Deterministic

Finite Automata

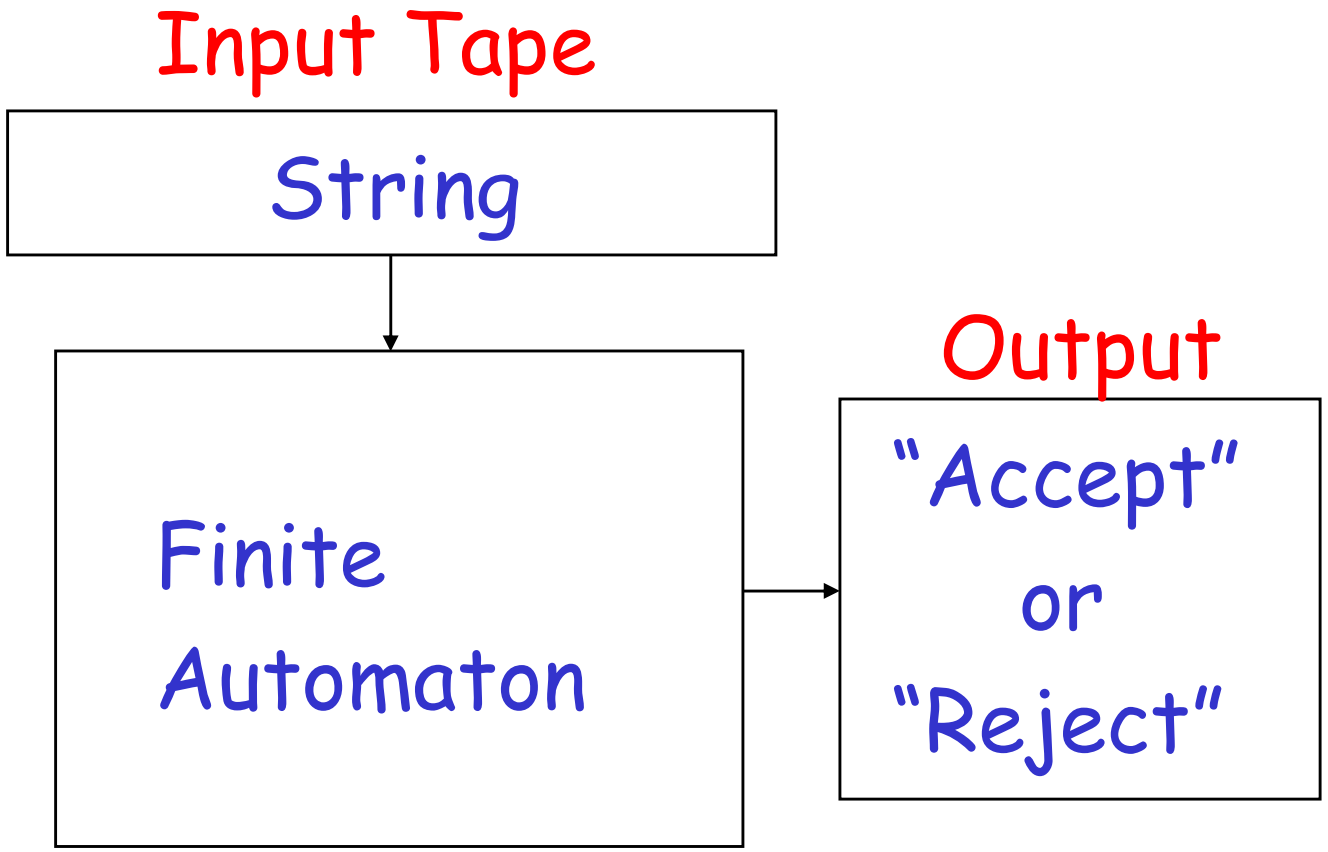
And Regular Languages

Finite Automata

- Languages are generally infinite, but must be describable in some finite way

  - Finite Automata - recognizer
  - Regular expression – notations to describe how strings of the language can be generated.

Deterministic Finite Automaton (DFA)



Finite Automata

- Finite automata are finite collections of states with transition rules that take you from one state to another.
- Original application was sequential switching circuits, where the "state" was the settings of internal bits.
- Informally, a state diagram that comprehensively captures all possible states and transitions that a machine can take while responding to a stream or sequence of input symbols
- Recognizer for "Regular Languages"

Finite Automaton (FA)

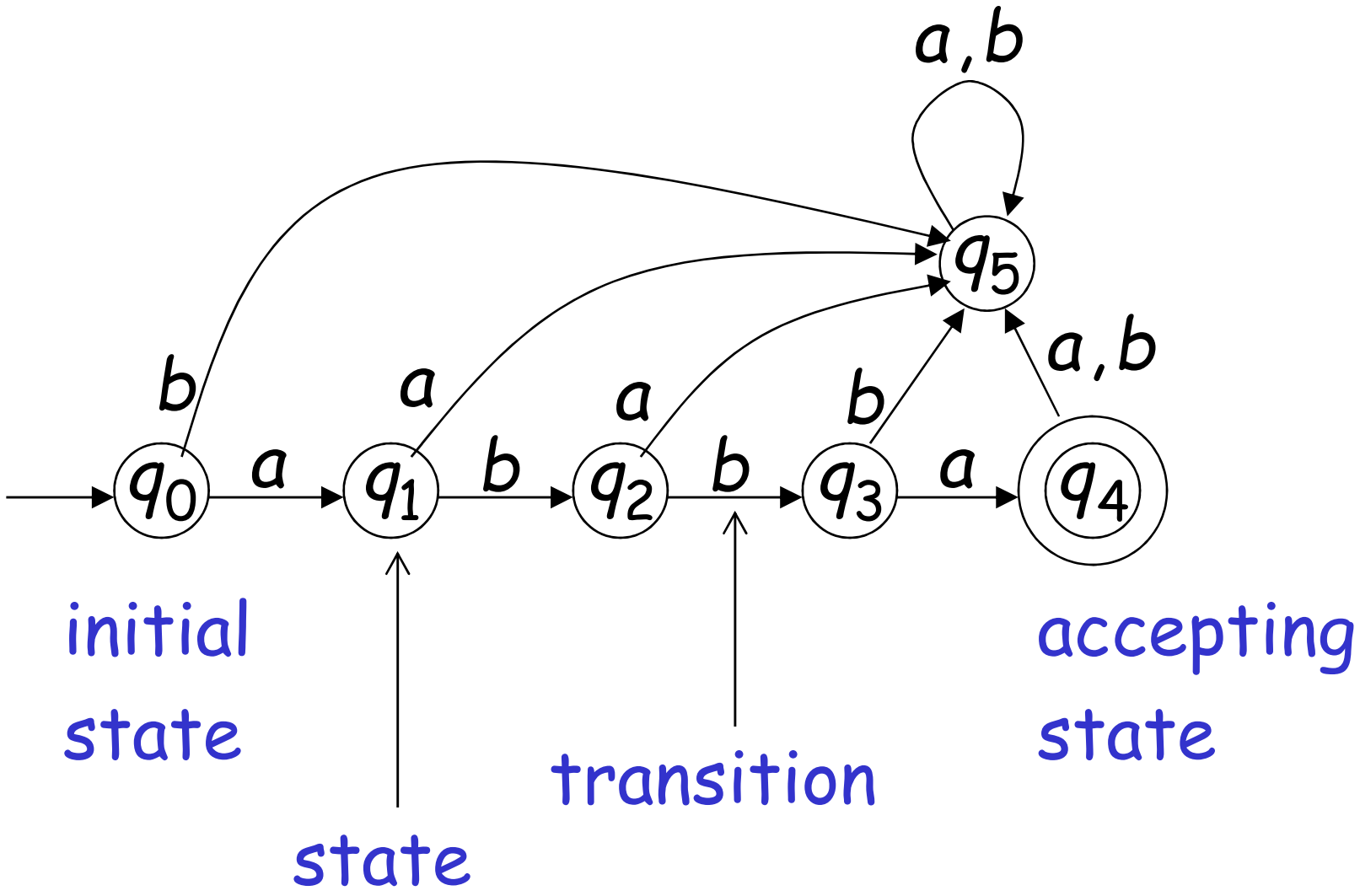
Deterministic Finite Automata (DFA)

- The machine can exist in only one state at any given time

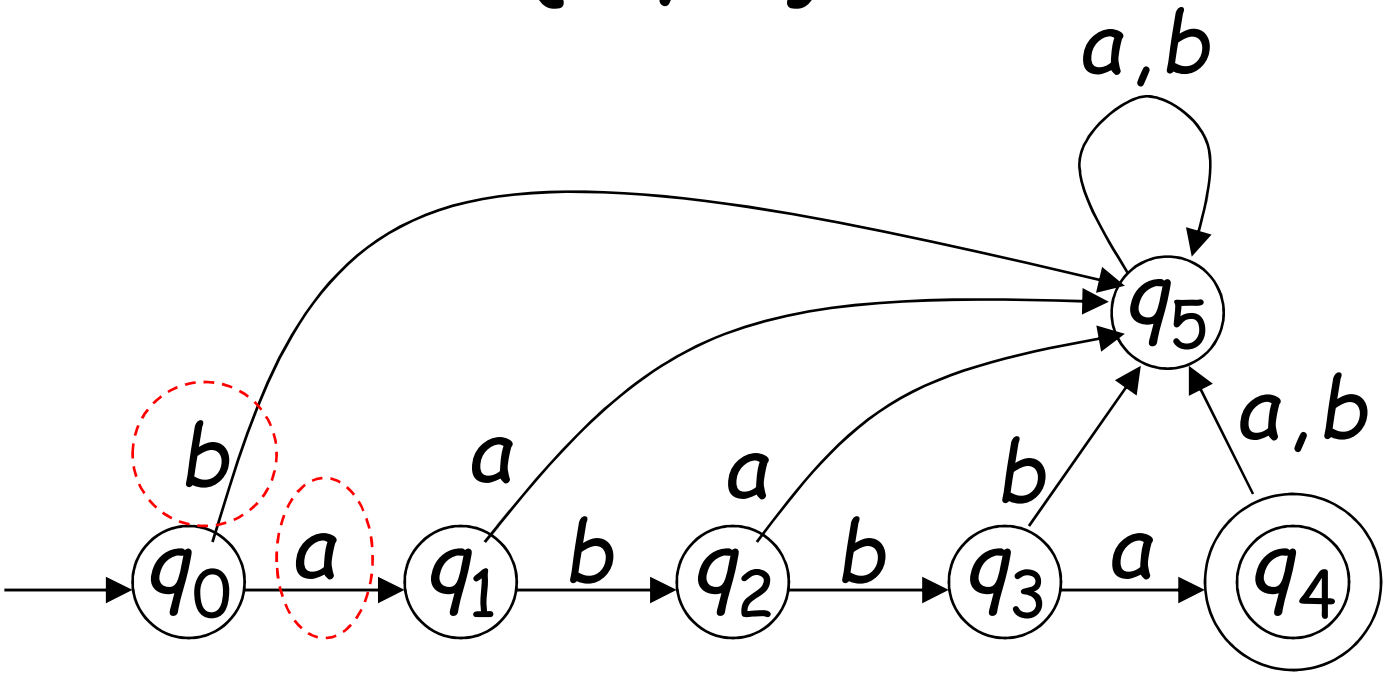
Non-deterministic Finite Automata (NFA)

