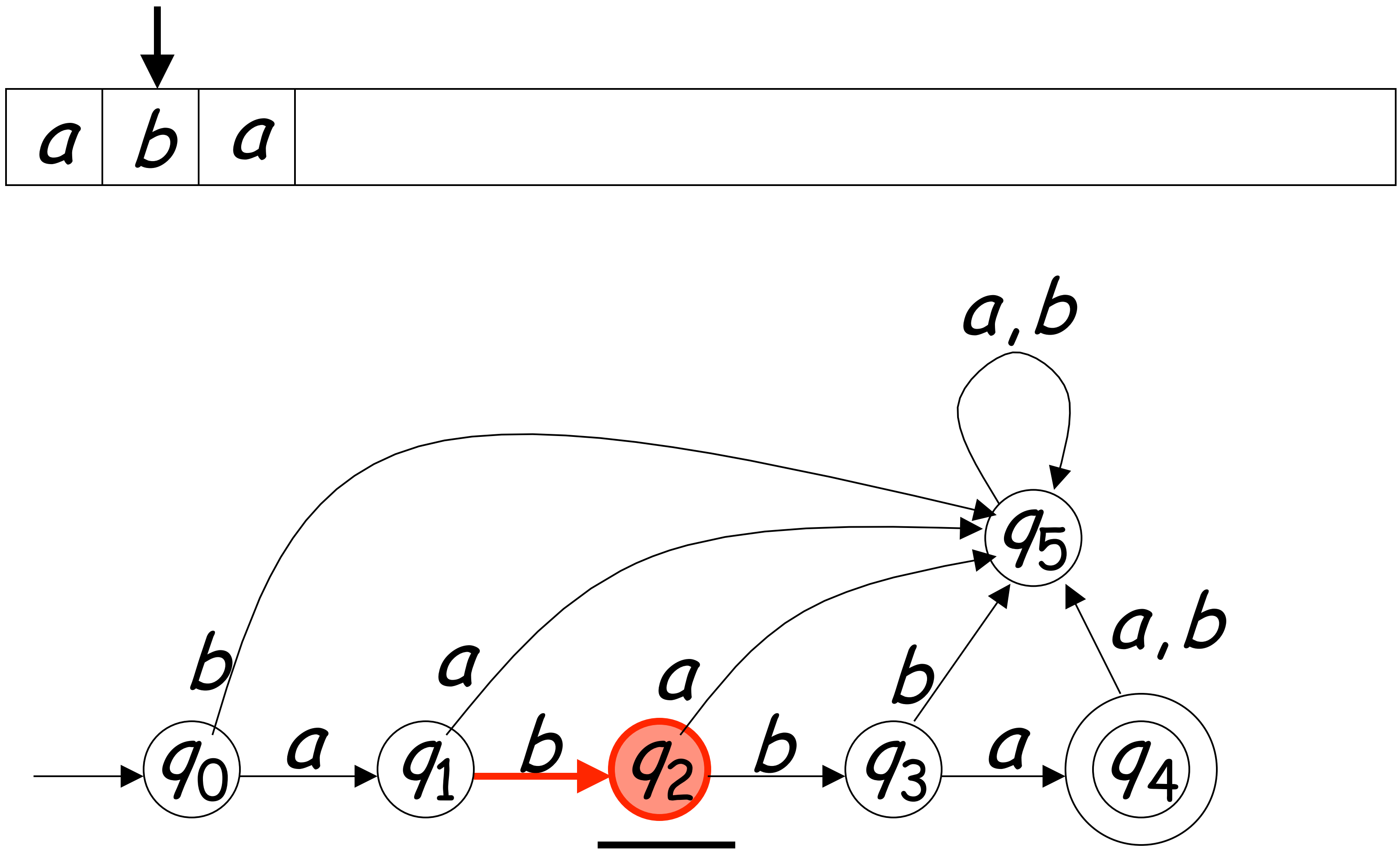


Input String

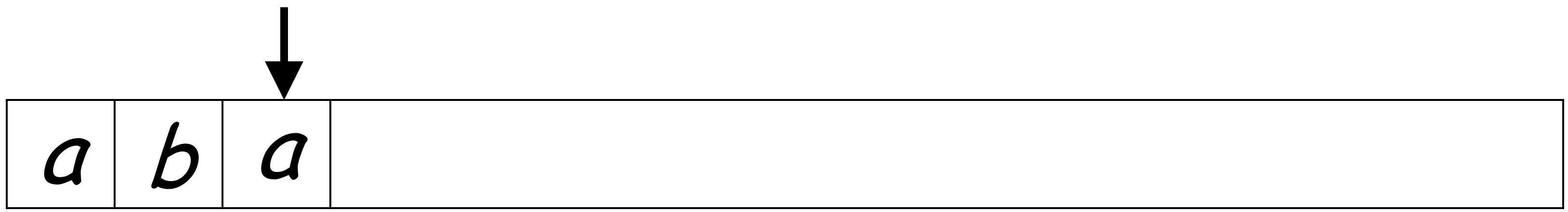


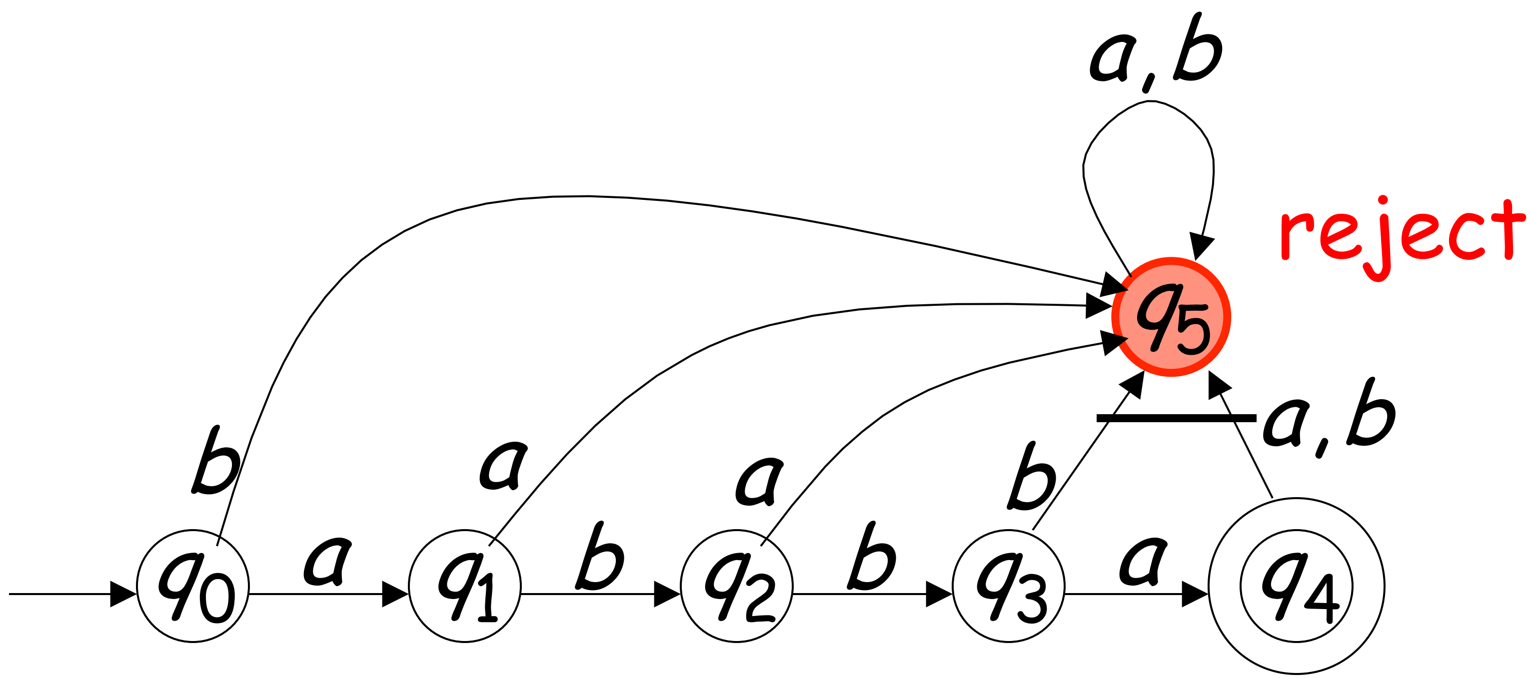

a
b
a


a,b
b
a
a
b
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





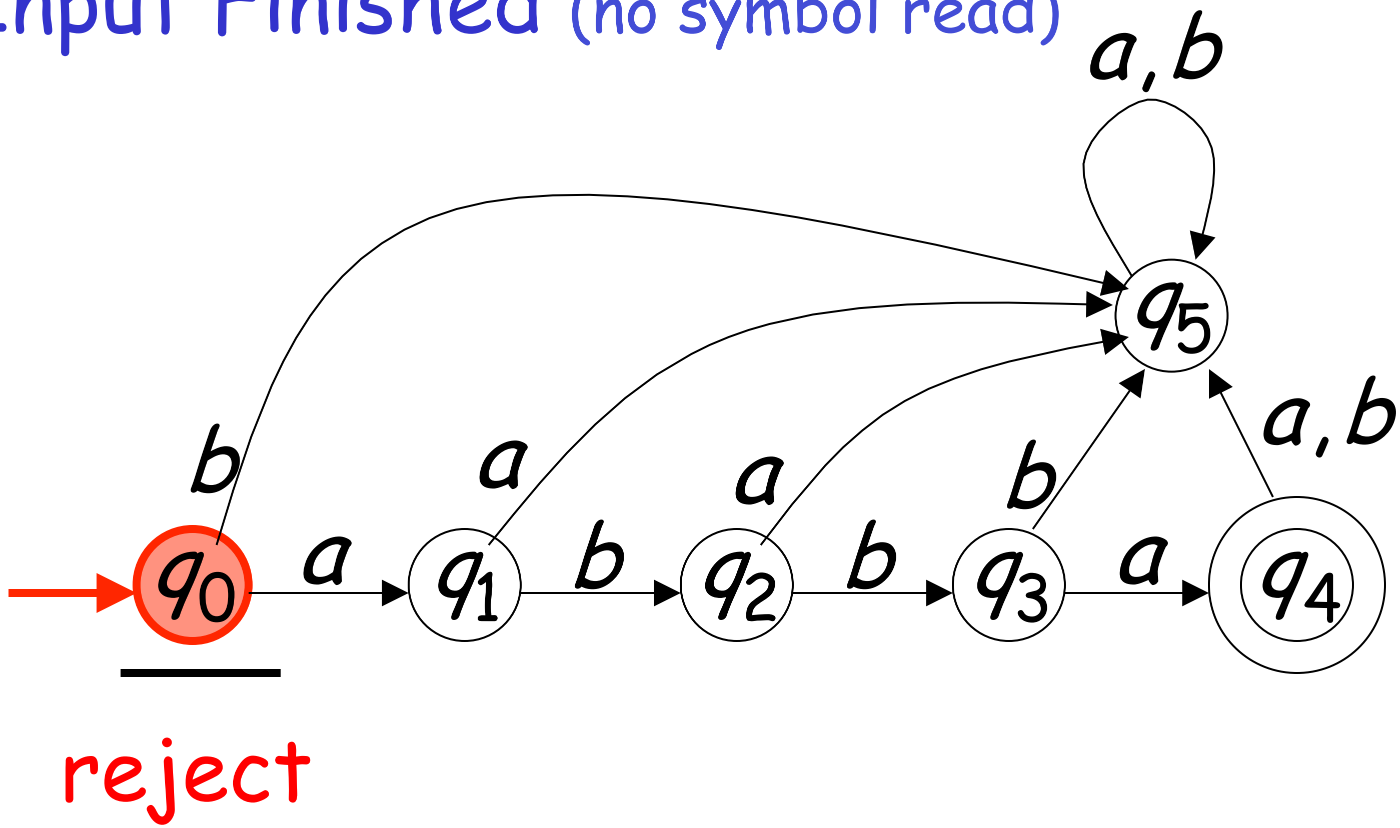
Last state determines the outcome

Another Rejection Case

Tape is empty

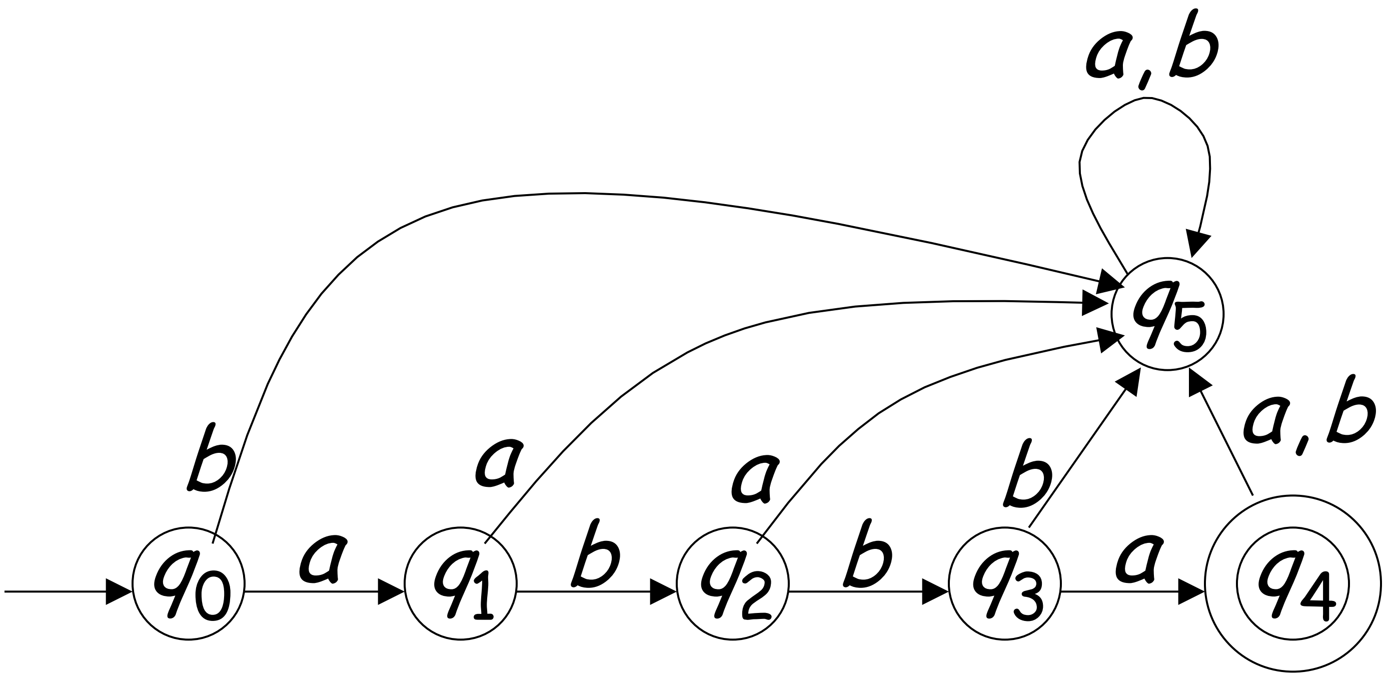
(λ)

Input Finished (no symbol read)



This automaton accepts only one string

Language Accepted: $L = \{abba\}$



To accept a string:

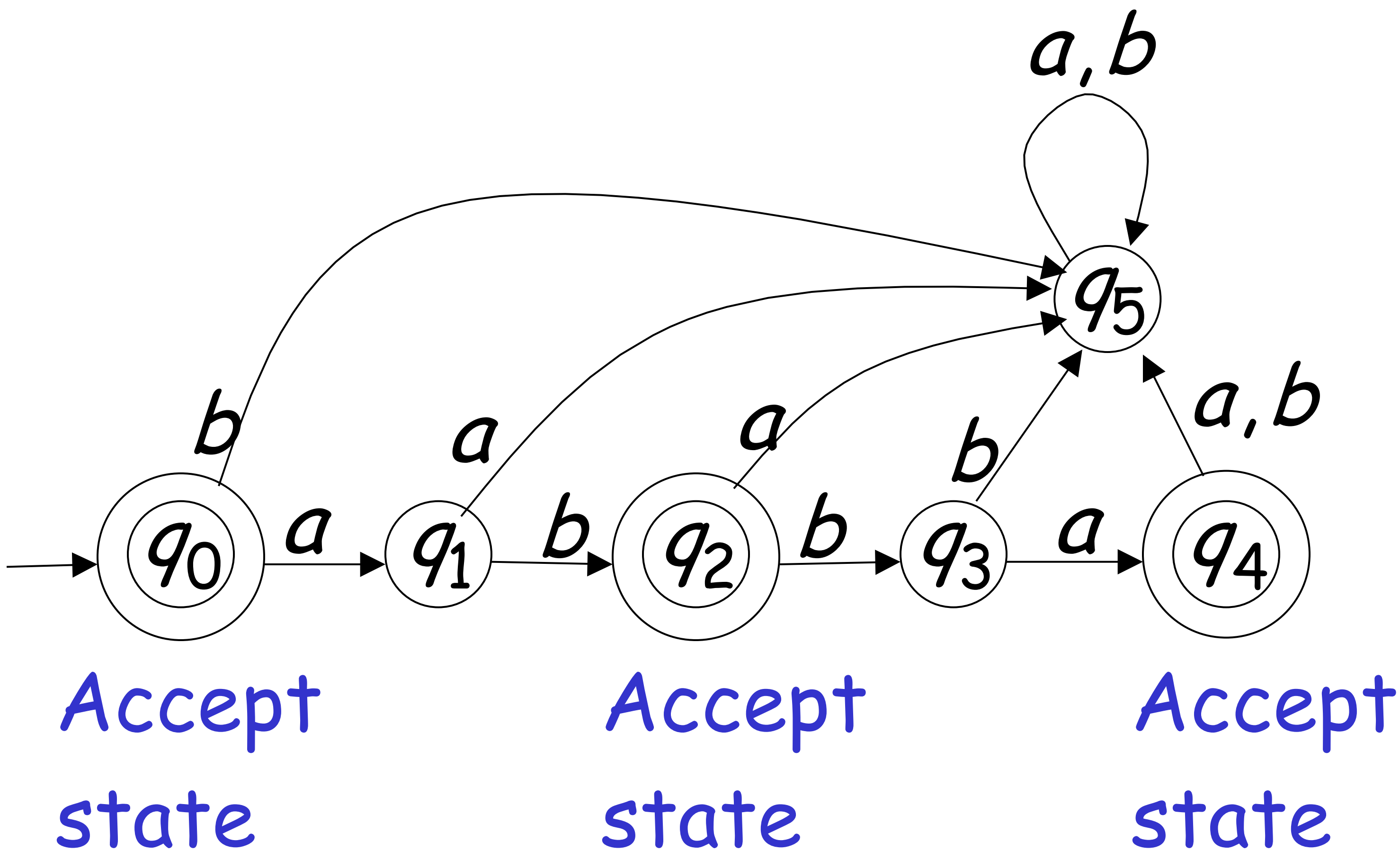
all the input string is scanned
and the last state is accepting

To reject a string:

all the input string is scanned
and the last state is non-accepting

Another Example

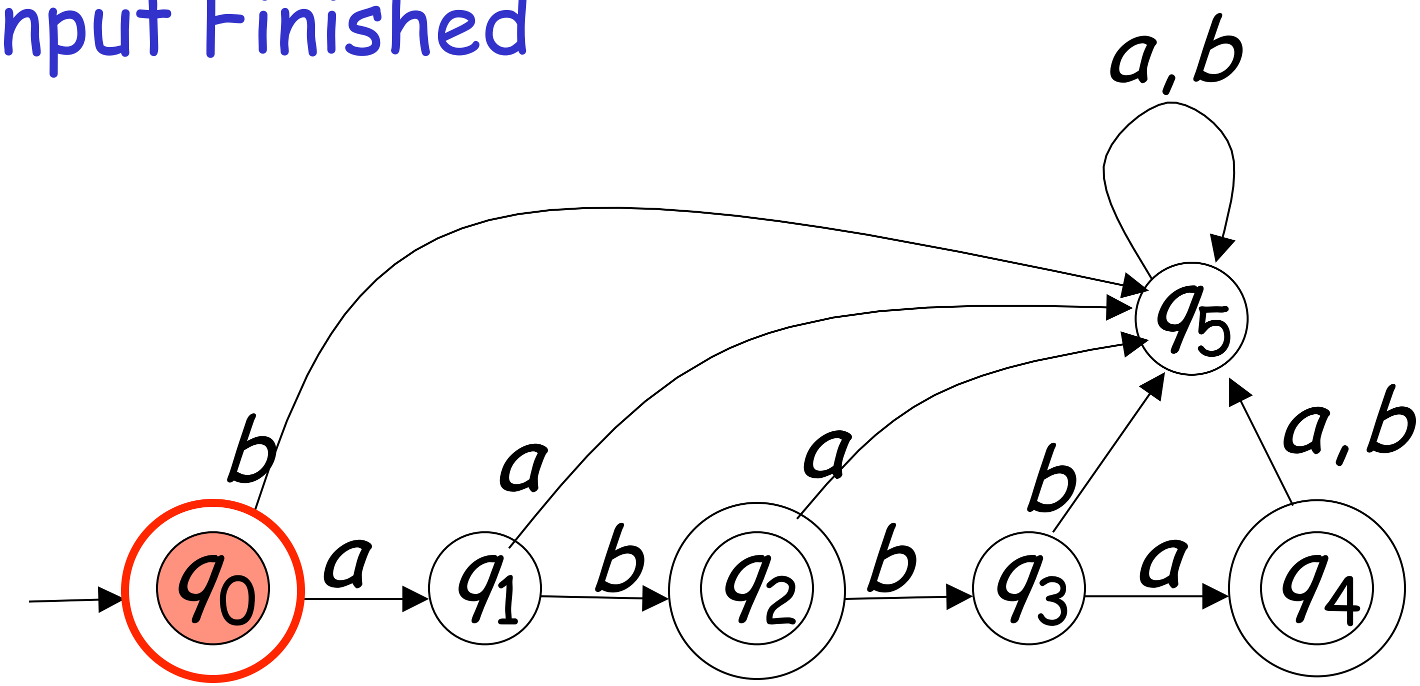
$$L = \{\lambda, ab, abba\}$$



Empty Tape

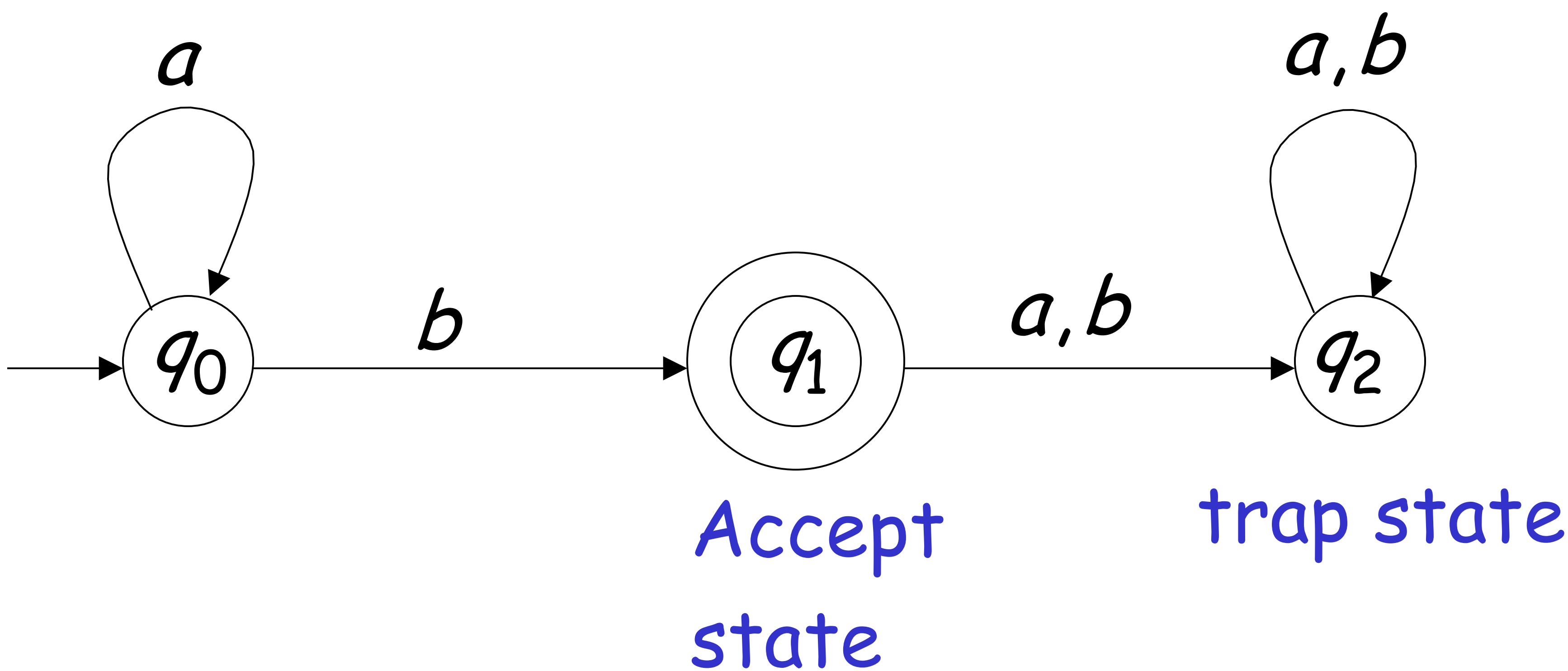
(λ)