- The machine can exist in multiple states at the same time

Transition Graph



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



For every state, there is a transition for every symbol in the alphabet

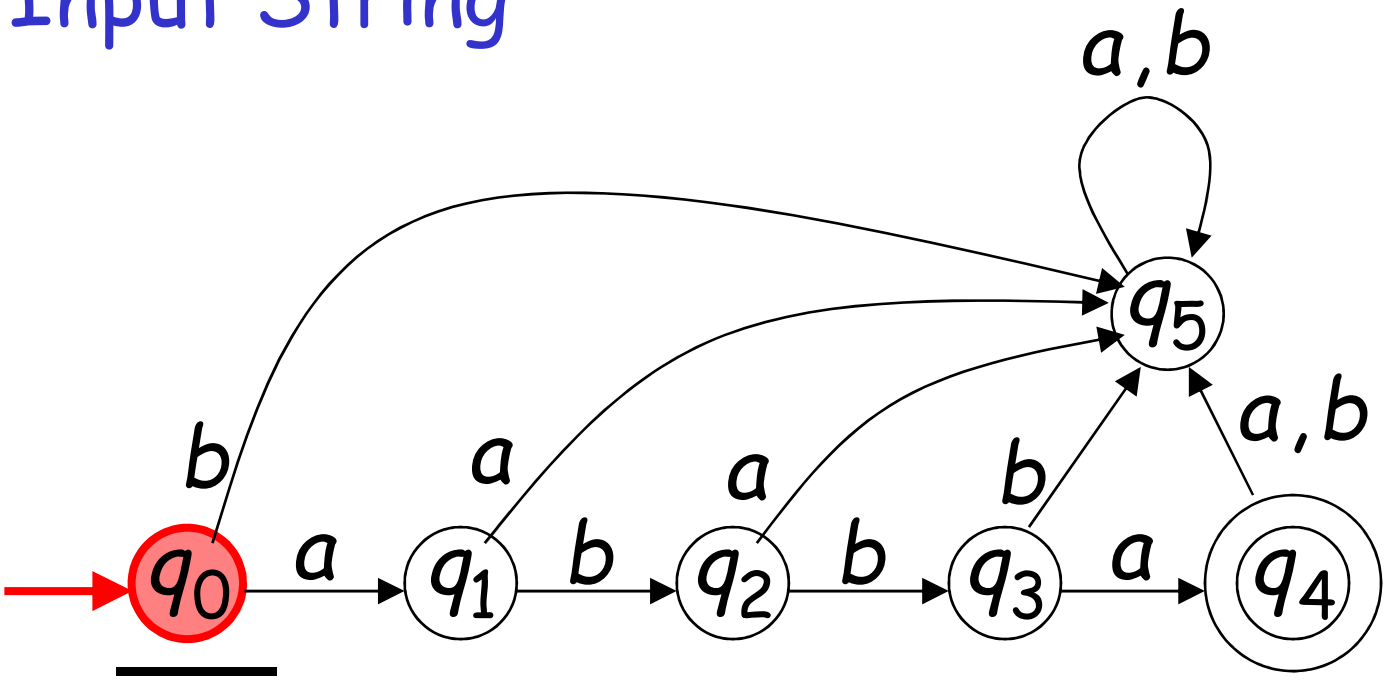
Initial Configuration

head

Input Tape

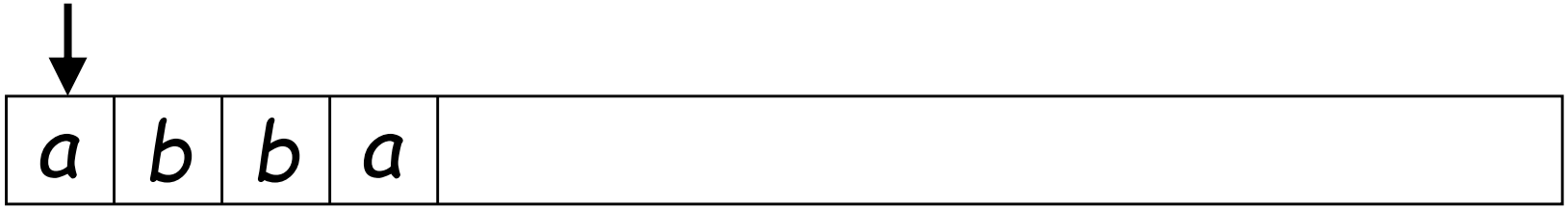
a	b	b	a	

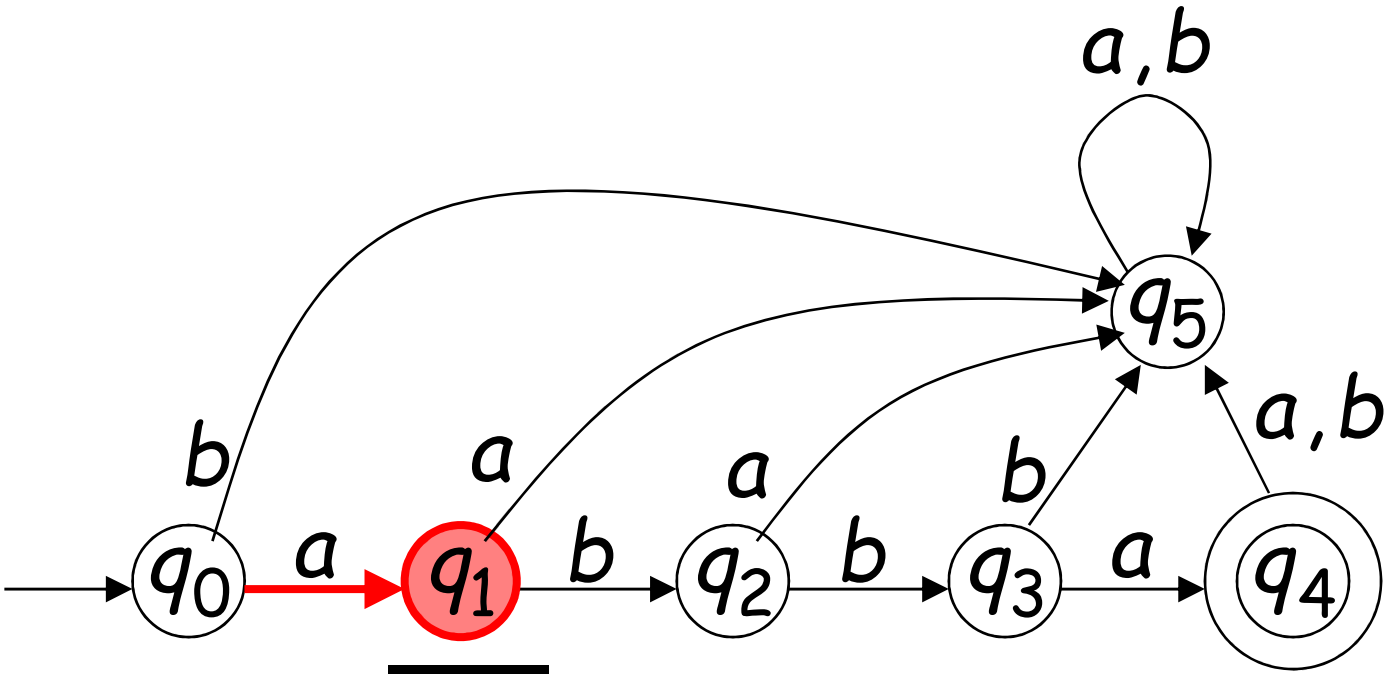
Input String

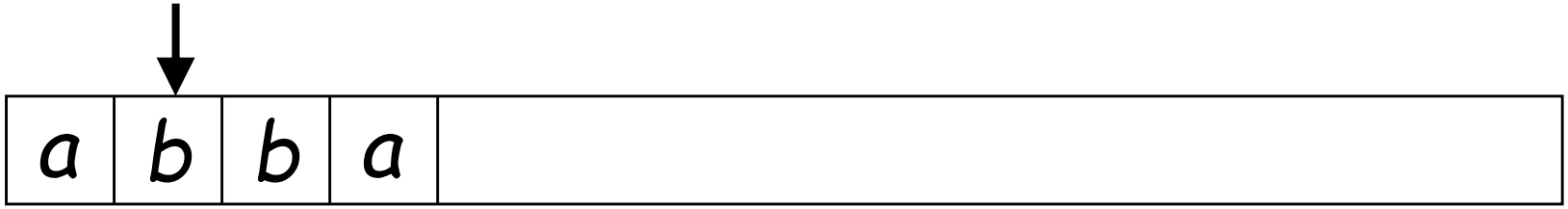


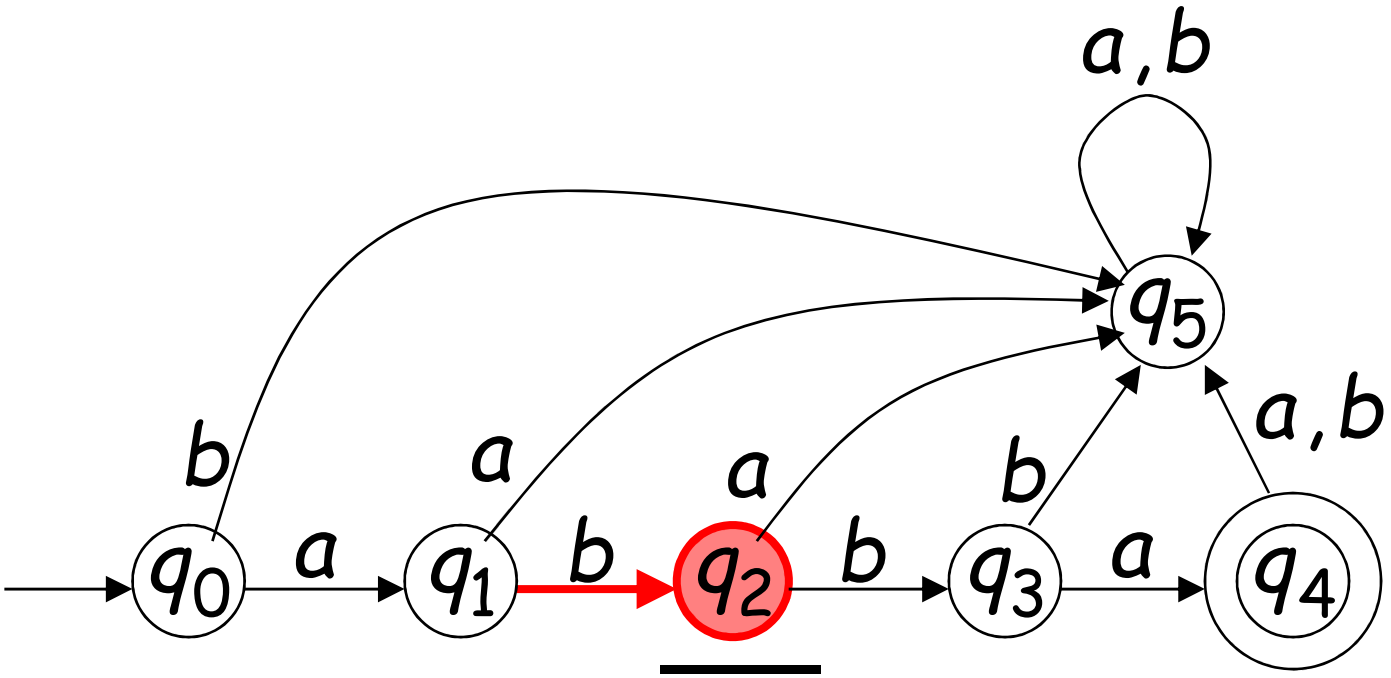
Initial state

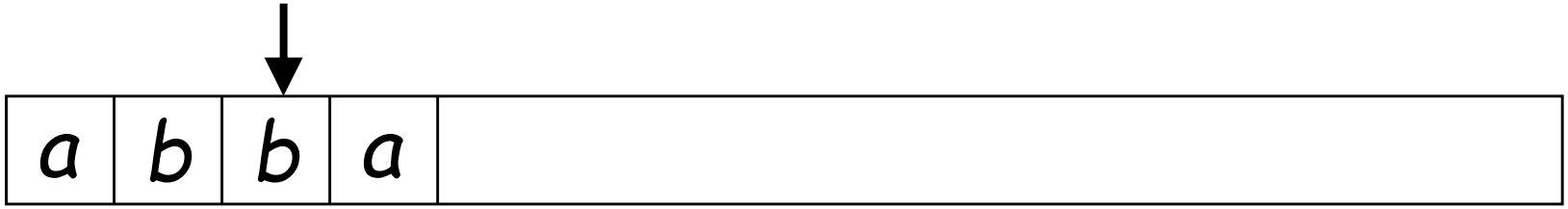
Scanning the Input

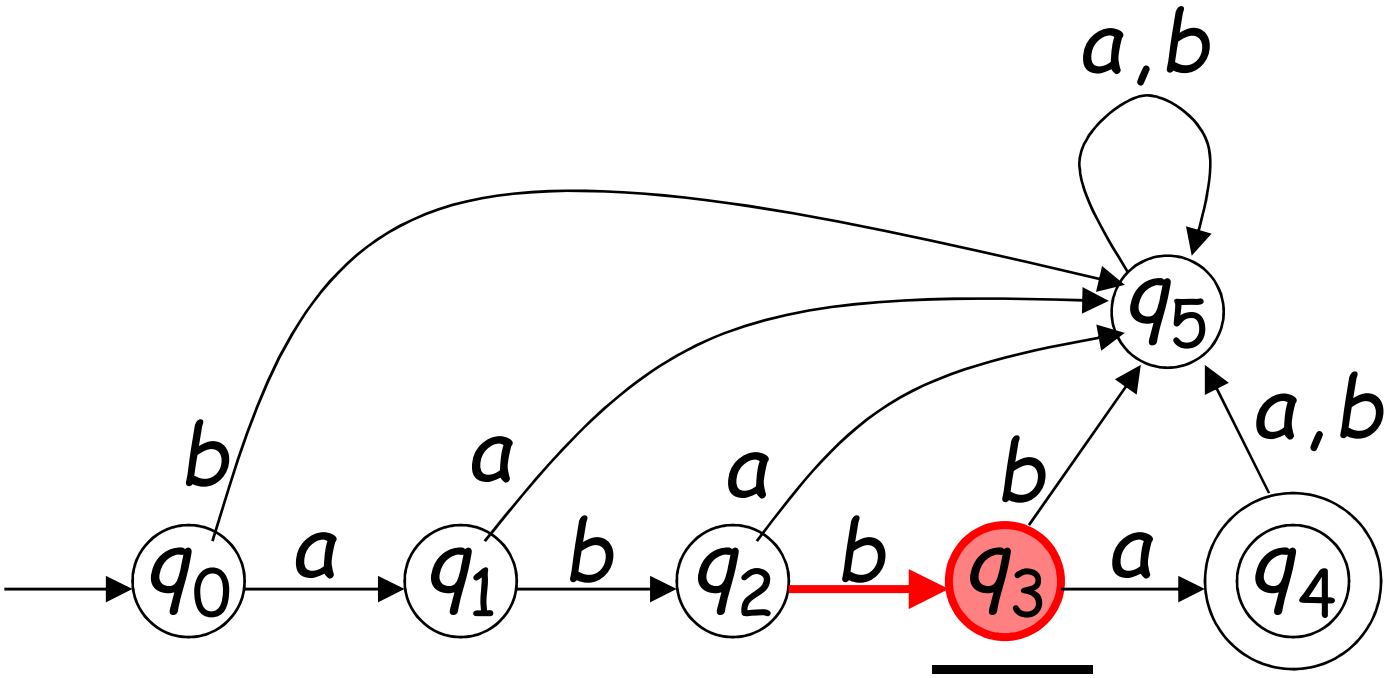




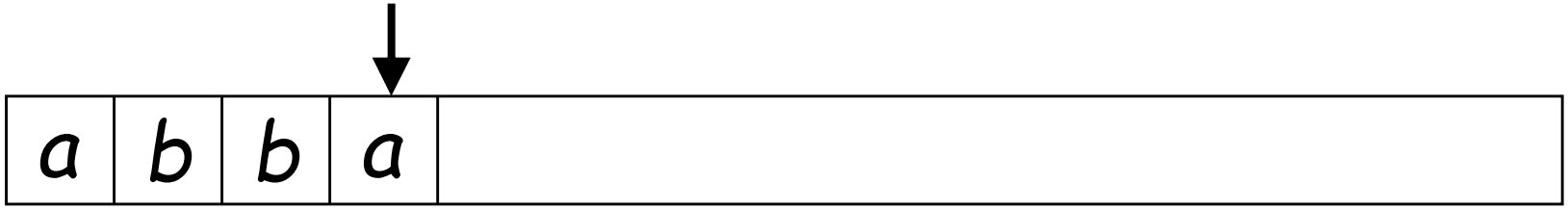

a
b
b
a

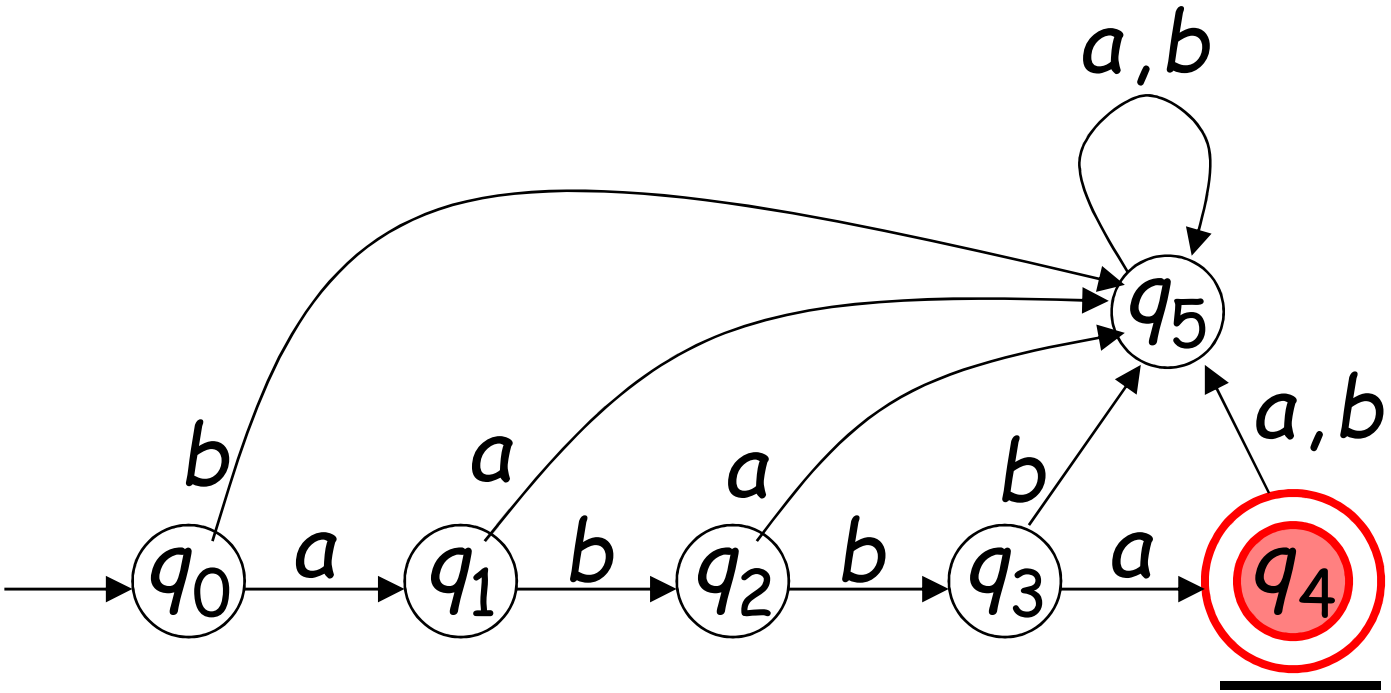

a, b
q5
b
a
a
b
a, b
q0
a
q1
b
q2
b
q3
a
q4


a
b
b
a


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

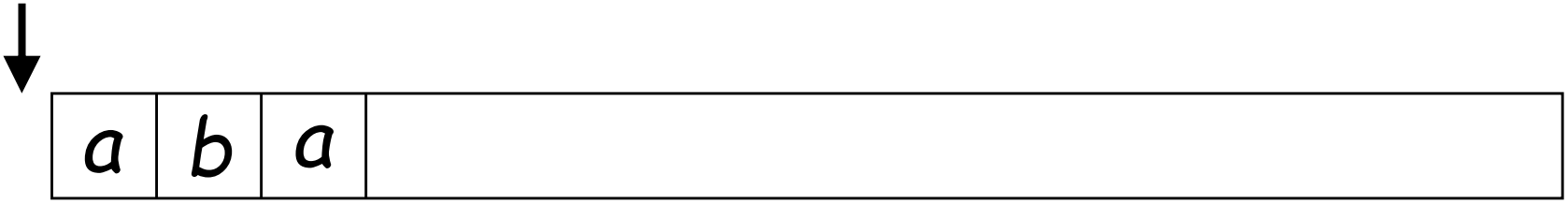
Input finished



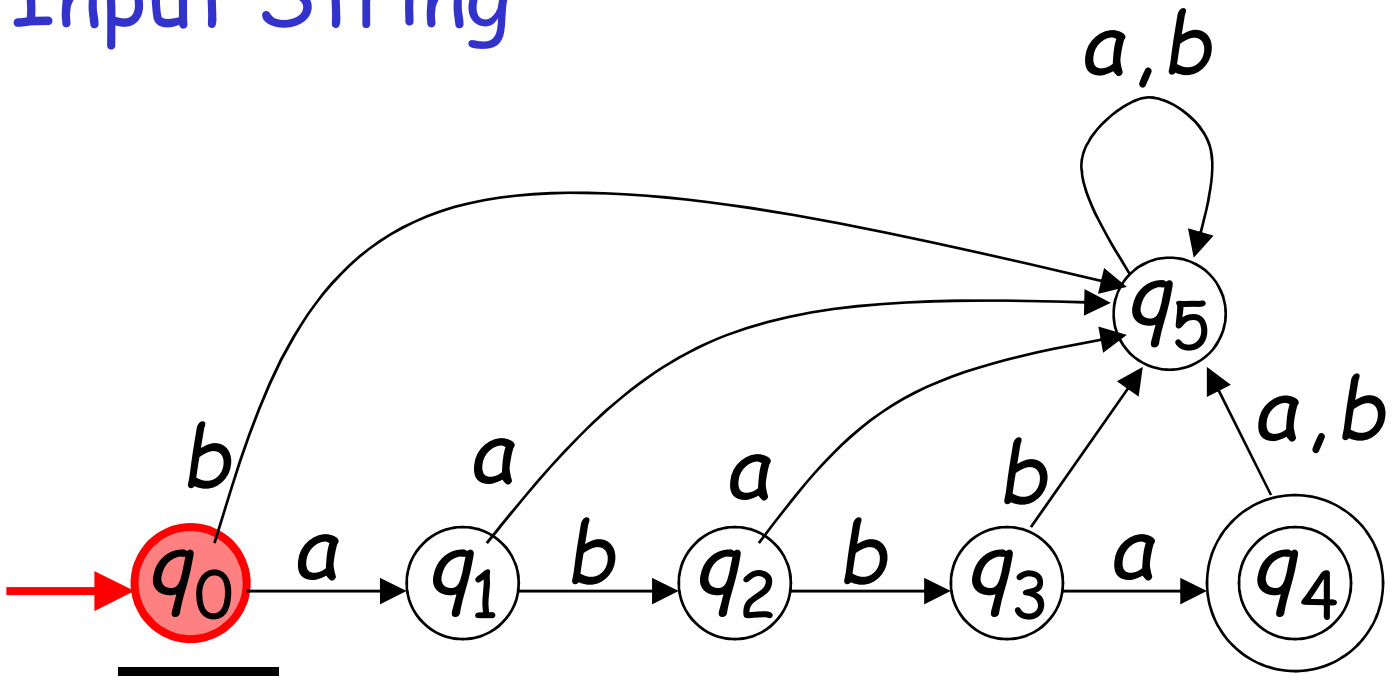


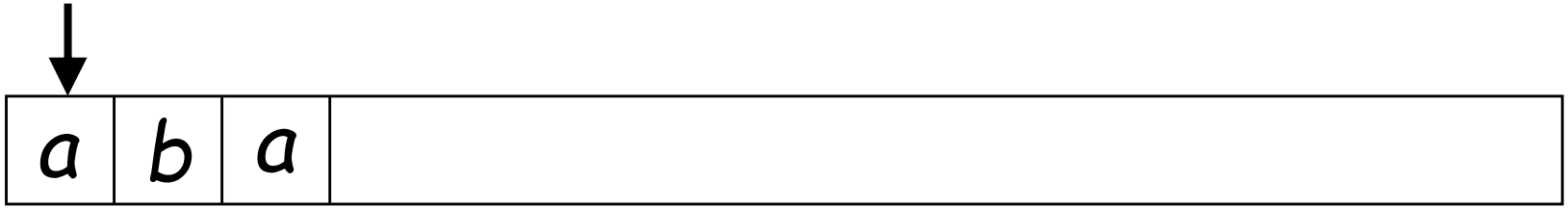
accept

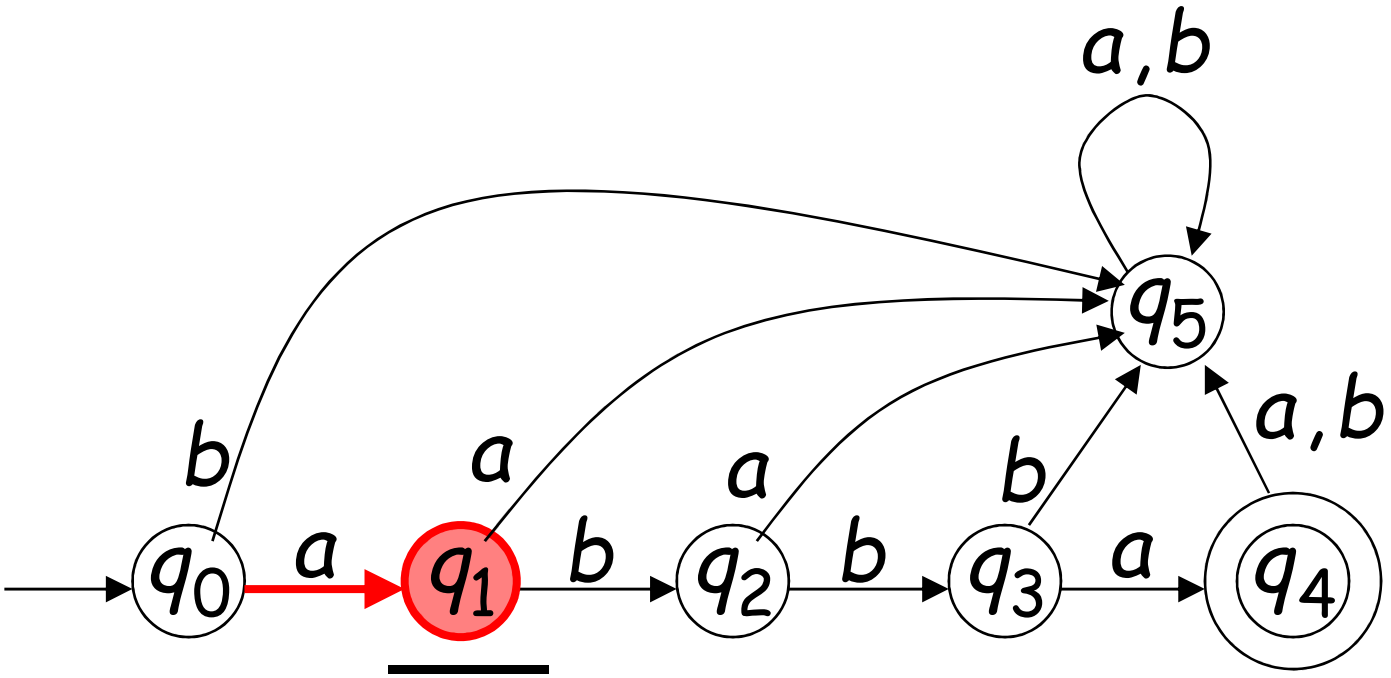
A Rejection Case

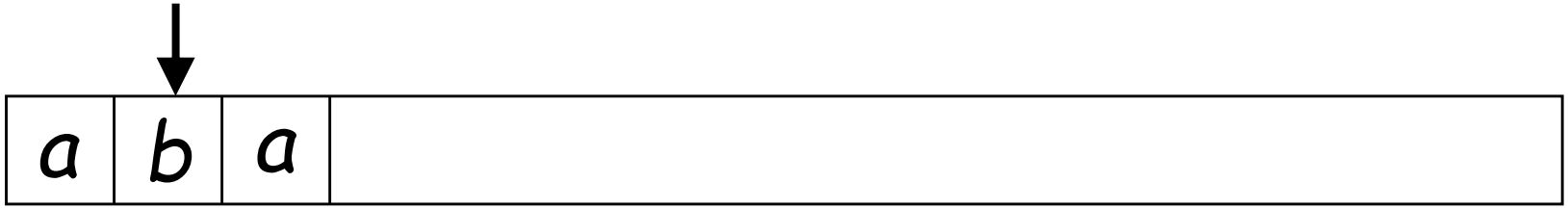


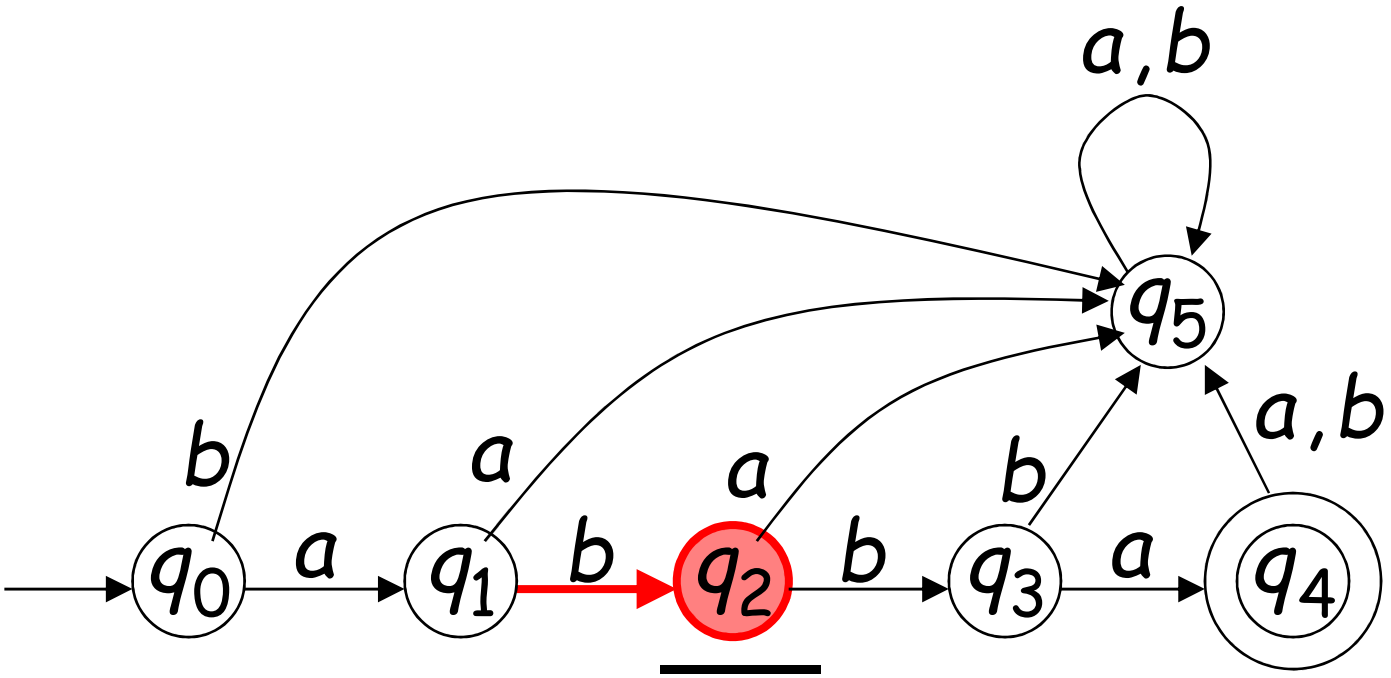
Input String



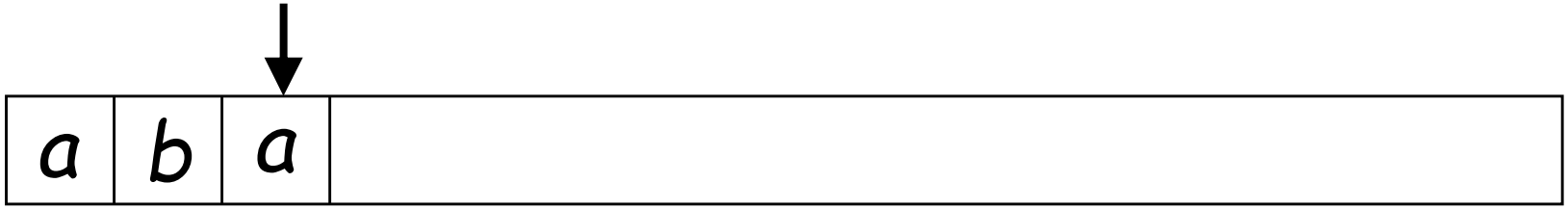

a
b
a

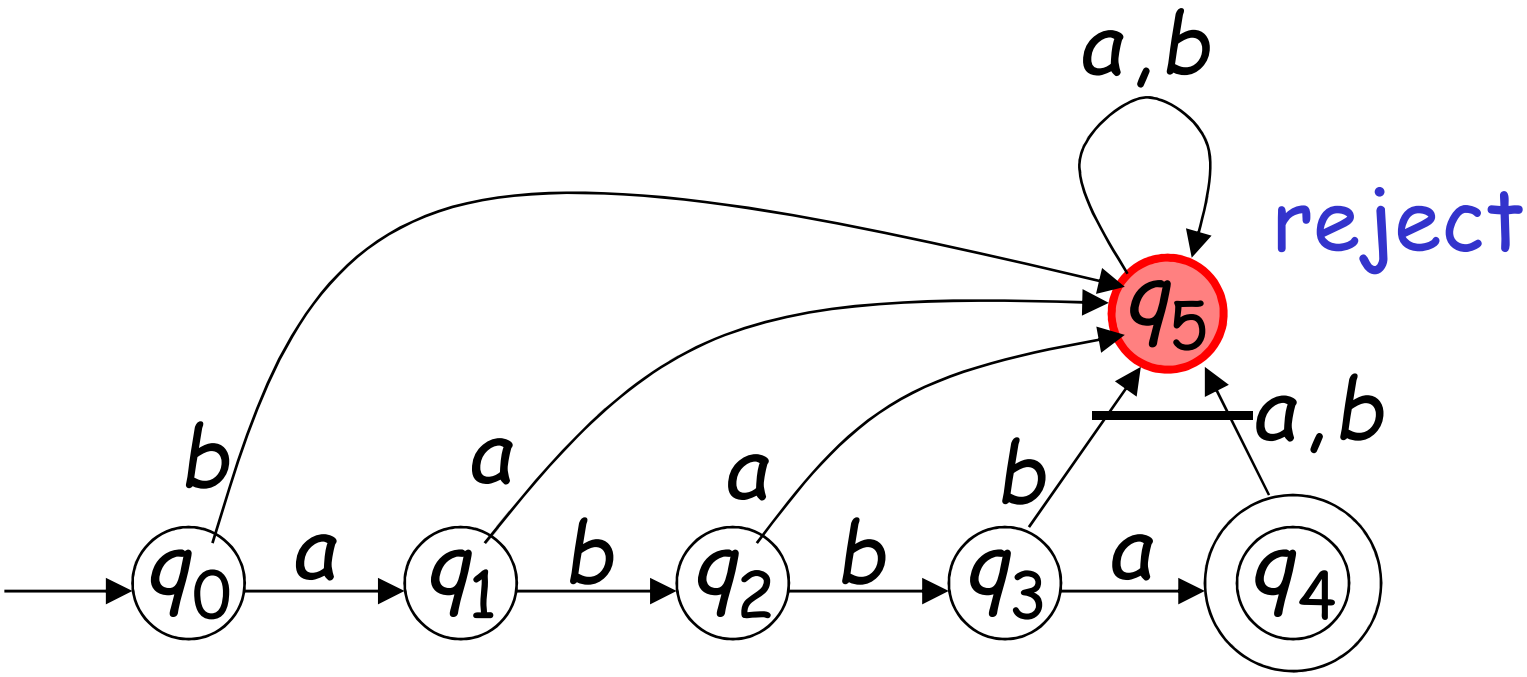

a, b
q5
b
a
a
b
a, b
q0
a
q1
b
q2
b
q3
a
q4


a
b
a

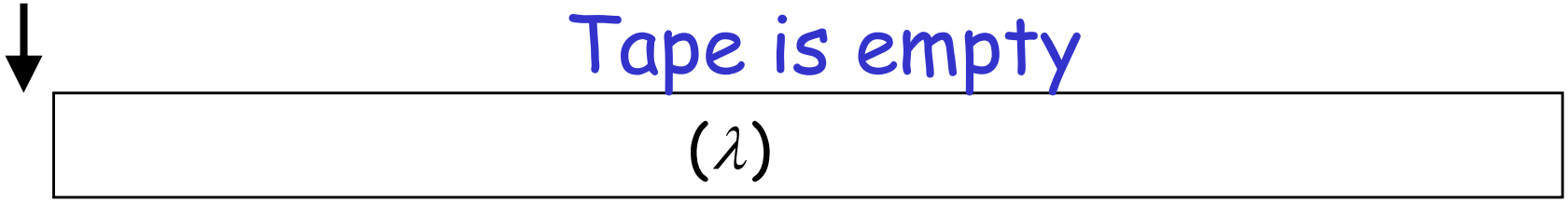

a, b
q5
b
a
a
b
a, b
q0
a
q1
b
q2
b
q3
a
q4

Input finished

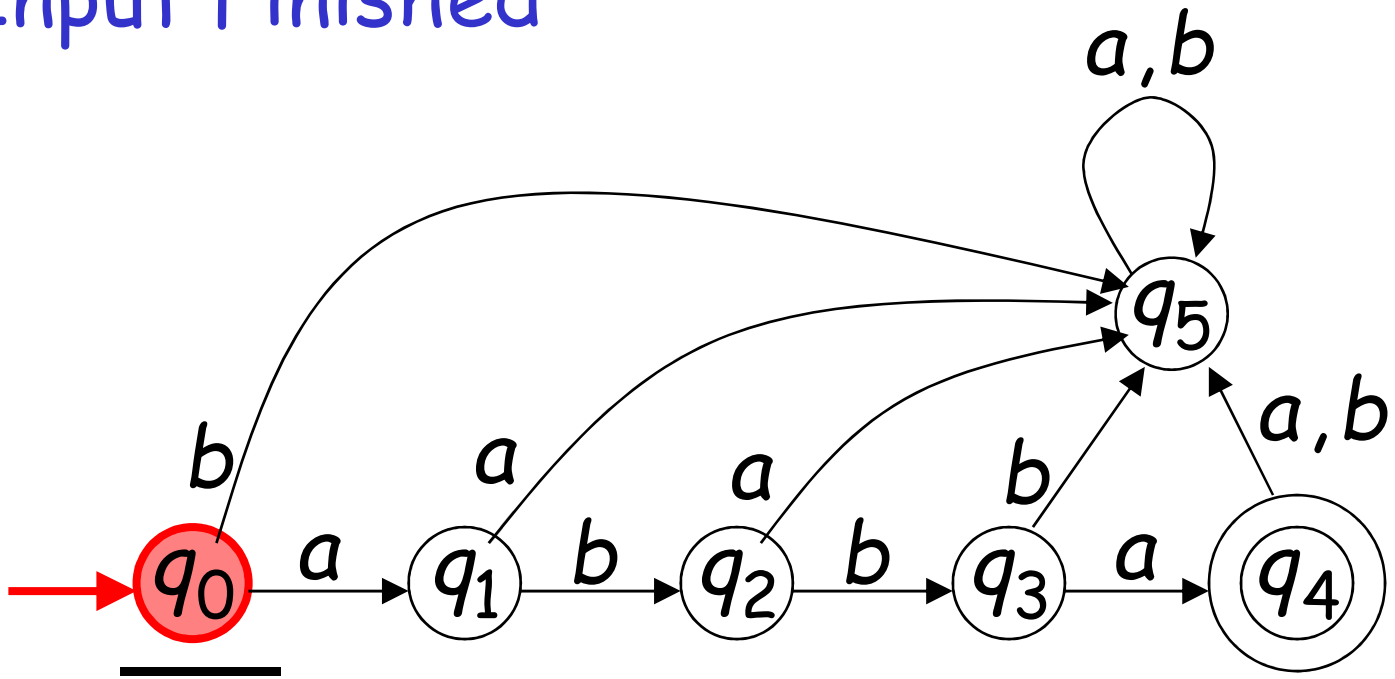




Another Rejection Case

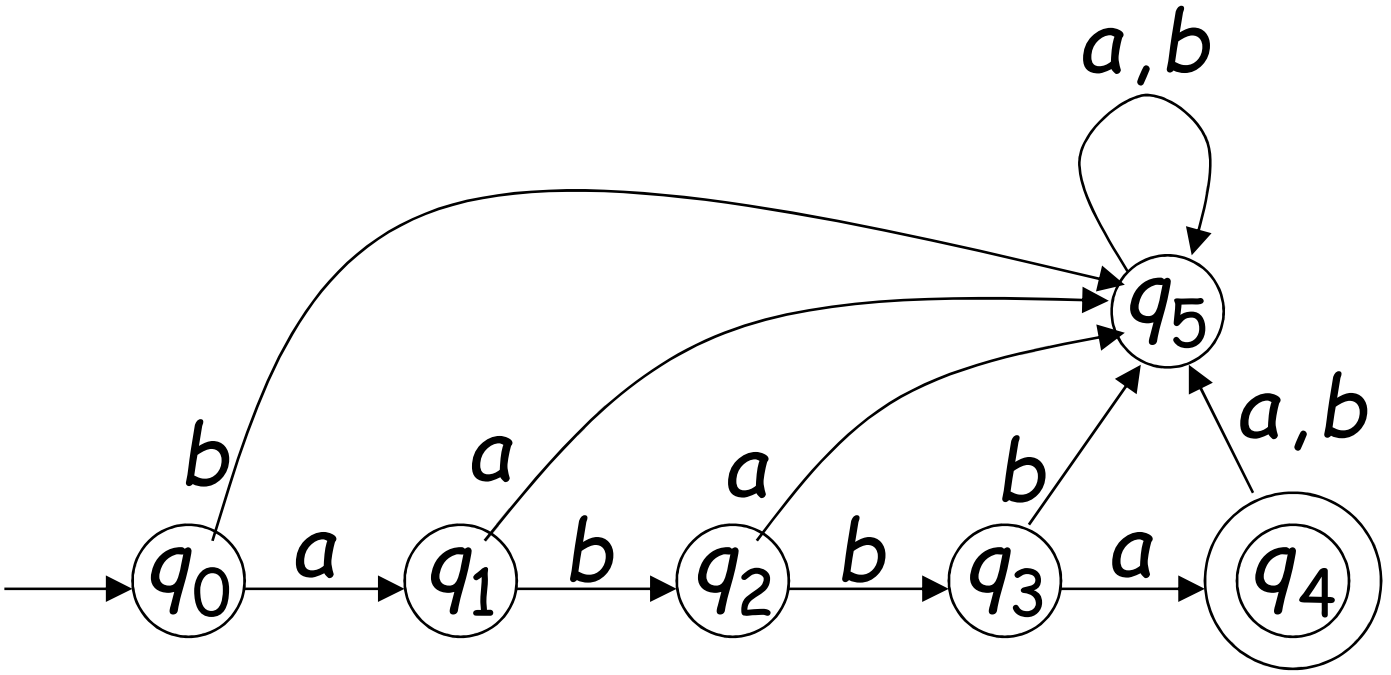


Input Finished



reject

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



To accept a string:

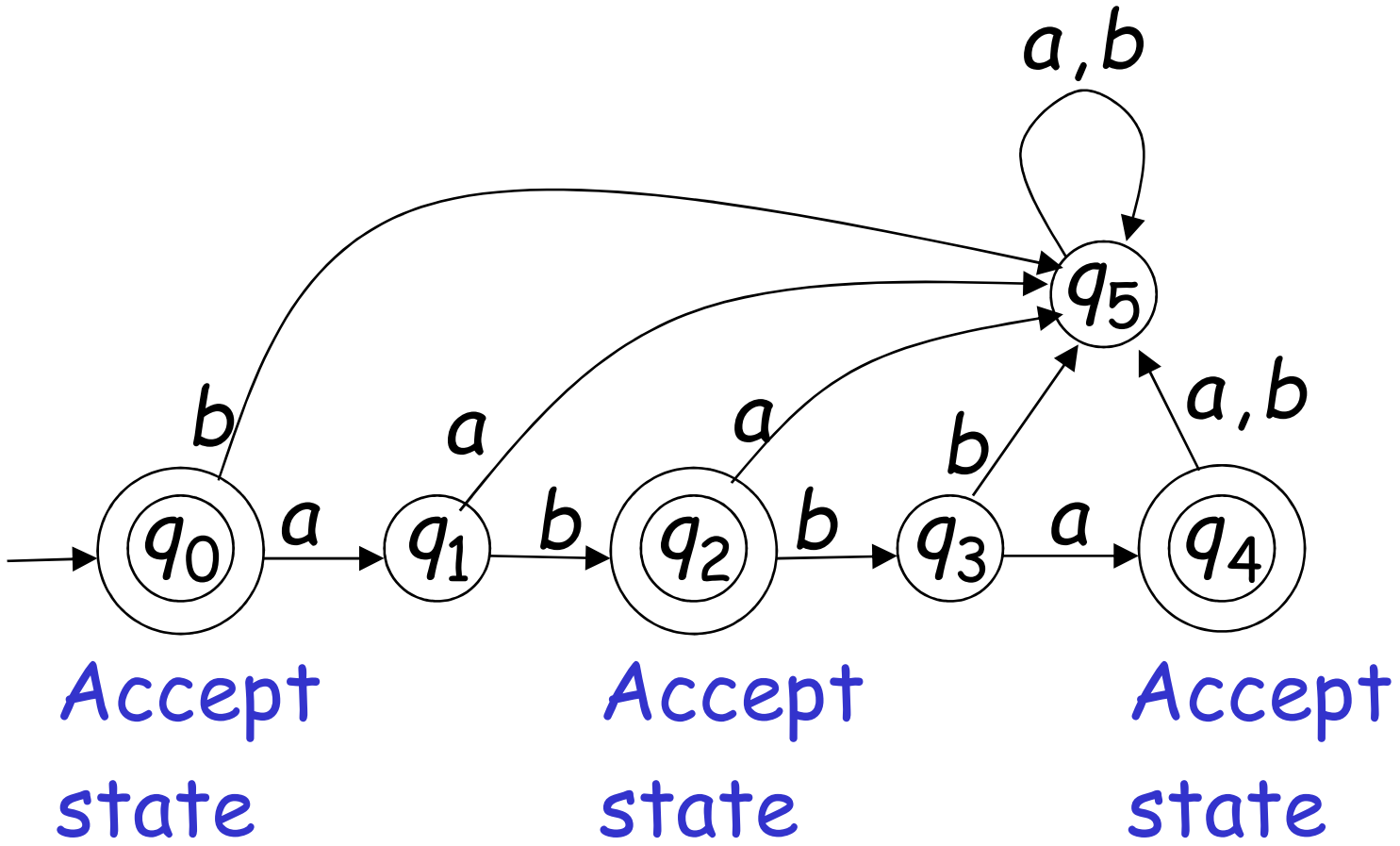
all the input string is scanned
and the last state is accepting

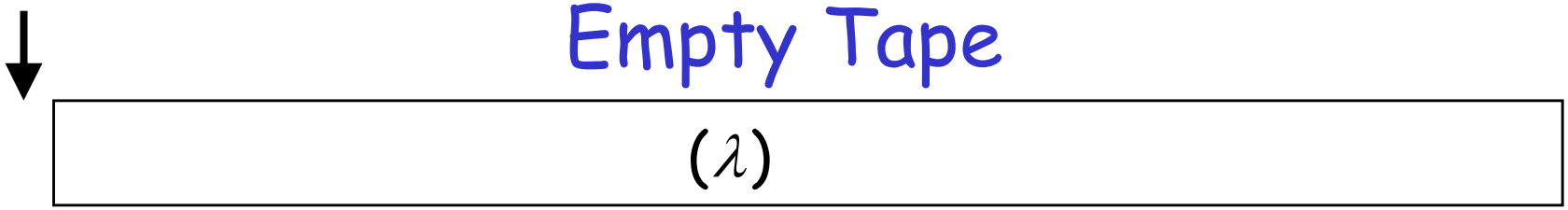
To reject a string:

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

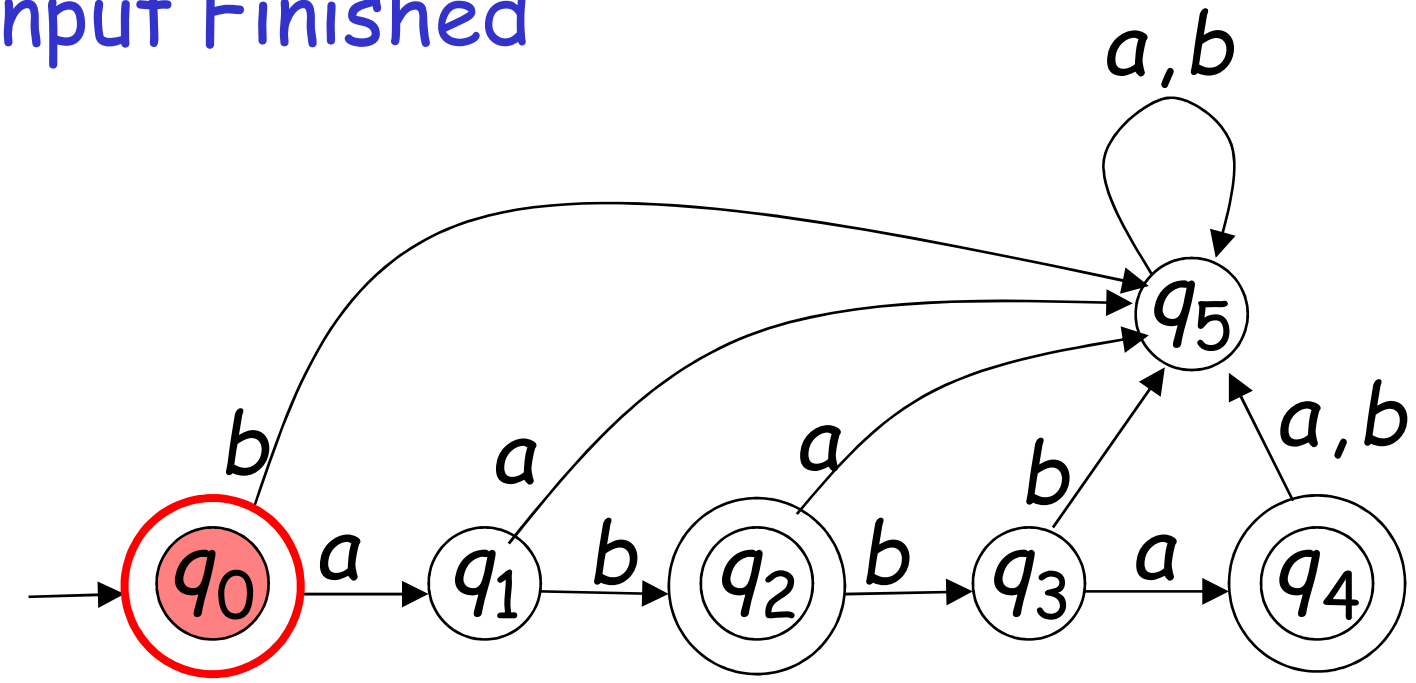
Another Example

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



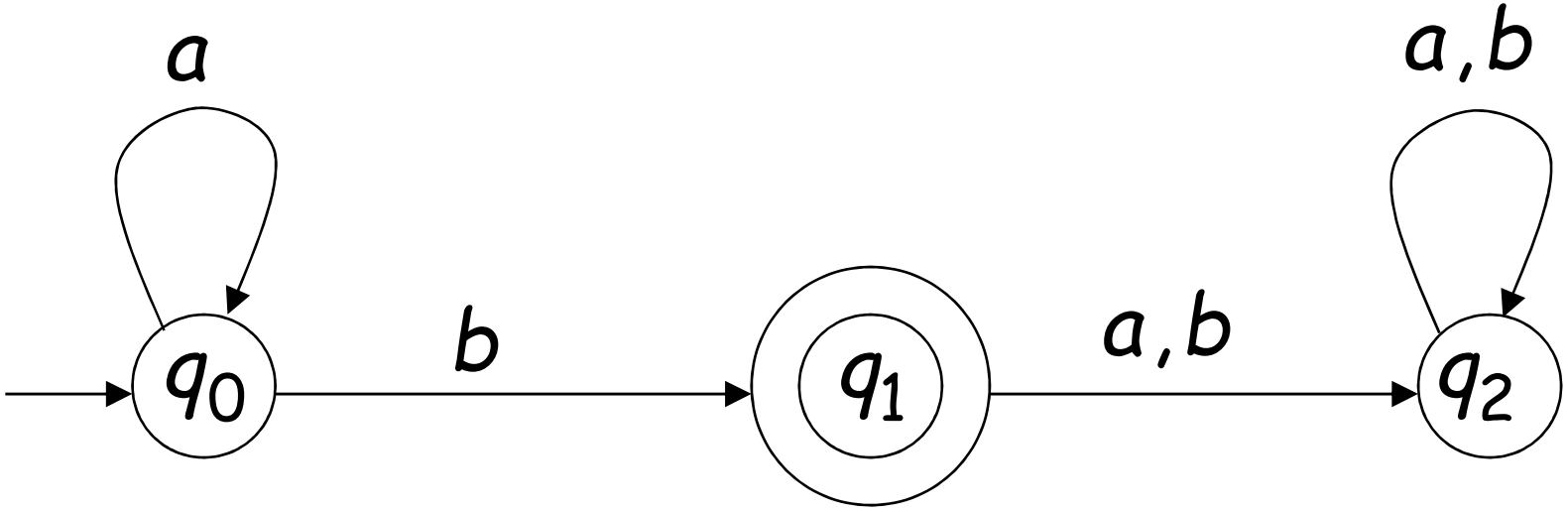


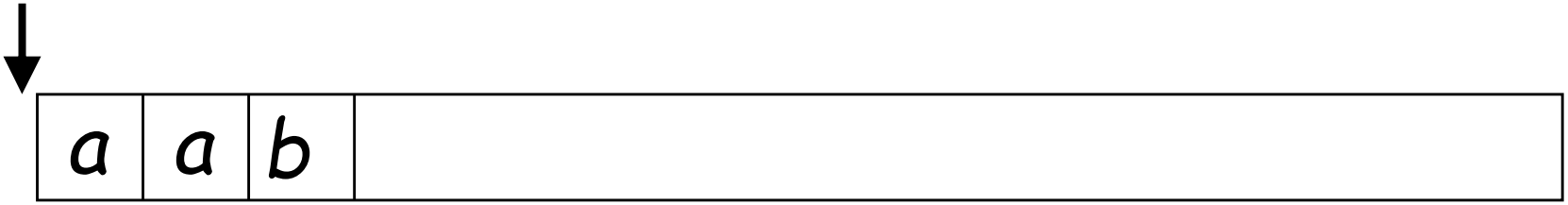
Input Finished



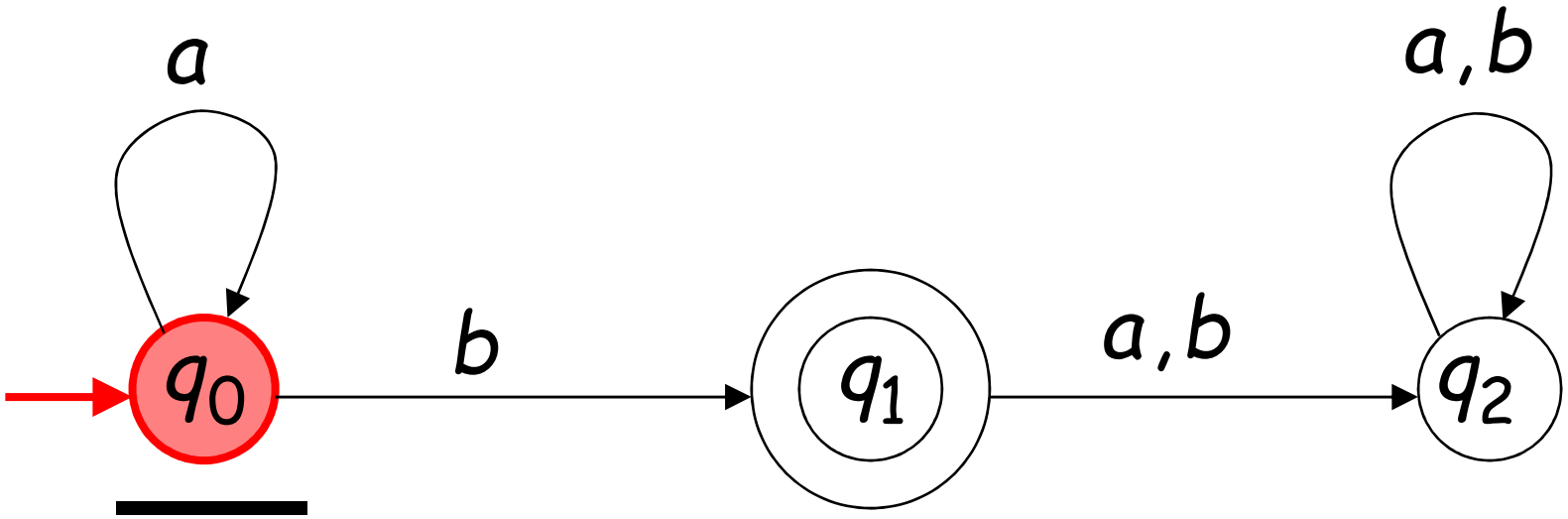
accept

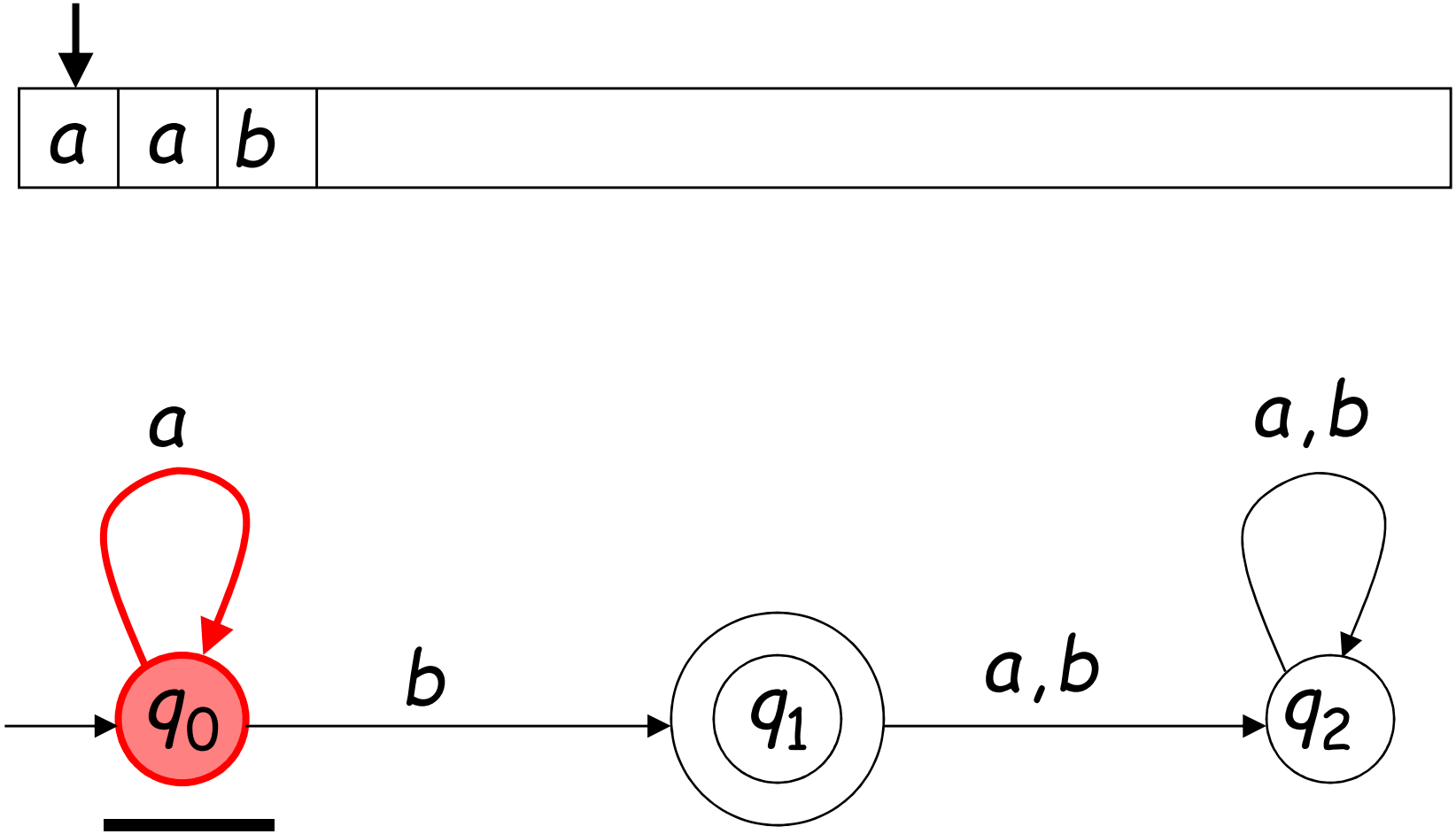
Another Example

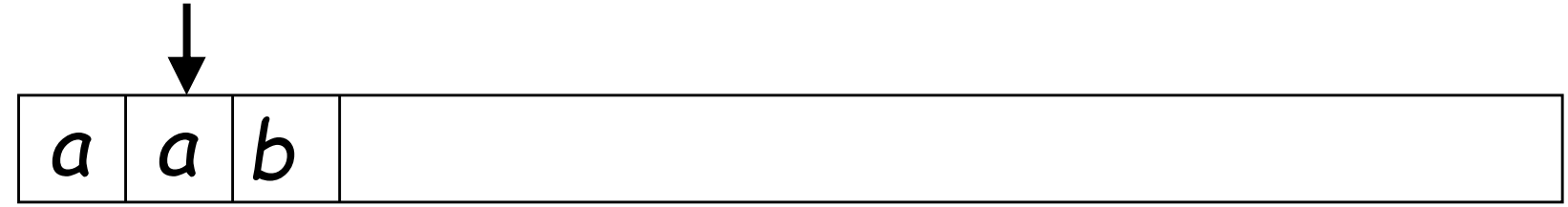


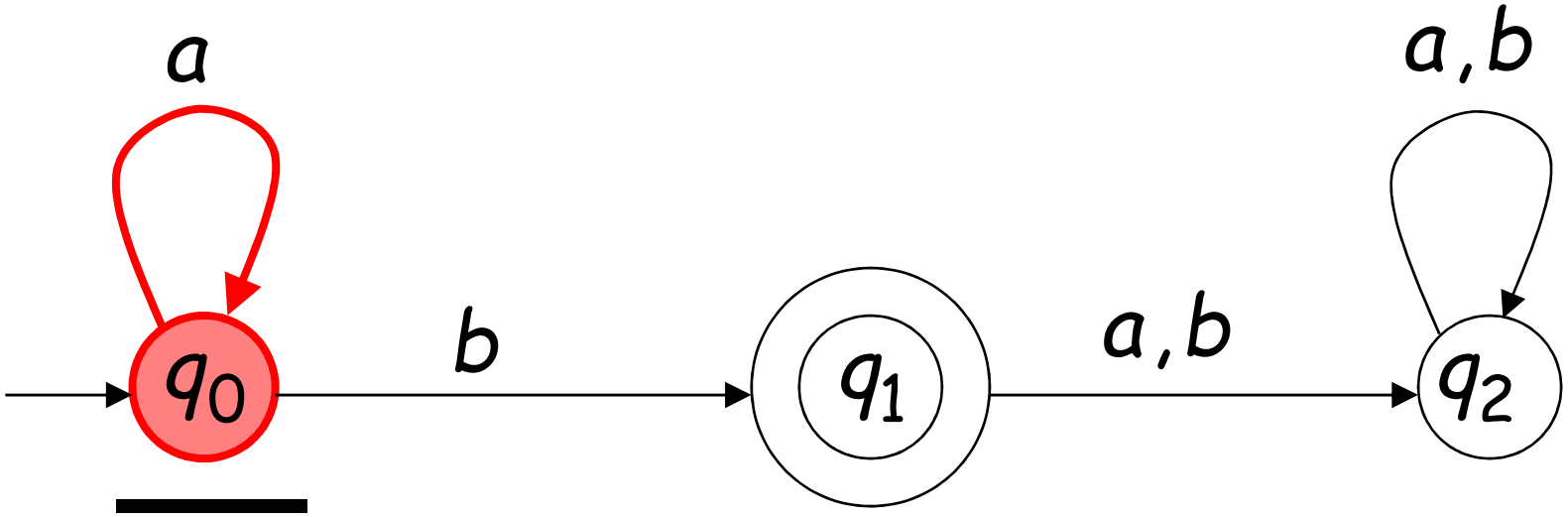


Input String

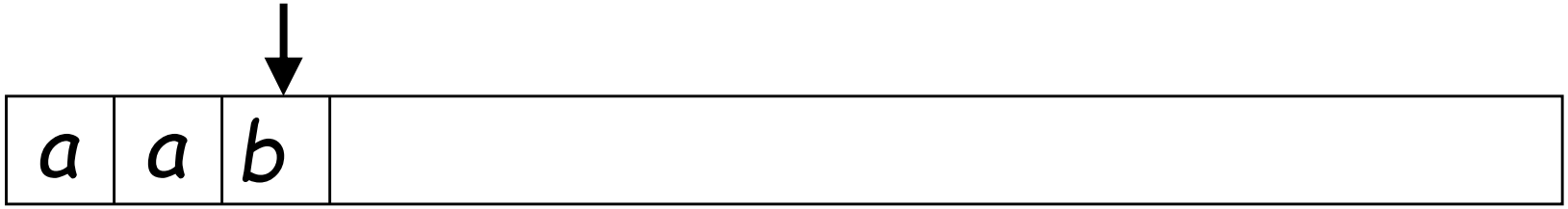


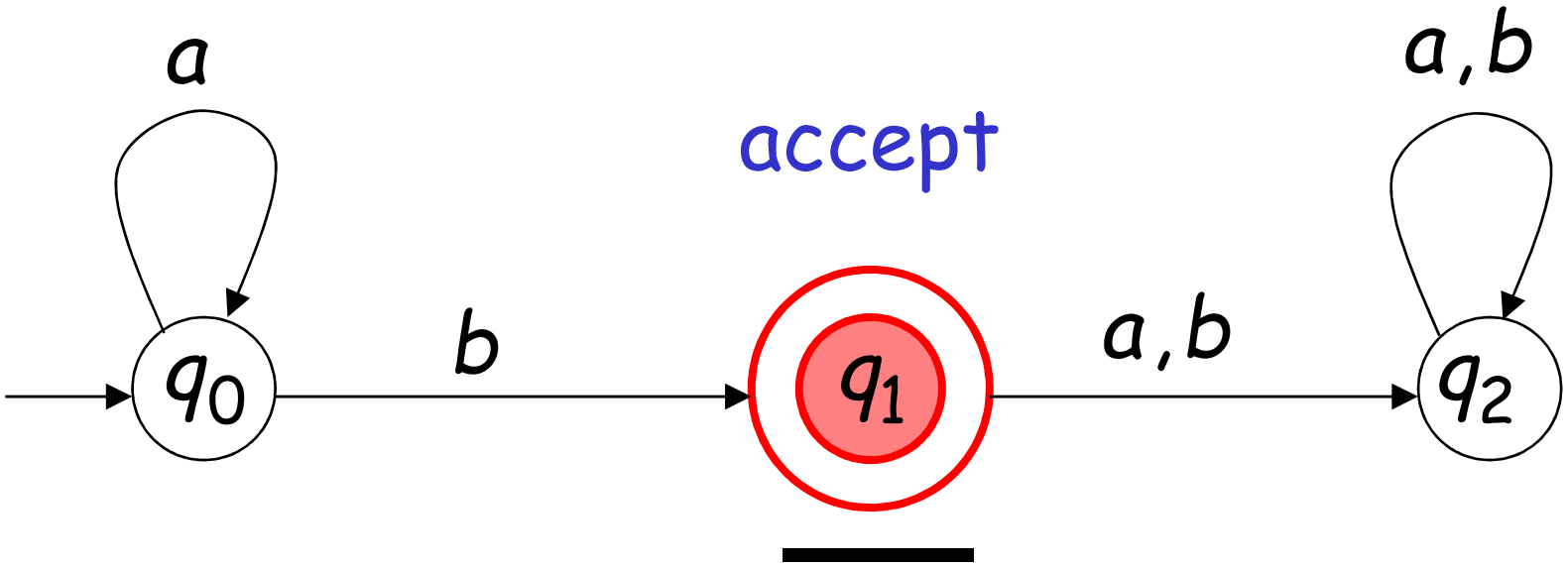

a
a
b
a
a,b
b
a,b
q_0
q_1
q_2


a
a
b

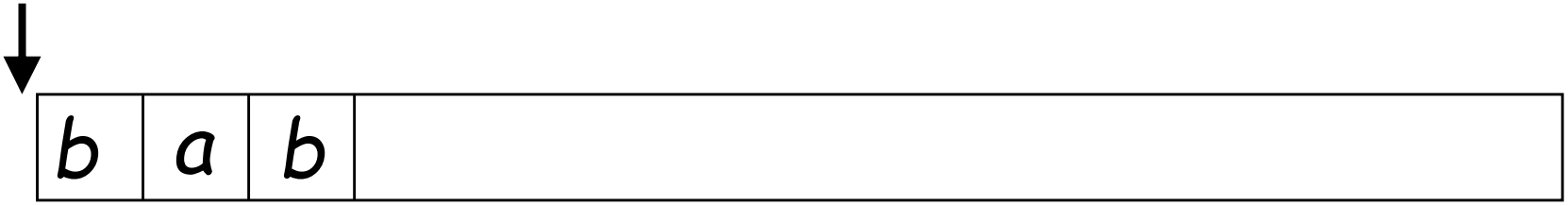

a
a,b
b
a,b
q_0
q_1
q_2

Input finished

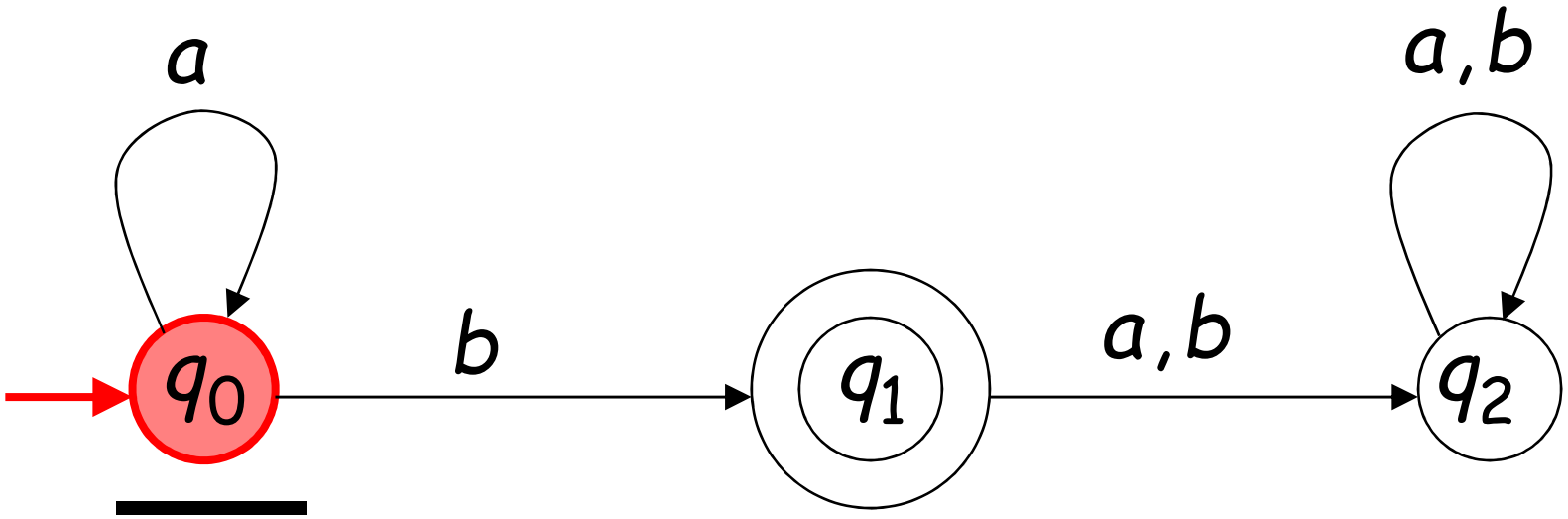


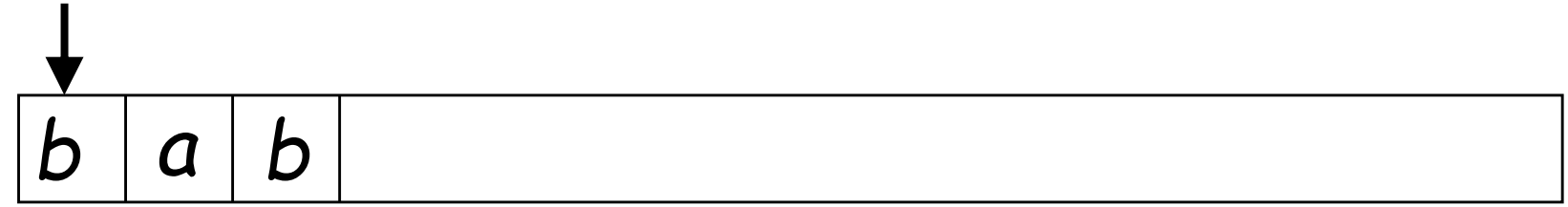


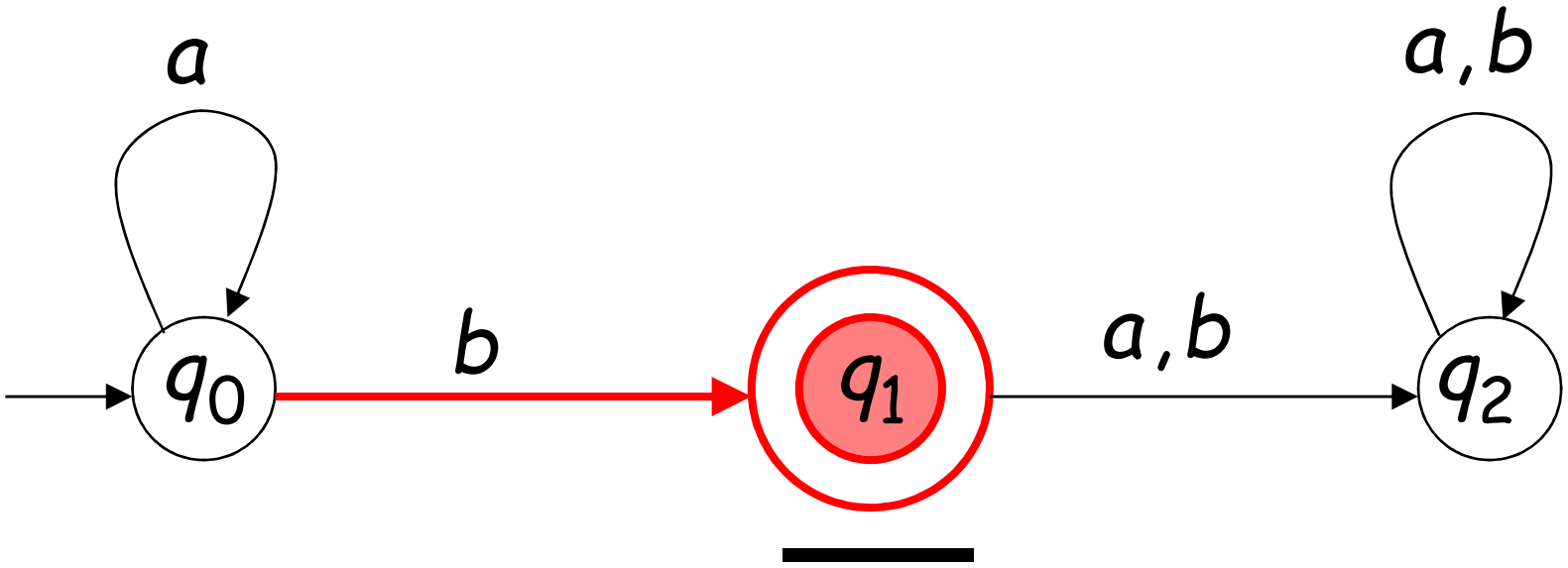
A rejection case

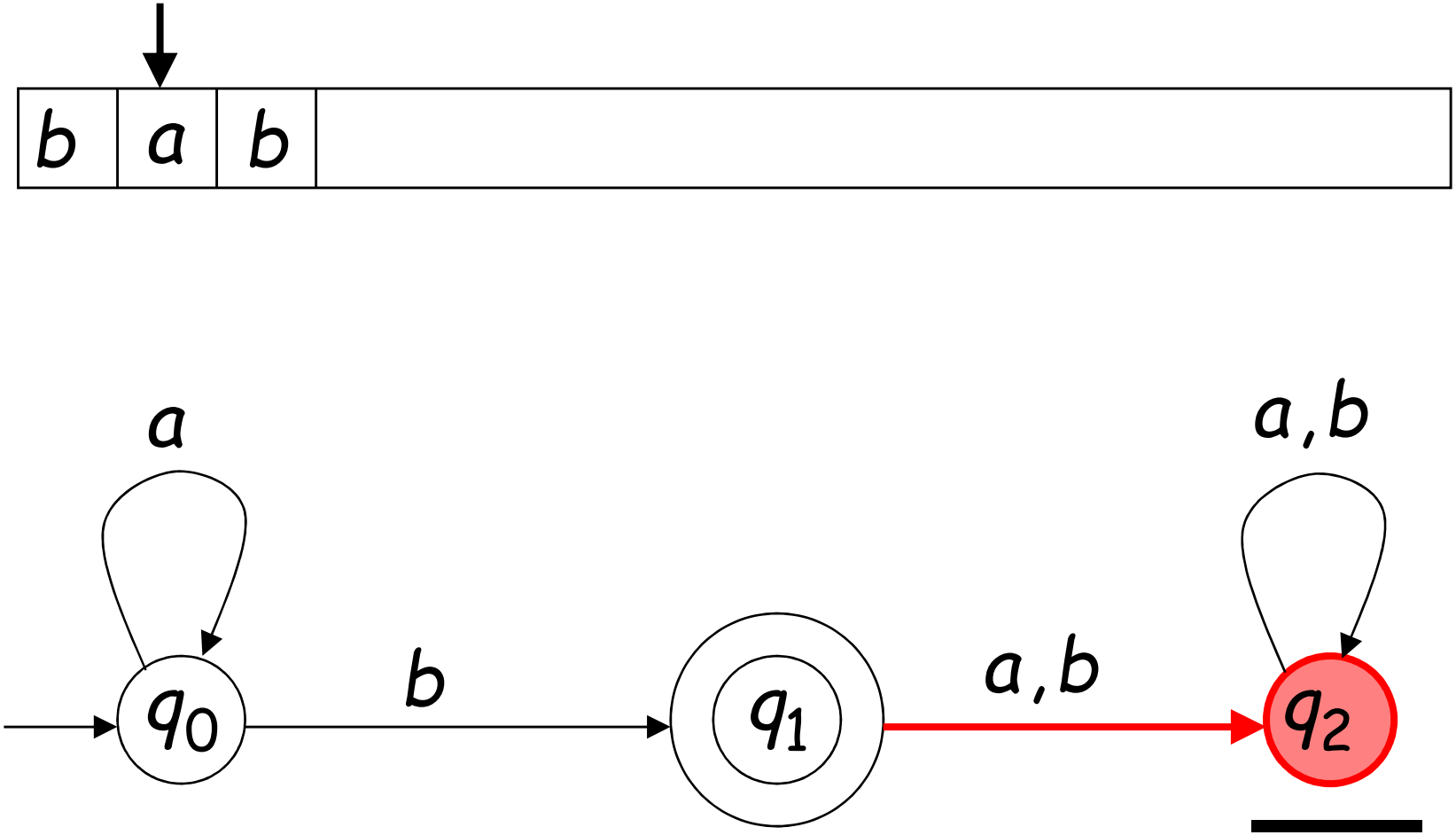


Input String

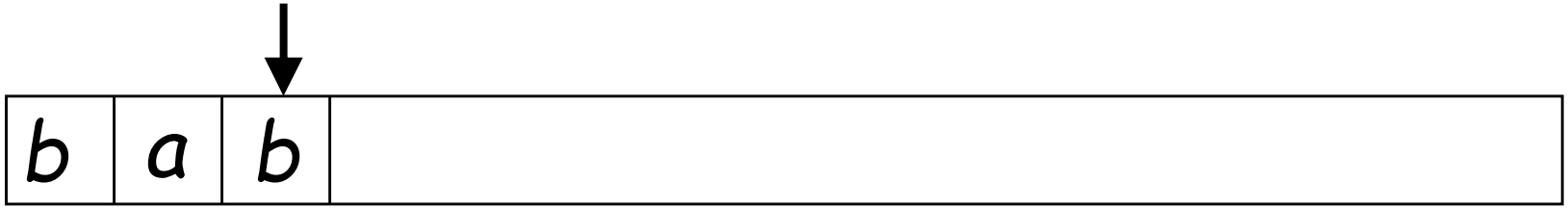


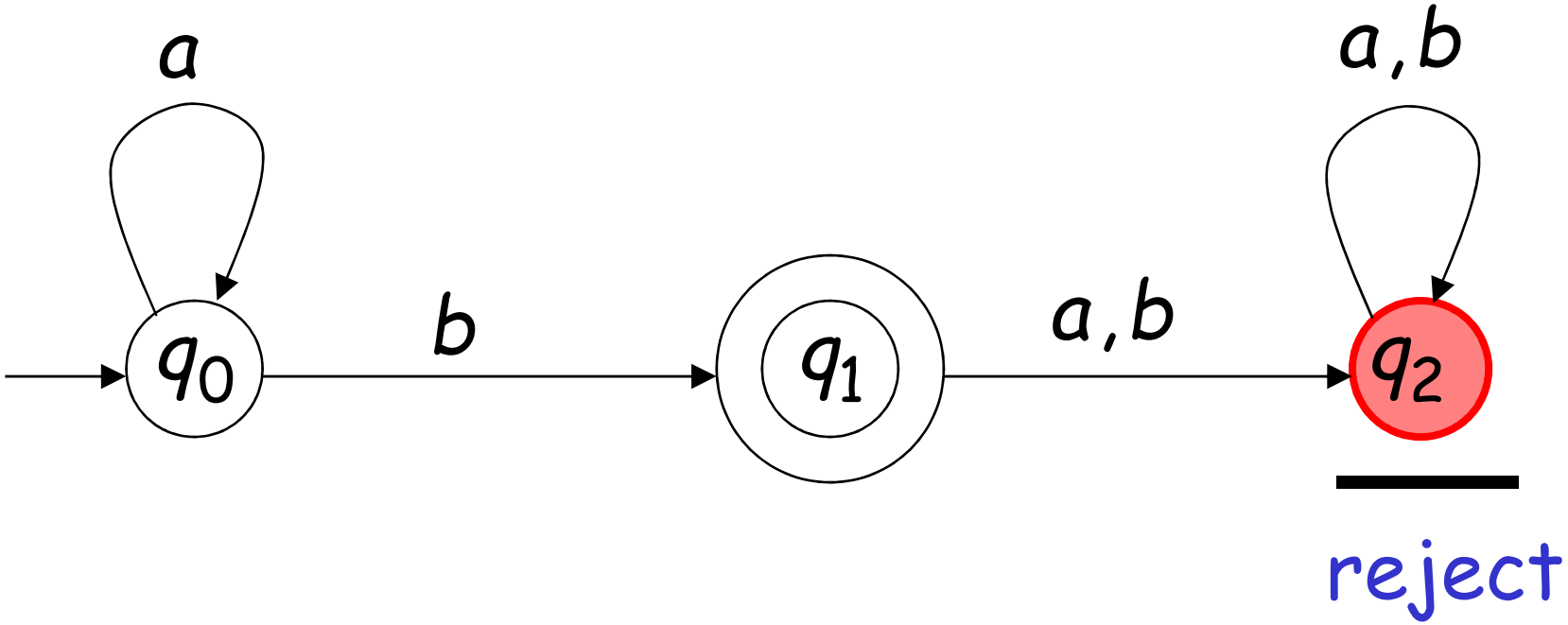

b
a
b


a
a,b
b
a,b
q_0
q_1
q_2

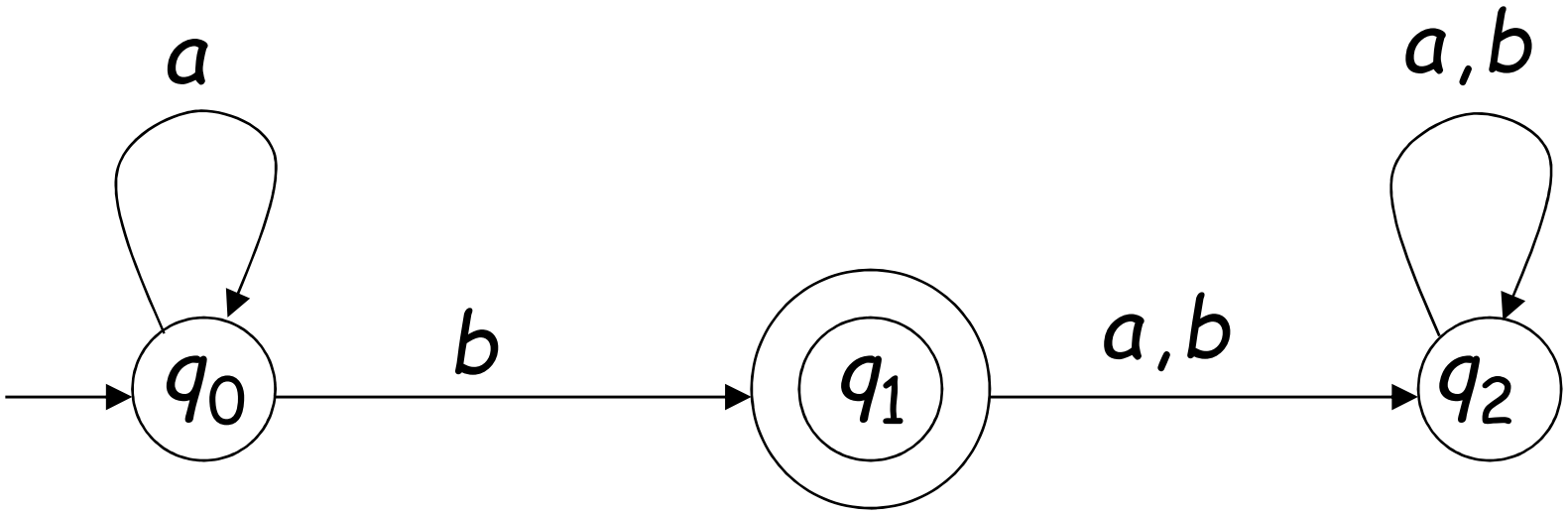

b
a
b
a
a,b
q0
b
q1
a,b
q2

Input finished



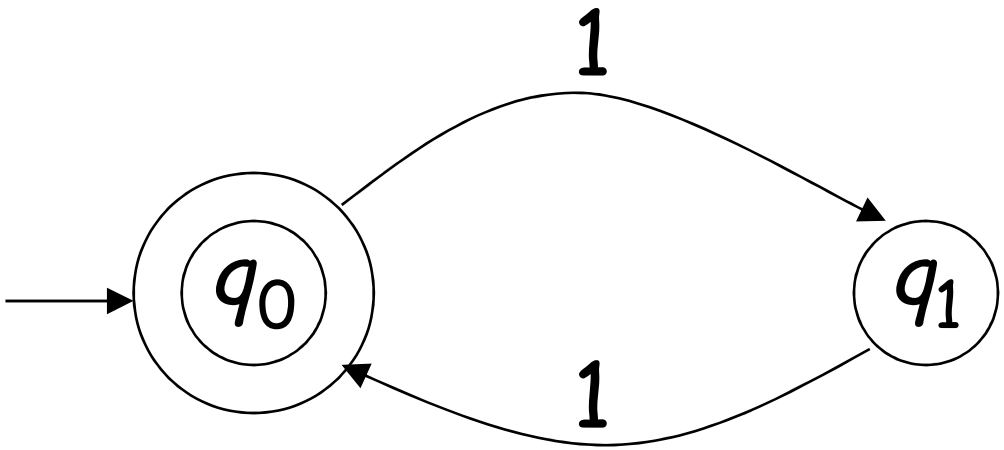


Language Accepted: $L = \{a^n b : n \geq 0\}$



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 $Q \times \Sigma \Longrightarrow Q$