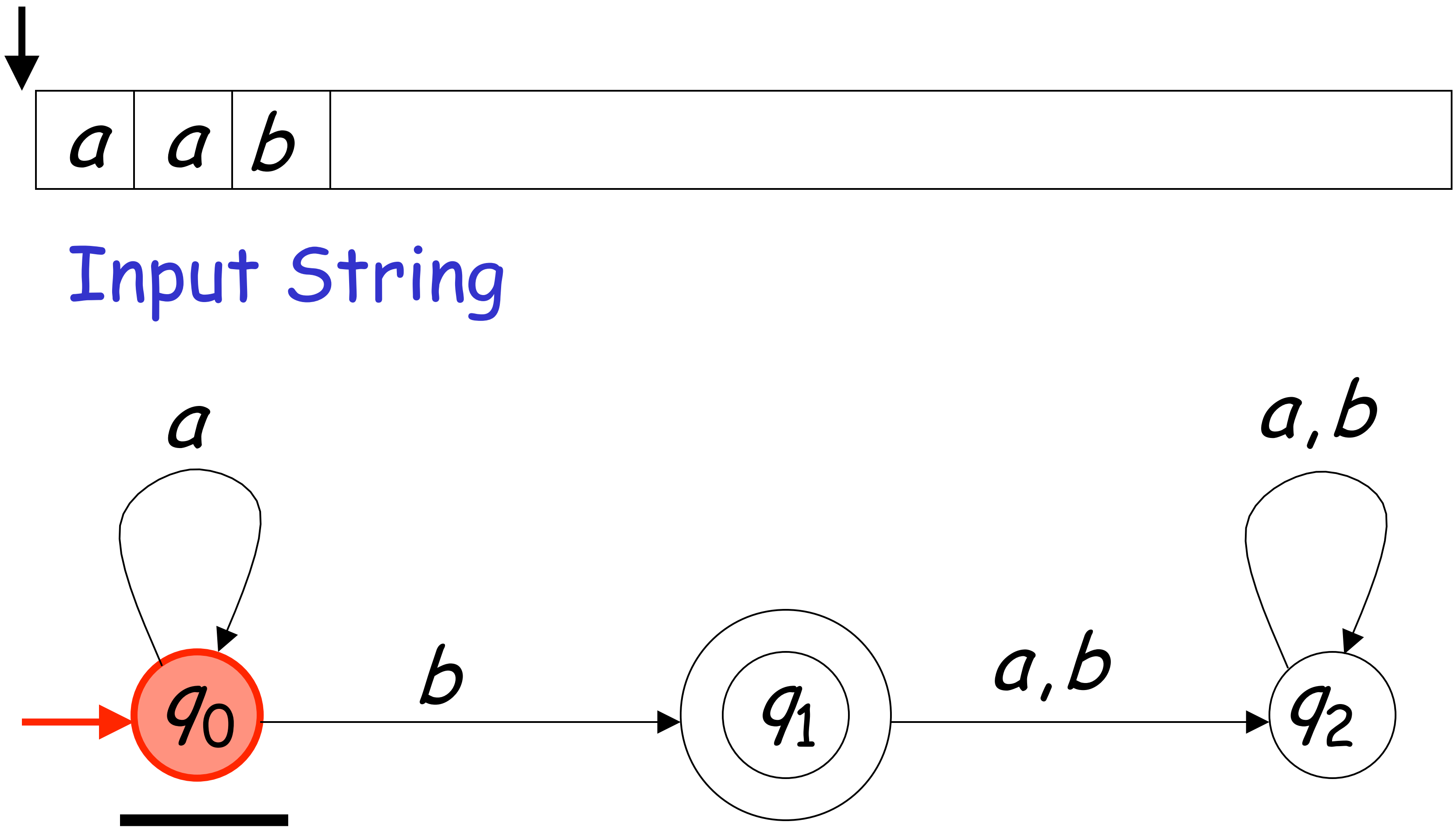
Input Finished

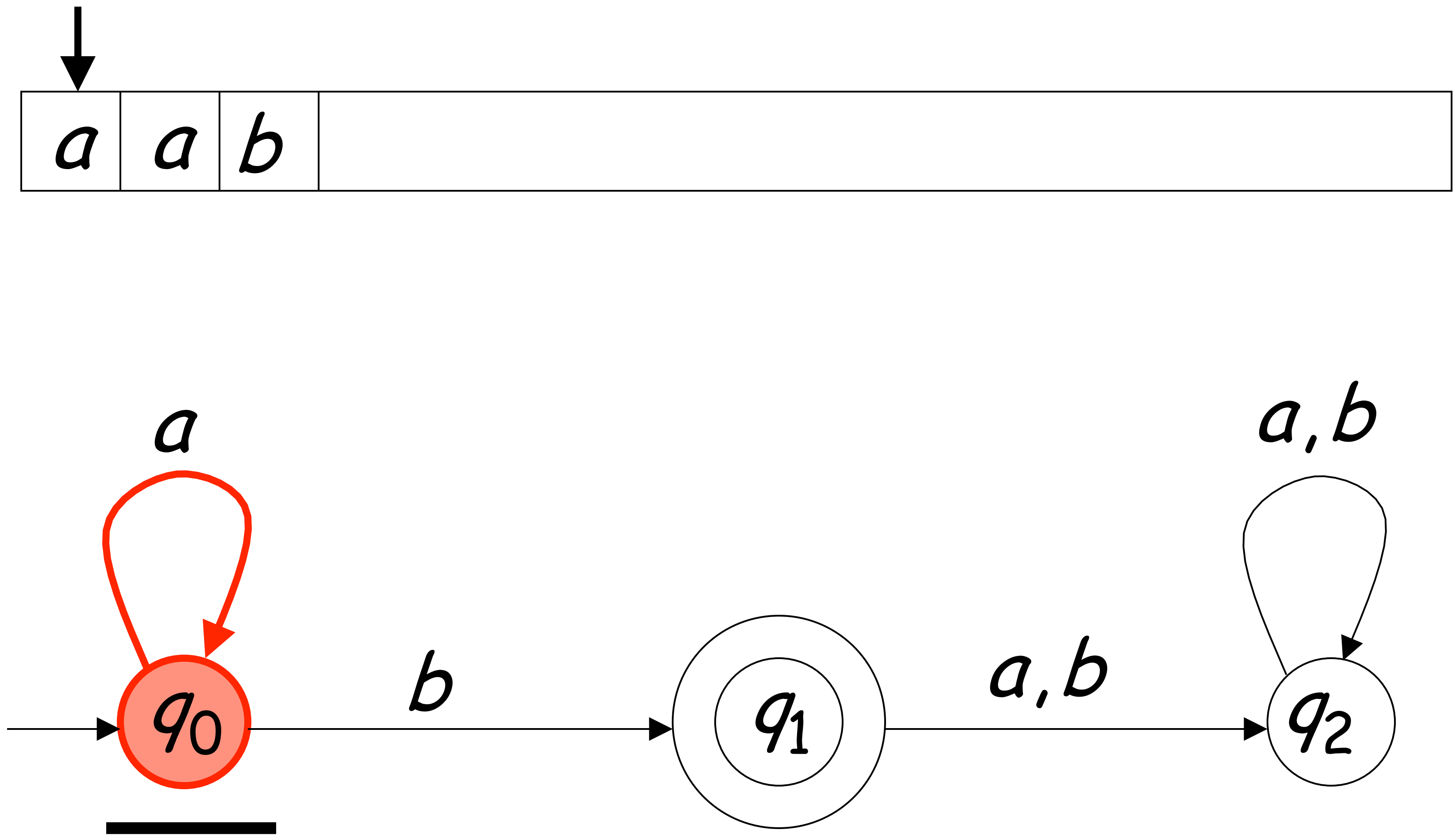


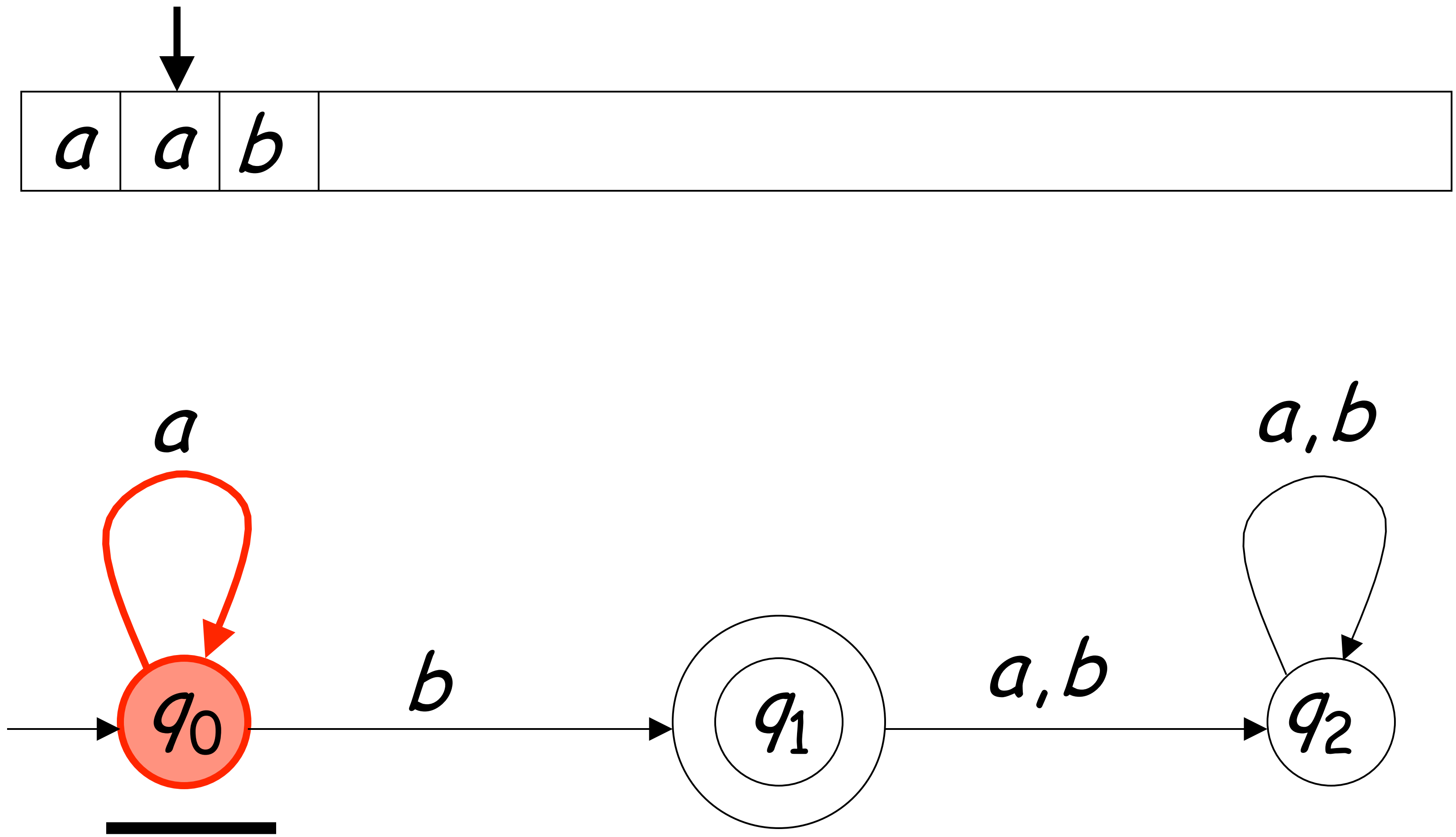
accept

Another Example

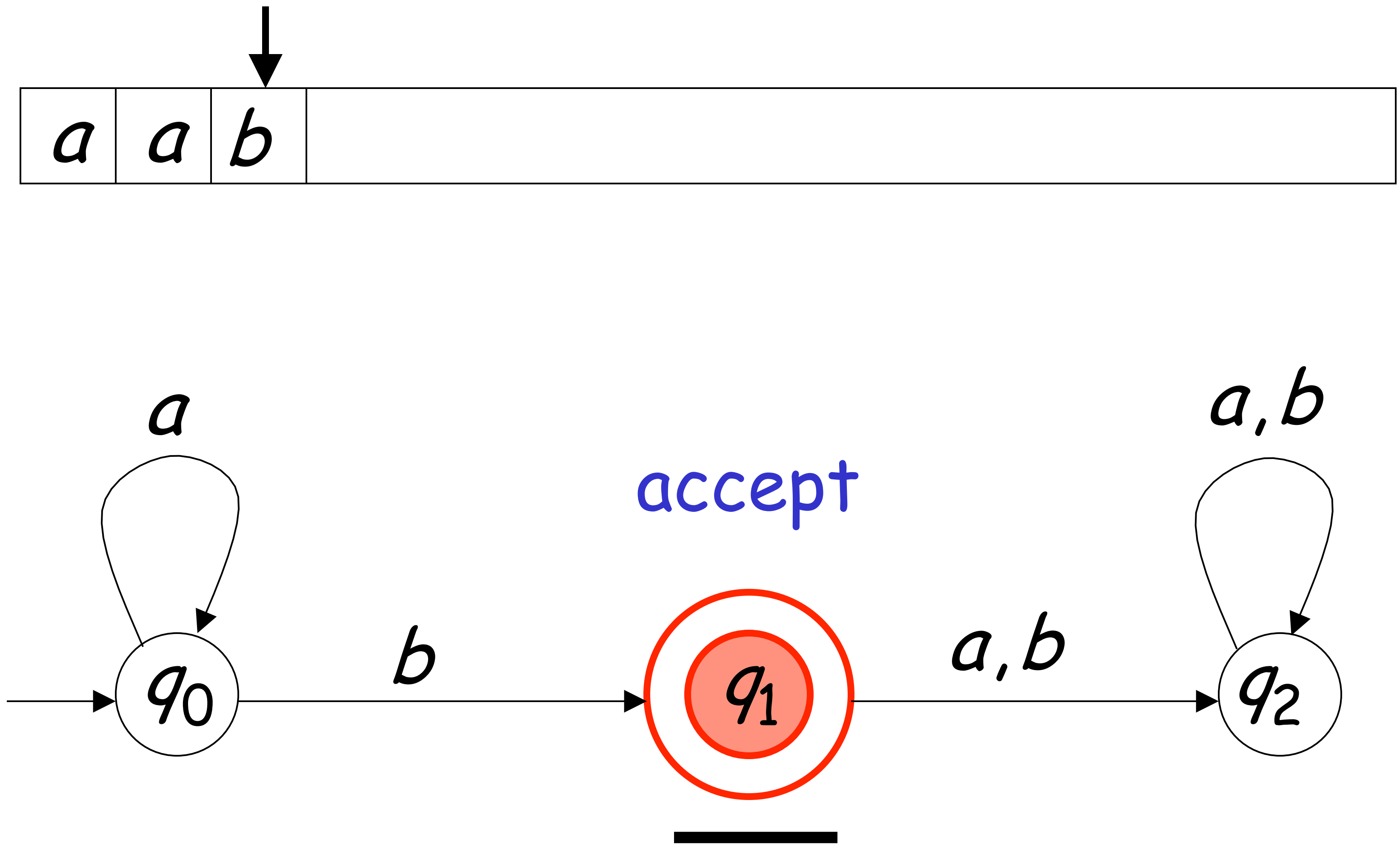



a
a
b
Input String
a
a, b
b
q_0
q_1
a, b
q_2

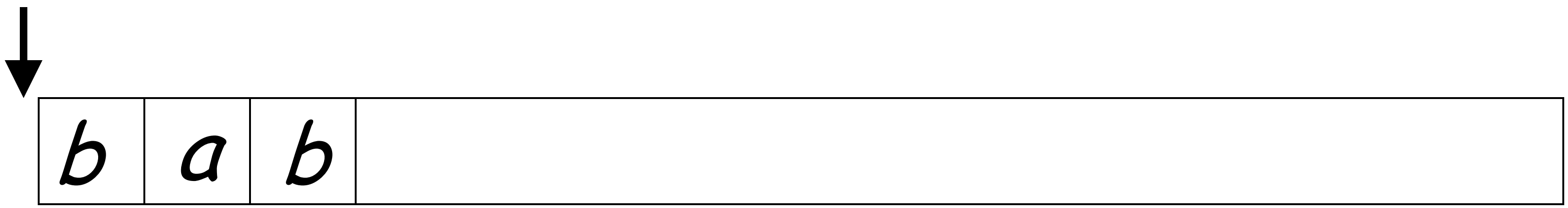

a
a
b
a
b
a,b
a,b
q0
q1
q2


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

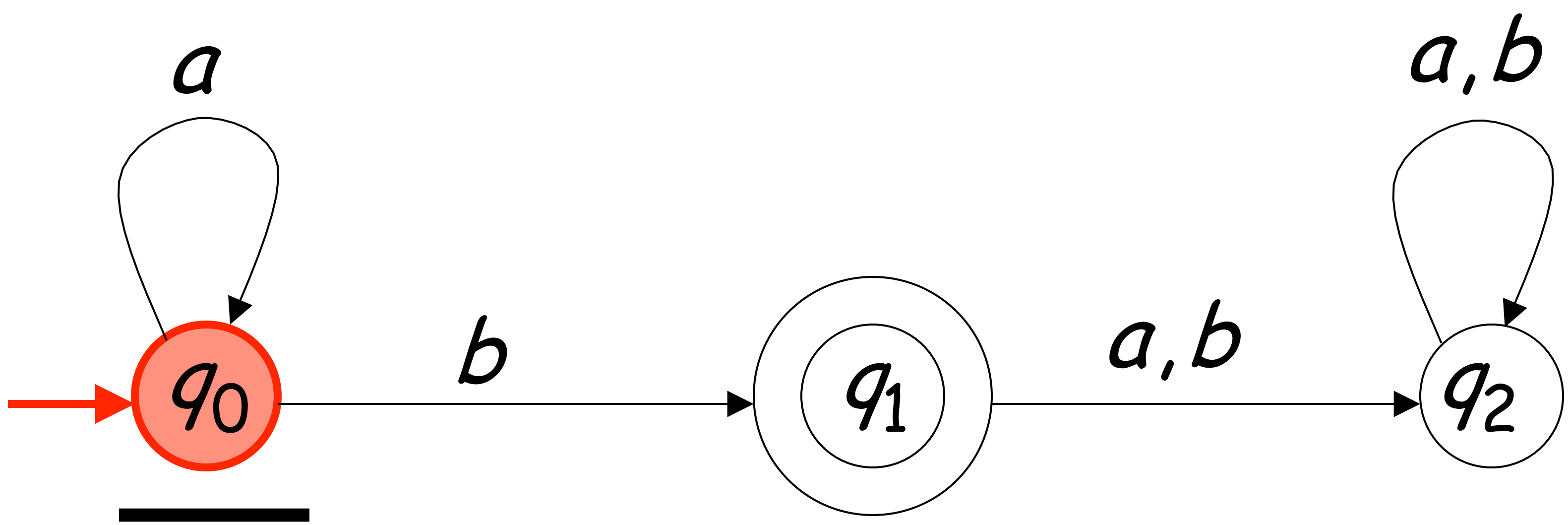
Input finished

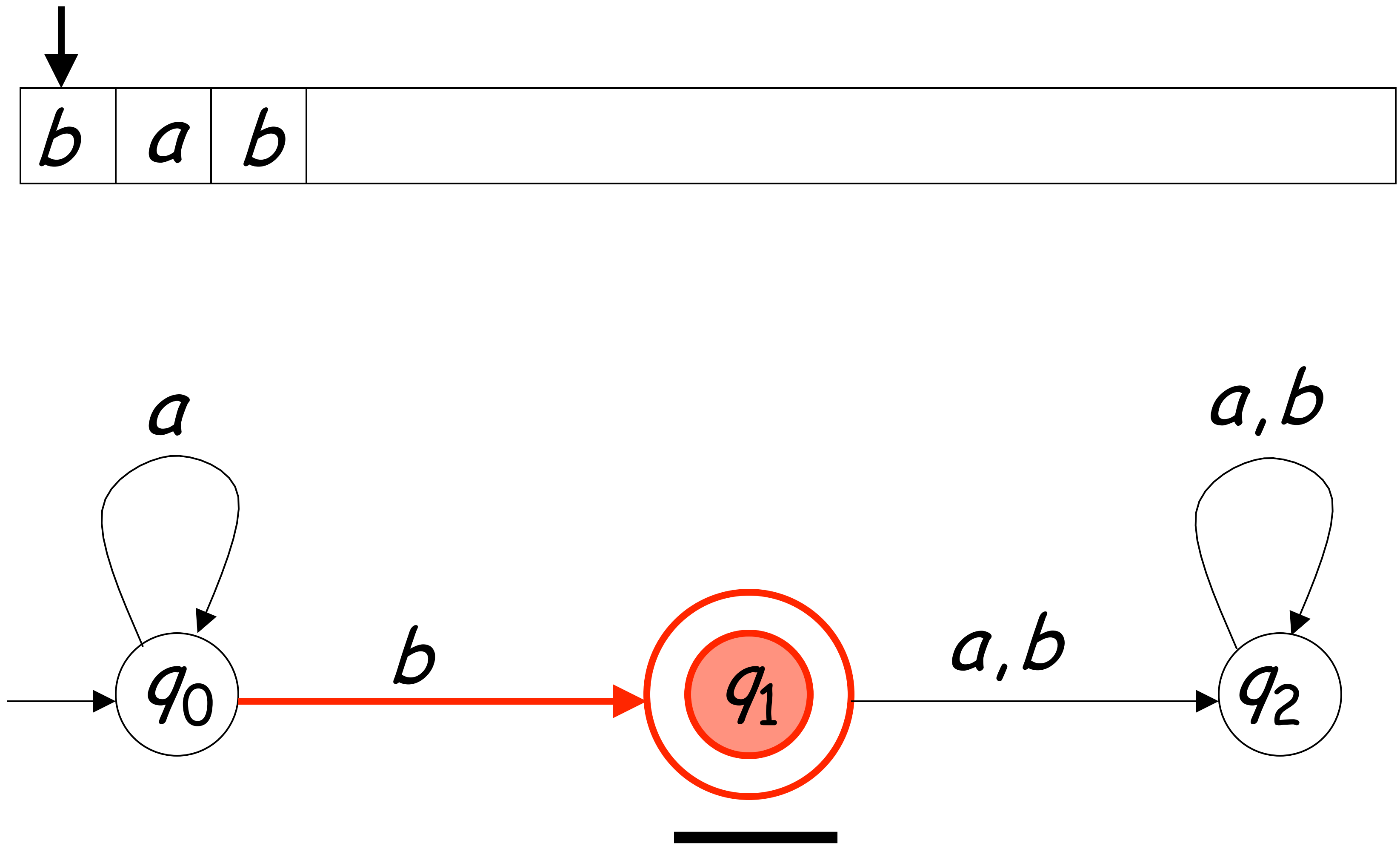


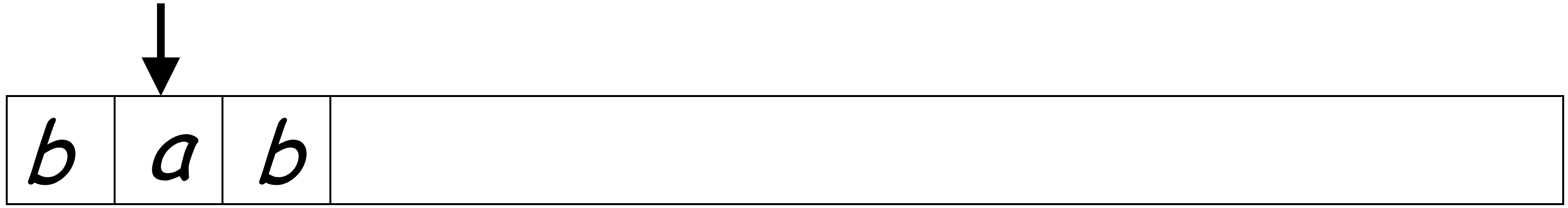
A rejection case

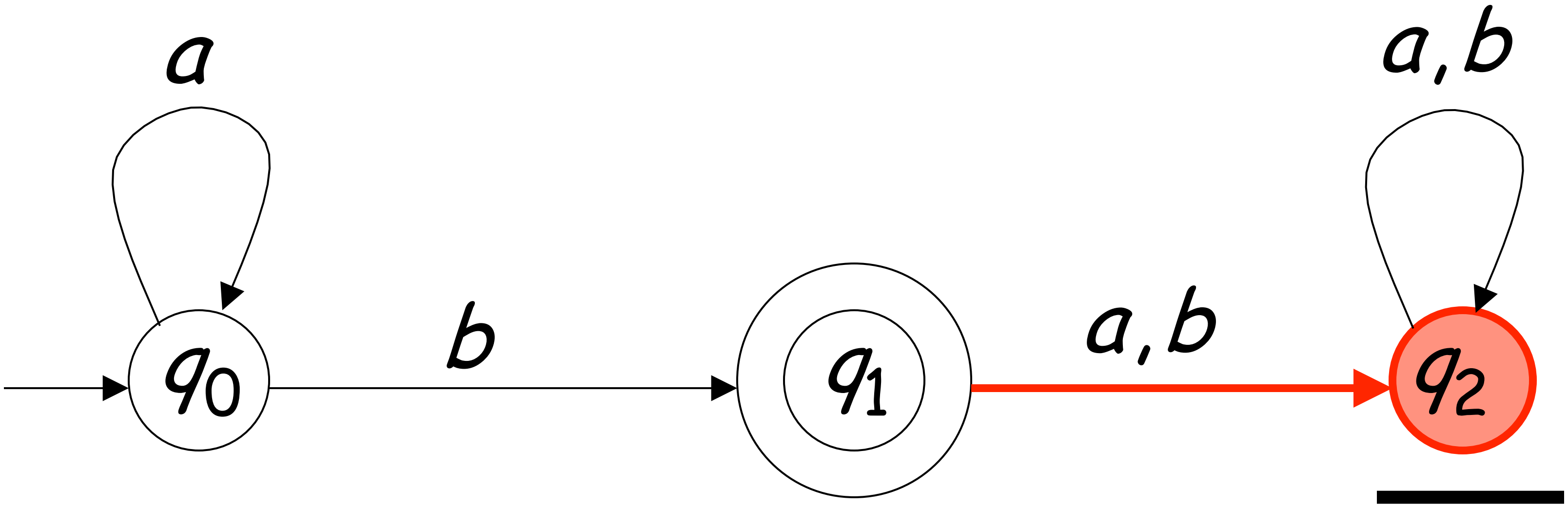


Input String

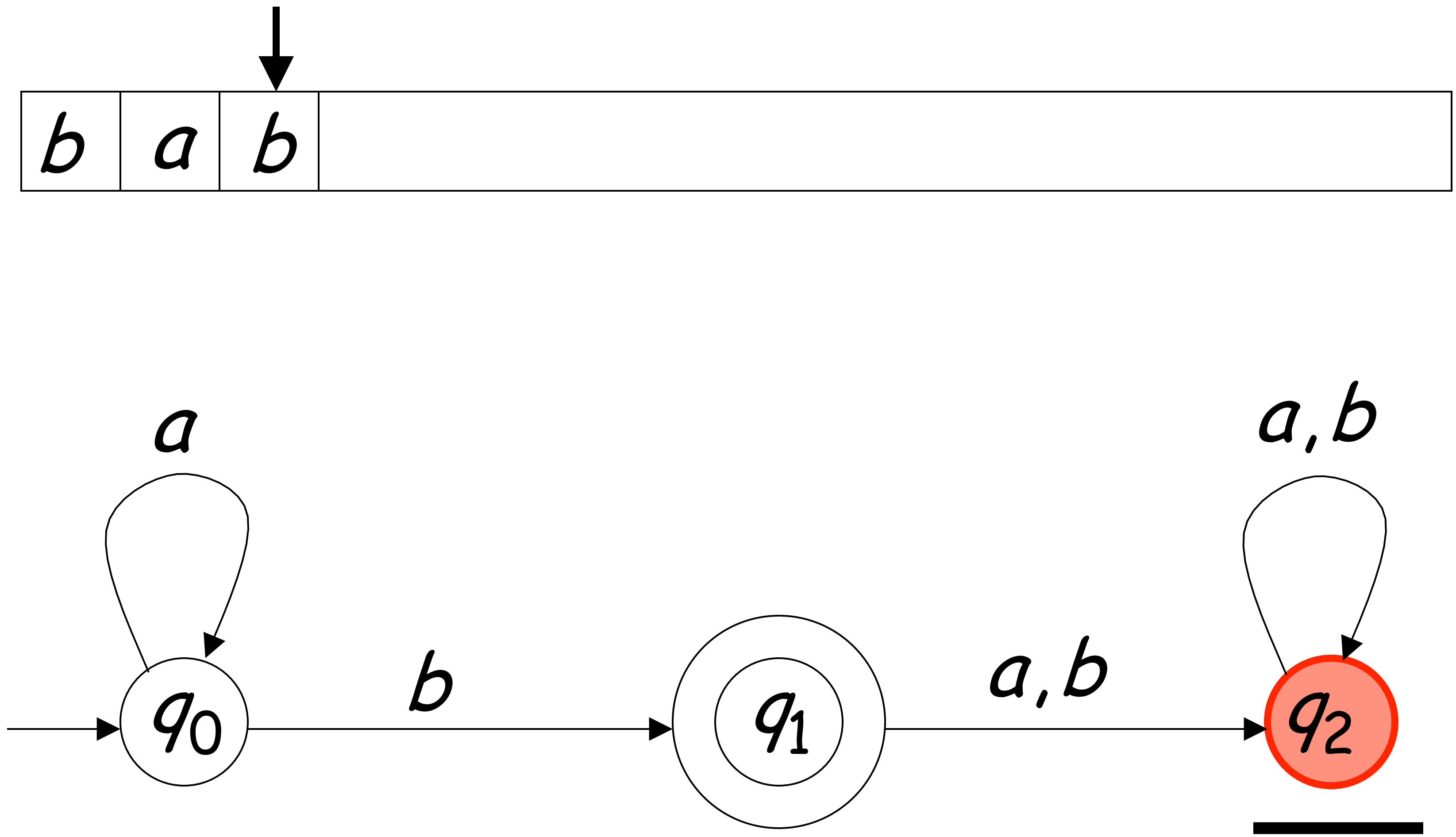



b
a
b
a
a,b
b
q_0
q_1
a,b
q_2


b
a
b

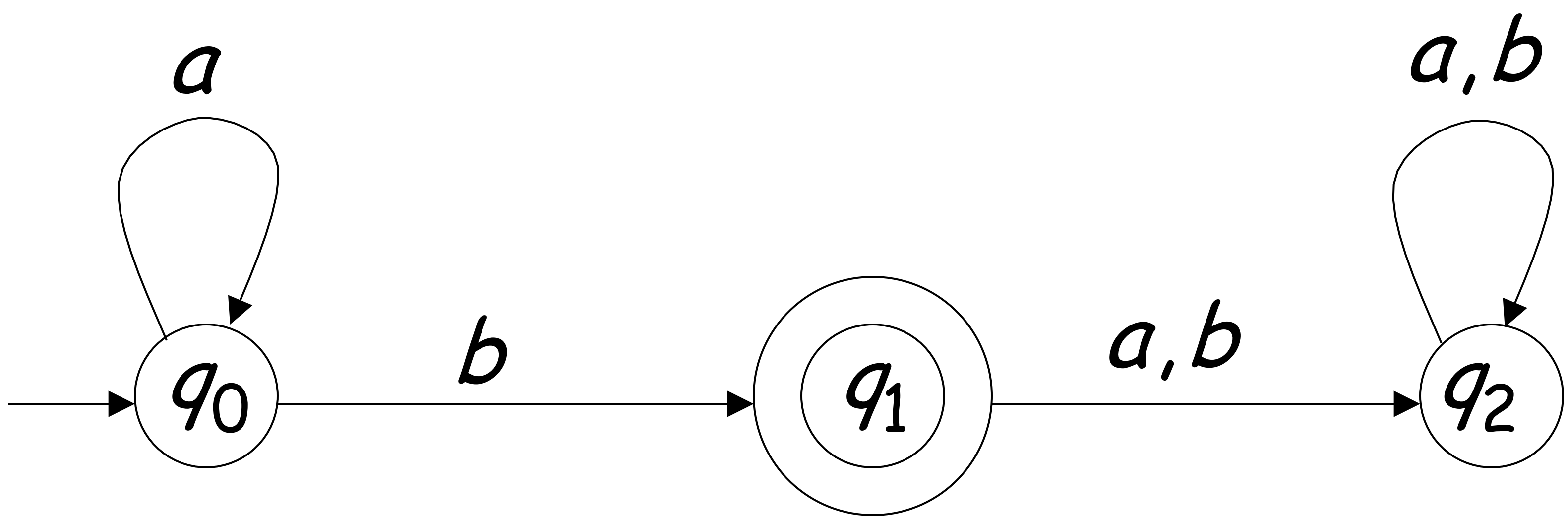

a
a, b
b
a, b
q_0
q_1
q_2

Input finished



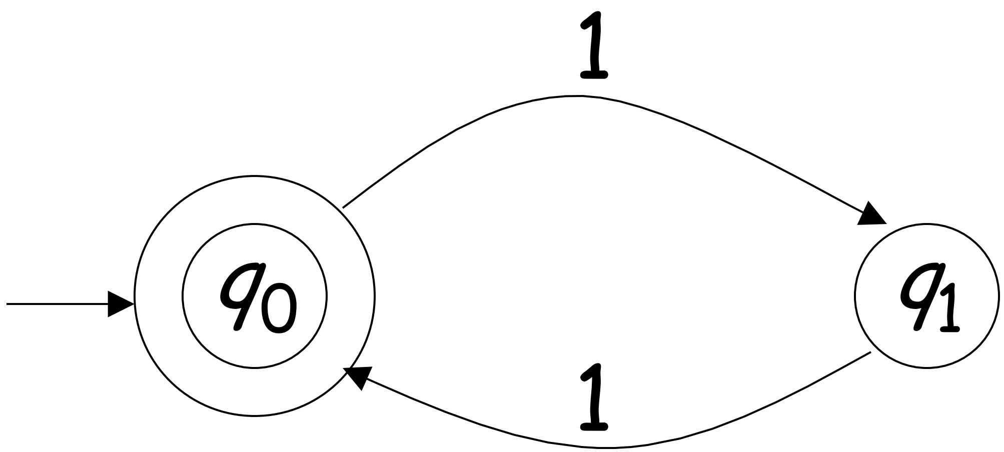
reject

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



Another Example

Alphabet: $\Sigma = \{1\}$



Language Accepted:

$$EVEN = \{x : x \in \Sigma^* \text{ and } x \text{ is even}\}$$
$$= \{\lambda, 11, 1111, 111111, \ldots\}$$

Formal Definition

Deterministic Finite Automaton (DFA)

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

Q : set of states

Σ : input alphabet $\lambda \notin \Sigma$

δ : transition function

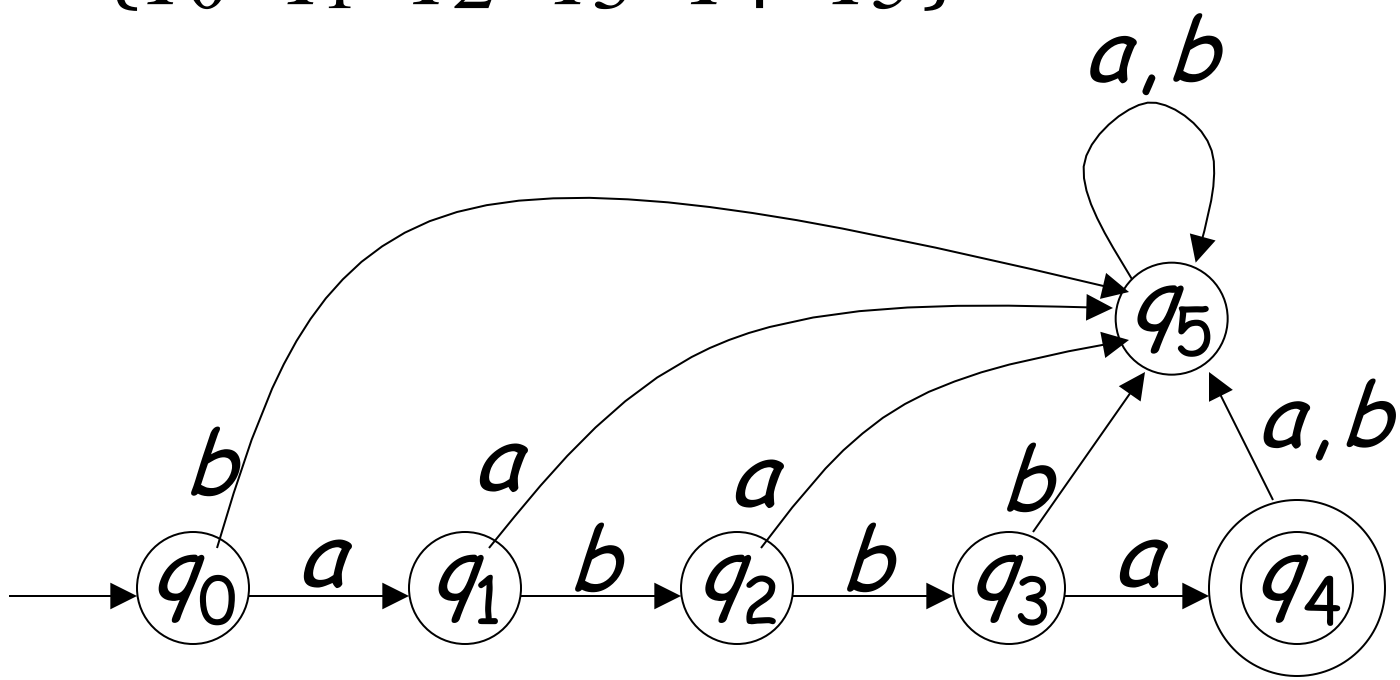
q_0 : initial state

F : set of accepting states

Set of States Q

Example

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

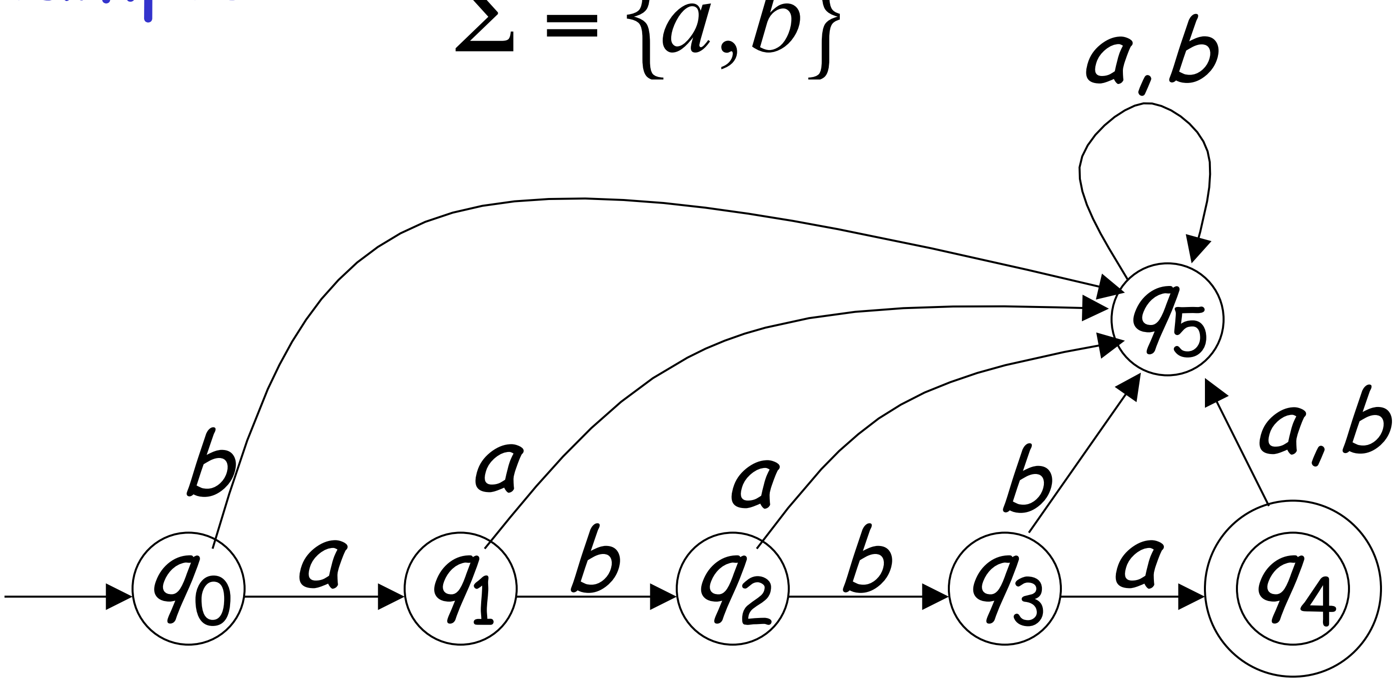


Input Alphabet Σ

$\lambda \notin \Sigma$:the input alphabet never contains λ

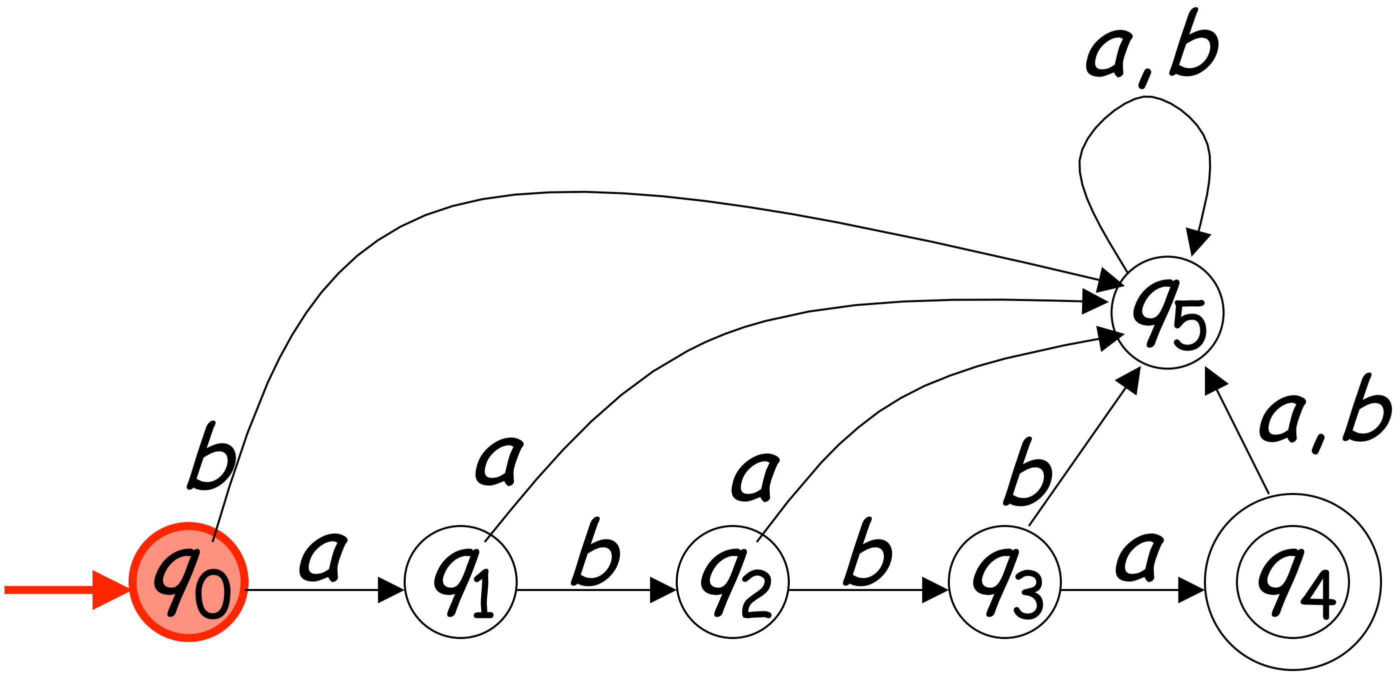
Example

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



Initial State q_0

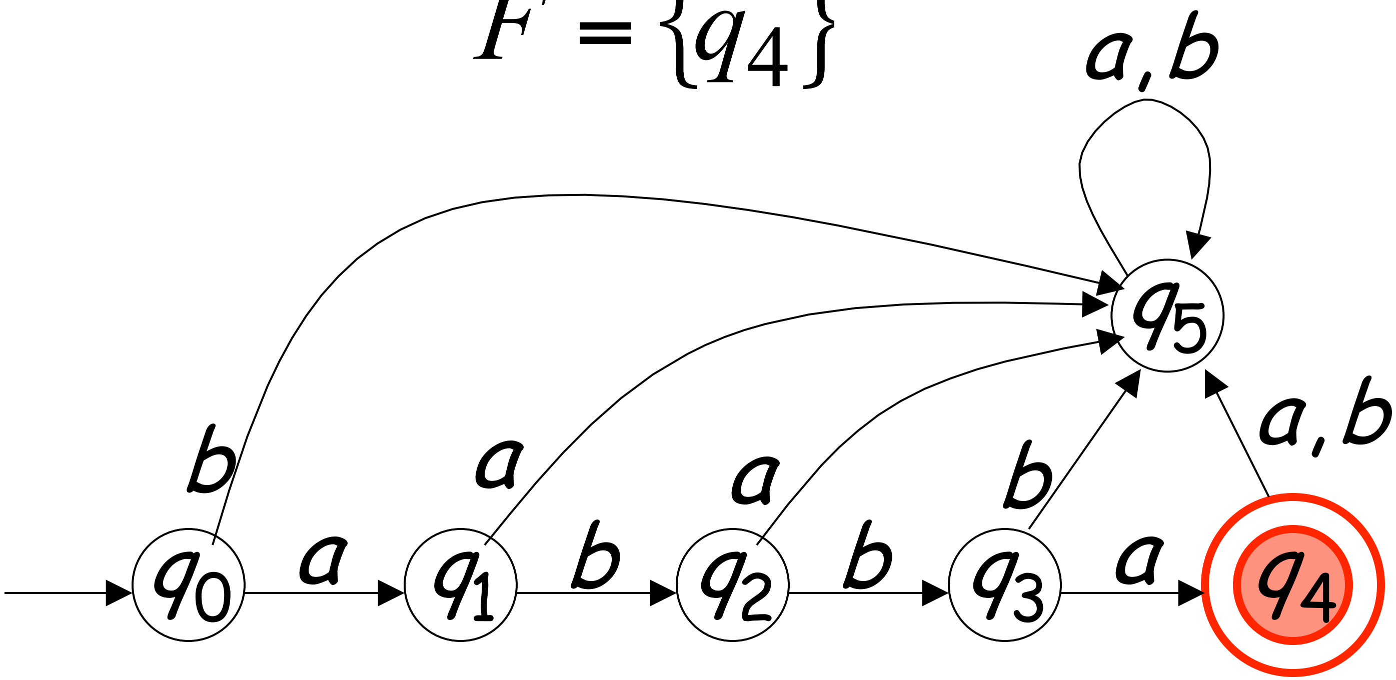
Example



Set of Accepting States $F \subseteq Q$

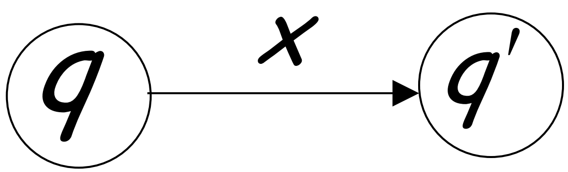
Example

$$F = \{q_4\}$$



Transition Function $\delta : Q \times \Sigma \to Q$

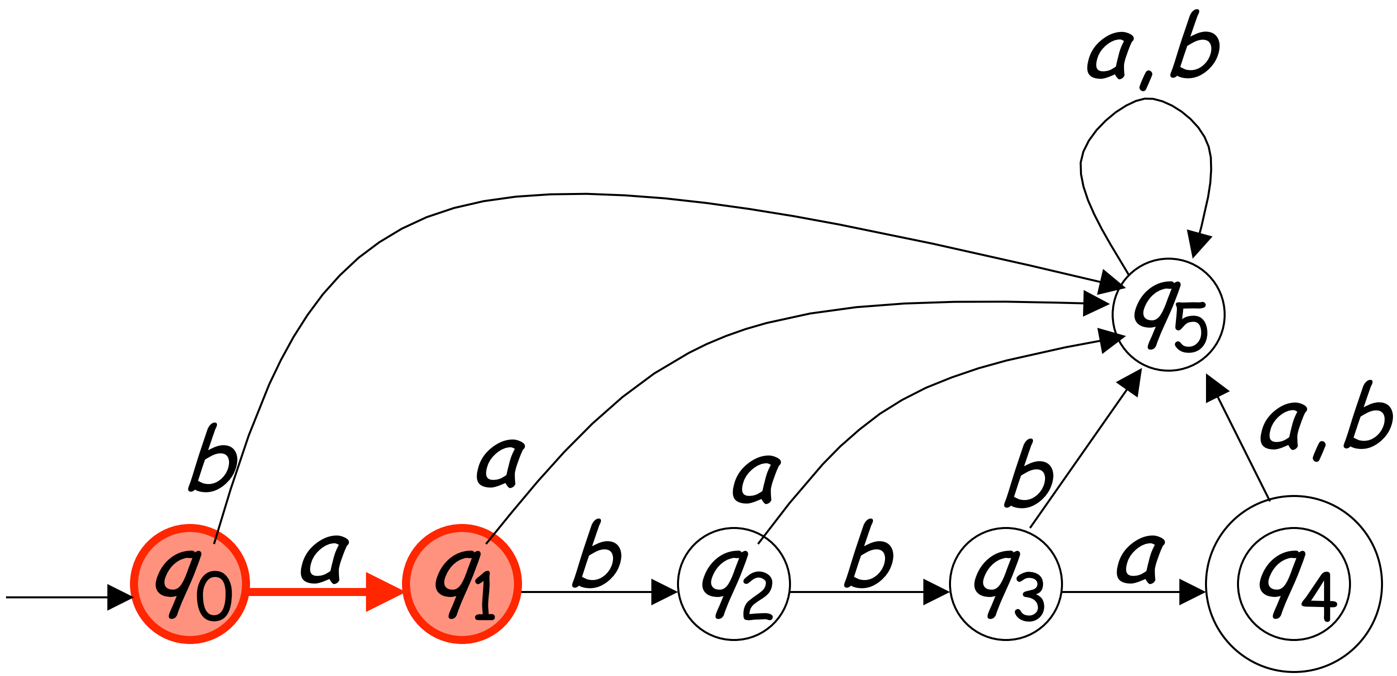
$$\delta(q, x) = q'$$



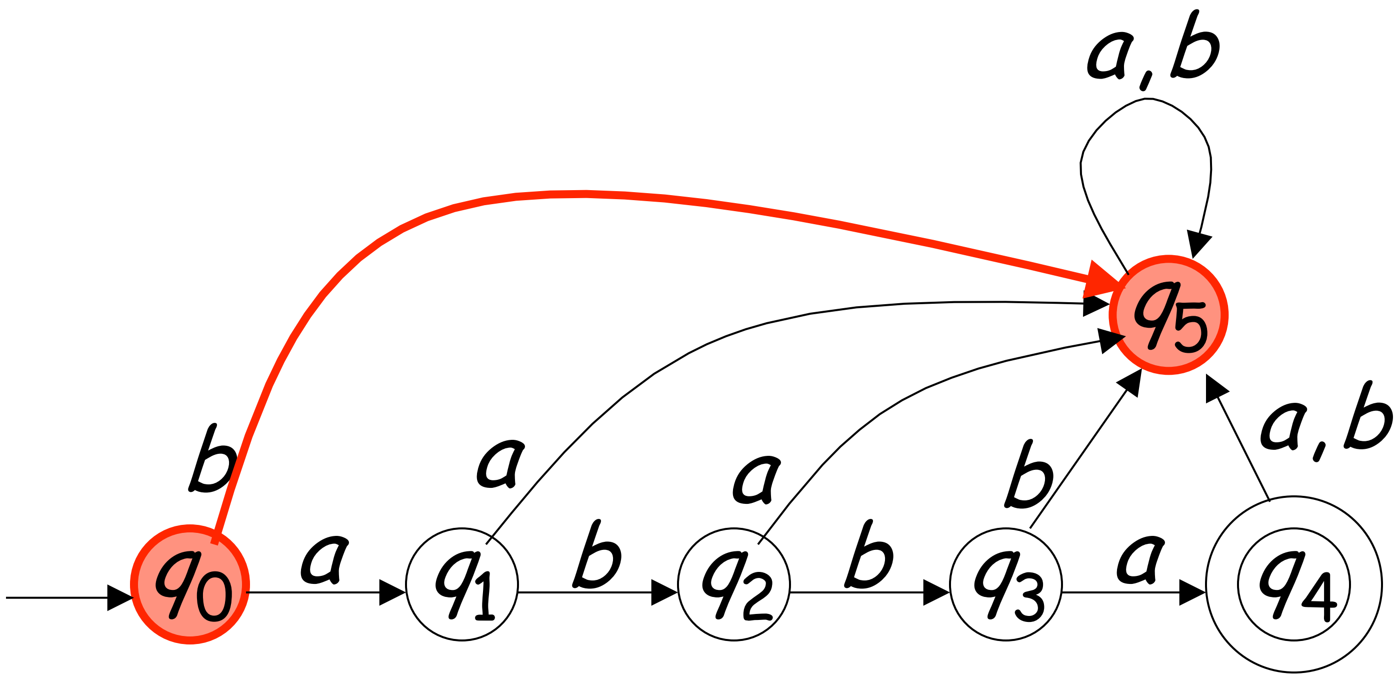
Describes the result of a transition from state q with symbol x

Example:

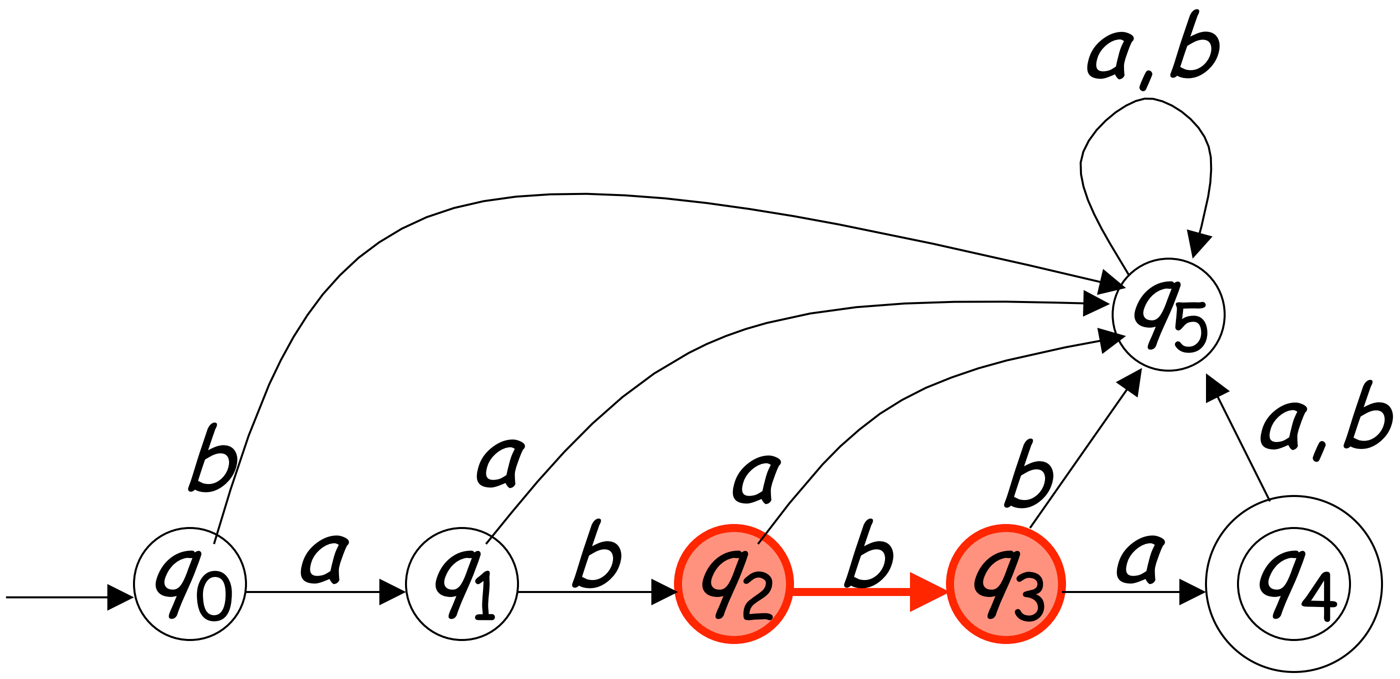
$$\delta(q_0, a) = q_1$$



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



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

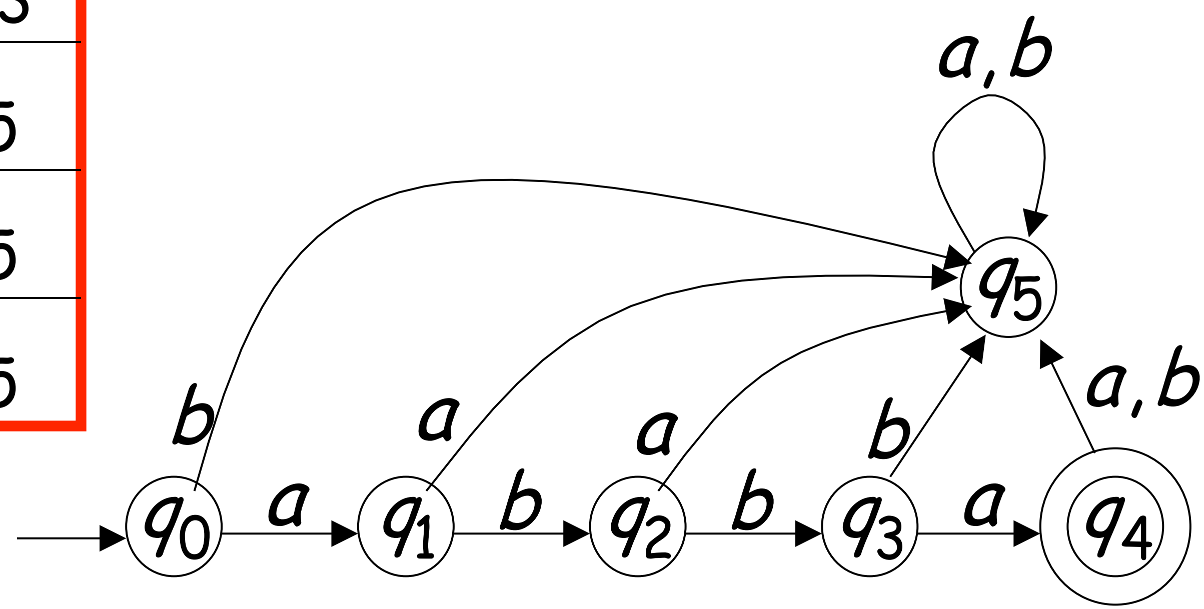


Transition Table for δ

symbols

states

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