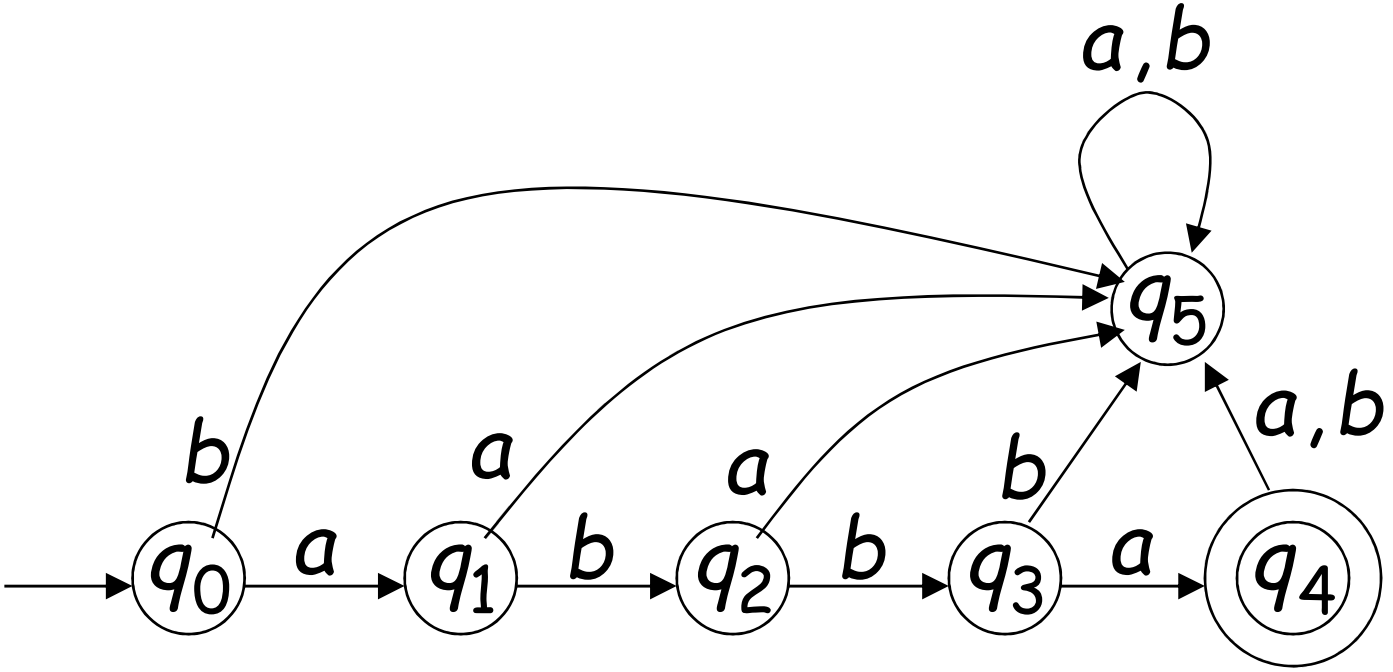
q_0 : initial state

F : set of accepting states

Set of States Q

Example

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

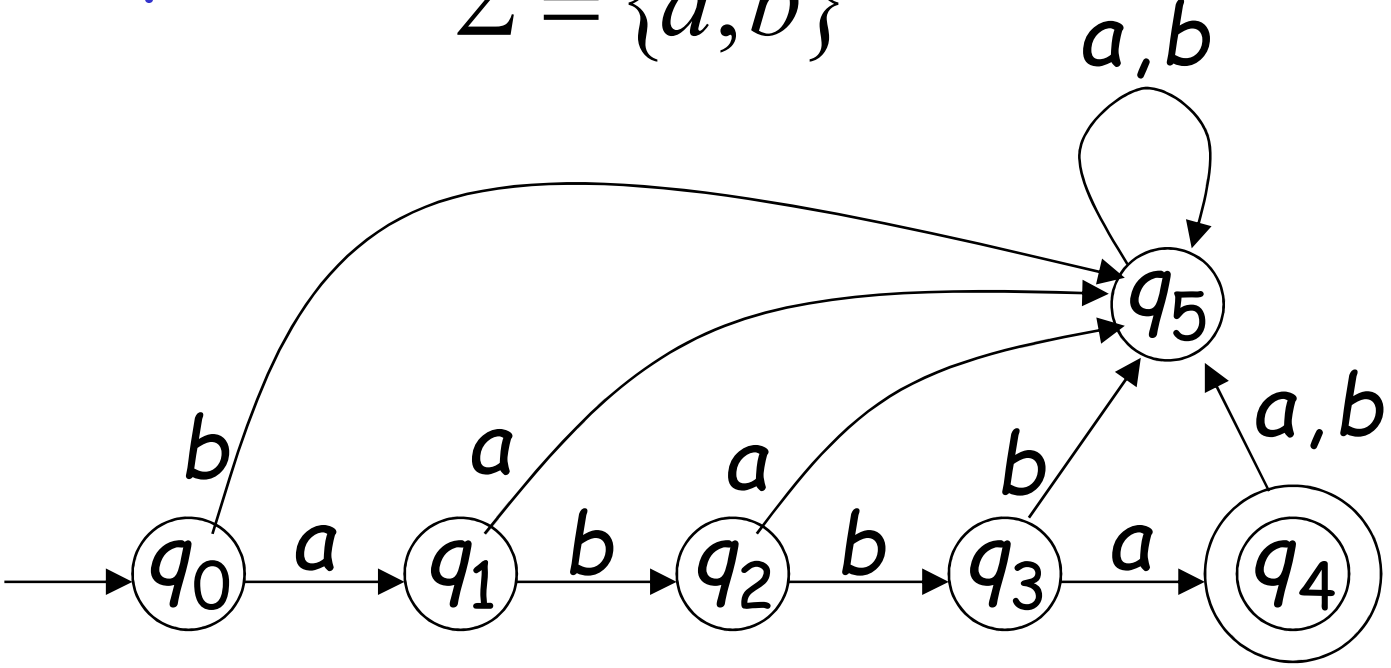


Input Alphabet Σ

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

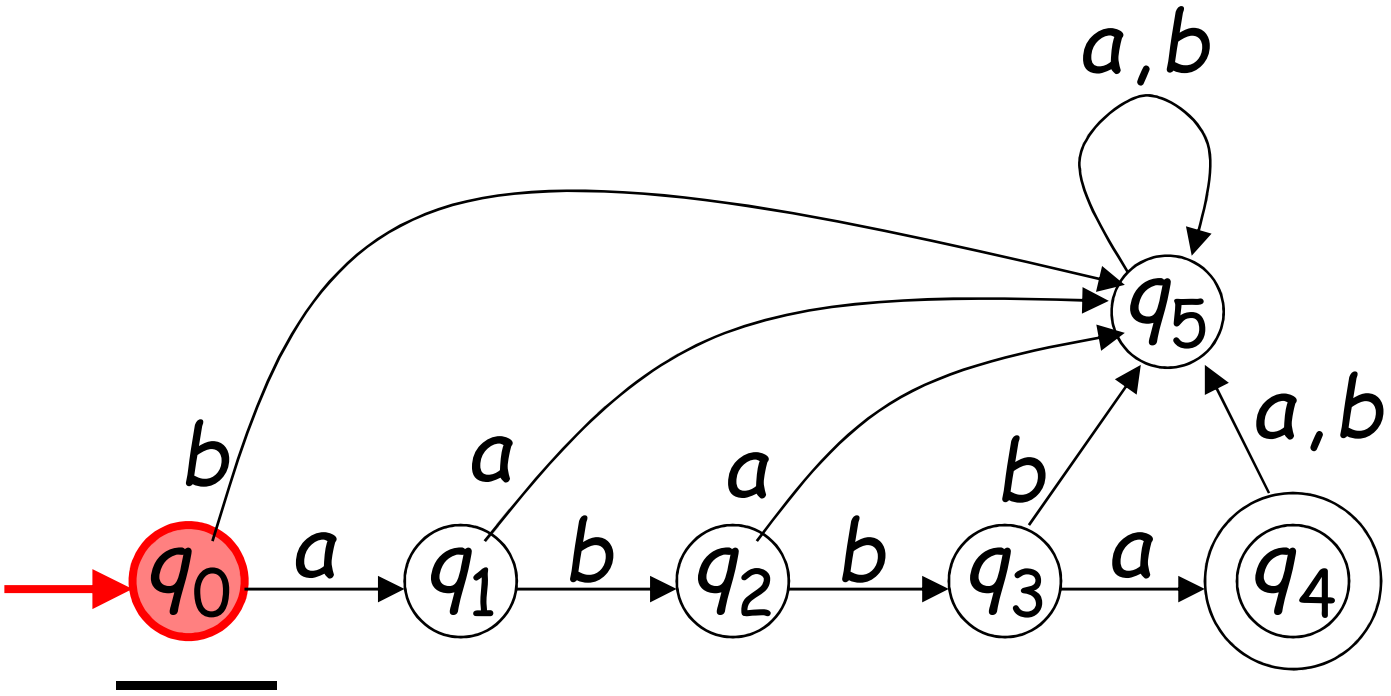
Example

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



Initial State q_0

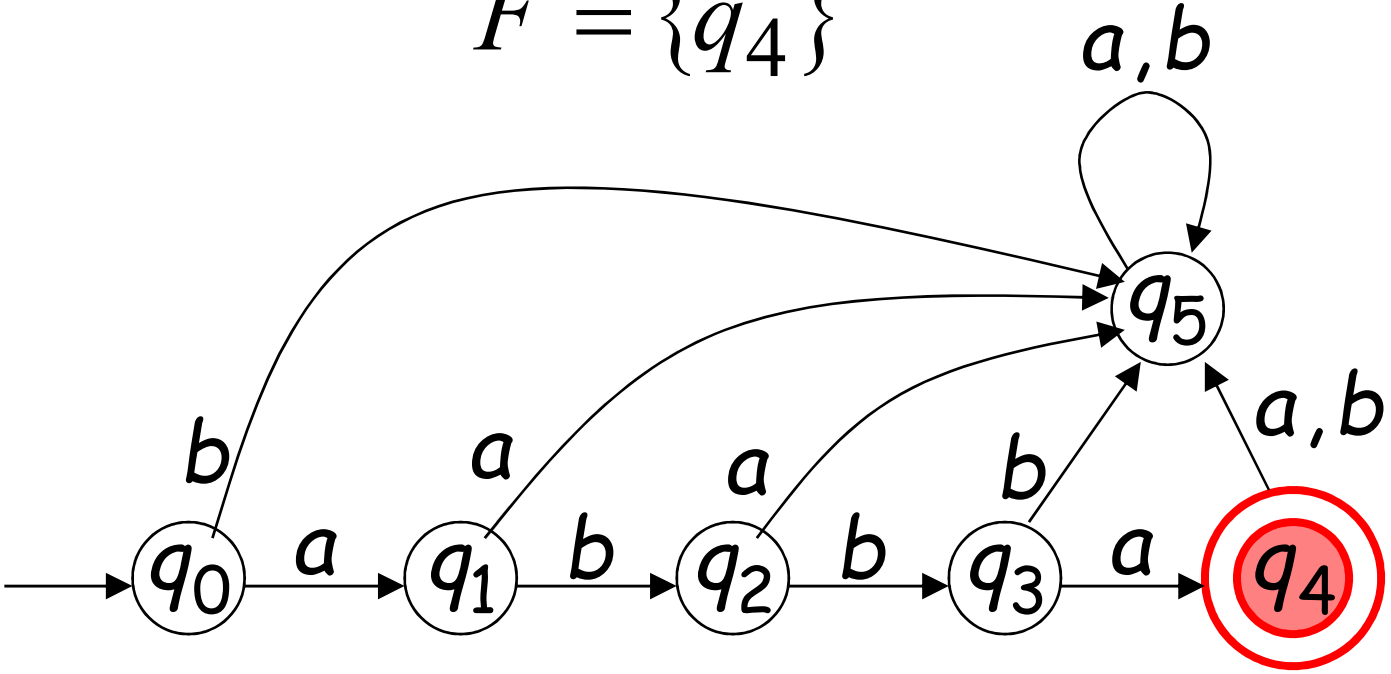
Example



Set of Accepting States $F \subseteq Q$

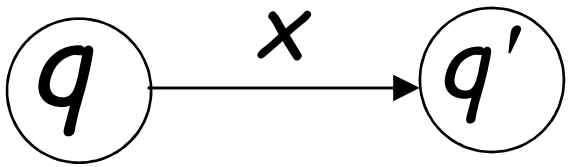
Example

$$F = \{q_4\}$$



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

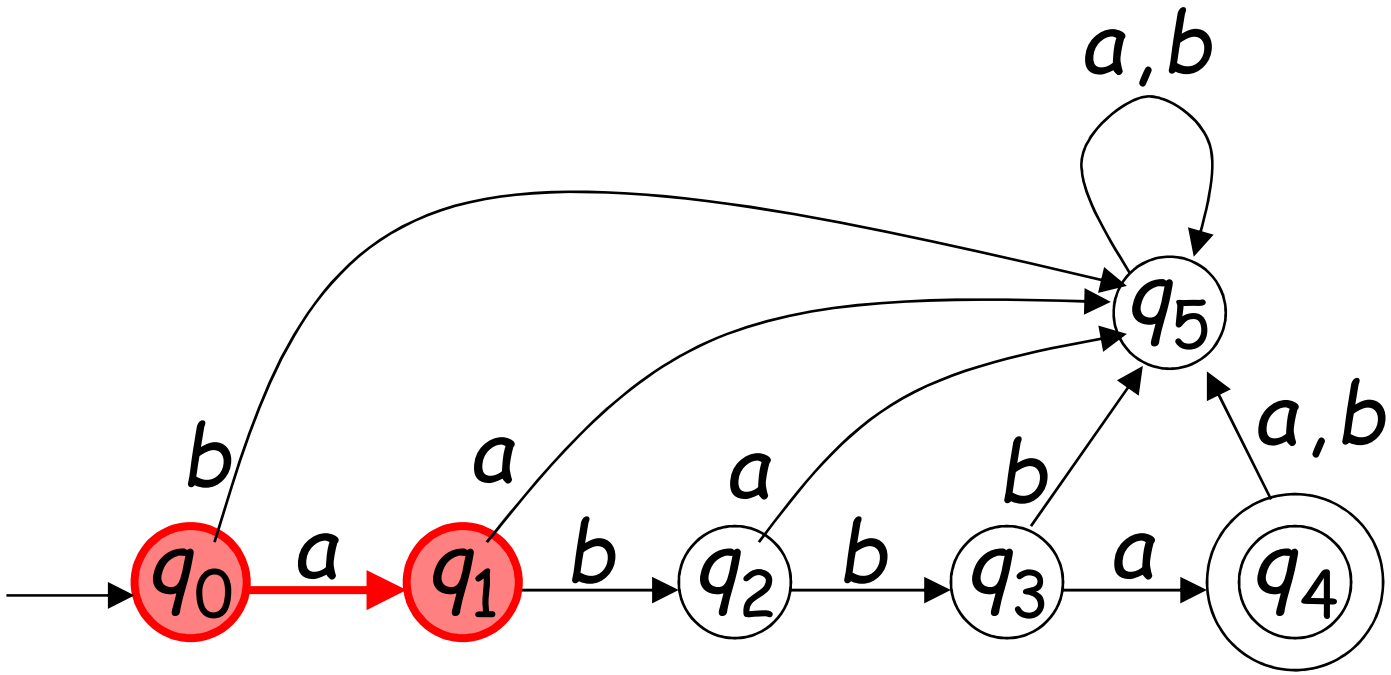
$$\delta(q, x) = q'$$



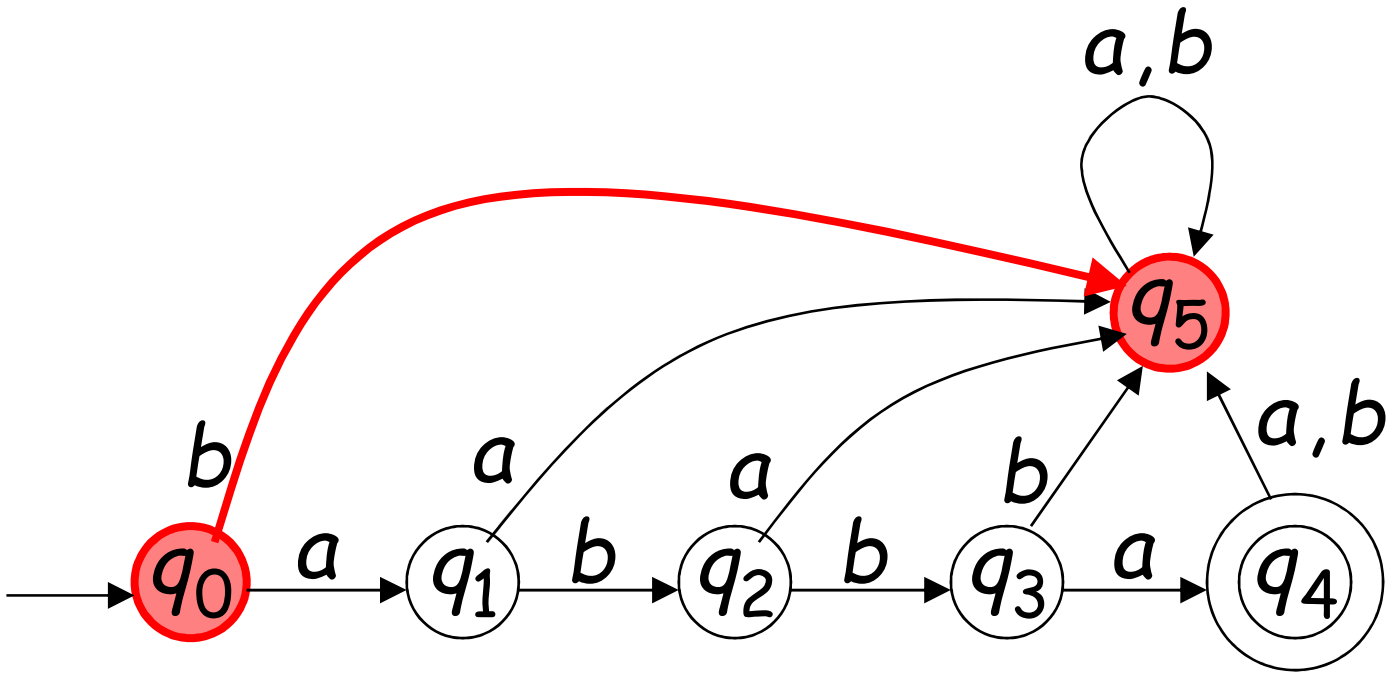
Describes the result of a transition from state q with symbol x

Example:

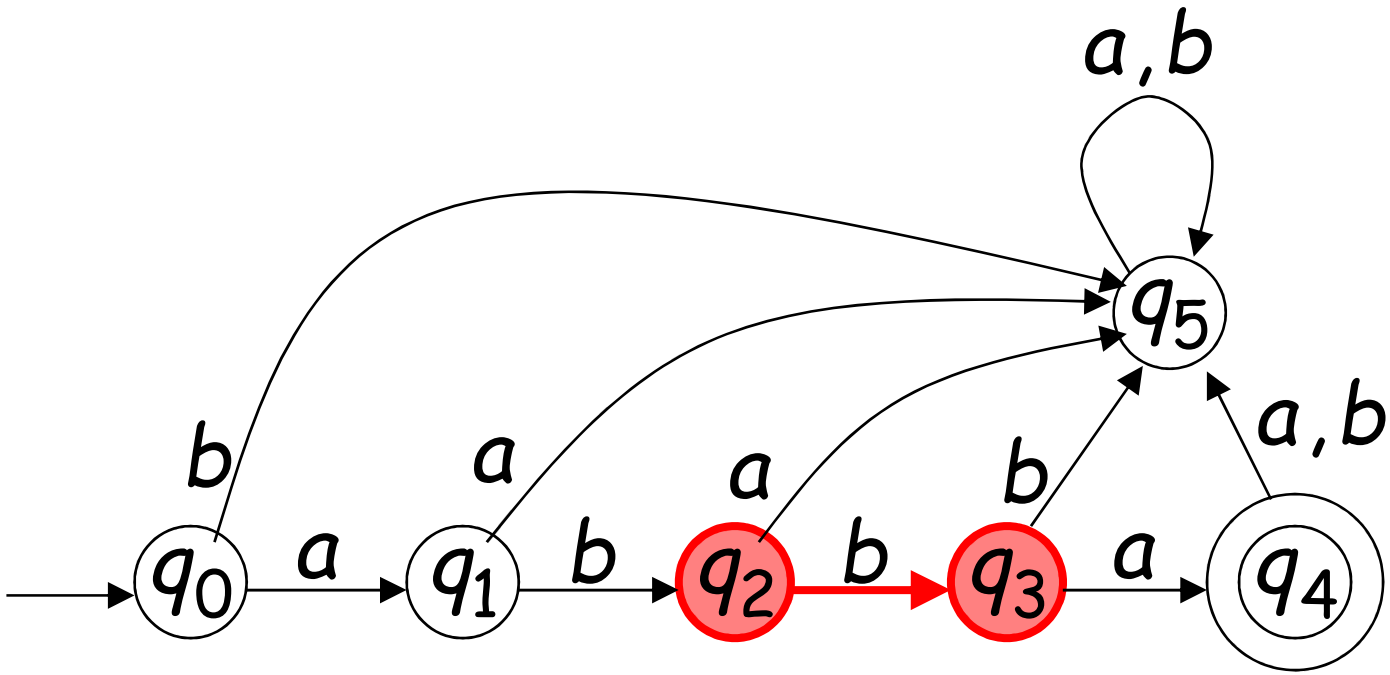
$$\delta(q_0, a) = q_1$$



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



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

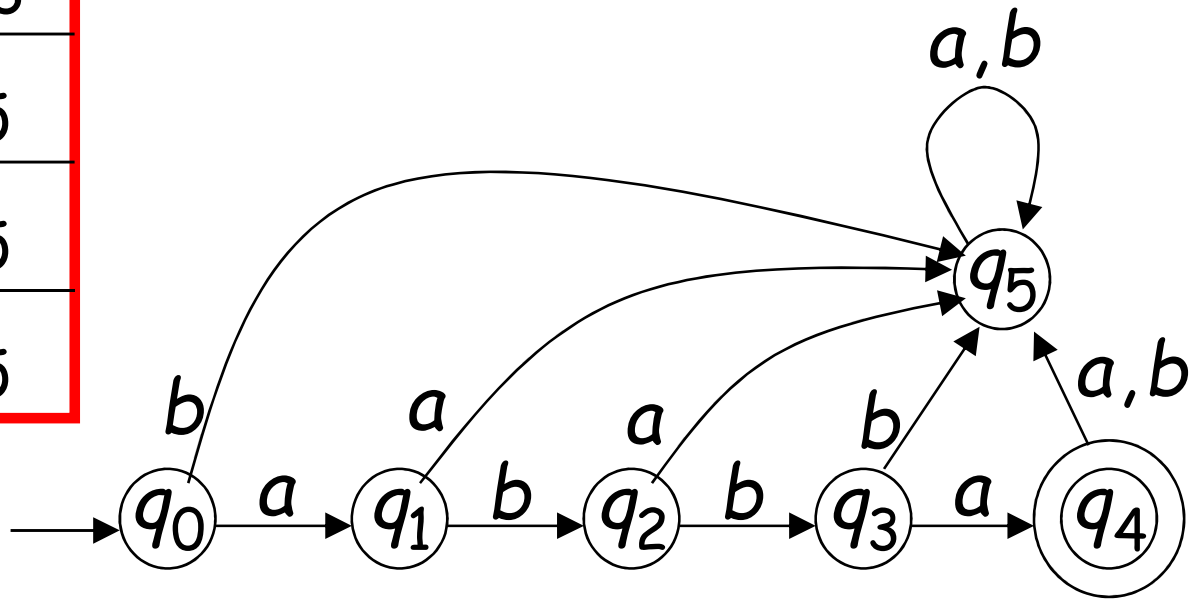


Transition Table for δ

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



What does a DFA do on reading an input string?

- Input: a word w in Σ^*
- Question: Is w acceptable by the DFA?
- Steps:
  - Start at the "start state" q_0
  - For every input symbol in the sequence w do
    - Compute the next state from the current state, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed, the current state is one of the accepting states (F) then *accept w;*
  - Otherwise, *reject w.*

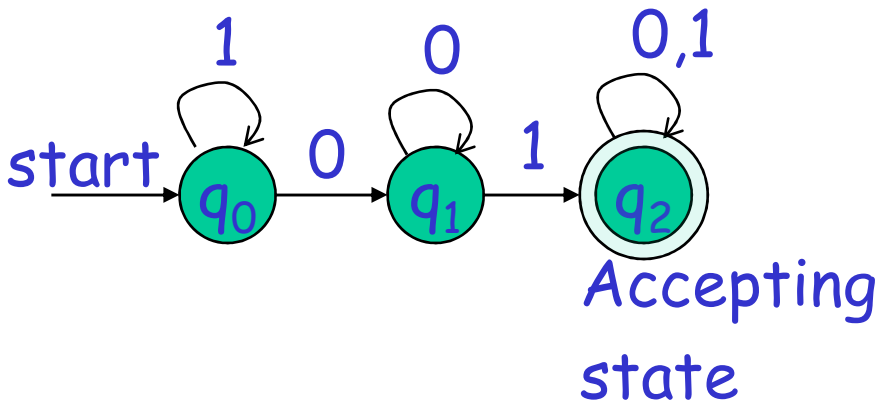
Example #1

- Build a DFA for the following language:
  - L = {w | w is a binary string that contains 01 as a substring}
- Steps for building a DFA to recognize L:
  - Σ = {0,1}
  - Decide on the states: Q
  - Designate start state and final state(s)
  - δ: Decide on the transitions:

- "Final" states == same as "accepting states"
- Other states == same as "non-accepting states"

DFA for strings containing 01

- What makes this DFA deterministic?



- What if the language allows empty strings?

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

states	δ	symbols 0	1
→	q_0	q_1	q_0
	q_1	q_1	q_2
	$*q_2$	q_2	q_2

Example #2

Clamping Logic:

- A clamping circuit waits for a "1" input, and turns on forever. However, to avoid clamping on spurious noise, we'll design a DFA that waits for *two consecutive 1s* in a row before clamping on.

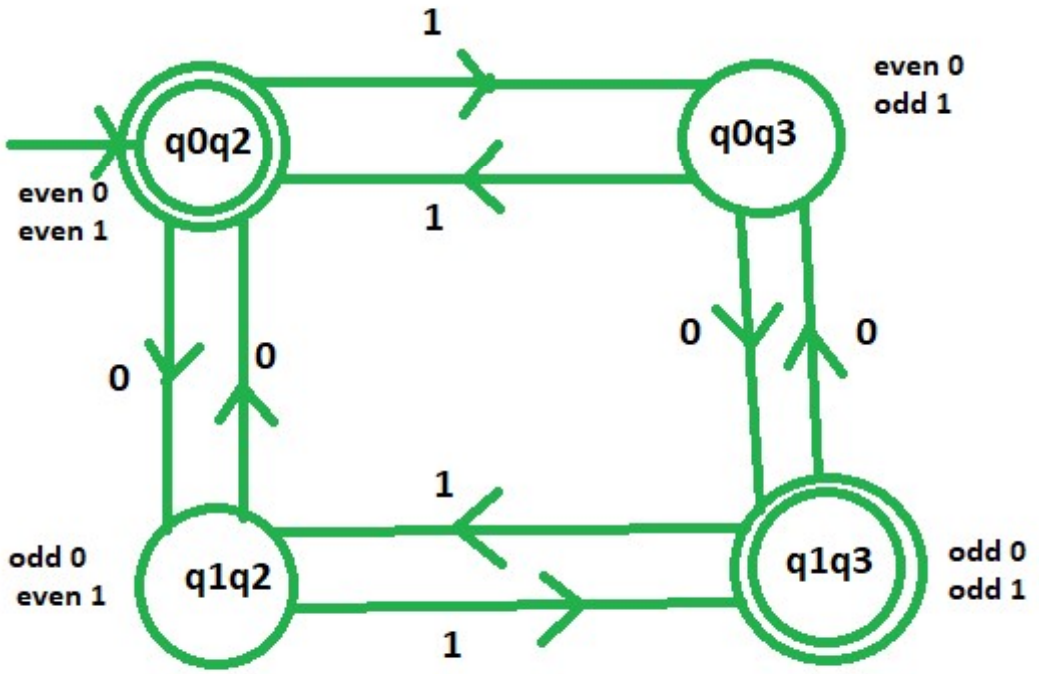
- Build a DFA for the following language:
  L = { w | w is a bit string which contains the substring 11}

- State Design:
  - q_0 : start state (initially off), also means the most recent input was not a 1
  - q_1: has never seen 11 but the most recent input was a 1
  - q_2: has seen 11 at least once

Example #3

Build a DFA for the following language:

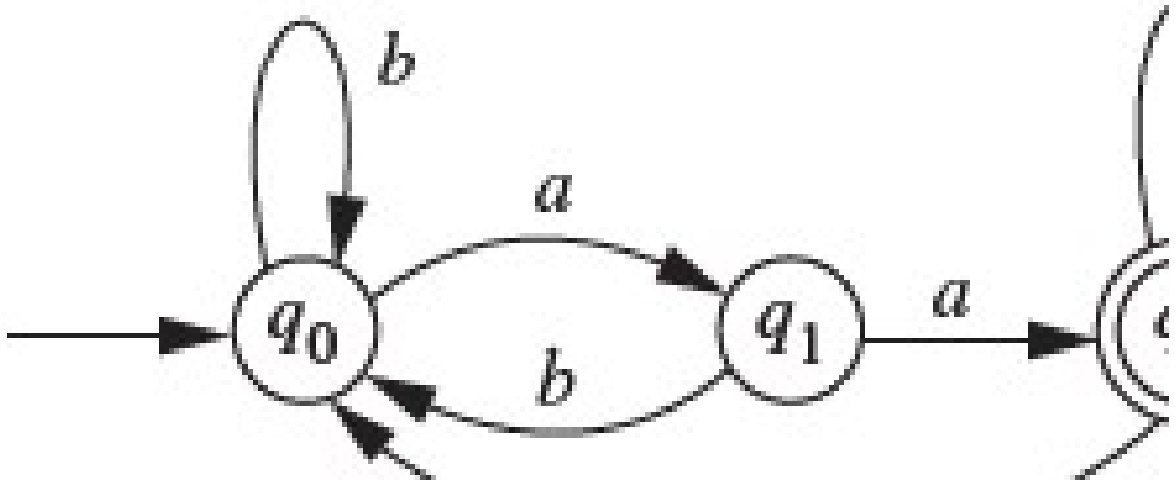
L = { w | w is a binary string that has even number of 1s and even number of 0s}



DFA

Example #4

L = { x ∈ {a,b}* / x ends with aa}



Extended Transition Function (for acceptance of string)