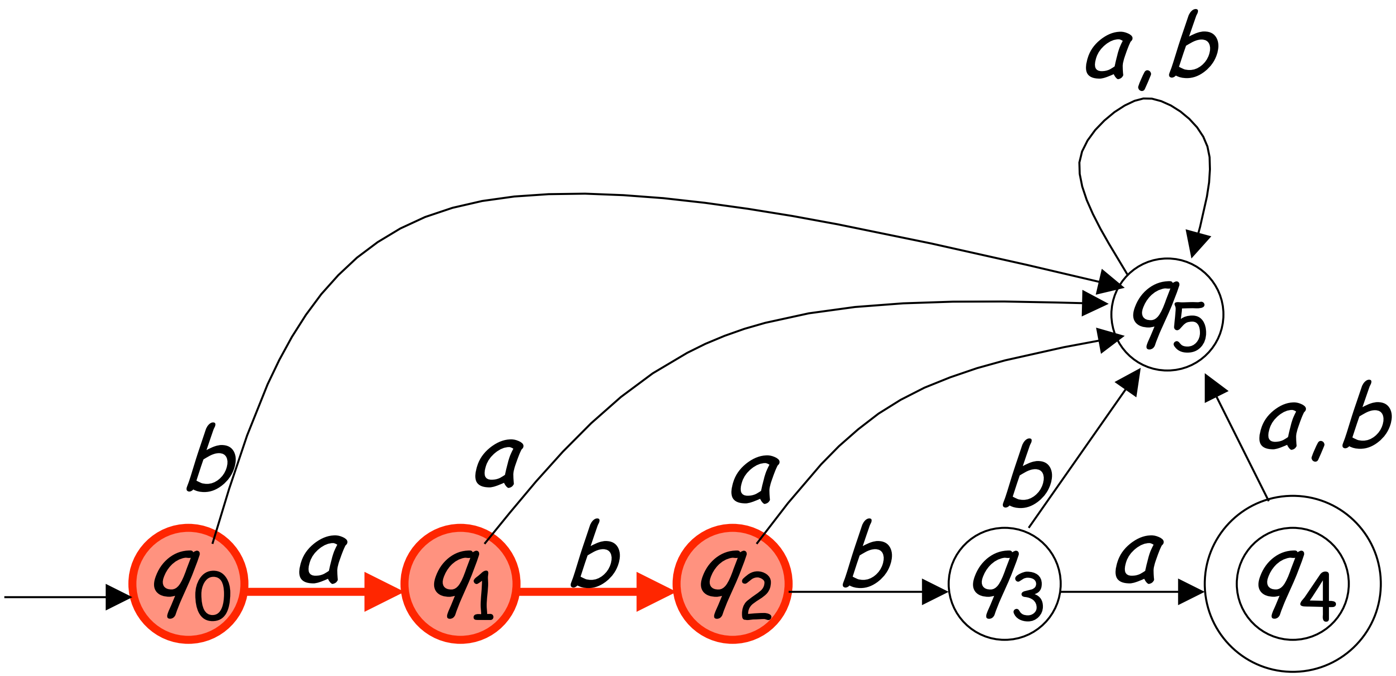
Extended Transition Function

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

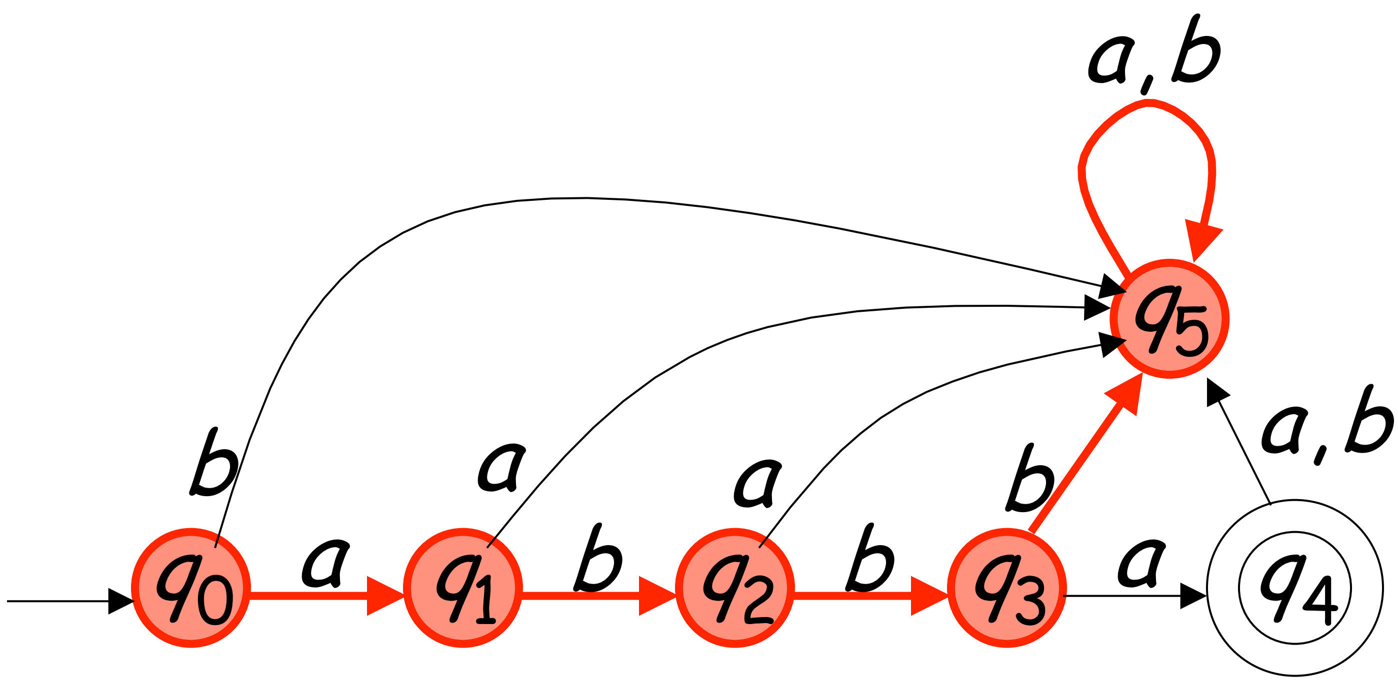
$$\delta^*(q, w) = q'$$

Describes the resulting state
after scanning string w from state q

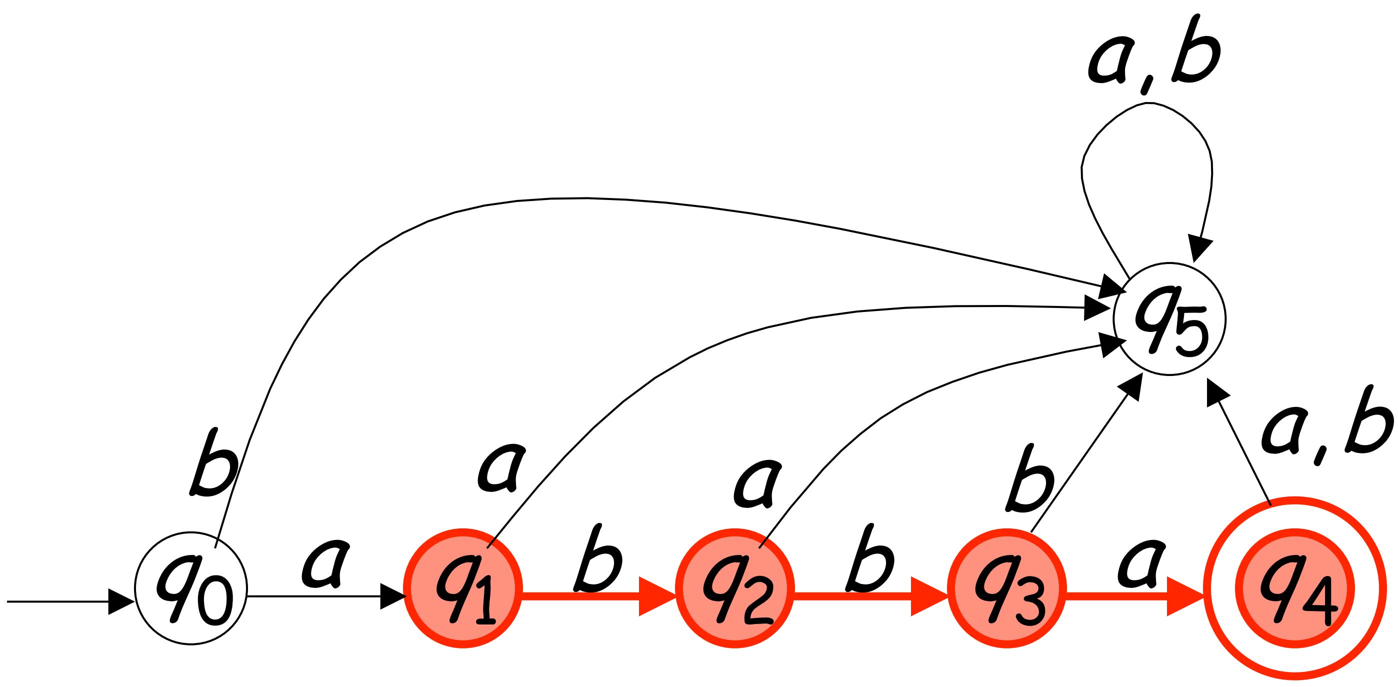
Example: $\delta^*(q_0, ab) = q_2$



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



$$\delta^*(q_1, bba) = q_4$$



Special case:

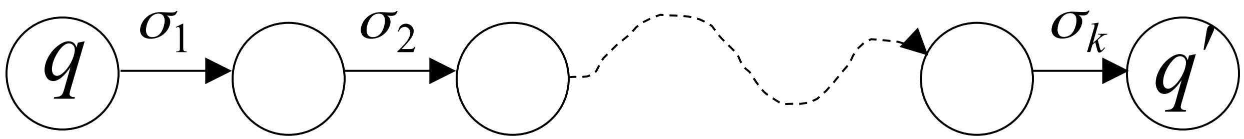
for any state q

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

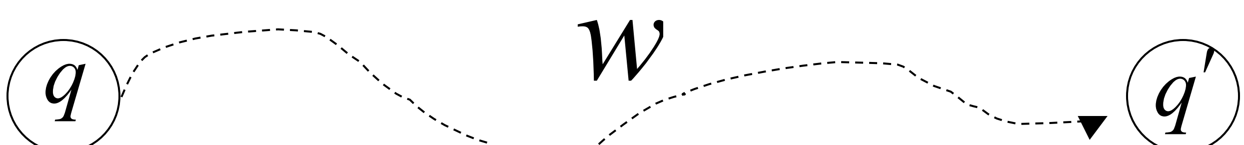
In general: $\delta^*(q,w) = q'$

implies that there is a walk of transitions

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



states may be repeated



Language Accepted by DFA

Language accepted by DFA M:

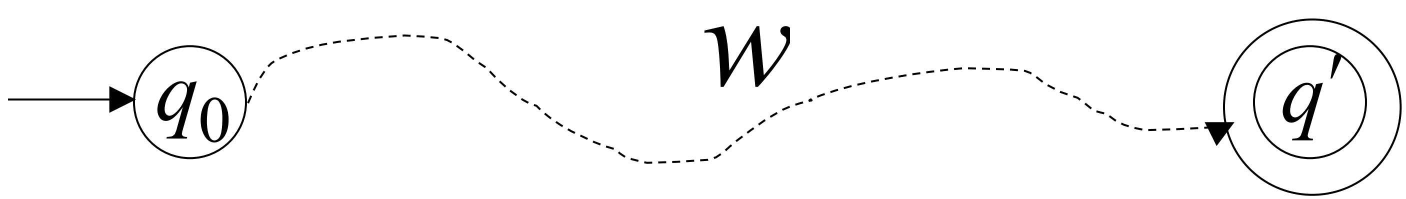
it is denoted as $L(M)$ and contains all the strings accepted by M

We also say that M recognizes $L(M)$

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

Language accepted by M:

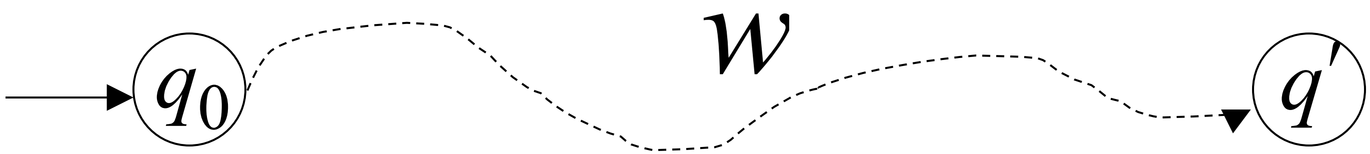
$$L(M) = \{ w \in \Sigma^* : \delta^*(q_0, w) \in F \}$$



$q' \in F$

Language rejected by M:

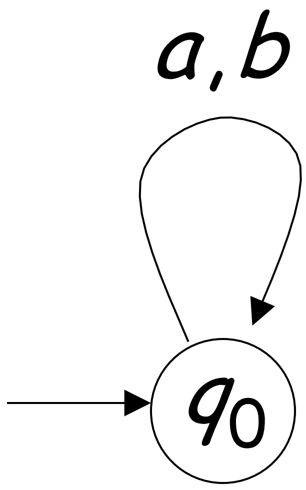
$$\overline{L(M)} = \{ w \in \Sigma^* : \delta^*(q_0, w) \notin F \}$$

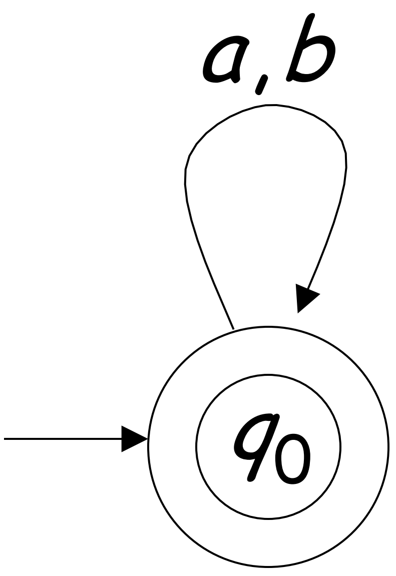


$q' \notin F$

More DFA Examples

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





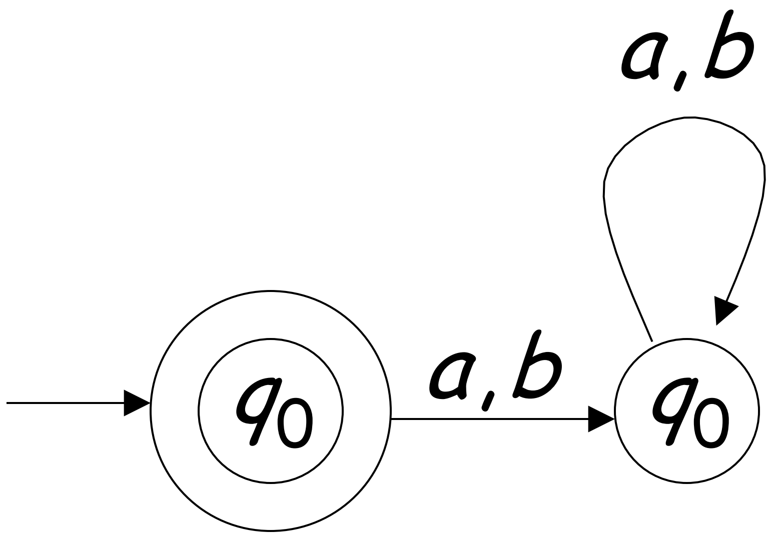
$$L(M) = \{\ \}$$

Empty language

$$L(M) = \Sigma^*$$

All strings

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

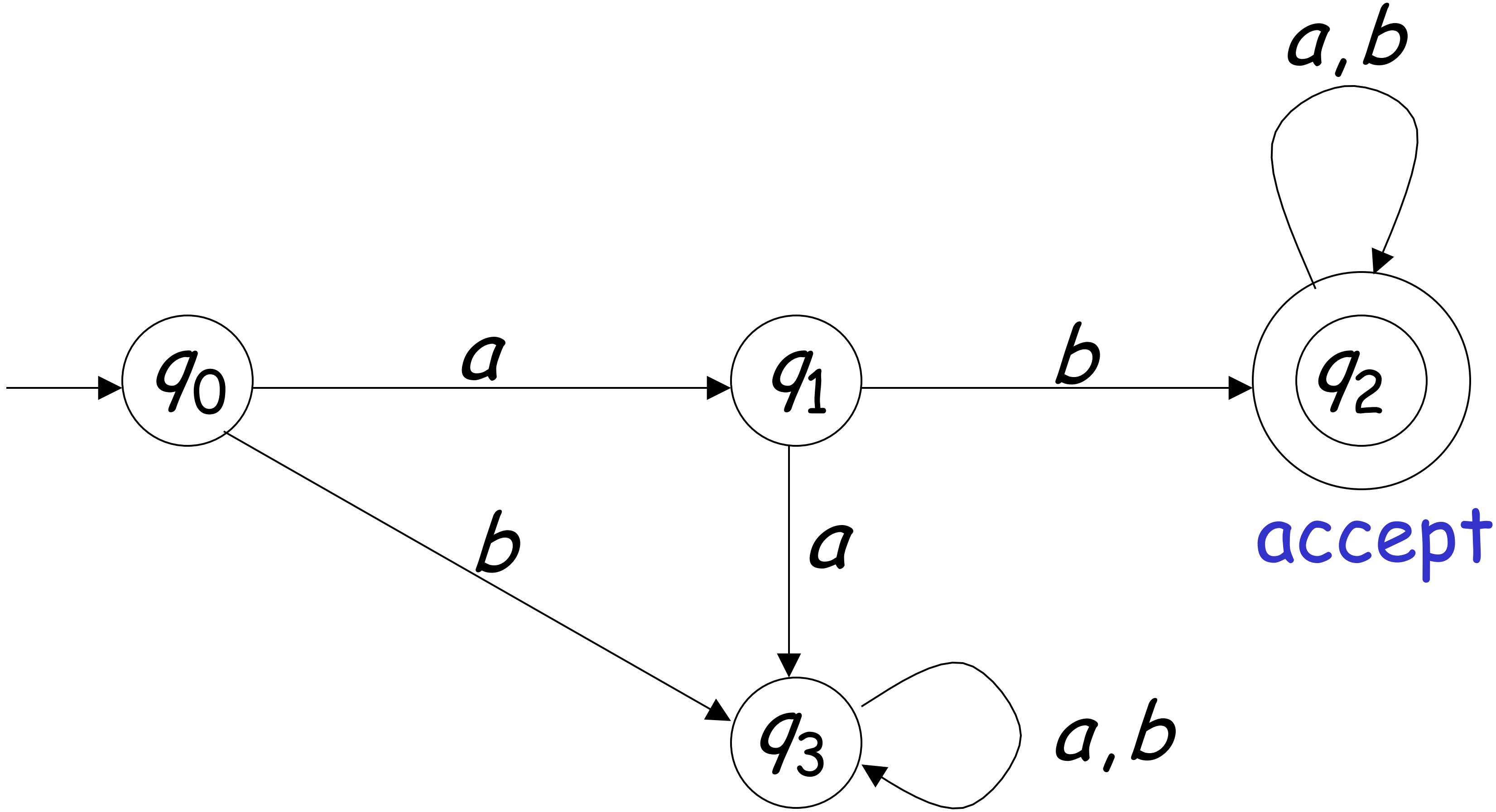


$$L(M) = \{\lambda\}$$

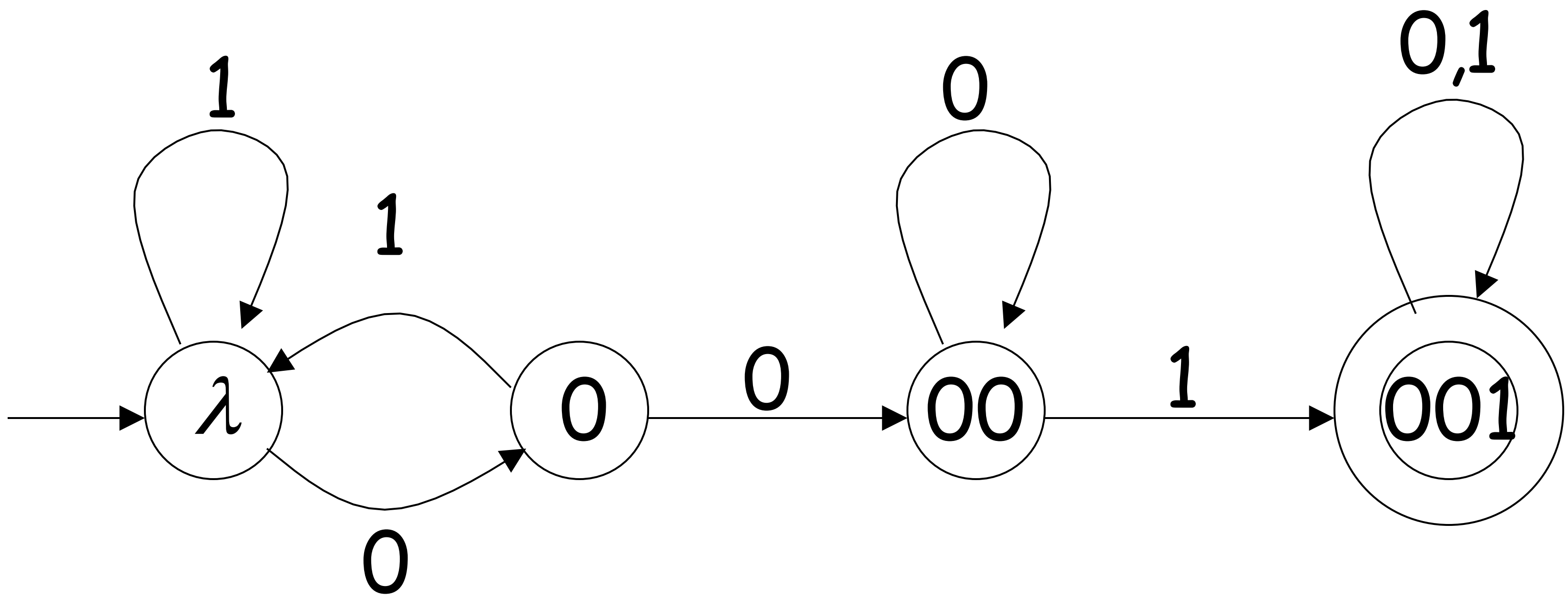
Language of the empty string

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

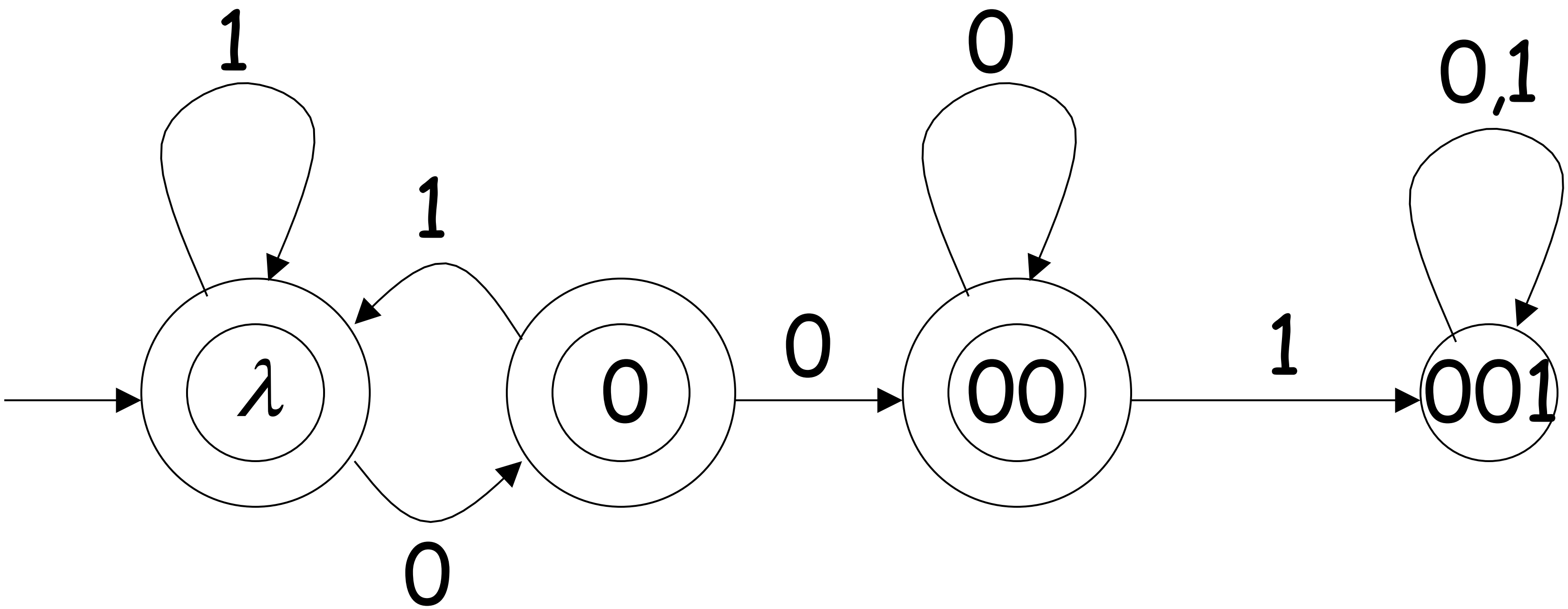
$L(M) =$ { all strings with prefix ab }



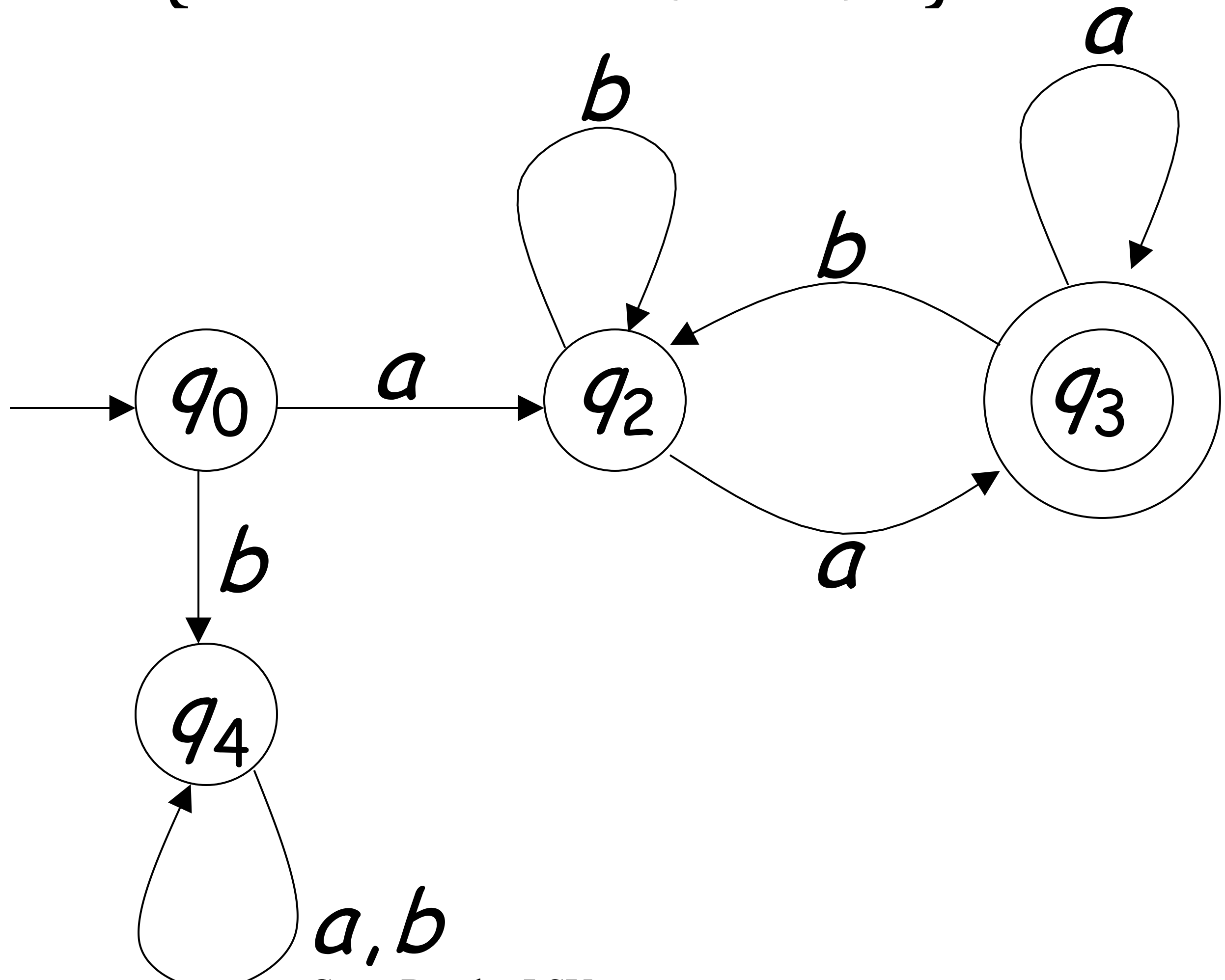
$L(M)$ = { all binary strings containing substring 001 }



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



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



Regular Languages

Definition:

A language L is regular if there is a DFA M that accepts it ($L(M) = L$)

The languages accepted by all DFAs form the family of regular languages

Example regular languages:

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

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

{ all strings in {a,b}* with prefix ab }

{ all binary strings without substring 001}

$\{x : x \in \{1\}^*$ and x is even$\}$

$\{\ \}$ $\{\lambda\}$ $\{a,b\}^*$

There exist DFAs that accept these languages (see previous slides).

There exist languages which are not Regular:

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

$$ADDITION = \{x + y = z \ : x = 1^n, y = 1^m, z = 1^k, \\ n + m = k\}$$

There are no DFAs that accept these languages

(we will prove this in a later class)

Properties of Regular Languages

For regular languages L_1 and L_2
we will prove that:

$$\left.\begin{array}{rl} \text{Union:} & L_1 \cup L_2 \\ \text{Concatenation:} & L_1 L_2 \\ \text{Star:} & L_1{}^* \\ \text{Reversal:} & L_1{}^R \\ \text{Complement:} & \overline{L_1} \\ \text{Intersection:} & L_1 \cap L_2 \end{array}\right\} \text{Are regular Languages}$$

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

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1{}^*$

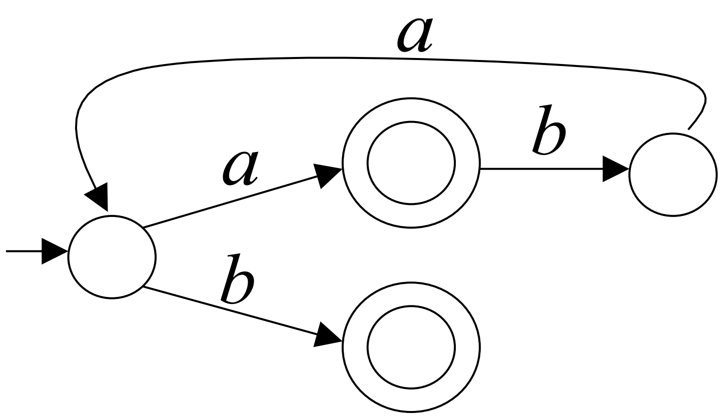
Reversal: $L_1{}^R$

Complement: $\overline{L_1}$

Intersection: $L_1 \cap L_2$

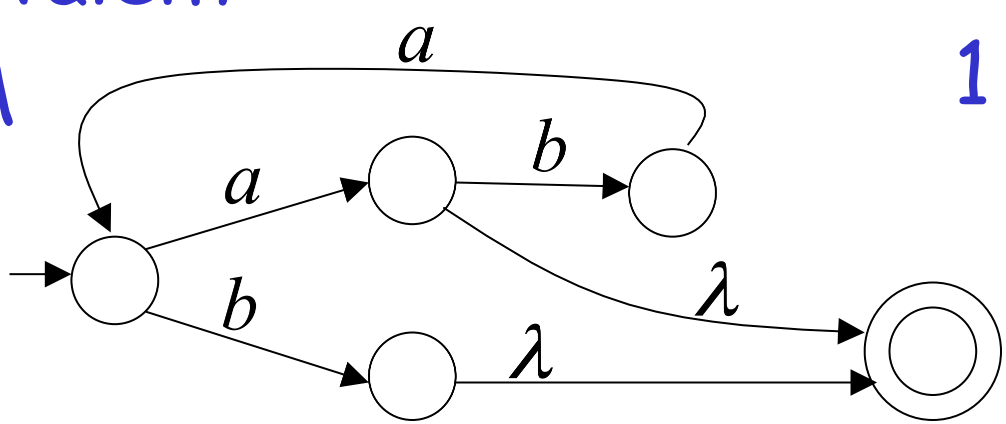
A useful transformation: use one accept state