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

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

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

Describes the resulting state
after scanning string **w** from state **q**

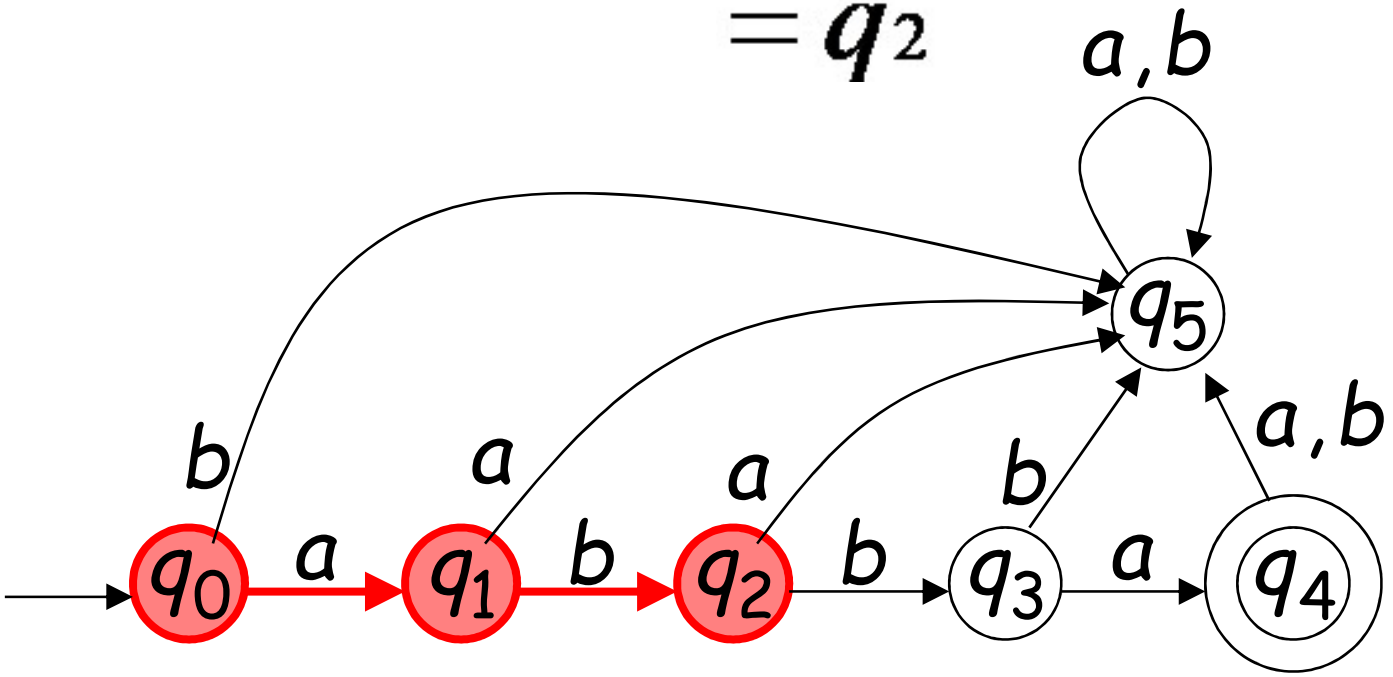
$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$

Extended Transition Function

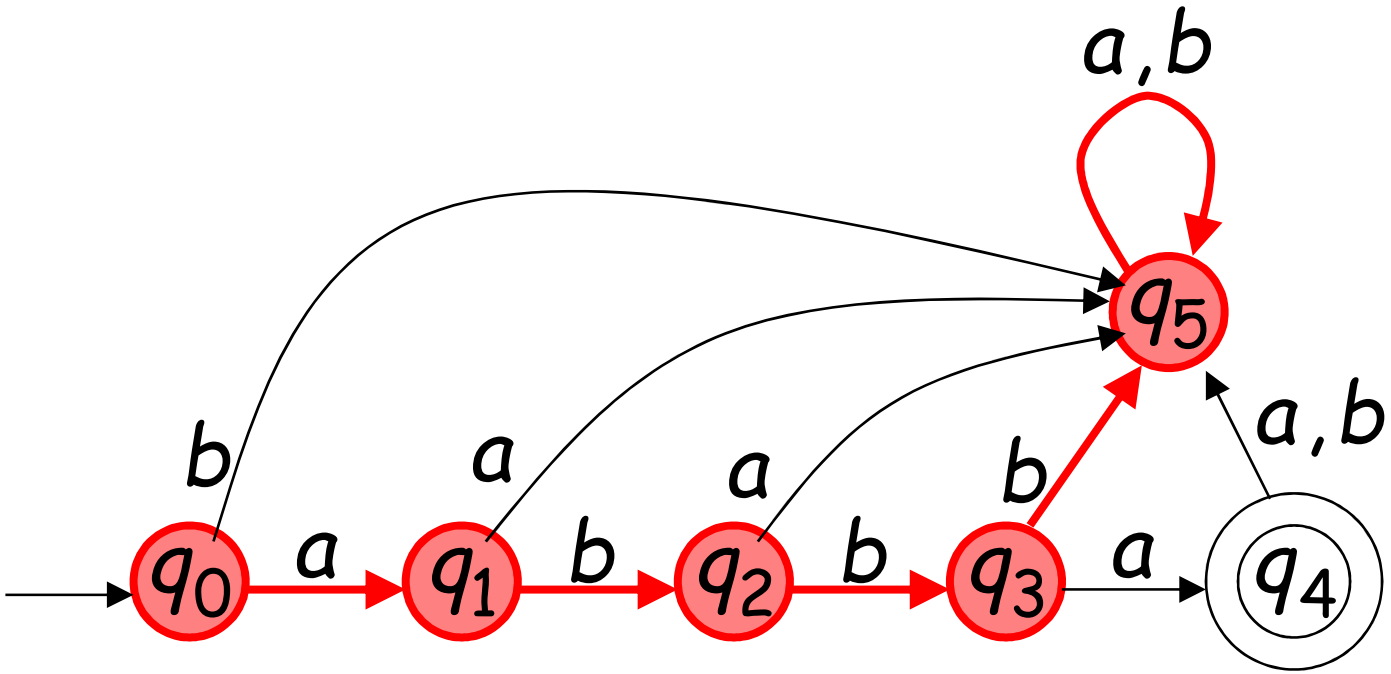
- δ may be extended to handle strings
- This gives a function δ^*: Q x Σ^* –> Q
- δ^* is defined for all states **q**, strings
  symbols **a** by
  - $\delta^*(q,\lambda) = q$
  - $\delta^*(q,wa) = \delta(\delta^*(q,w), a)$
- Note that δ^* extends δ in the sense

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

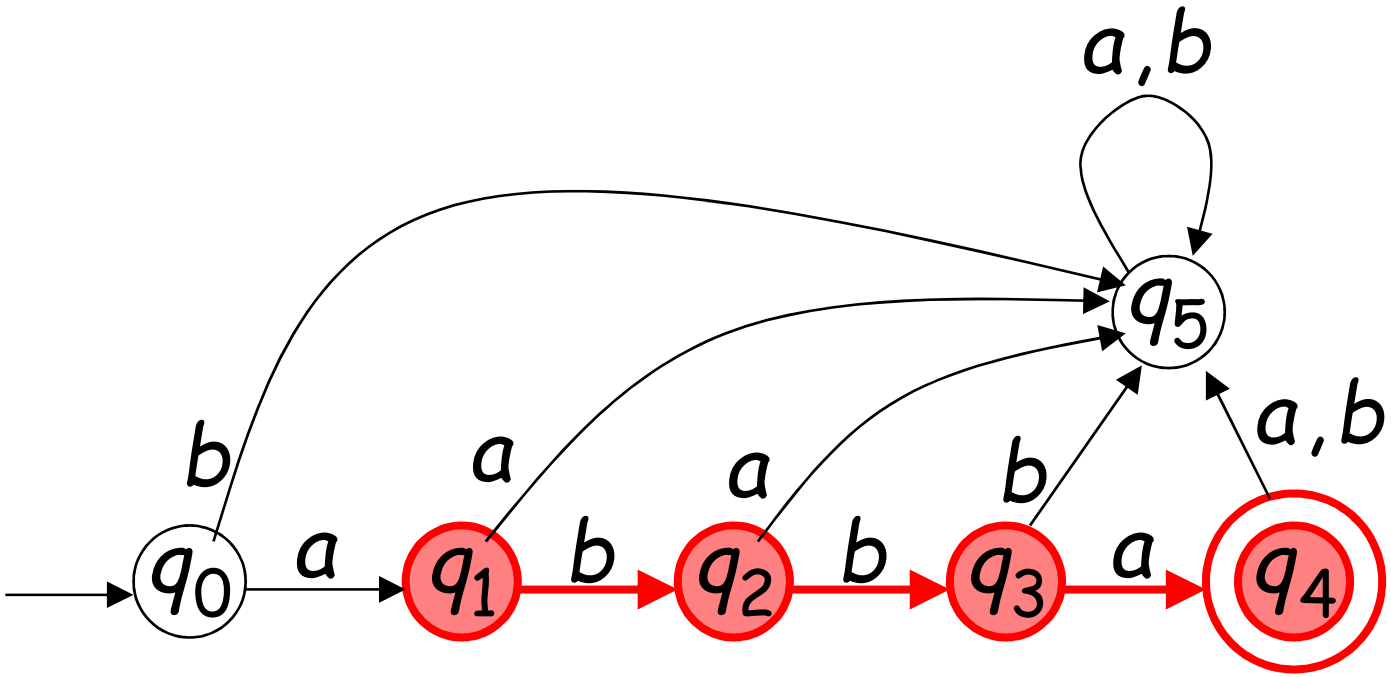
$$\delta^*(q_0, ab) = \delta(\delta^*(a_{0\_}$$
$$= \delta(\delta(\delta^*(a_{0\_}\wedge)a)\_b)$$
$$= \delta(\delta(a_0\ a)\ b)$$
$$= \delta(a_1, b)$$
$$= q_2$$



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



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



Special case:

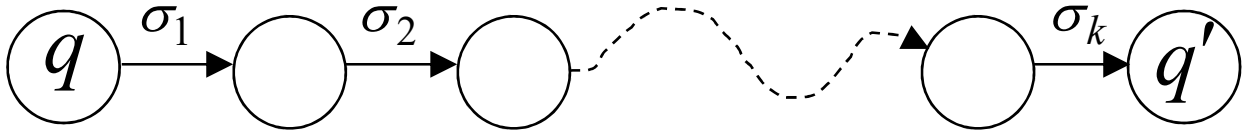
for any state q

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

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 of DFA

Language of DFA M :

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

We say that a language L' is accepted (or recognized) by DFA M if $L(M) = L'$

Language accepted by DFA

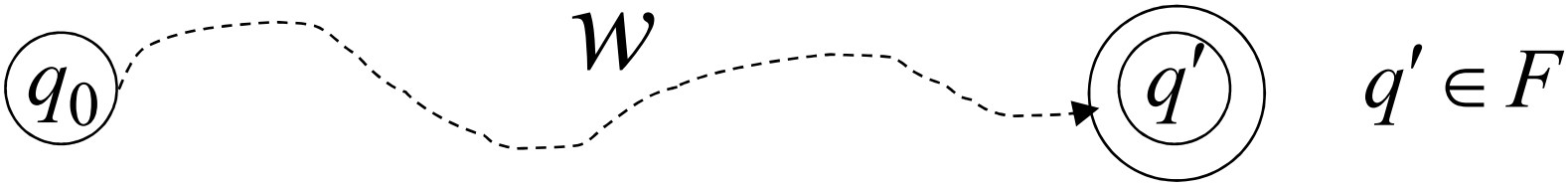
A DFA M accepts string w if there is a path from q_0 to an accepting (or final) state that is labeled by w

I.e., $L(M)$ = all strings that lead to an accepting state from q_0

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

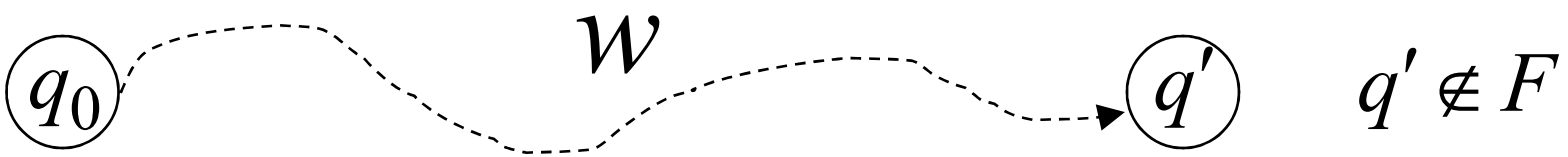
Language accepted by M:

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



Language rejected by M:

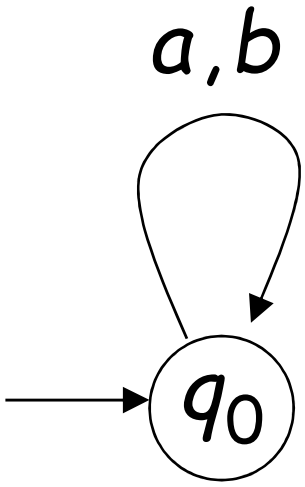
$$\overline{L(M)} = \{w \in \Sigma^* : \delta^*(q_0, w) \notin F\}$$



More DFA Examples

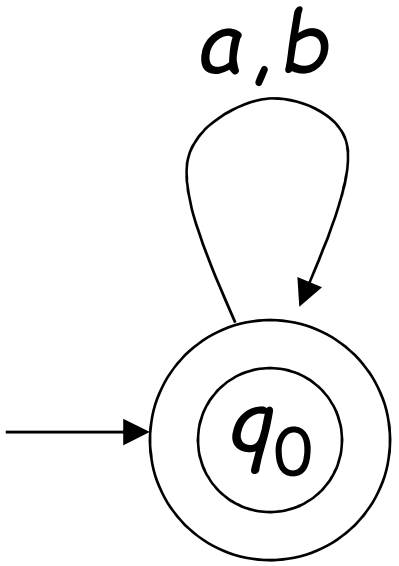
$$\Sigma = \{a, b\}$$

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



Empty language

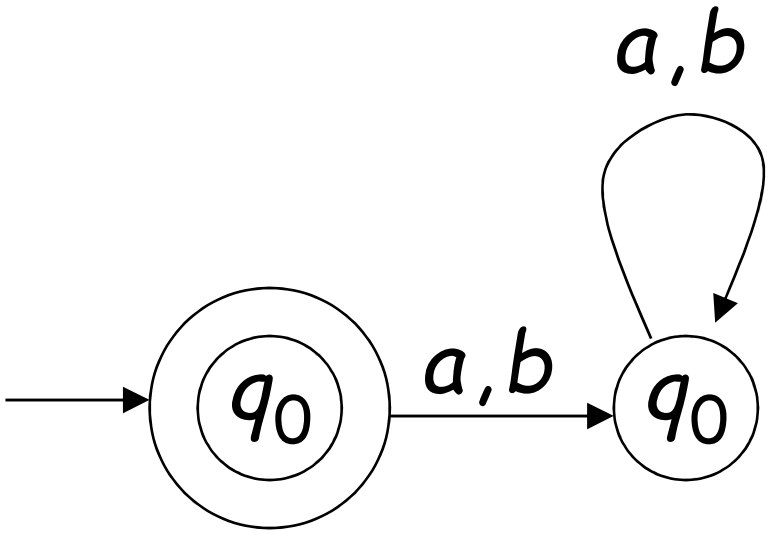
$L(M) = \Sigma^*$



All strings

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

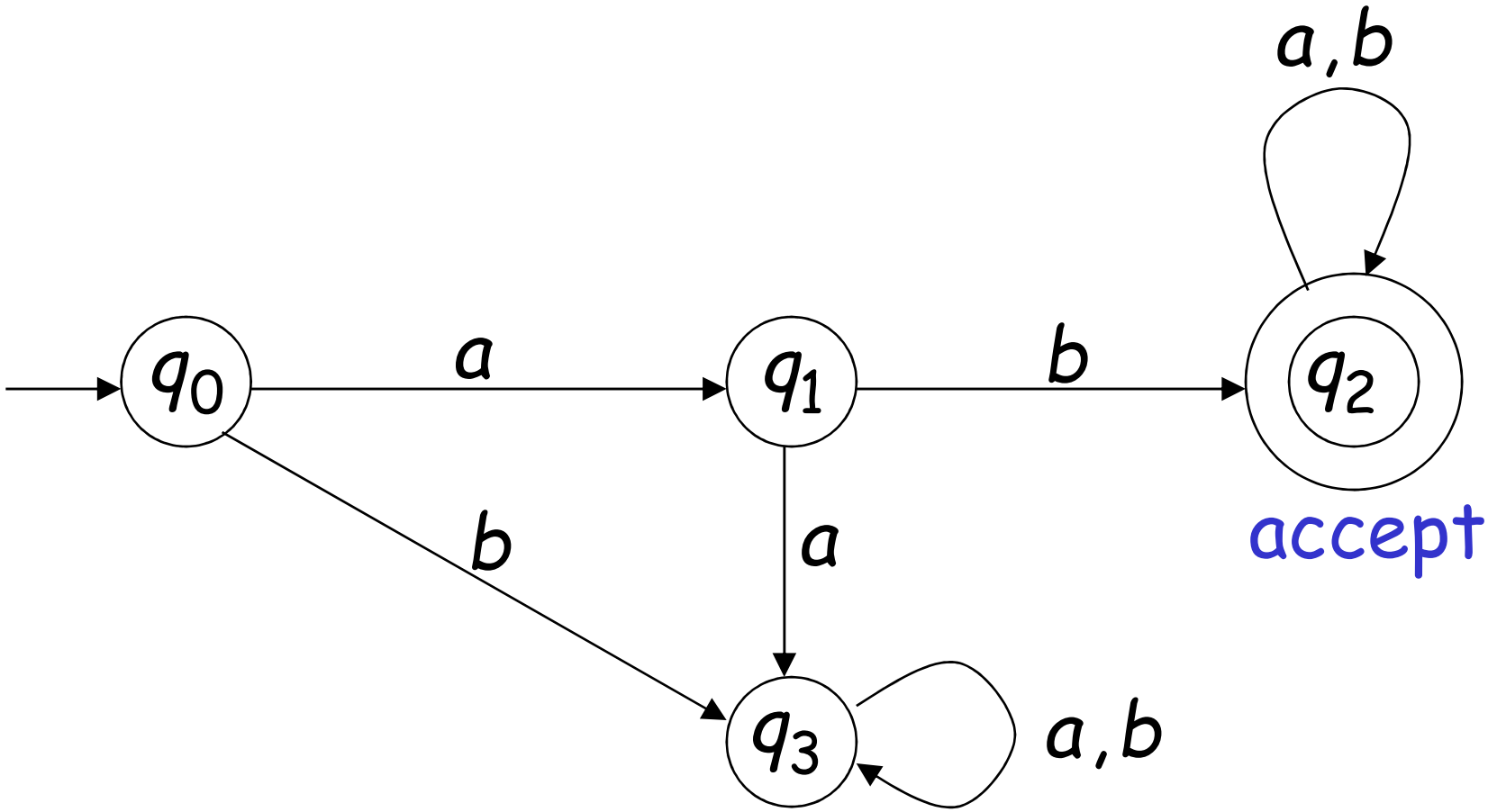
Language of the empty string



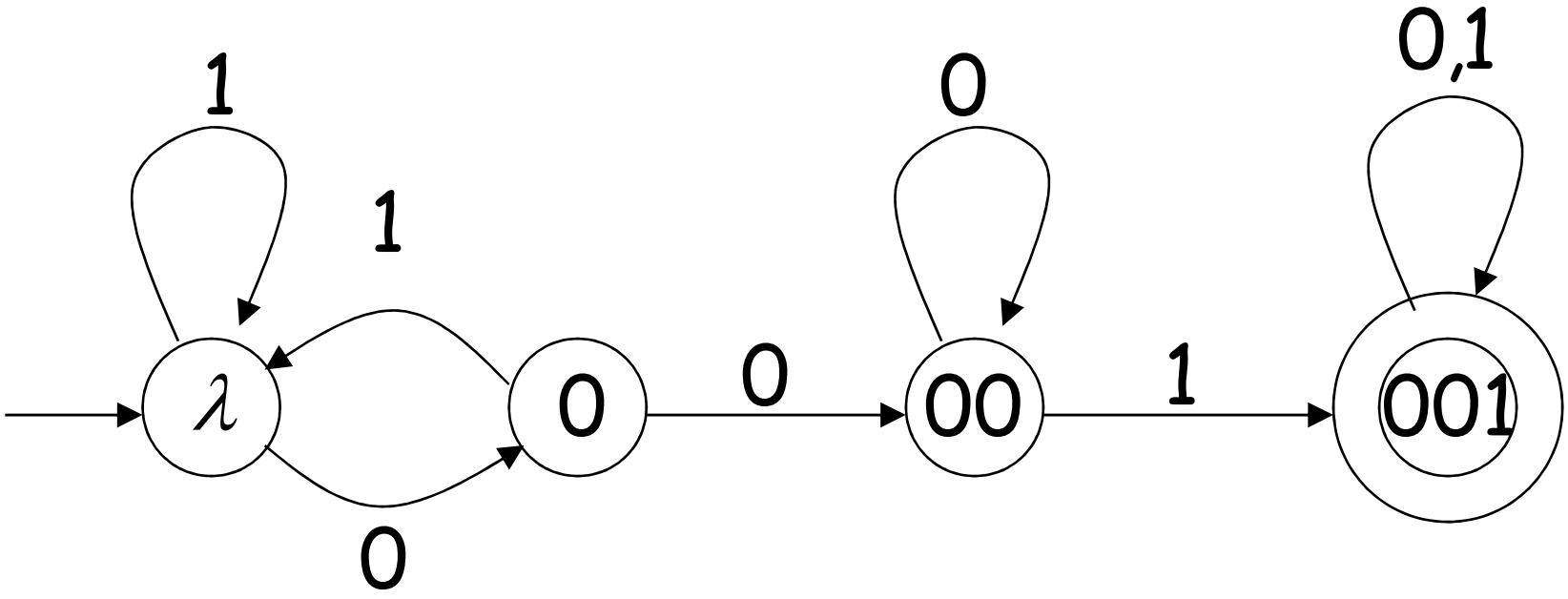
$$L(M) = \{\lambda\}$$

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

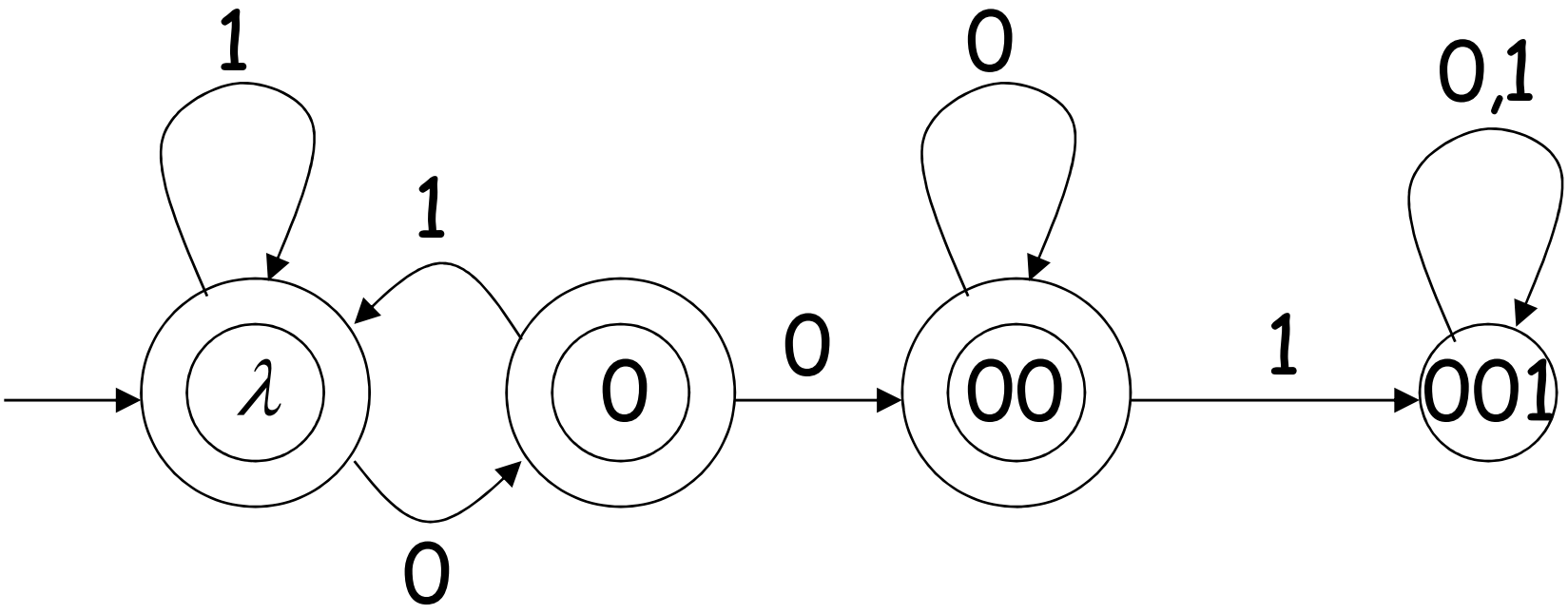
$L(M)$= { all strings with prefix ab }



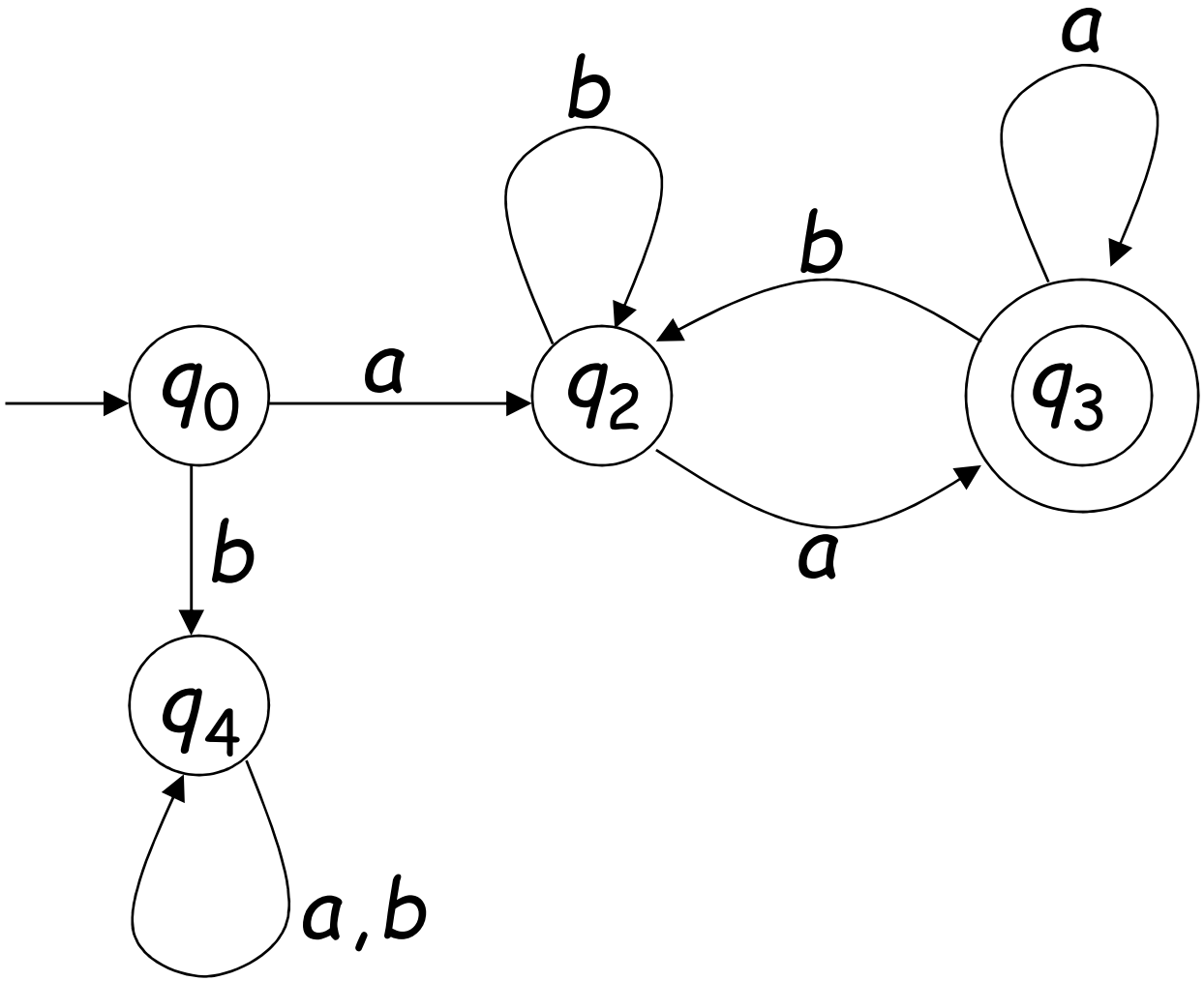
$L(M)$ = { all binary strings containing substring 001 }