NFA



2 accept states

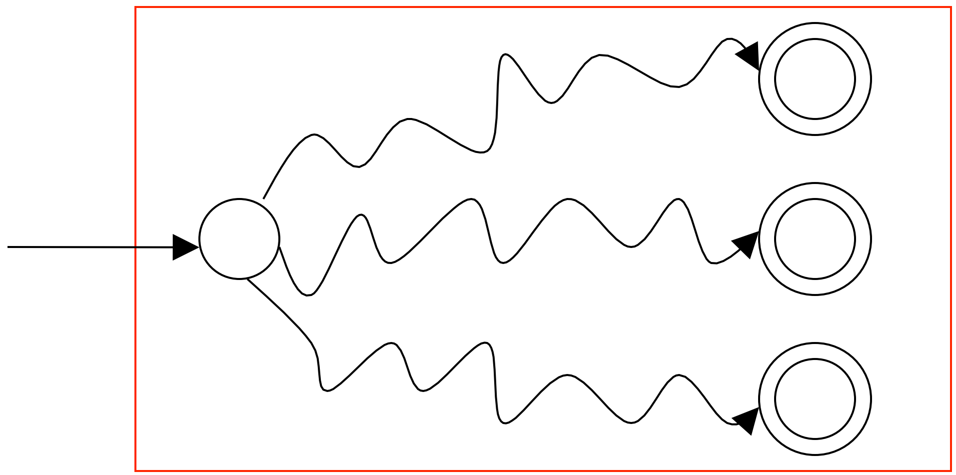
Equivalent
NFA



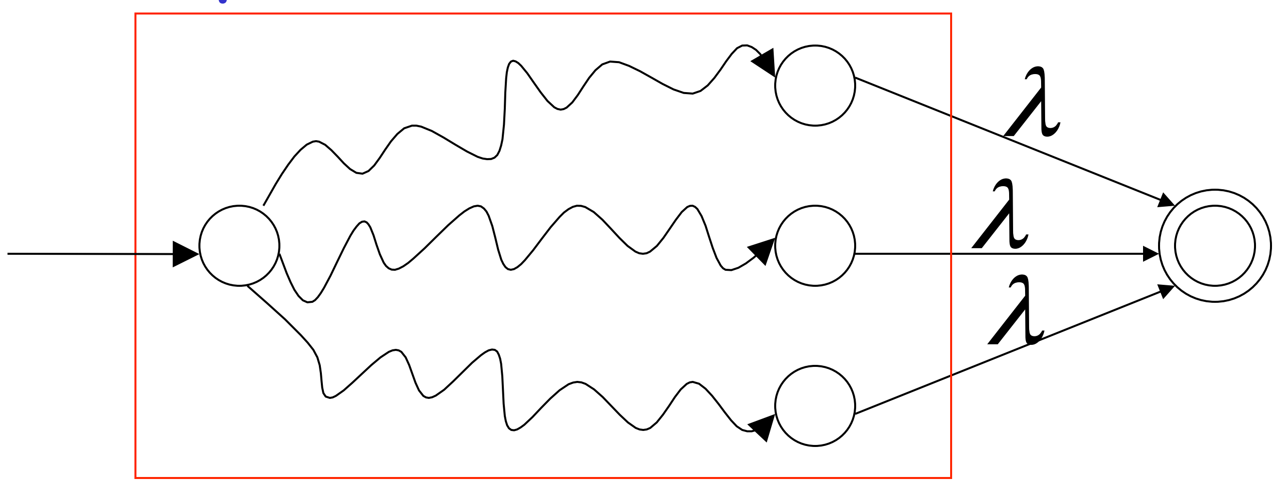
1 accept state

In General

NFA



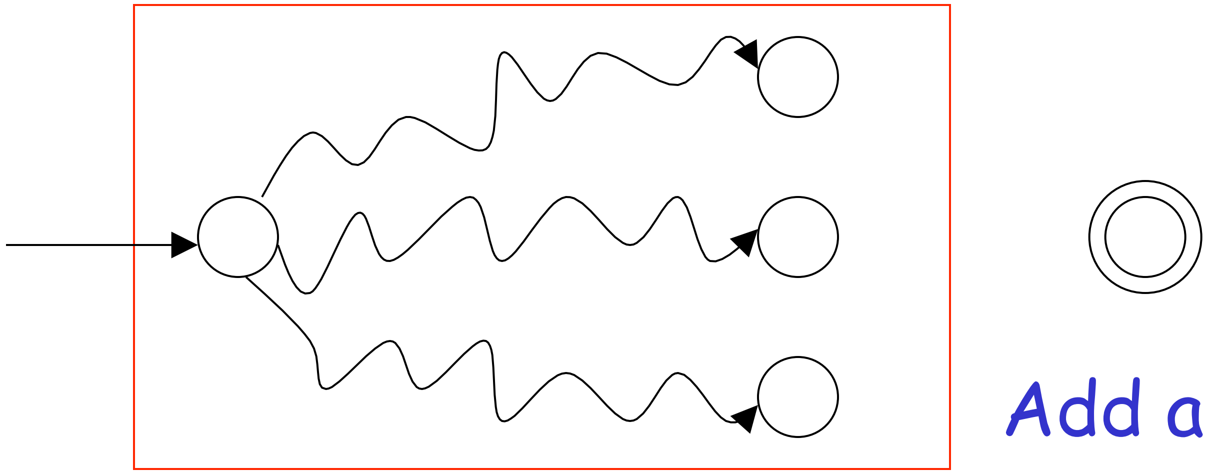
Equivalent NFA



Single accepting state

Extreme case

NFA without accepting state



Add an accepting state
without transitions

Take two languages

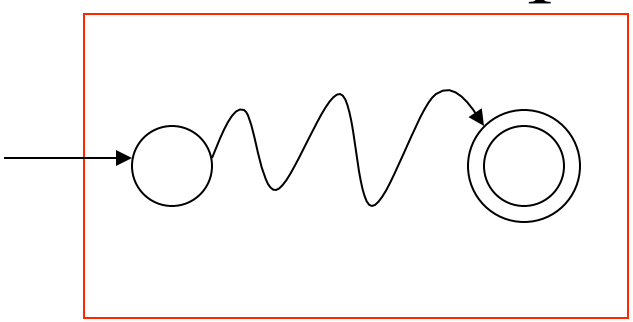
Regular language L_1

Regular language L_2

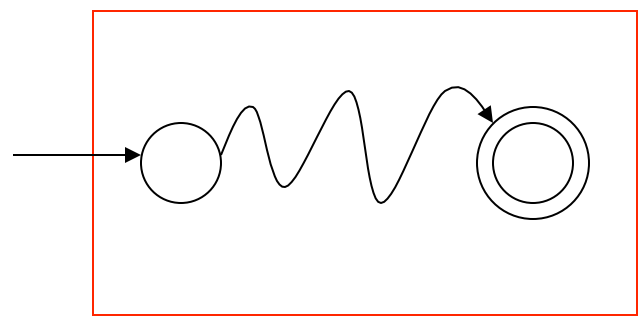
$L(M_1) = L_1$

$L(M_2) = L_2$

NFA M_1



NFA M_2



Single accepting state

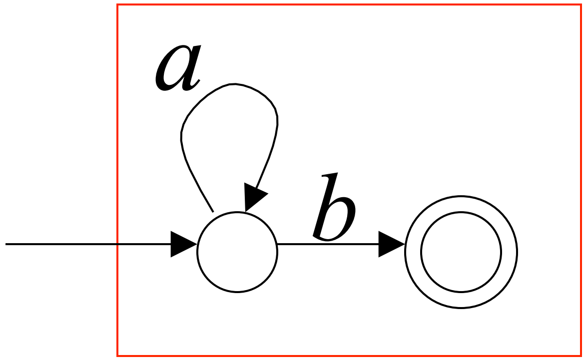
Single accepting state

Example

$n \geq 0$

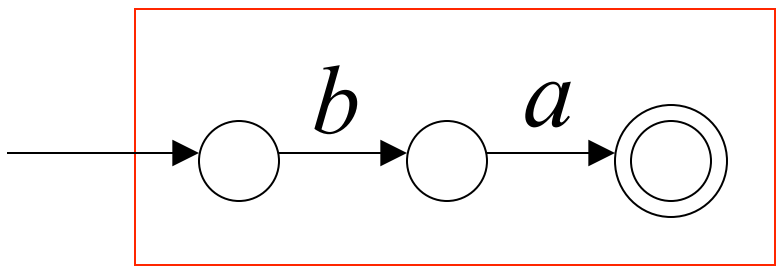
$L_1 = \{a^n b\}$

M_1



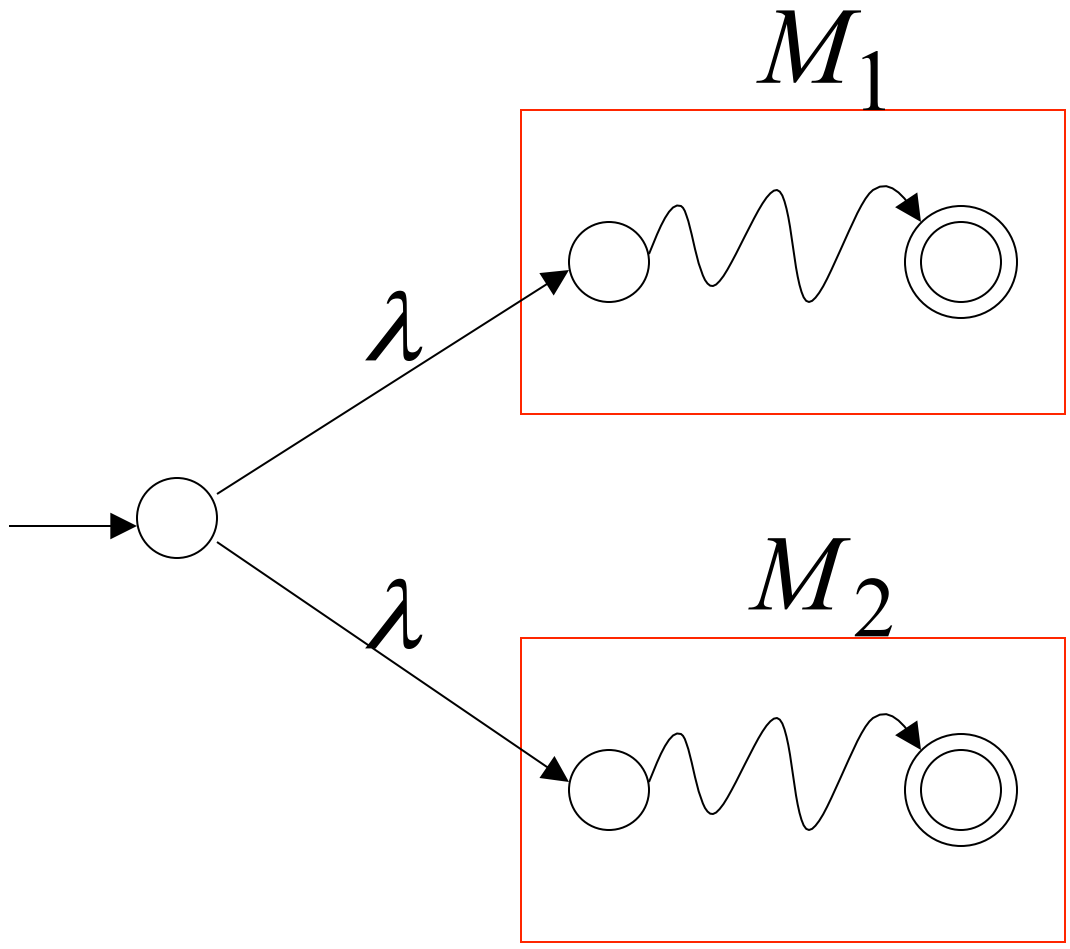
$L_2 = \{ba\}$

M_2



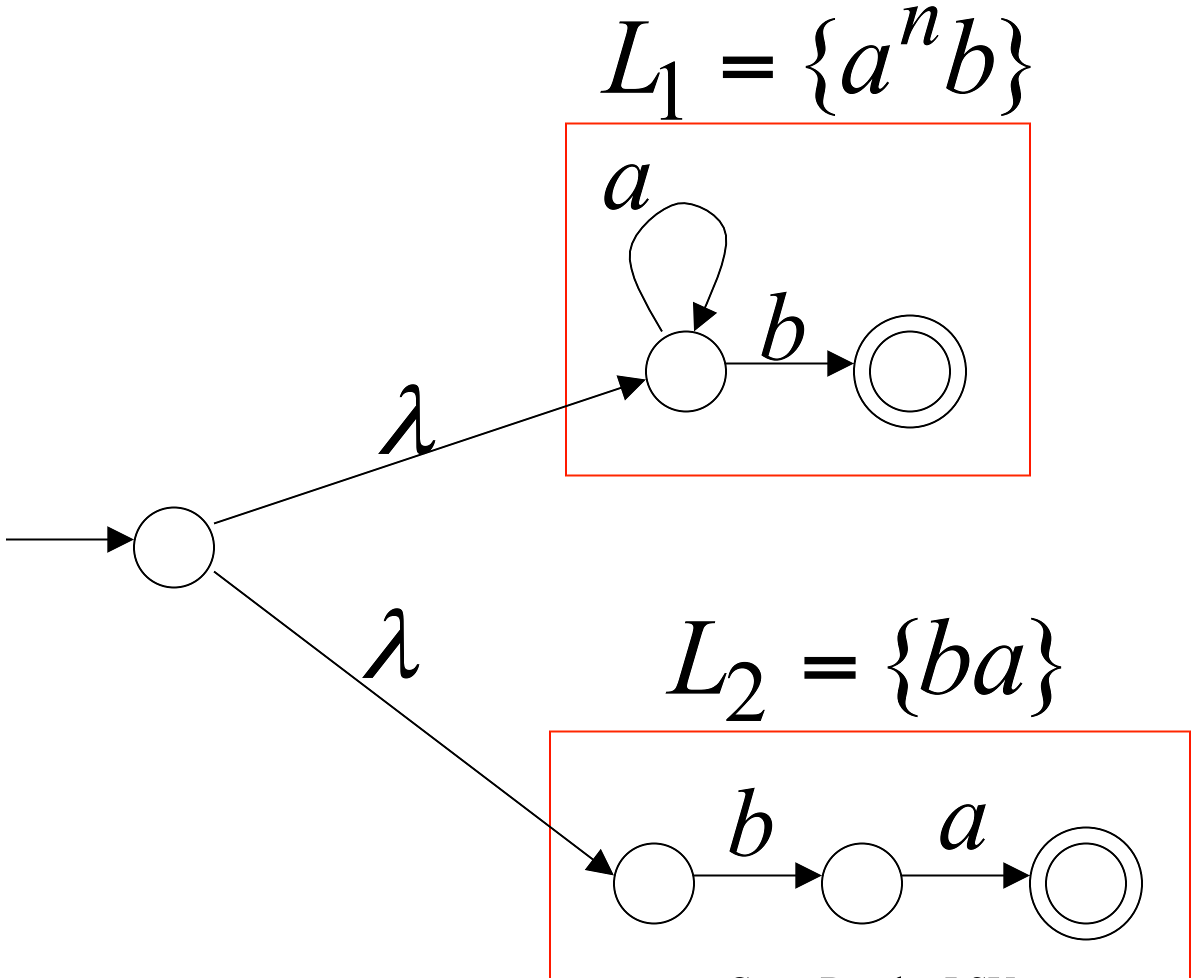
Union

NFA for $L_1 \cup L_2$



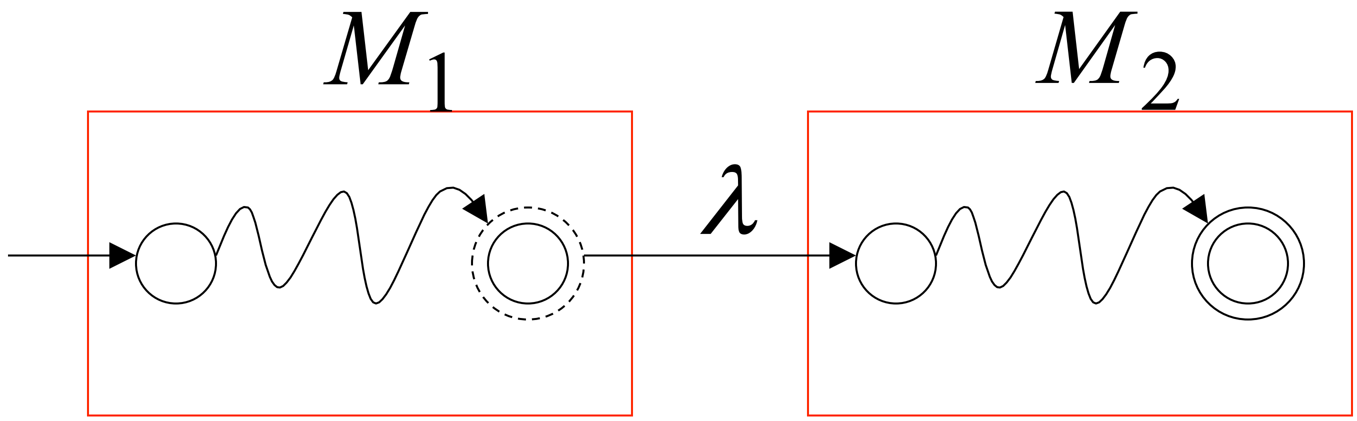
Example

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



Concatenation

NFA for $L_1 L_2$

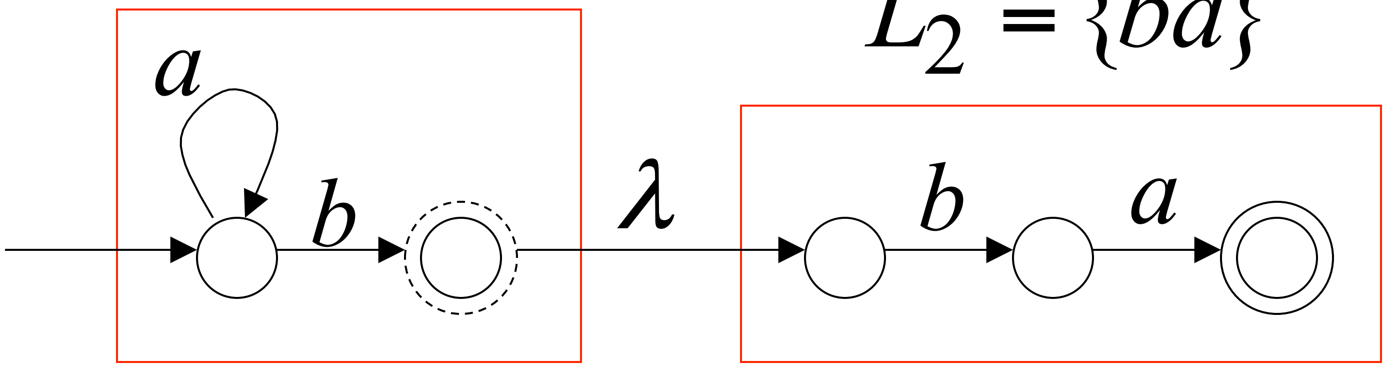


Example

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

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

$L_2 = \{ba\}$

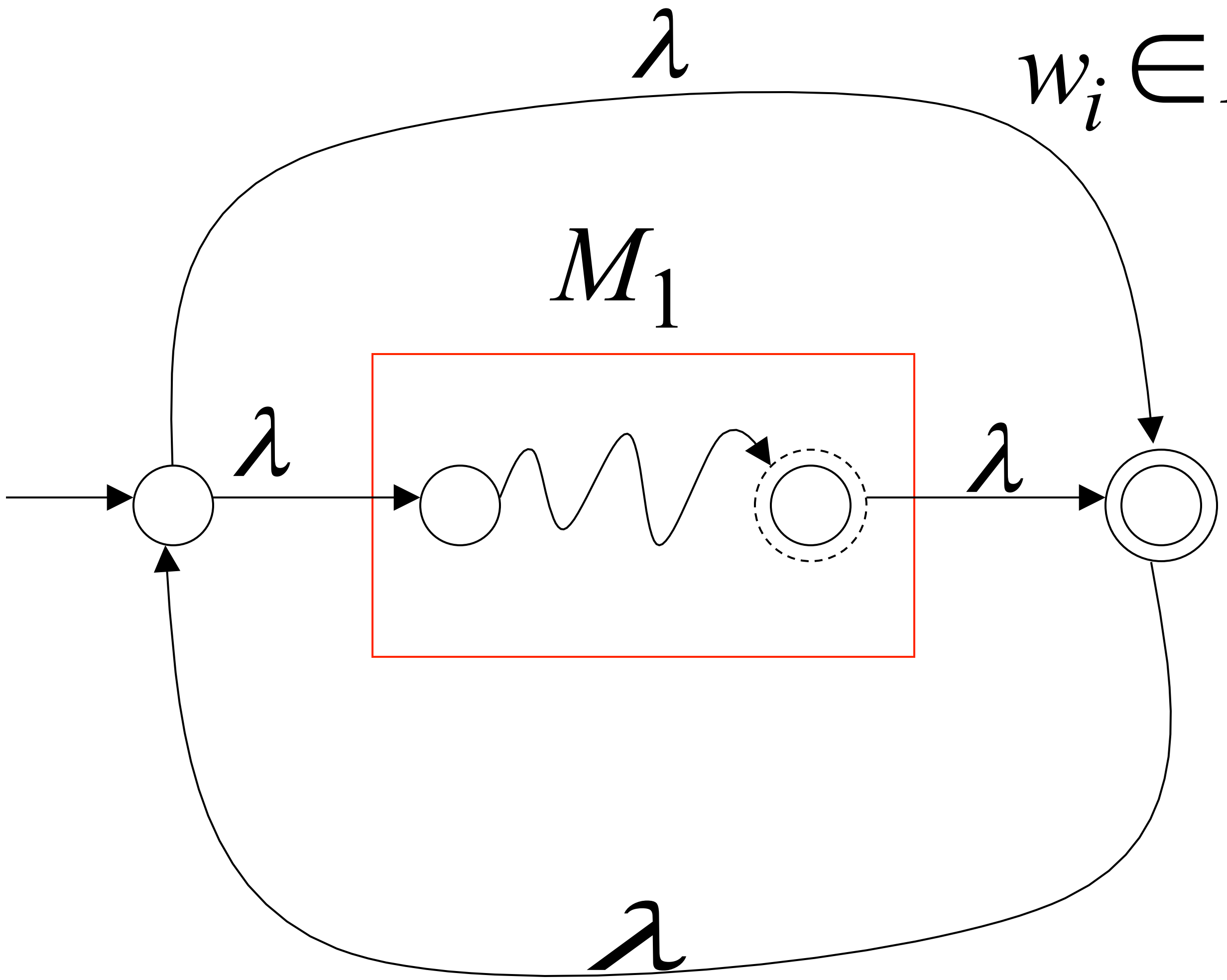


Star Operation

NFA for $L_1{}^*$

$w = w_1 w_2 \cdots w_k$

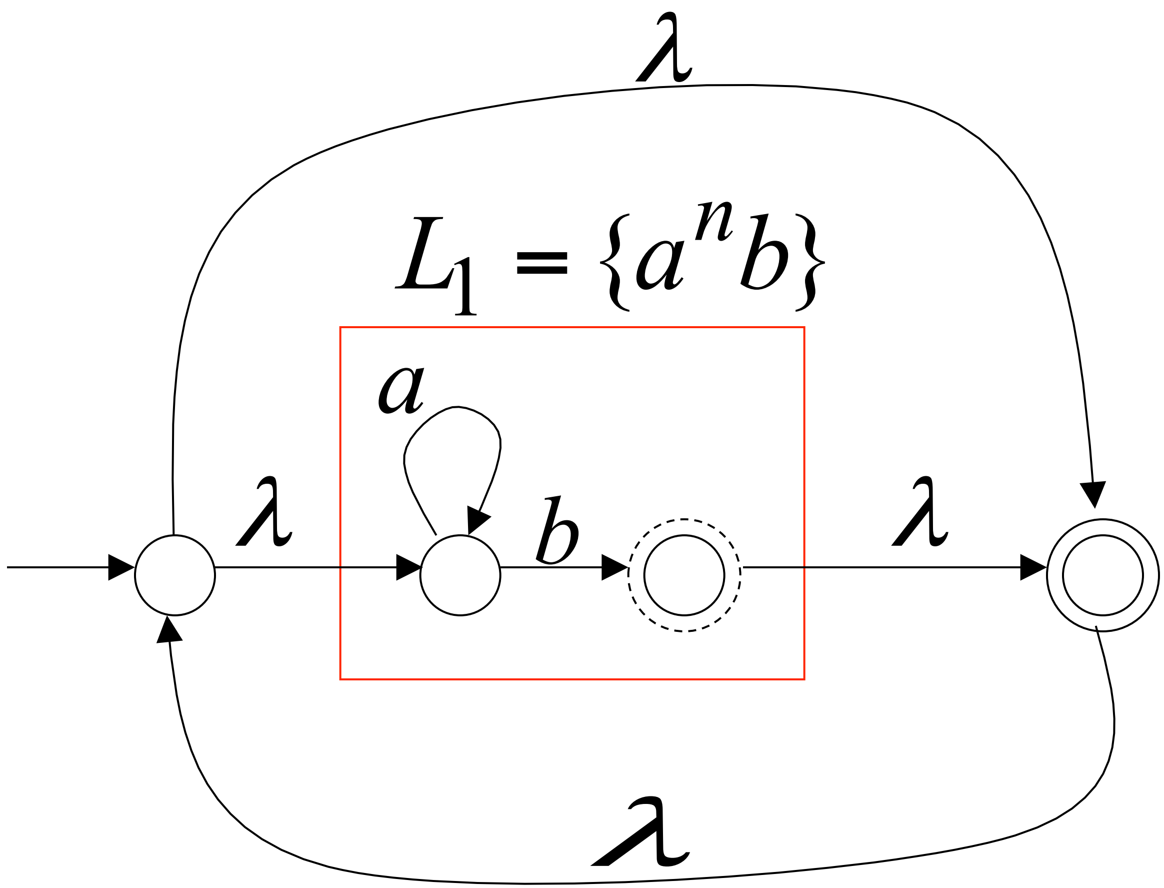
$w_i \in L_1$



$\lambda \in L_1{}^*$

Example

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



Reverse

NFA for L_1^R

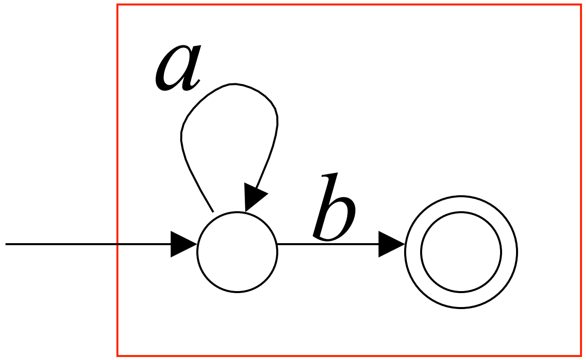
1. Reverse all transitions

2. Make initial state accepting state and vice versa

Example

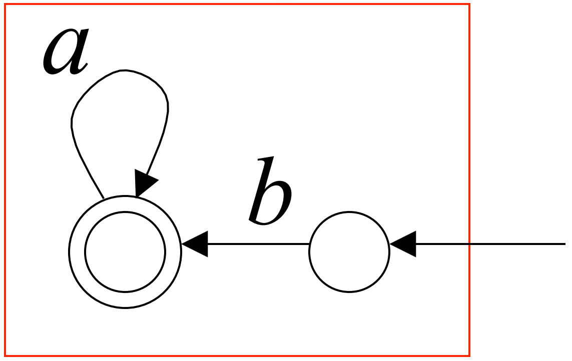
M_1

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

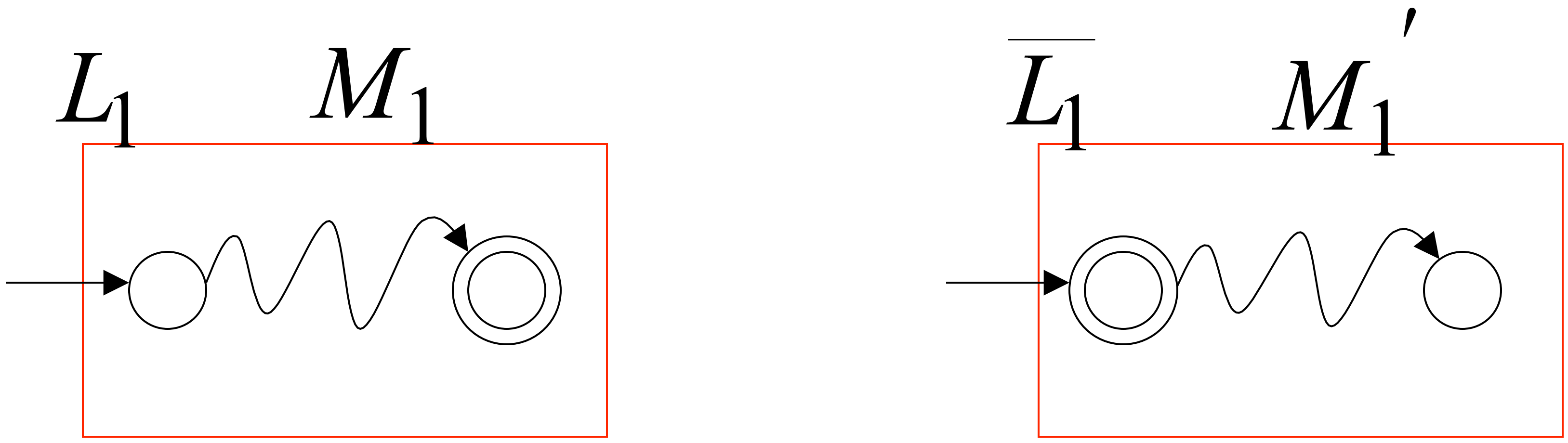


M_1'

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



Complement

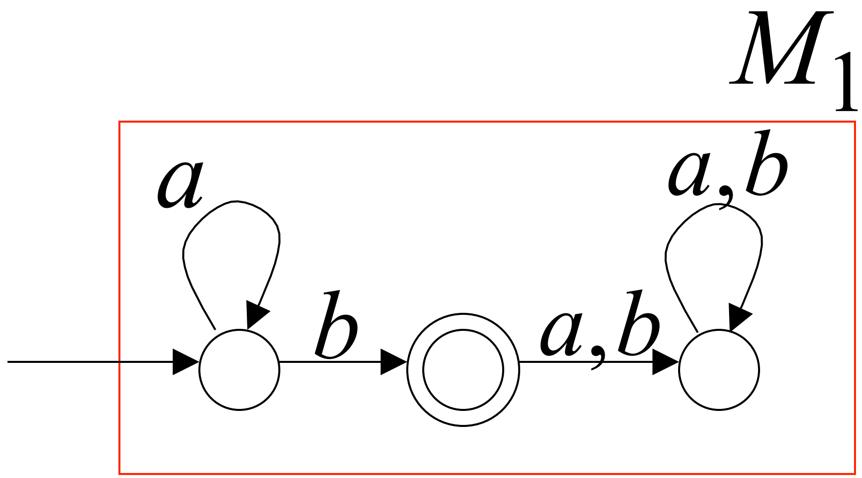


1. Take the DFA that accepts L_1

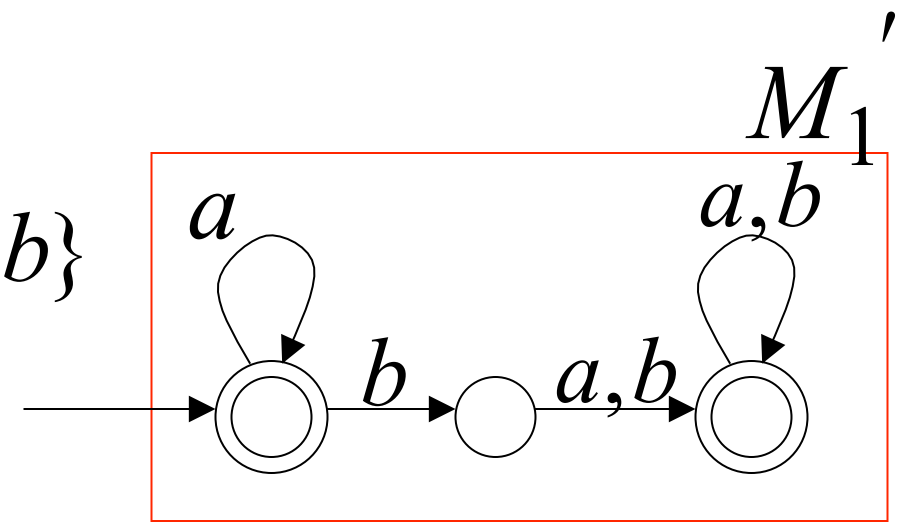
2. Make accepting states non-final, and vice-versa

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

DFA for L_1

Machine M_2

DFA for L_2

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

M simulates in parallel M_1 and M_2

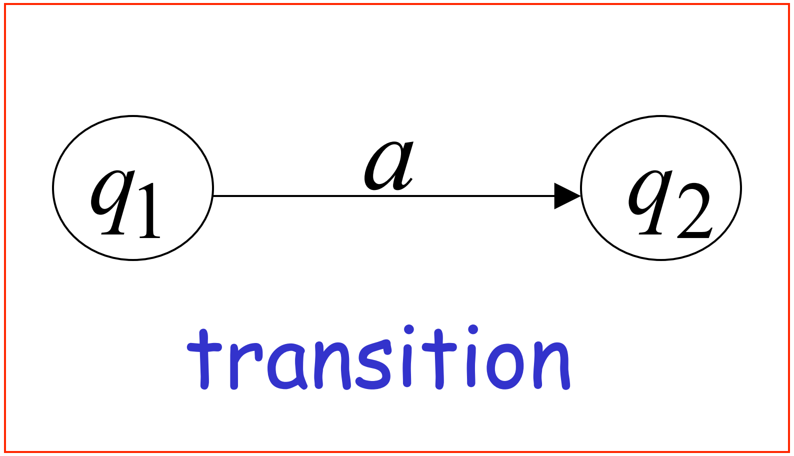
States in M

q_i, p_j

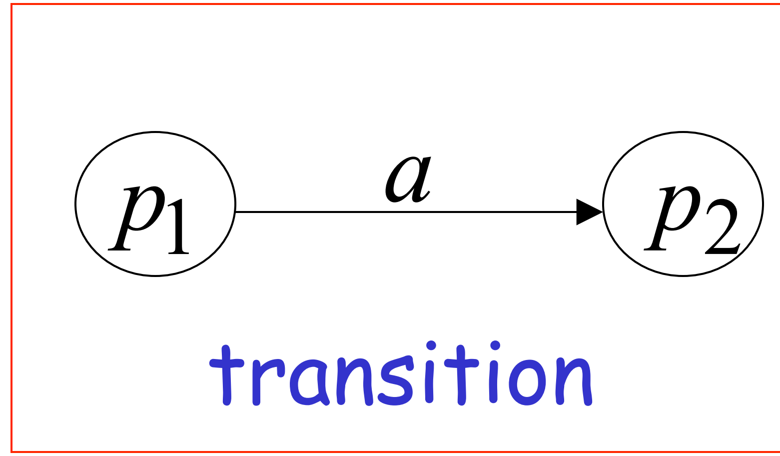
State in M_1

State in M_2

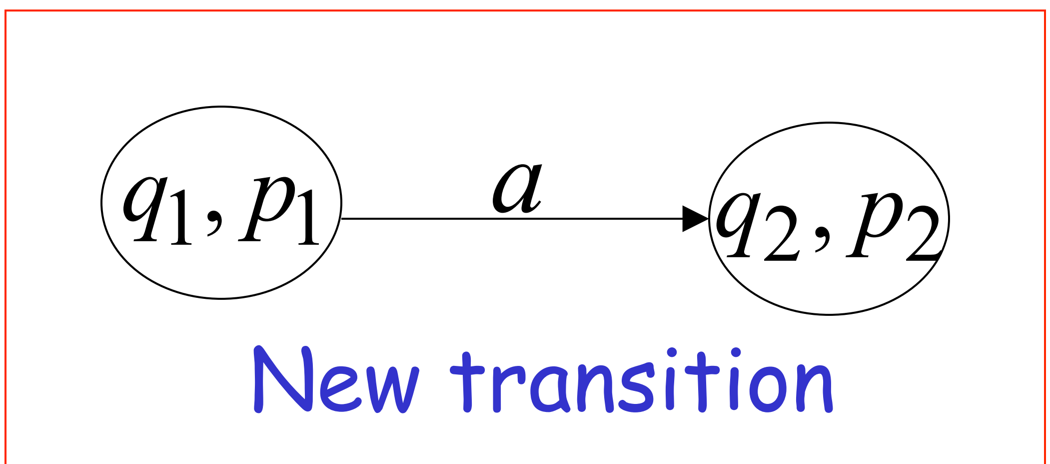
DFA M_1

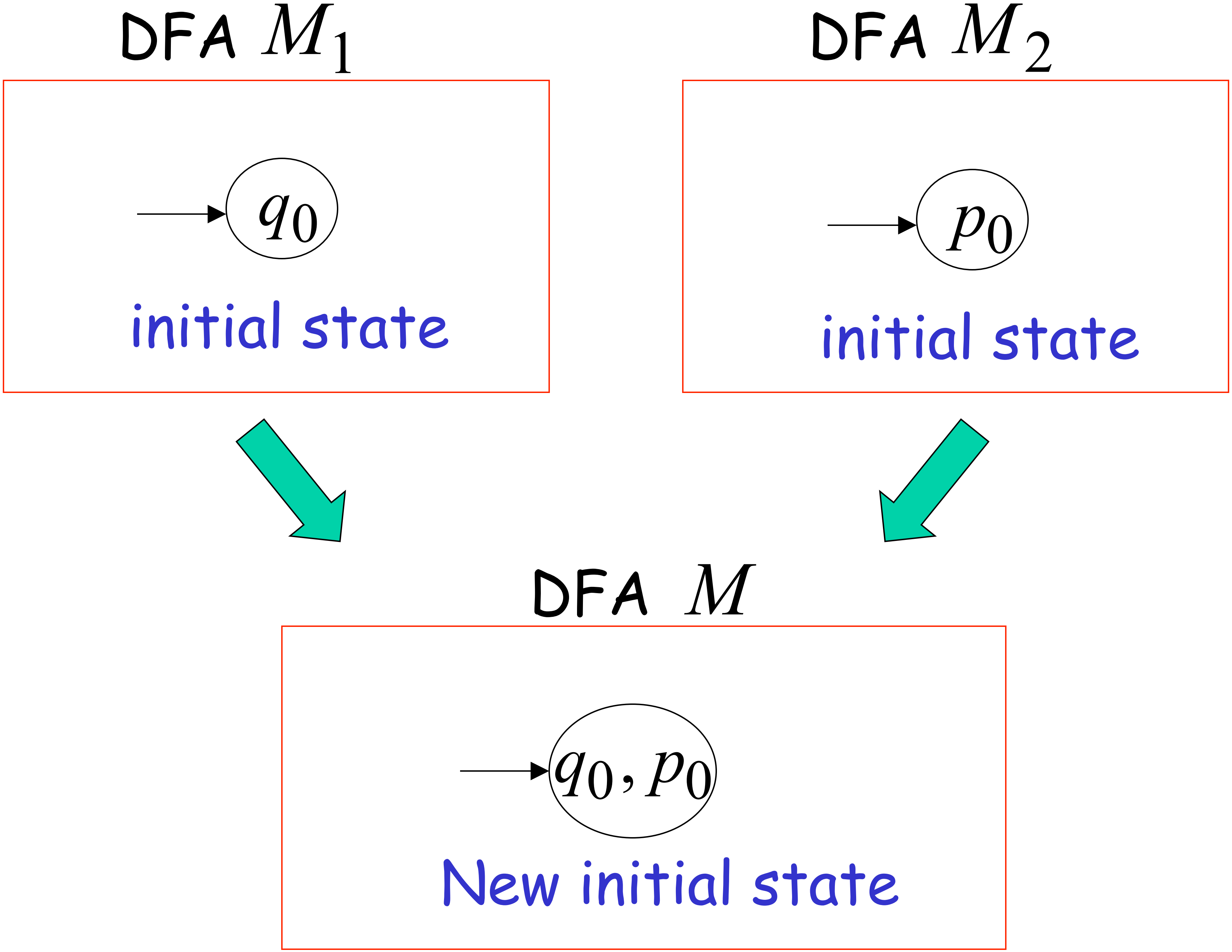


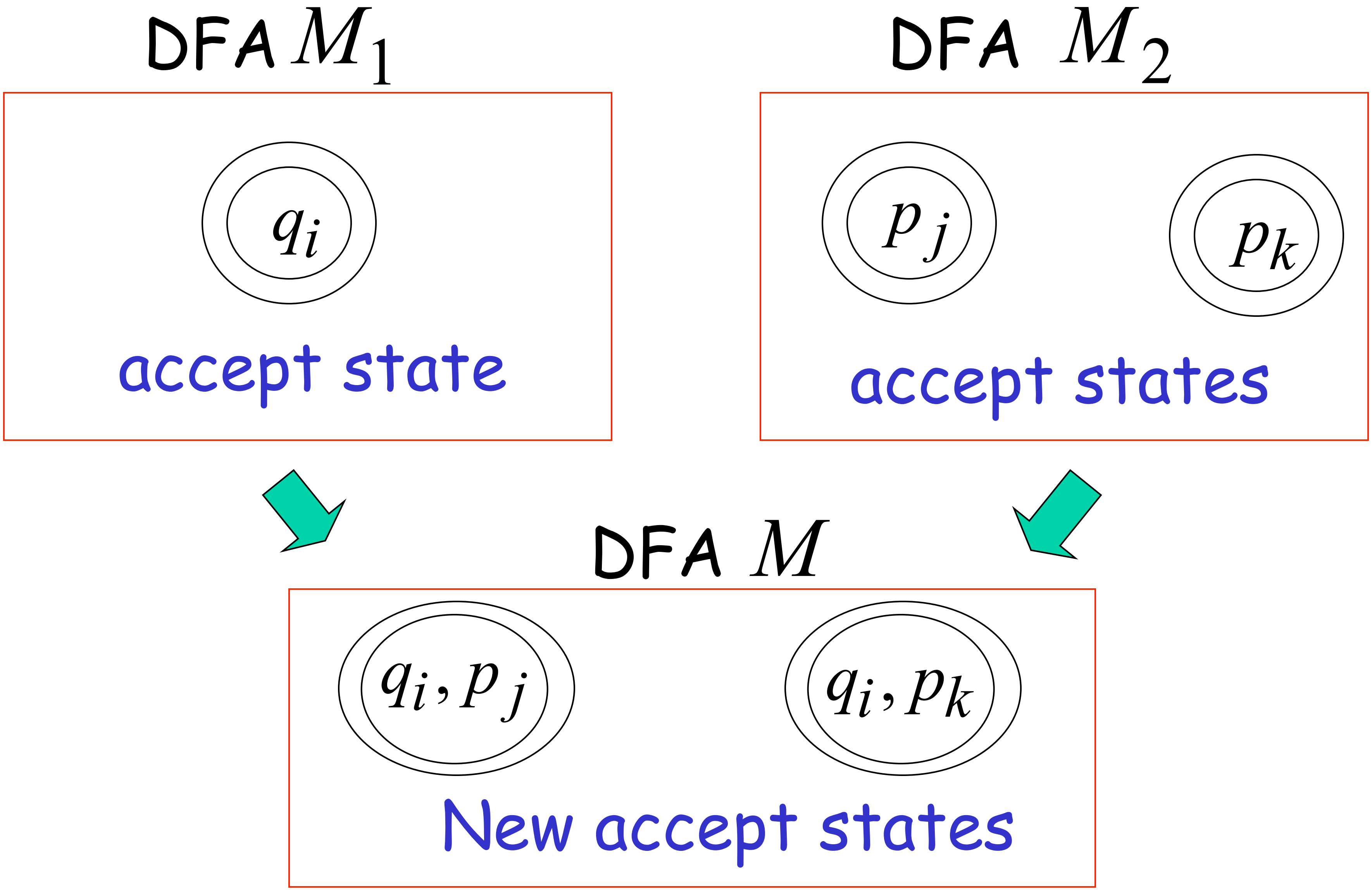
DFA M_2



DFA M




DFA M_1
q_0
initial state
DFA M_2
p_0
initial state
DFA M
q_0, p_0
New initial state



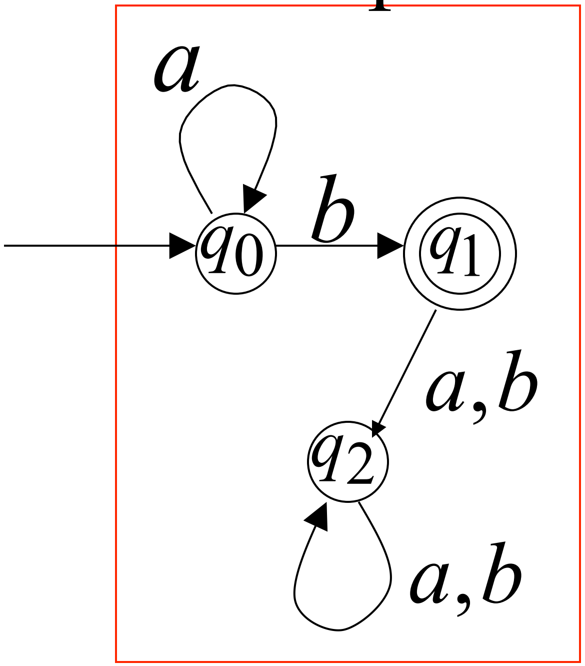
Both constituents must be accepting states

Example:

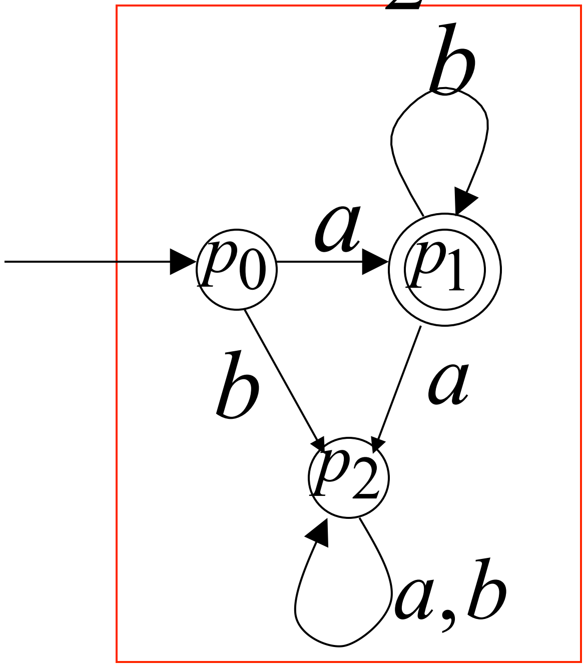
$$L_1 = \{a^n b\} \quad n \geq 0$$

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

M_1

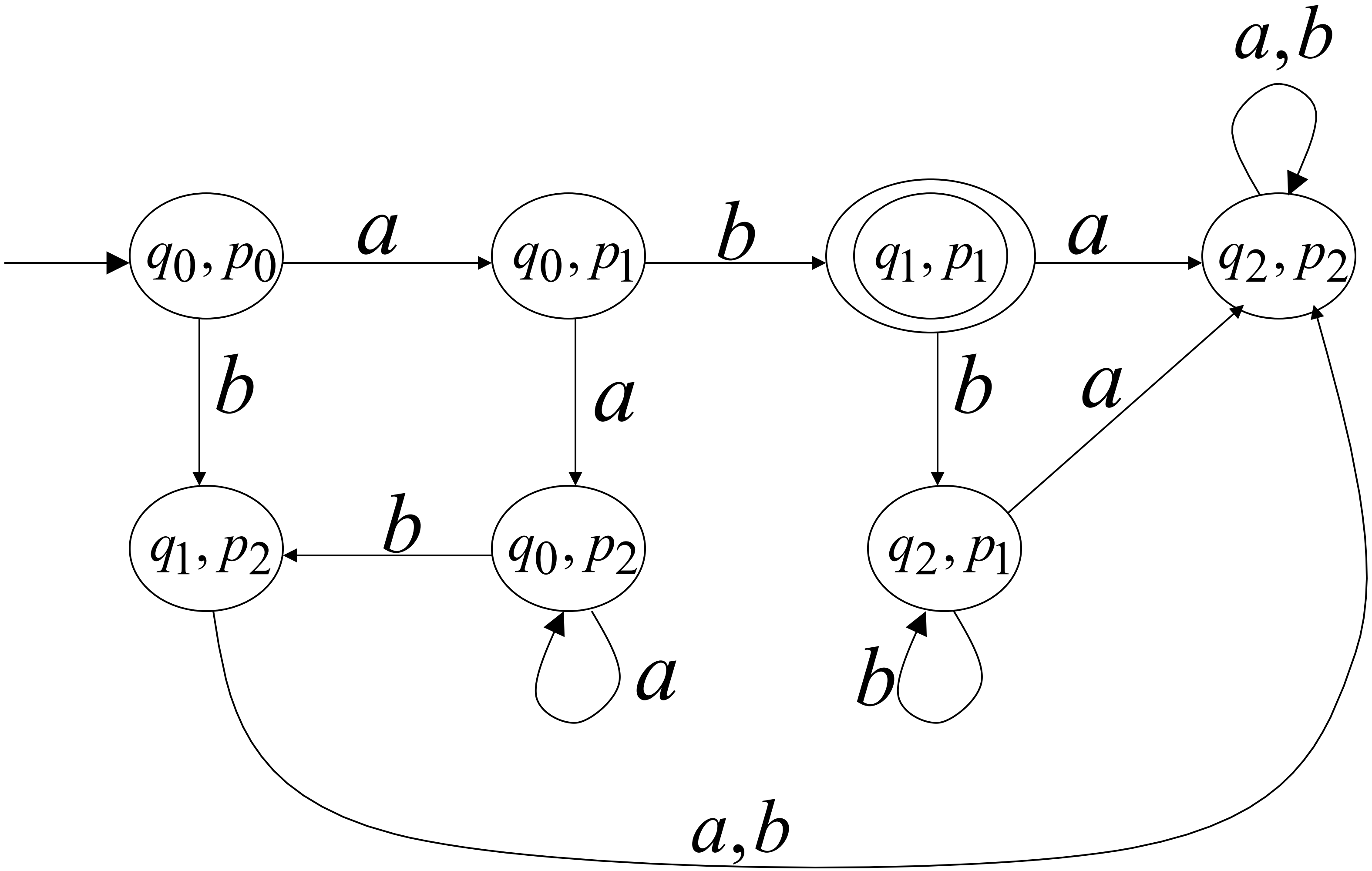


M_2



Automaton for intersection

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



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

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

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:

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

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

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

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

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

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 containing substring 00 }

Example

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

$L(r)$ = { all strings without substring 00 }

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 substring 00 }

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

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

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

r_1 and r_2
are equivalent
regular expressions

Regular Expressions and Regular Languages

Theorem

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

Proof:

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

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

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

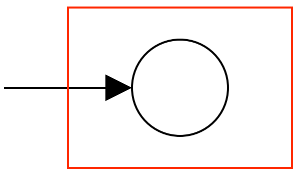
Proof by induction on the size of r

Induction Basis

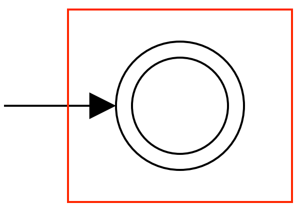
Primitive Regular Expressions: $\emptyset, \quad \lambda, \quad \alpha$

Corresponding

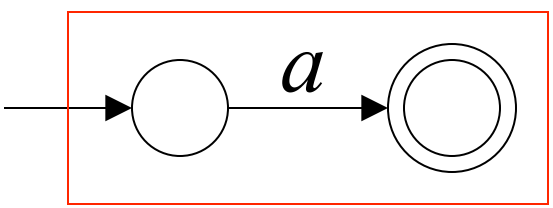
NFAs



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



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



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

regular languages

Inductive Hypothesis

Suppose

that for regular expressions r_1 and r_2,

$L(r_1)$ 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:

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

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

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

Are regular languages

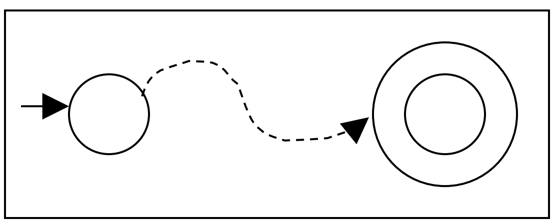
$L((r_1)) = L(r_1)$ is trivially a regular language (by induction hypothesis)

End of Proof-Part 1

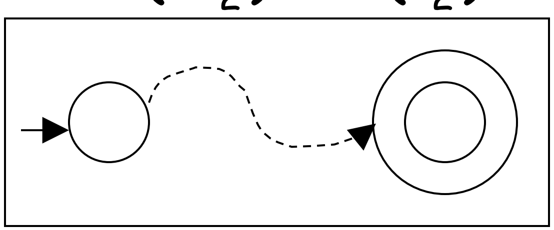
Using the regular closure of these operations, we can construct recursively the NFA M that accepts $L(M) = L(r)$