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

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

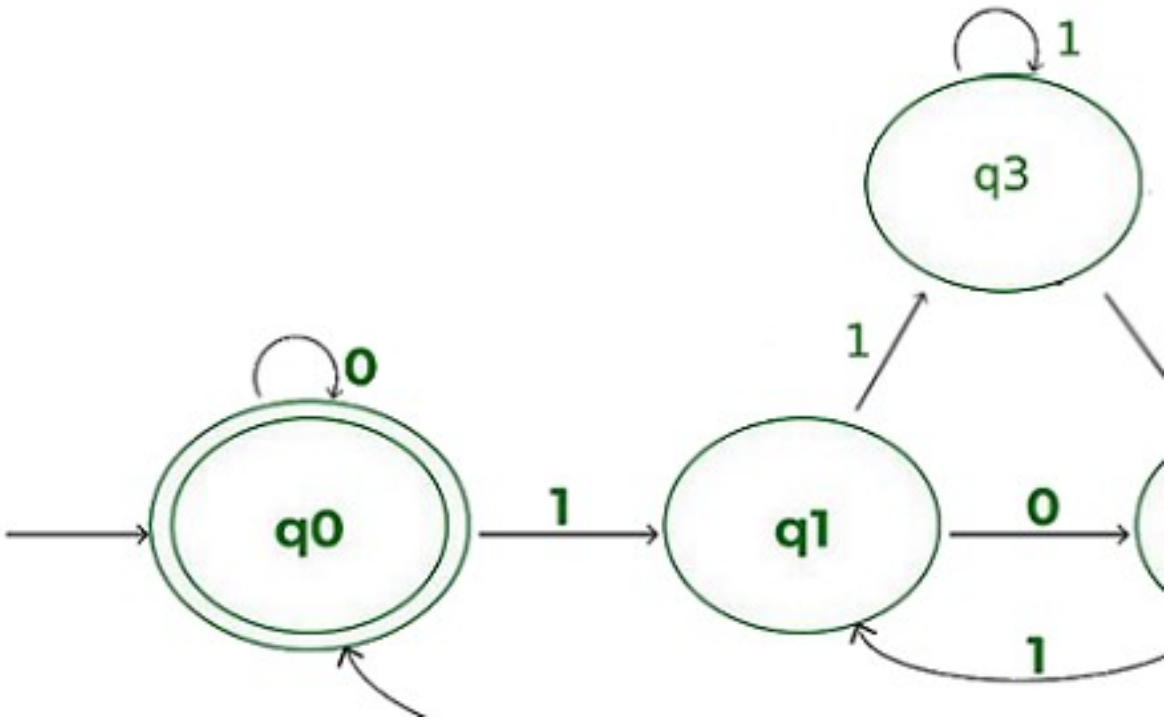
There exist automata that accept these languages (see previous slides).

Draw DFA for the following regular languages:

- FA accepting Binary representation of integer
  Divisible by 3
- FA accepting Binary representation of integer
  Divisible by 4
- FA accepting all string w over {a,b} that begins and ends with the same symbol
- Draw an FA having $N_a(w) \cong 0 \bmod 3$
  i.e. number of a in string is divisible by 3
- $L=\{ w \mid |w|=3\}$
- FA accepting binary strings that begins and ends with different symbols

Example

- FA accepting Binary representation of integer Divisible by 4



There exist languages which are not Regular:

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

Language of even/odd length palindrome

There is no DFA that accepts these languages

(we will prove this in a later class)

Closure Properties of Regular Languages

If we have a finite automata accepting the Language L1 and L2 then there is a FA accepting

$L1 \cup L2$

$L1 \cap L2$

L1 - L2

Complement of L

Union of Two Languages

Theorem 1.12: If L_1 and L_2 are regular languages, then so is $L_1 \cup L_2$.

(The regular languages are 'closed' under the union operation.)

Proof idea: L_1 and L_2 are regular, hence there are two DFA M_1 and M_2, with $L_1=L(M_1)$ and $L_2=L(M_2)$.

Out of these two DFA, we will make a third automaton M_3 such that $L(M_3) = L_1 \cup L_2$.

Proof Union-Theorem (1)

$M_1=(Q_1,\Sigma,\delta_1,q_1,F_1)$ and $M_2=(Q_2,\Sigma,\delta_2,q_2,F_2)$

Think of running both the m/c parallelly:

Define $M_3 = (Q_3,\Sigma,\delta_3,q_3,F_3)$ by:

- $Q_3 = Q_1 \times Q_2 = \{(r_1,r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
- $\delta_3((r_1,r_2),a) = (\delta_1(r_1,a), \delta_2(r_2,a))$
- $q_3 = (q_1,q_2)$
- If $F_3 = \{(r_1,r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$, M_3 accept $L_1 \cup L_2$.

Proof Union-Theorem (2)

$\delta_3((r_1,r_2),a) = (\delta_1(r_1,a), \delta_2(r_2,a))$

This will follow immediately from the formula

$\delta^*(q_3,x) = (\delta_1^*(q_1,x), \delta_2^*(q_2,x))$ for every $x \in \Sigma^*$

For every string x, x is accepted by M_3 if $\delta^*(q_3,x) \in F_3$

And according to definition of F_3 and formula for δ^* this is true for every string x

Proof Union-Theorem (2)

The automaton $M_3 = (Q_3,\Sigma,\delta_3,q_3,F_3)$ runs M_1 and M_2 in 'parallel' on a string w.

In the end, the final state (r_1,r_2) 'knows' if $w \in L_1$ (via $r_1 \in F_1$?) and if $w \in L_2$ (via $r_2 \in F_2$?)

The accepting states F_3 of M_3 are such that $w \in L(M_3)$ if and only if $w \in L_1$ *or* $w \in L_2$, for:
$F_3 = \{(r_1,r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$.

Proof: i) if $w \in L(M_3) \Rightarrow w \in L_1$ *or* $w \in L_2$

ii) if $w \in L_1$ *or* $w \in L_2 \Rightarrow w \in L(M_3)$

Intersection of L_1 and L_2

Definition: $L_1 \cap L_2$

Theorem 1.13: If L_1 and L_2 are regular languages, then so is $L_1 \cap L_2$.

(The regular languages are 'closed' under concatenation.)

$$F_3 = \{(r_1, r_2) \mid r_1 \in F_1 \text{ and } r_2 \in F_2\}$$

Example - Difference of L_1 and L_2

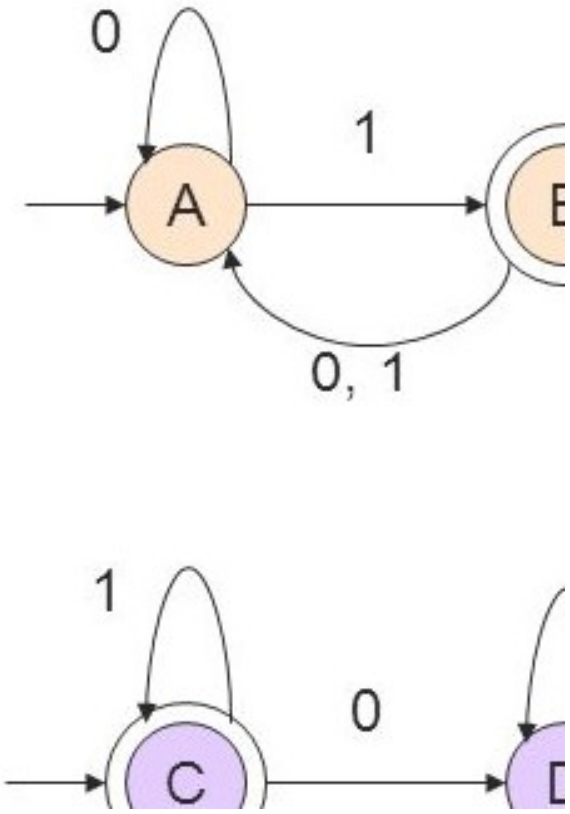
Definition: $L_1 - L_2$

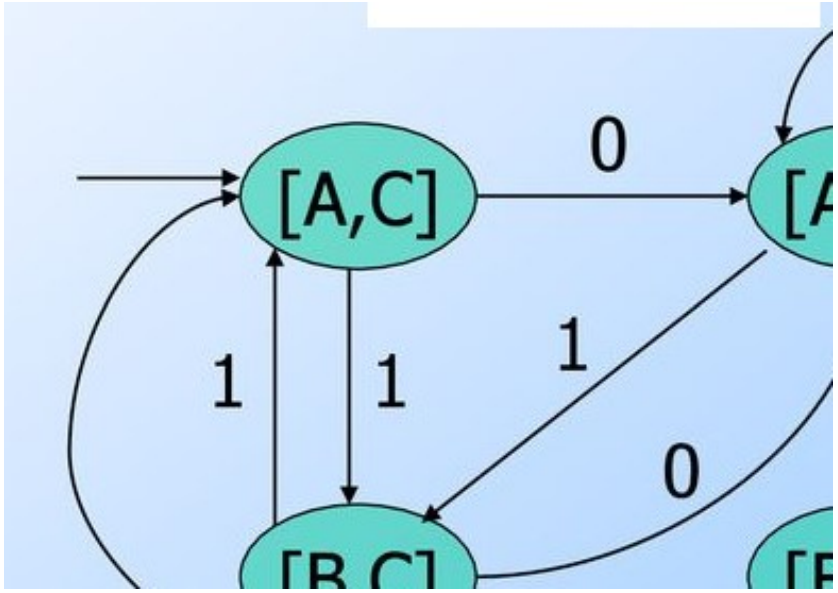
Theorem 1.13: If L_1 and L_2 are regular langues, then so is $L_1 - L_2$.

(The regular languages are 'closed' under difference.)

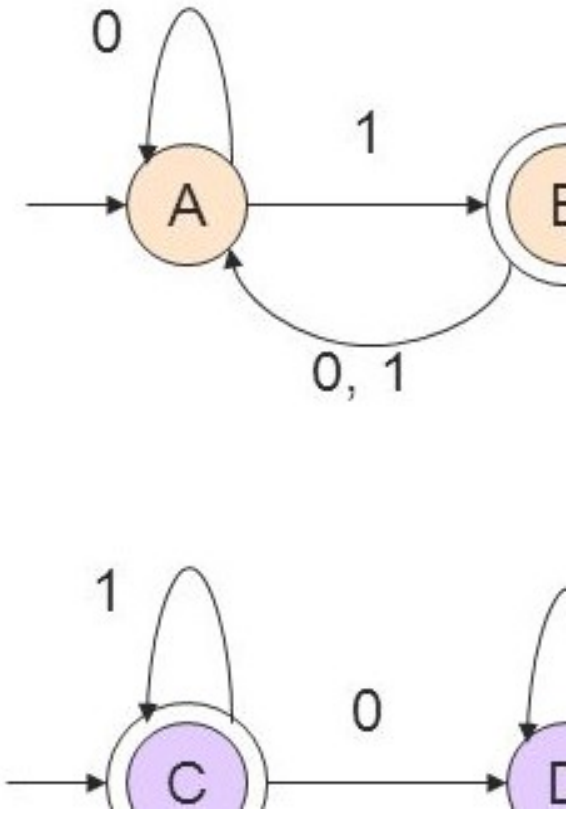
$$F_3 = \{(r_1, r_2) \mid r_1 \in F_1 \text{ and } r_2 \notin F_2\}$$

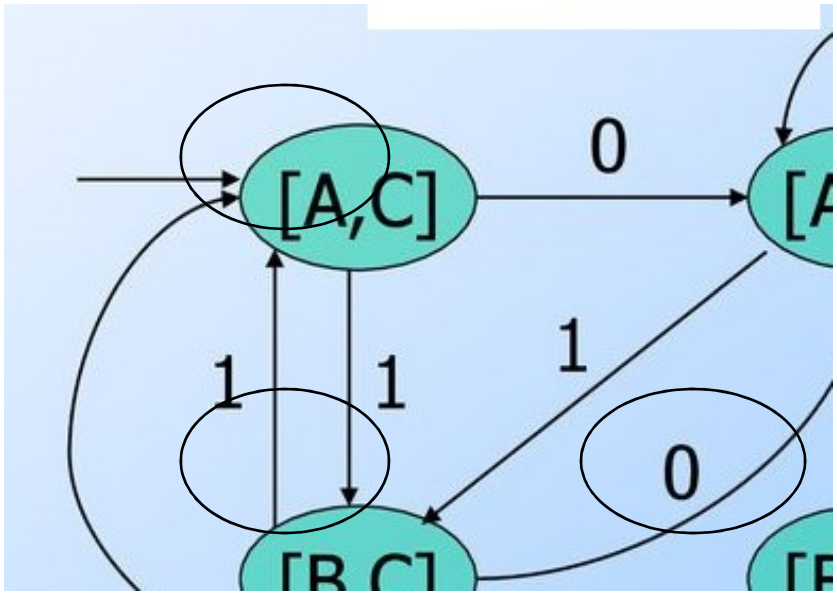
Example - Union



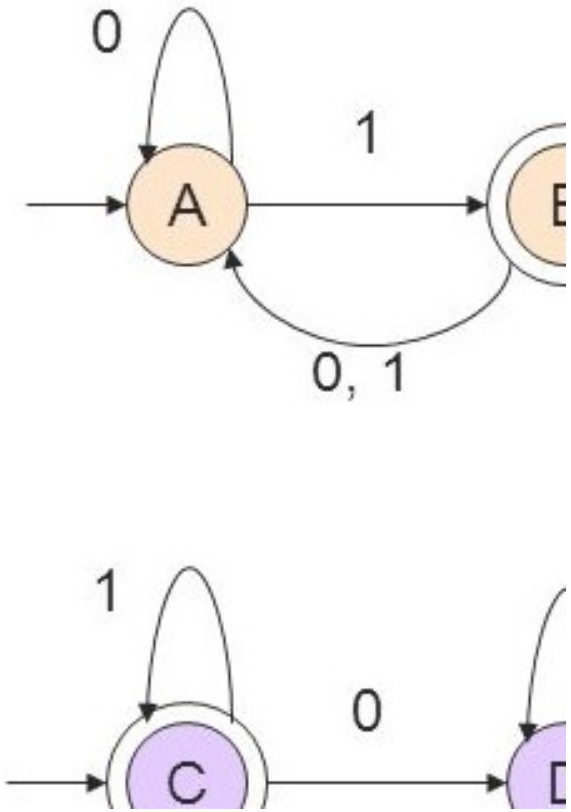


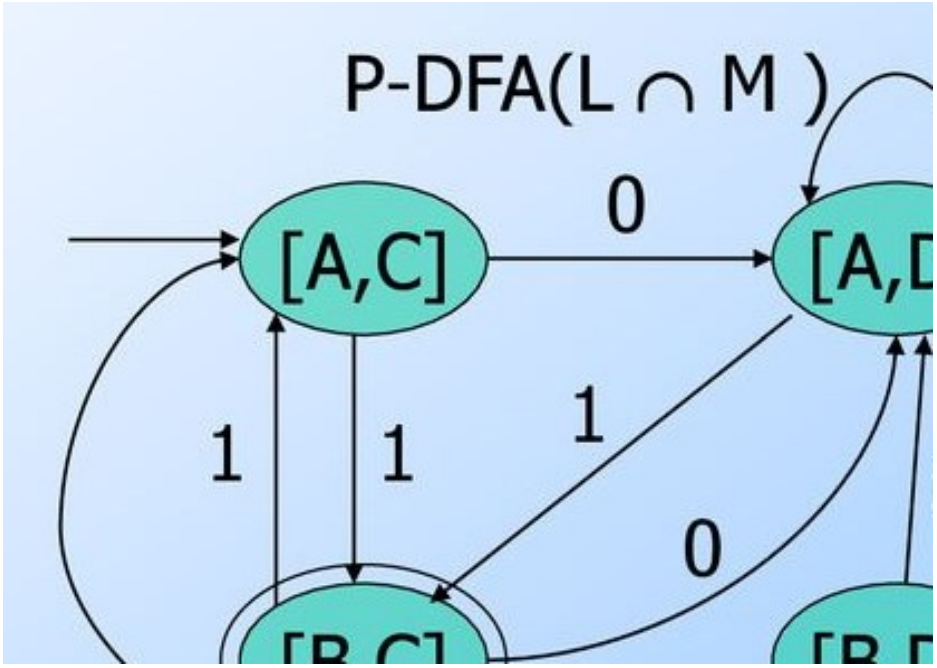
Example - Union



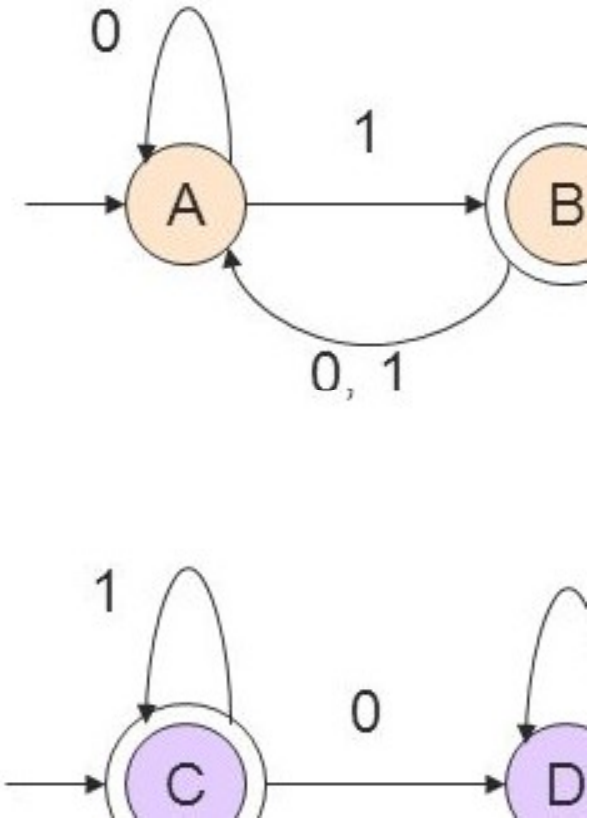


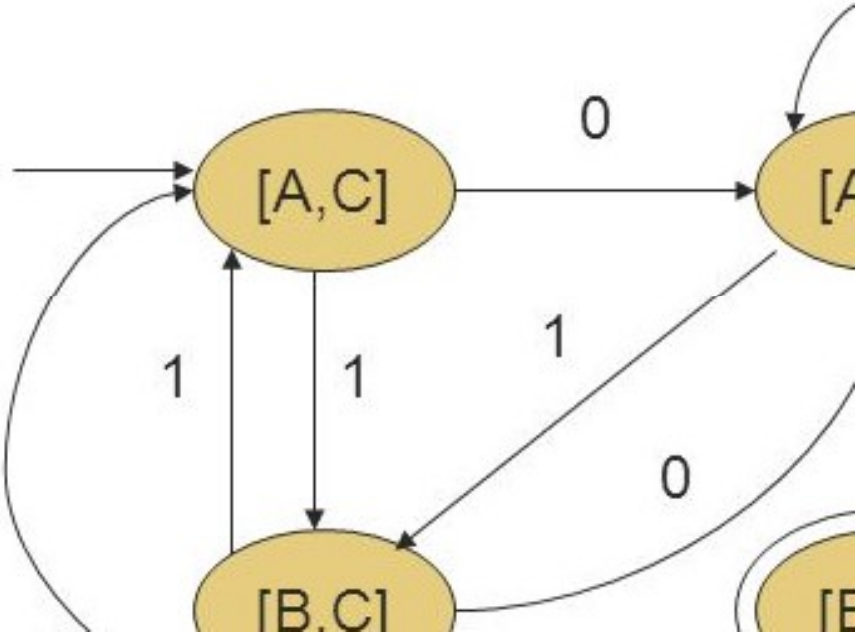
Example -Intersection





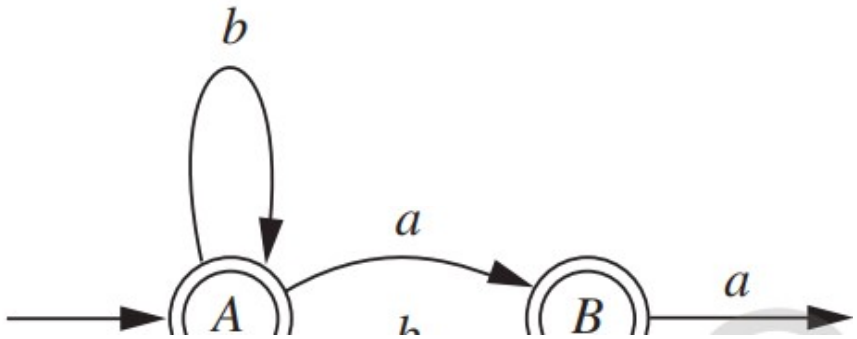
Example - Differene

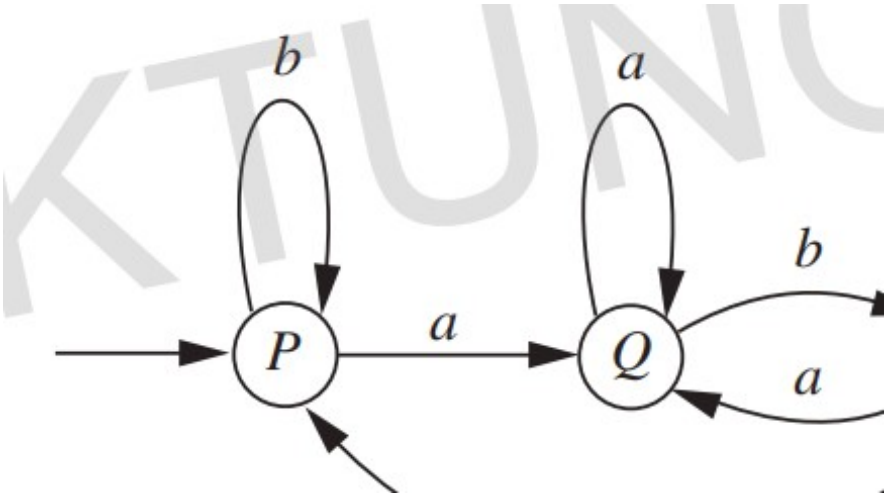




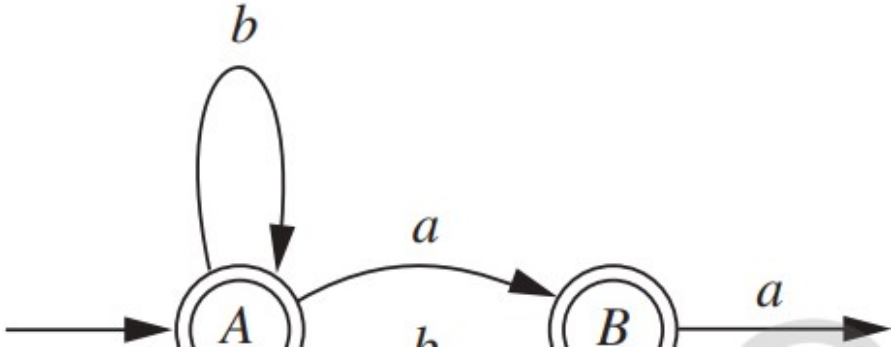
Example

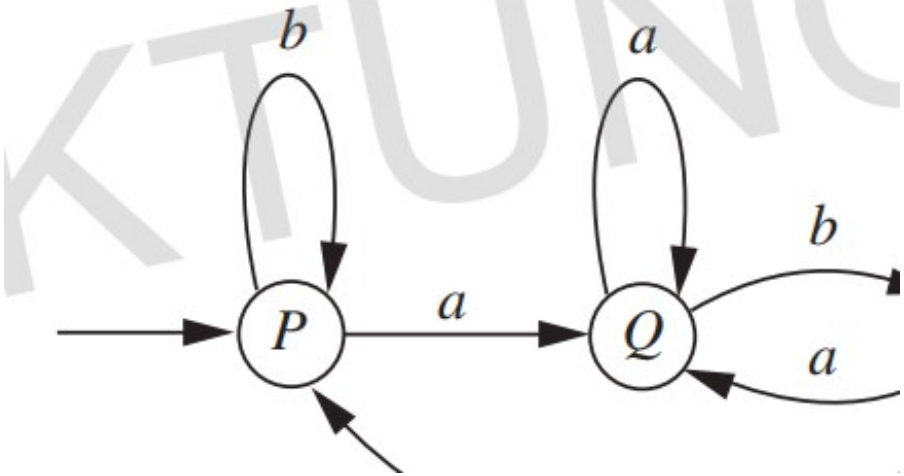
- L1= {x∈{a,b}*| aa is not a substring of x}
- L2 = {x∈{a,b}*|x ends with ab}