Example: $r = r_1 + r_2$

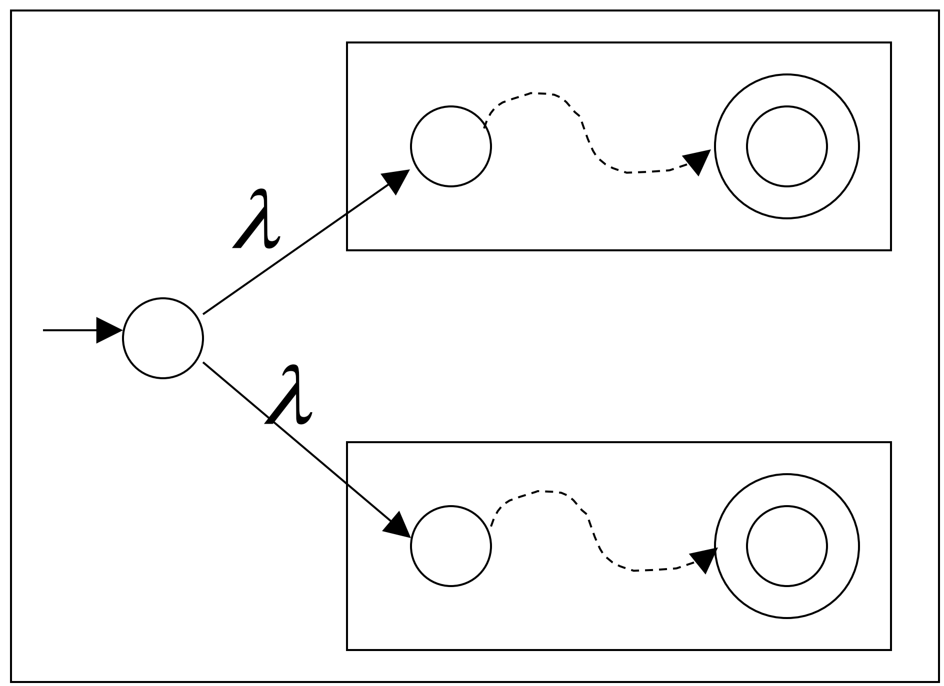
$L(M_1) = L(r_1)$



$L(M_2) = L(r_2)$



$L(M) = L(r)$



Proof - Part 2

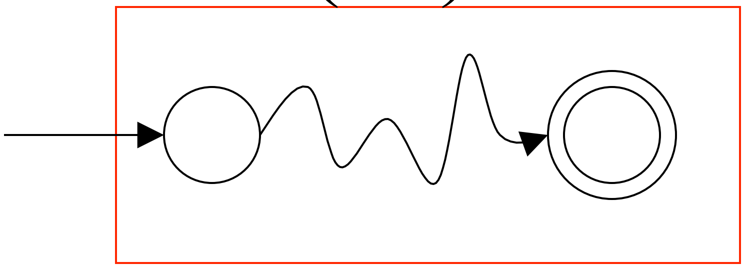
$$\left\{\begin{array}{l}\text{Languages} \\ \text{Generated by} \\ \text{Regular Expressions}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular} \\ \text{Languages}\end{array}\right\}$$

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

We will convert an NFA that accepts L to a regular expression

Since L is regular, there is a NFA M that accepts it

$$L(M) = L$$



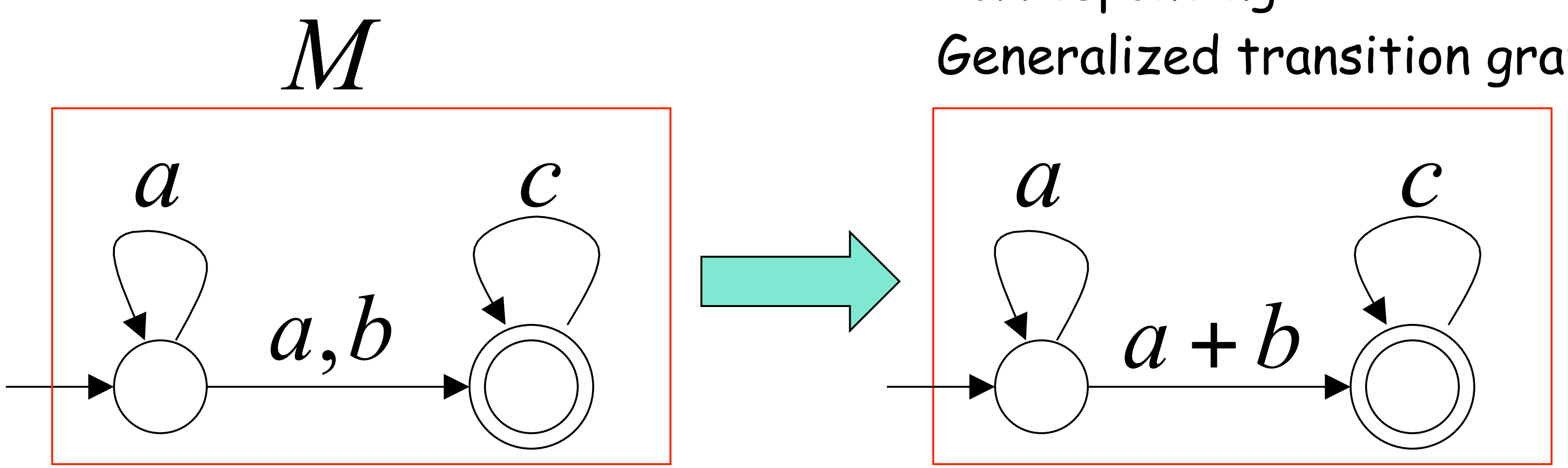
Take it with a single accept state

From M construct the equivalent

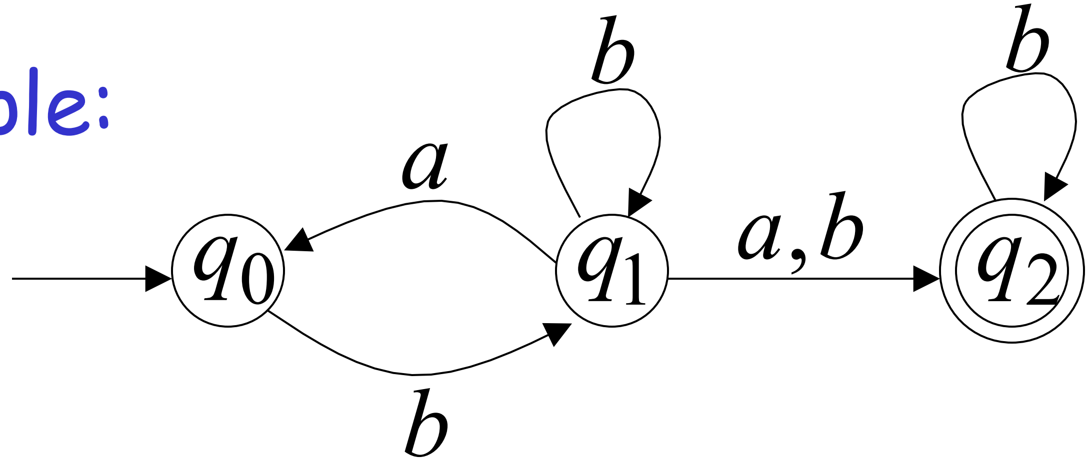
Generalized Transition Graph

in which transition labels are regular expressions

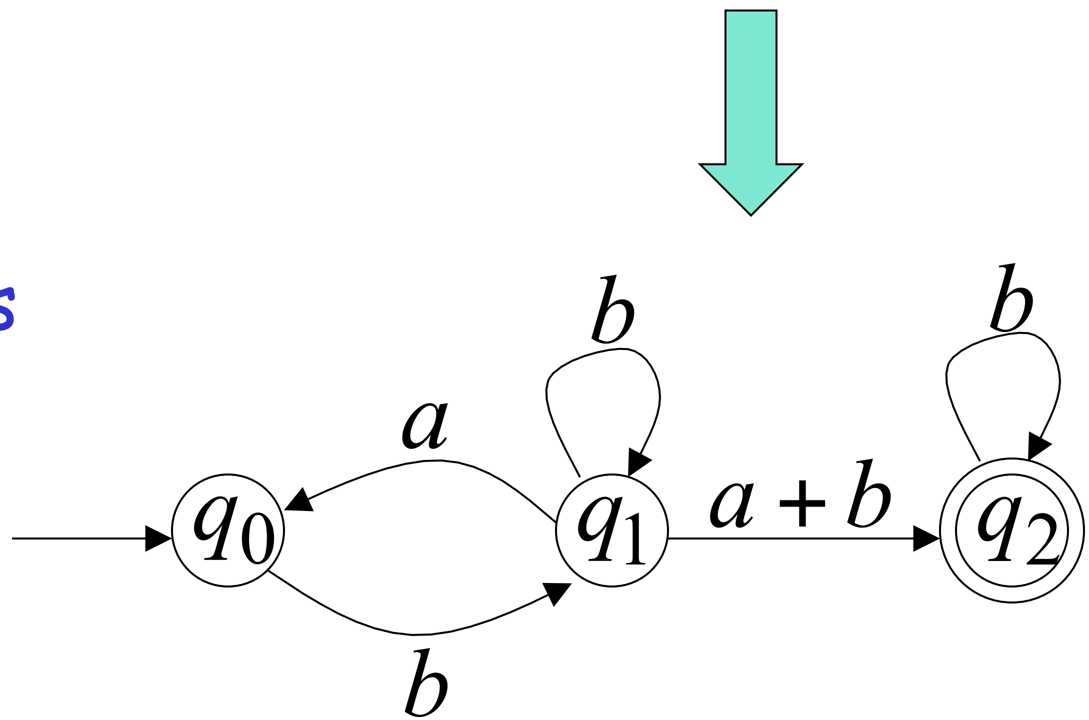
Example:



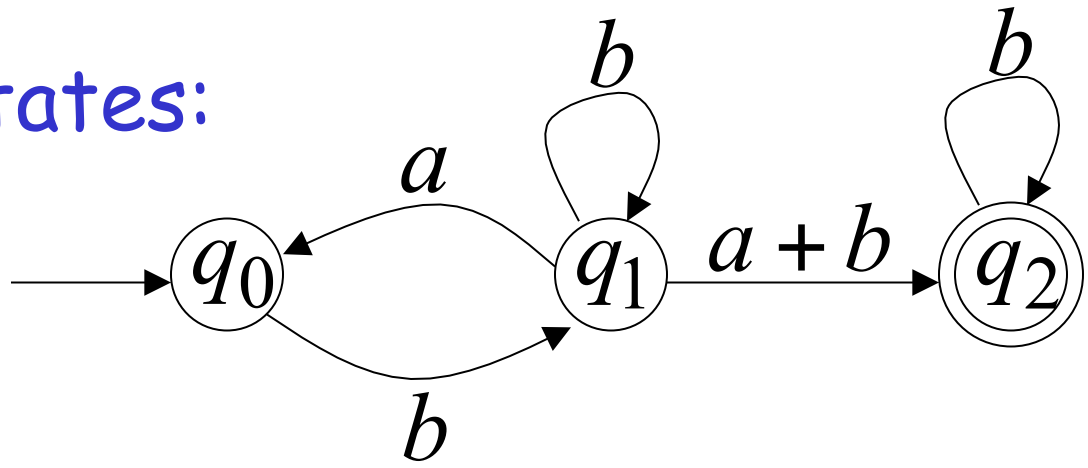
Another Example:



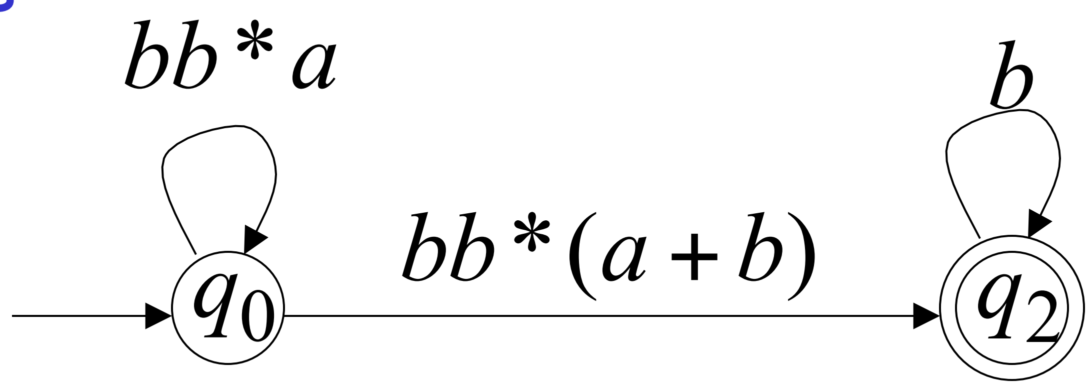
Transition labels are regular expressions



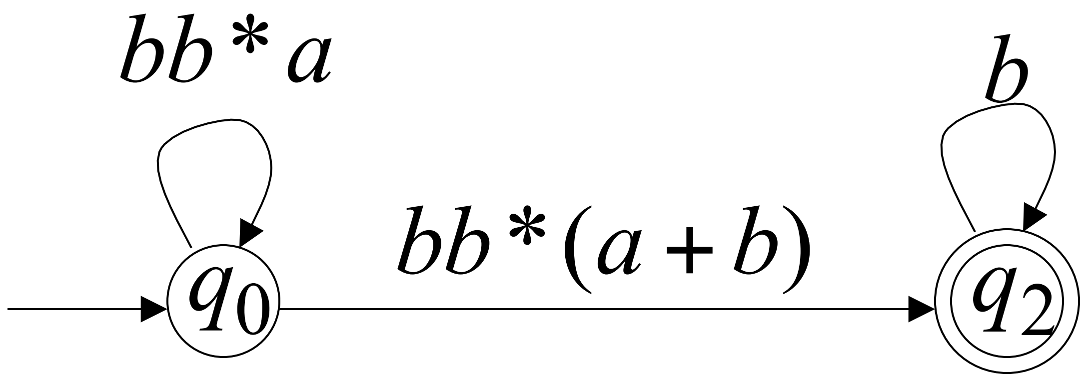
Reducing the states:



Transition labels are regular expressions



Resulting Regular Expression:

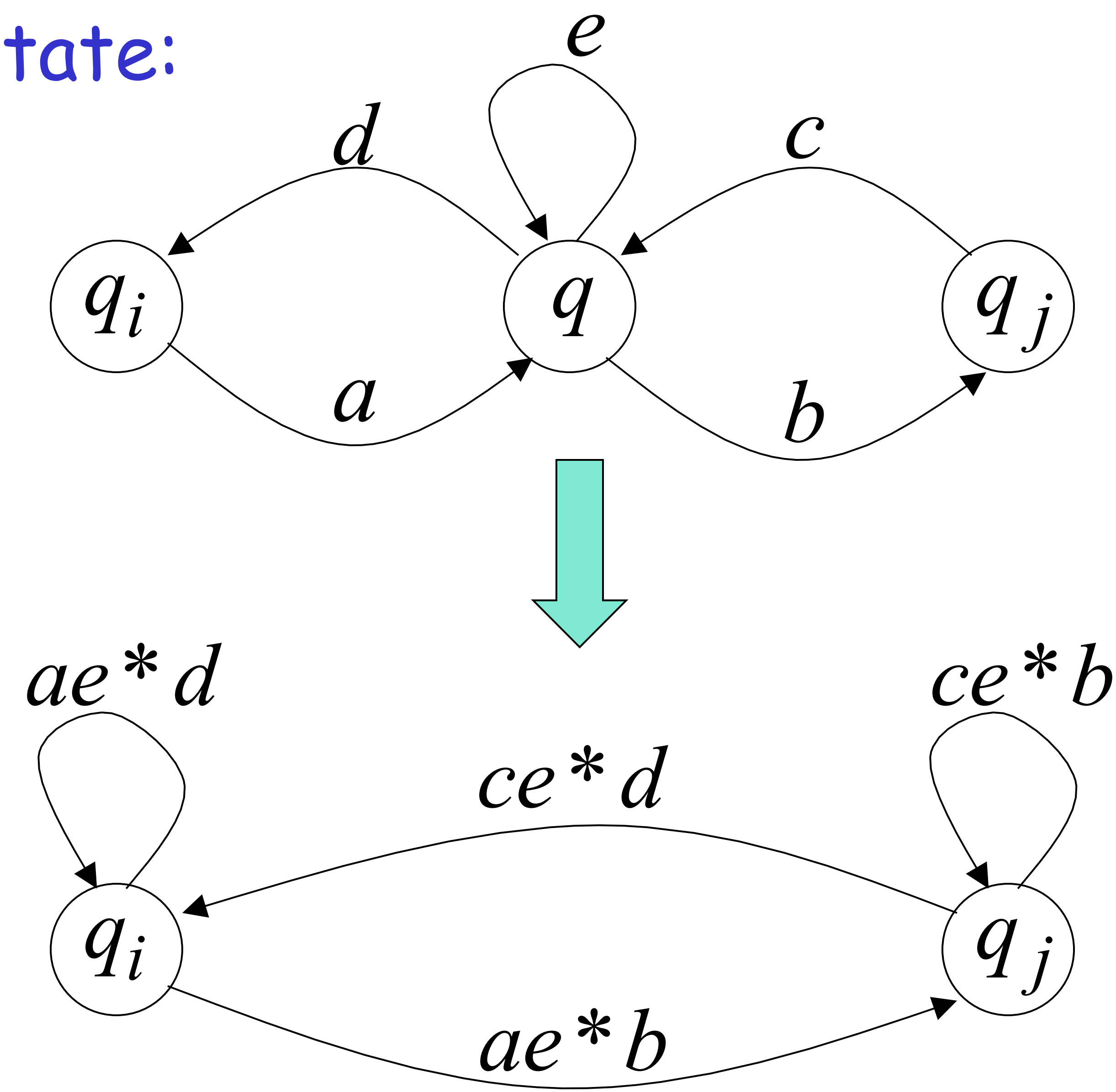


$$r = (bb^*a)^* \cdot bb^*(a+b) \cdot b^*$$

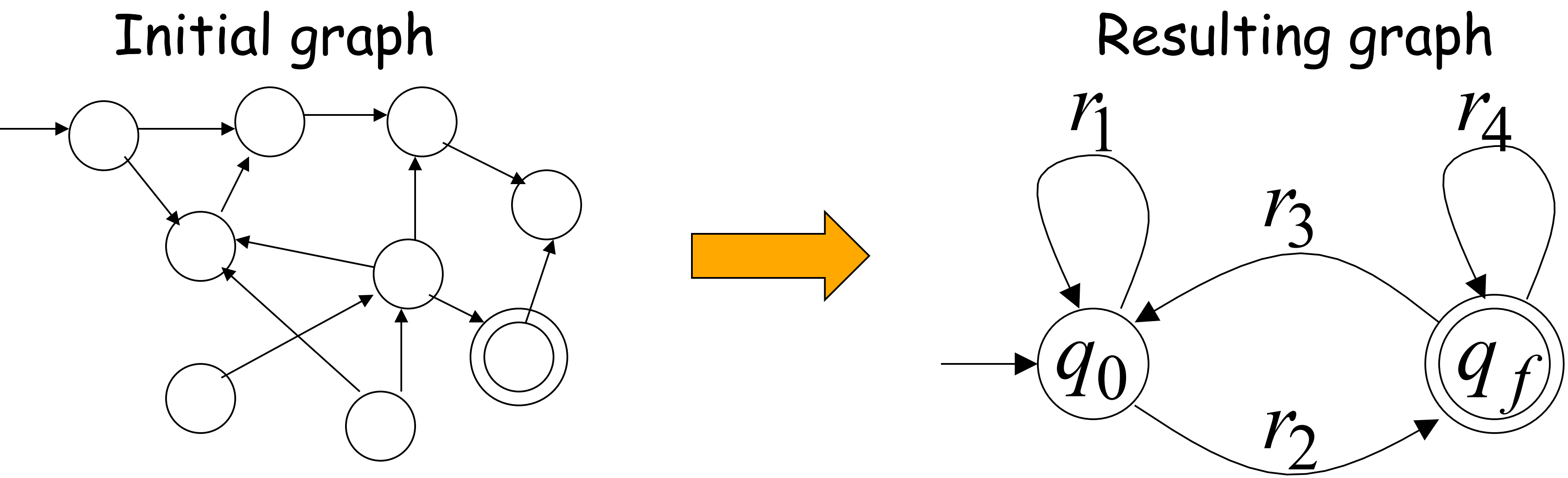
$$L(r) = L(M) = L$$

In General

Removing a state:



By repeating the process until two states are left, the resulting graph is



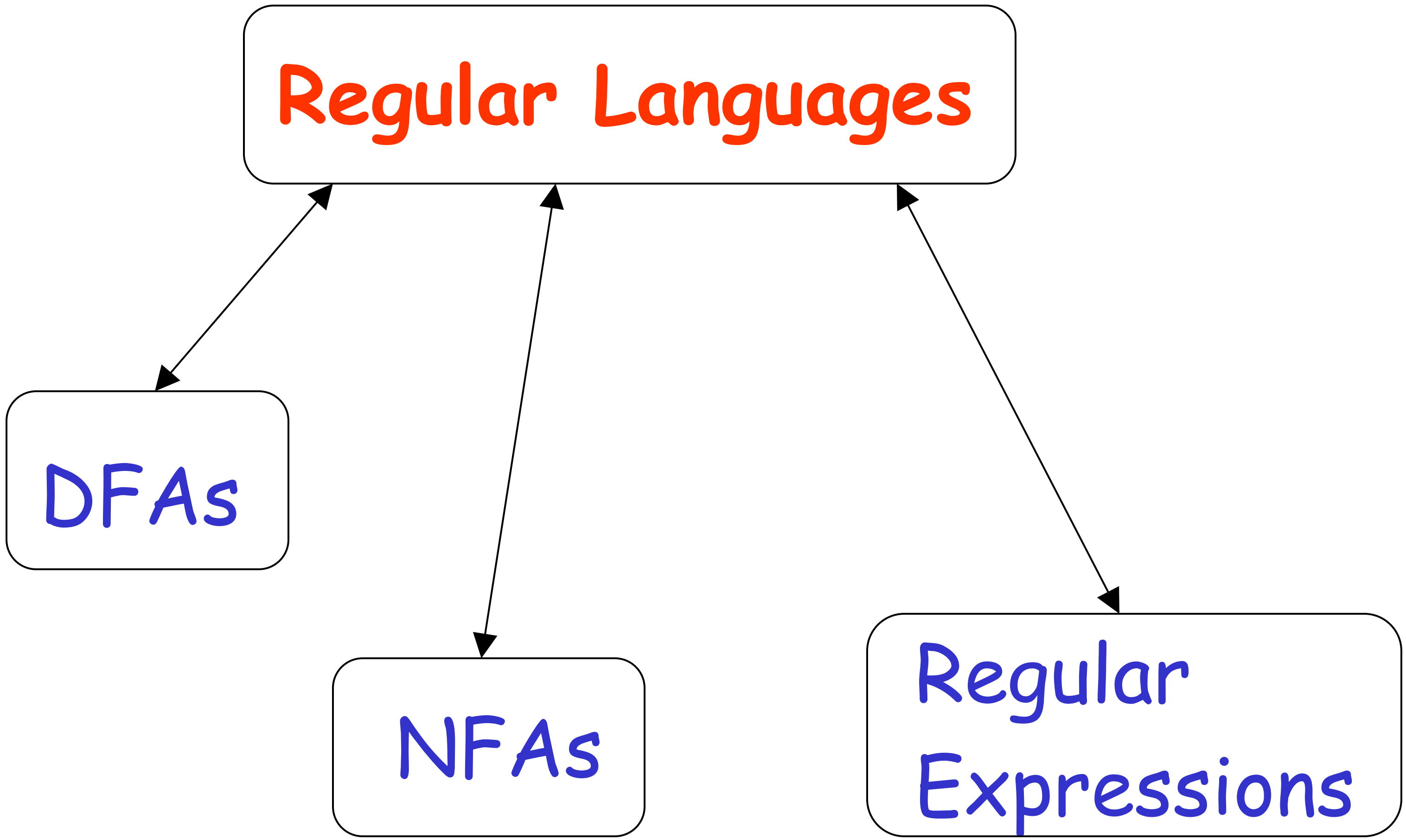
The resulting regular expression:

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

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

End of Proof-Part 2

Standard Representations of Regular Languages



When we say: We are given a Regular Language L

We mean: Language L is in a standard representation

(DFA, NFA, or Regular Expression)