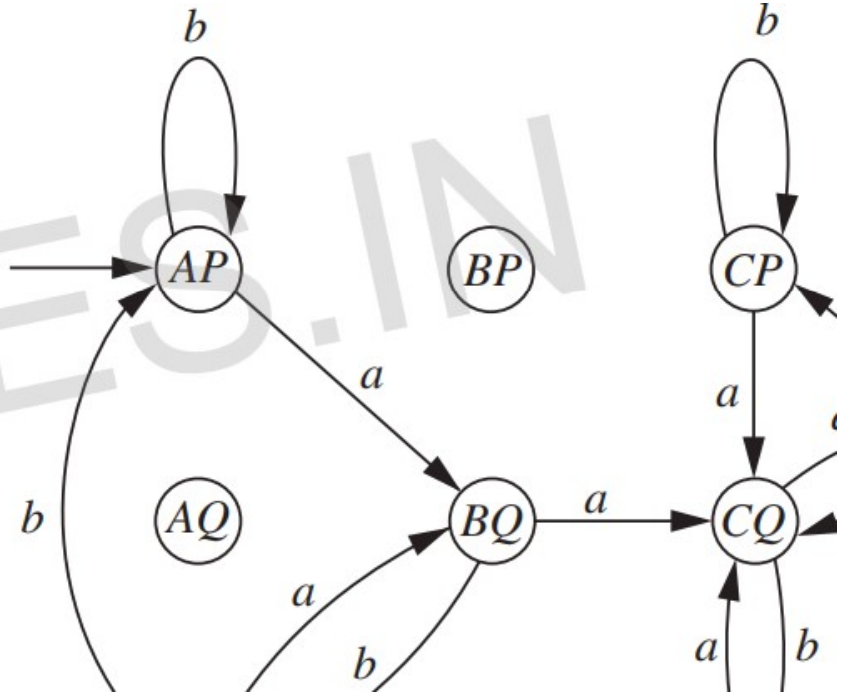




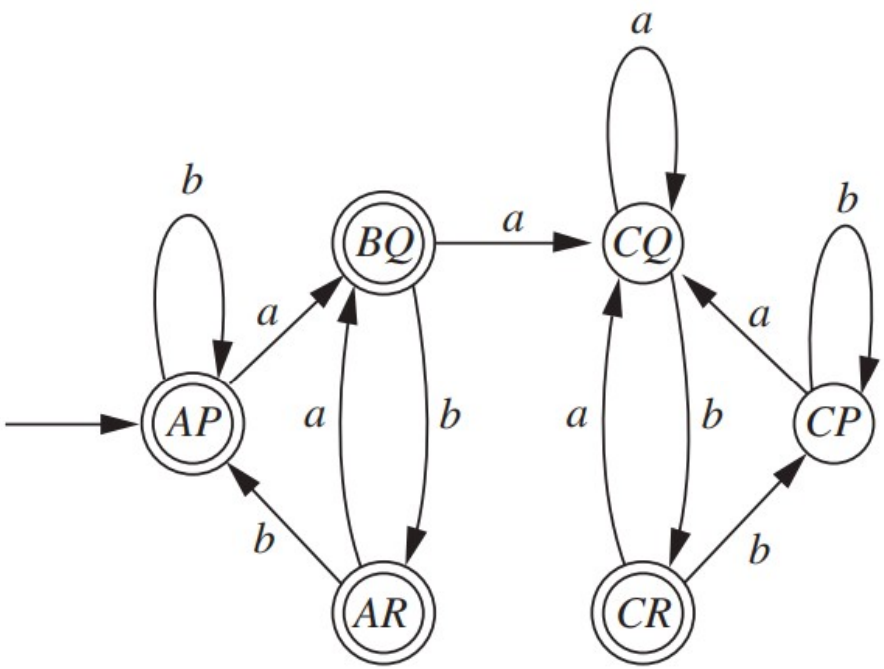
Example



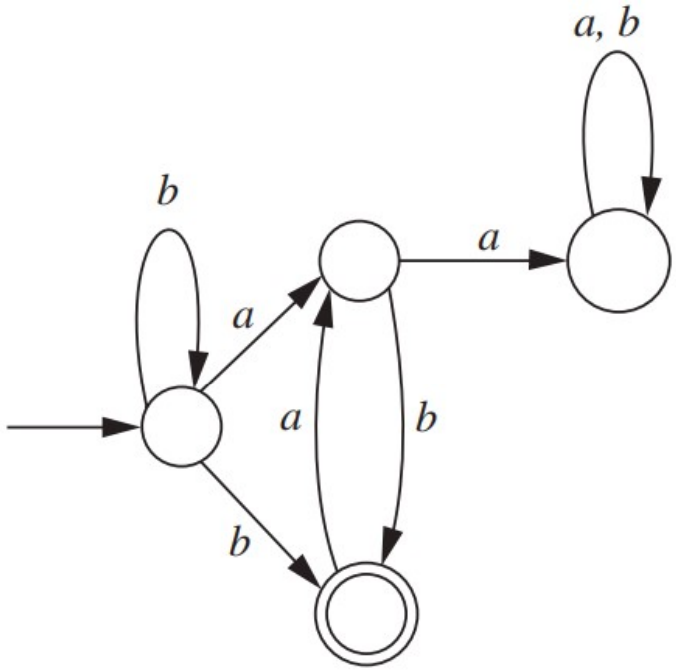




Example



L1 ∪ L2



L1 ∩ L2

Distinguishing One String from Another

Eg. FA M accepting L of string in {a,b}* ending with aa

1)^ and a

2)^ and aa

3)a and aa

Could FA be constructed with fewer than 3 states? **Or**

Can we be sure that 3 states are enough?

Distinguishing One String from Another

Definition:

- Let L be a language in Σ^*
- Two strings x and y in Σ^* are distinguishable with respect to L if there is a string $z \in \Sigma^*$ (which may depend on x and y) so that exactly one of the strings xz and yz is in L
- The string z is said to distinguish x and y with respect to L

Distinguishing One String from Another

- We may say that x and y are indistinguishable with respect to L if there is no such string z;
- In other words, if for every z, both xz and yz have the same status---either both in L or both not in L

  Eg: $L=\{x\in\{0,1\}^* \mid x \text{ ends with } 10\}$
- The strings 01011 and 100 are distinguishable with respect to L because for z=0, $01011z\in L$ and $100z\notin L$.
- The strings 0 and 100 are indistinguishable with respect to L

Distinguishing One String from Another

- Lemma

  Suppose that $L \subseteq \Sigma^*$ and $M=(Q, \Sigma, q_0, F, \delta)$ is any FA recognizing L.

  If x and y are two strings in Σ^* for which $\delta^*(q_0,x)=\delta^*(q_0,y)$, then x and y are indistinguishable with respect to L

Proof:

Let z be any string in Σ^* , and consider the two strings xz and yz.

$$\delta^*(q_0,xz)= \delta^*(\delta^*(q_0,x),z)$$

$$\delta^*(q_0,yz)= \delta^*(\delta^*(q_0,y),z)$$

and therefore, by our assumption, $\delta^*(q_0,xz)=\delta^*(q_0,yz)$.

Since M is assumed to recognize L, these two strings are either both in L or both not in L. Therefore, x and y are indistinguishable with respect to L. ■

Distinguishing One String from Another

Theorem

Suppose that $L \subseteq \Sigma^*$ and, for some positive integer n, there are n strings in Σ^* , any two of which are distinguishable with respect to L. Then there can be no FA recognizing L with fewer than n states.

P.T a machine recognizing a string having 1 in the nth position from right have atleast 2^n states

- L = {w | w is a binary string s.t., the n^{th} symbol from its right end is a 1}
  - NFA has n+1 states
  - But an equivalent DFA needs to have at least 2^n states
- Proof (By contradiction)

Proof

(Pigeon hole principle)

- *m* holes and pigeons >*m*
  - => at least one hole has to contain two or more pigeons

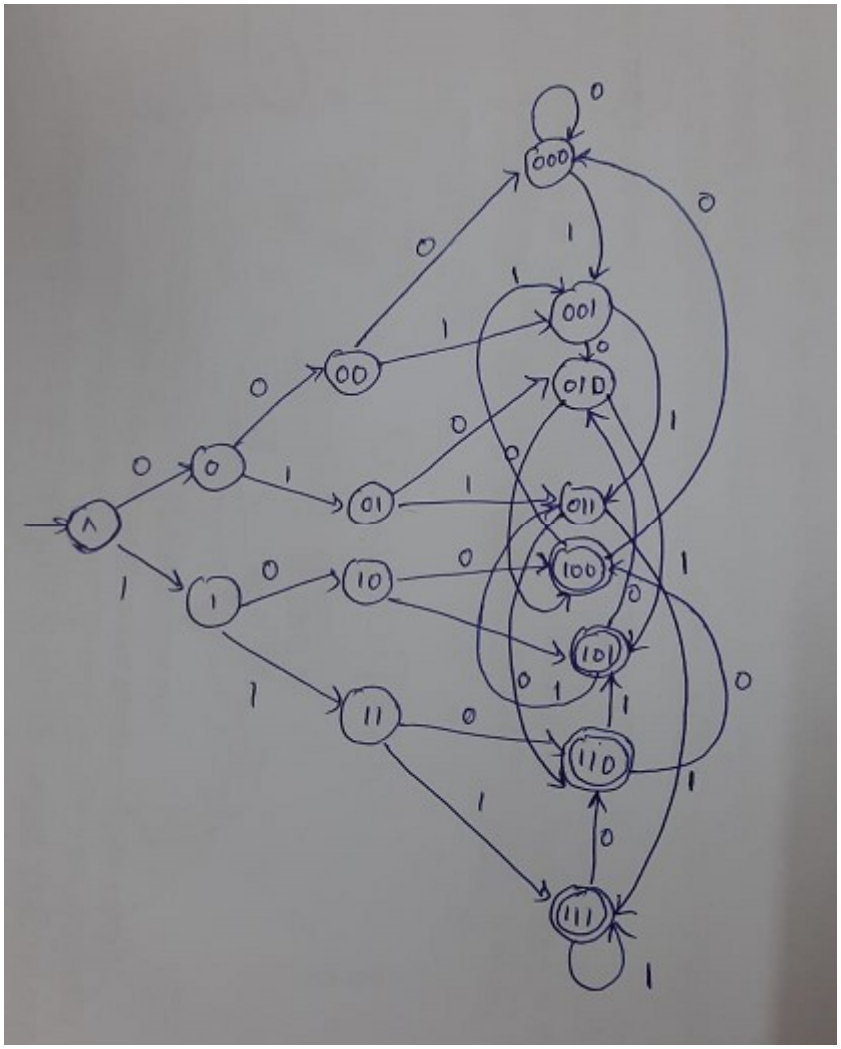
e.g. let n=3, and M accepts language L and has less than 8 states

- No. of strings = 8 and No. of states are 7 or less
- i.e. two strings takes machine to same state
  - E.g. 010 but 010 ∈L and 110 does not ∈ L
    110
  - E.g. 100 considering substring 0, 1000 does not ∈ L
    111 , 1110 ∈L

Proof

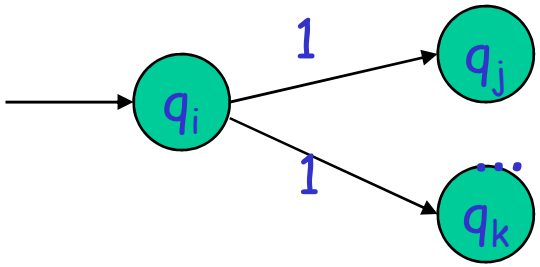
- For each n, the no. of stings of length exactly n is 2^n.
- So if we add these no. for the values of i from 0 to n, we obtain total

  $2^{n+1}-1$ states.
- So, a machine recognizing a string having 1 in the nth position from right have atleast 2^n states


λ
0
1
00
01
10
11
000
001
010
011
100
101
110
111

Non-deterministic Finite Automata (NFA)

- A Non-deterministic Finite Automaton (NFA)
  - is of course "non-deterministic"
    - Implying that the machine can exist in more than one state at the same time
    - Transitions could be non-deterministic (0,1, or more transition for an input symbol)



• Each transition function therefore maps to a set of states

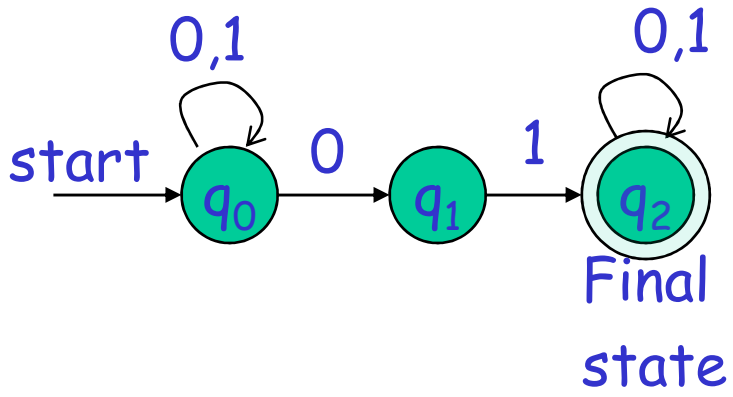
Non-deterministic Finite Automata (NFA)

- An NFA is also defined by the 5-tuple:
  - $\{Q, \Sigma, q_0, F, \delta\}$
- A Non-deterministic Finite Automaton (NFA) consists of:
  - Q ==> a finite set of states
  - Σ ==> a finite set of input symbols (alphabet)
  - q_0 ==> a start state
  - F ==> set of accepting states
  - δ ==> a transition function, which is a mapping between $Q \times \Sigma$ ==> subset of Q (2^Q)

Regular expression: (0+1)*01(0+1)*

NFA for strings containing 01

Why is this non-deterministic?



What will happen if at state q_1 an input of 0 is received?

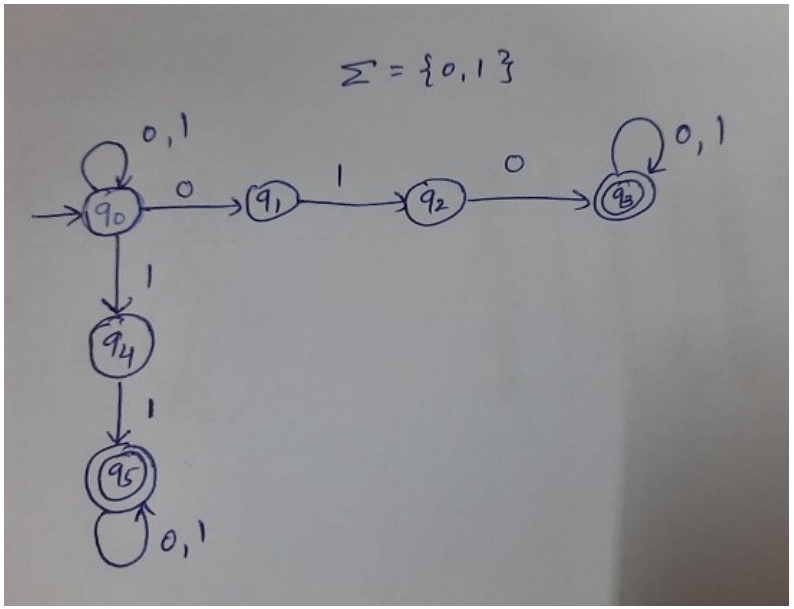
- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

	symbols	
states / δ	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Φ	$\{q_2\}$
$*q_2$	$\{q_2\}$	$\{q_2\}$

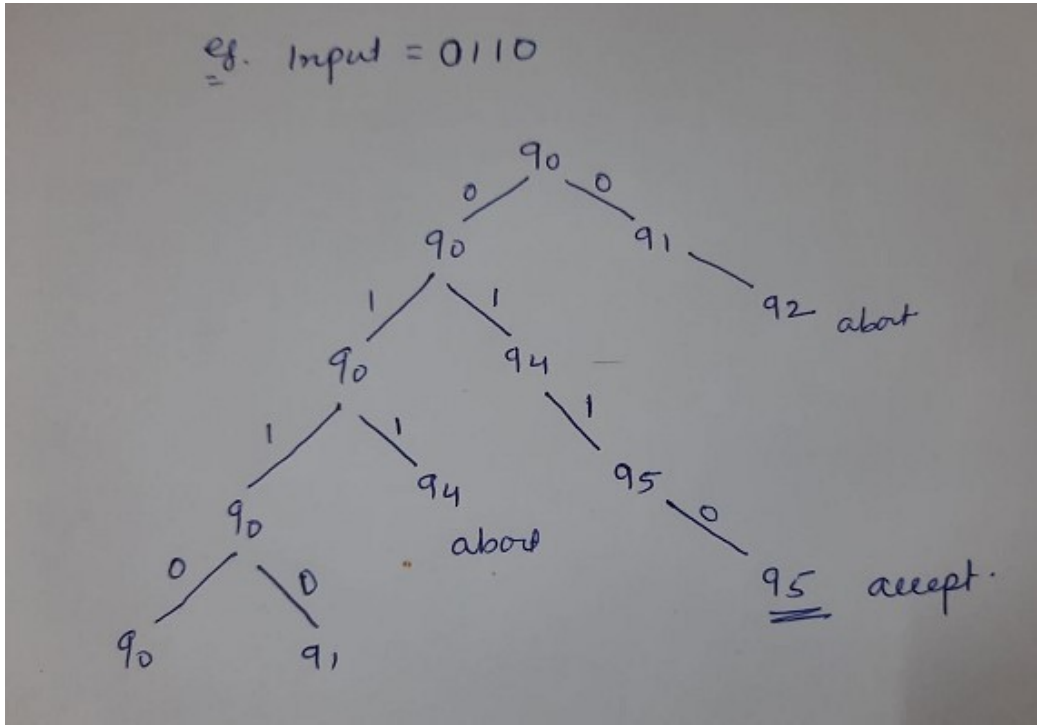
How to use an NFA?

- Input: a word w in Σ^*
- Question: Is w acceptable by the NFA?
- Steps:
  - Start at the "start state" q_0
  - For every input symbol in the sequence w do
    - Determine all possible next states from all current states, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed and if at least one of the current states is a final state then *accept w;*
  - Otherwise, *reject w.*

Example



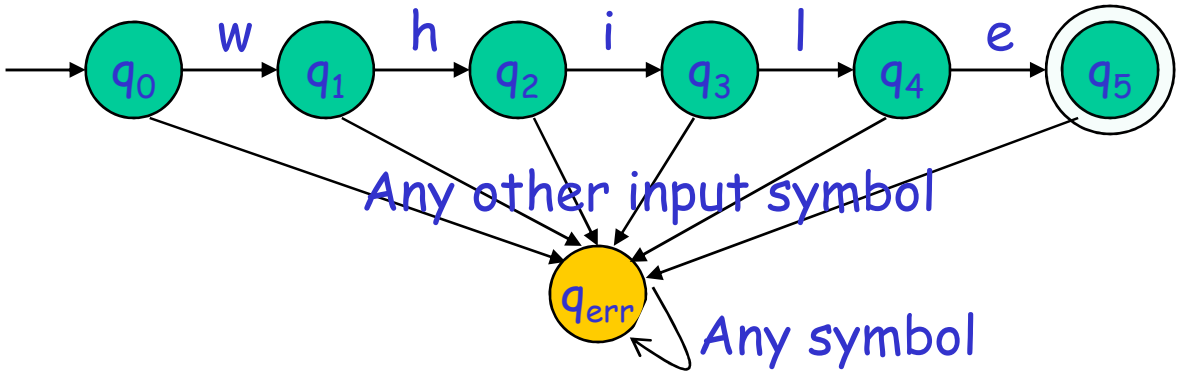
Input: 0110



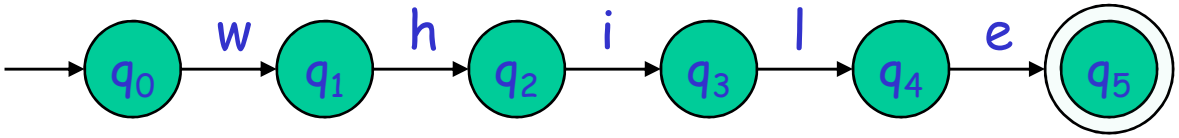
Note: Omitting to explicitly show error states is just a matter of design convenience
(one that is generally followed for NFAs), and
i.e., this feature should not be confused with the notion of non-determinism.

What is an "error state"?

- A DFA for recognizing the key word "*while*"



- An NFA for the same purpose:



Transitions into a dead state are implicit

Example #2

- Build an NFA for the following language:
  L = { w | w ends in ab}
- Other examples
  - Keyword recognizer (e.g., if, then, else, while, for, include, etc.)
  - L = { w | w starts with ab}
  - L = { w | w consists 10 }
  - L = {aa,aab}*{b}
  - L = {w | w is a binary string s.t., the 4th symbol from its right end is a 0}
  - L = {w|either 11 or 010 is a substr of w}

Examples

L = { w | w ends in ab}

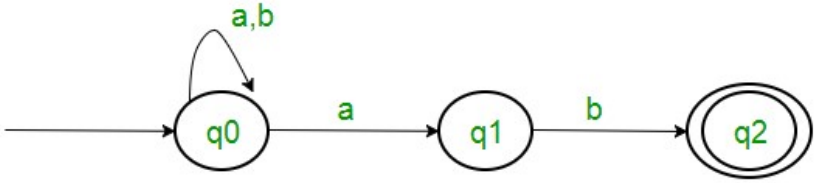
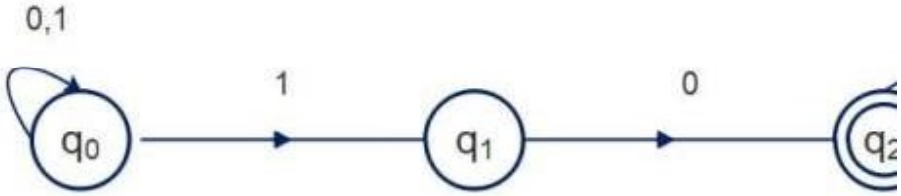
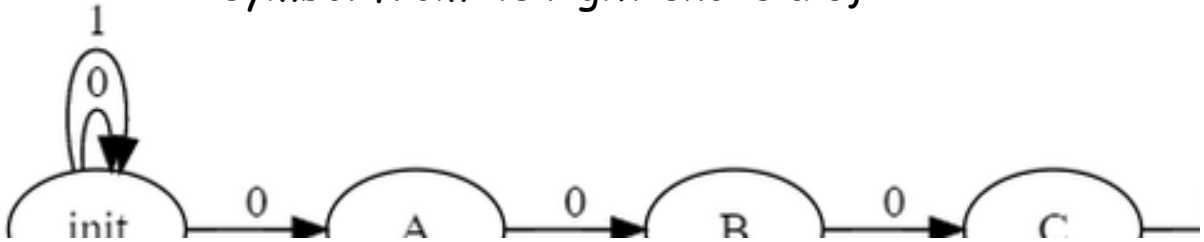


Figure 1

L = { w | w consists 10 }

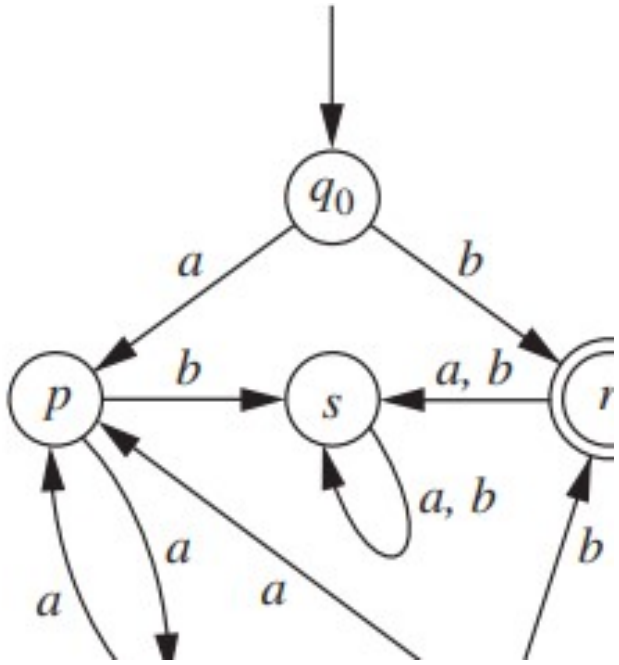


L = {w | w is a binary string s.t., the 4th symbol from its right end is a 0}

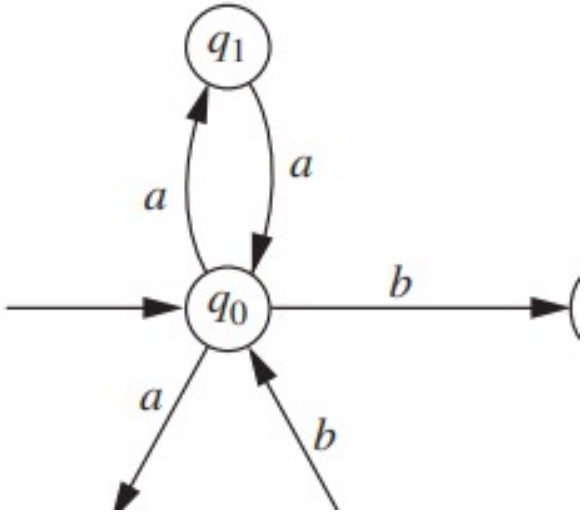


Examples

DFA for L = {aa,aab}*{b}



NFA for L = {aa,aab}*{b}



Extension of δ to NFA Paths

- Let $M = \{Q, \Sigma, q_0, F, \delta\}$ be an NFA
- The function $\delta^* : Q \times \Sigma^* \Longrightarrow 2^Q$ is defined as

- Basis: for any $q \in Q$, $\delta^*(q, \varepsilon) = \{q\}$

- Induction:
  - Let $\delta^*(q_0, w) = \{p_1, p_2 \ldots, p_k\}$
  - $\delta(p_i, a) = S_i$ for $i = 1, 2 \ldots, k$

  - Then, $\delta^*(q_0, wa) = S_1 \cup S_2 \cup \ldots \cup S_k$

    or $\delta^*(q_0, wa) = \bigcup_{s \in \delta^*(q_0, w)} \delta(s, a)$

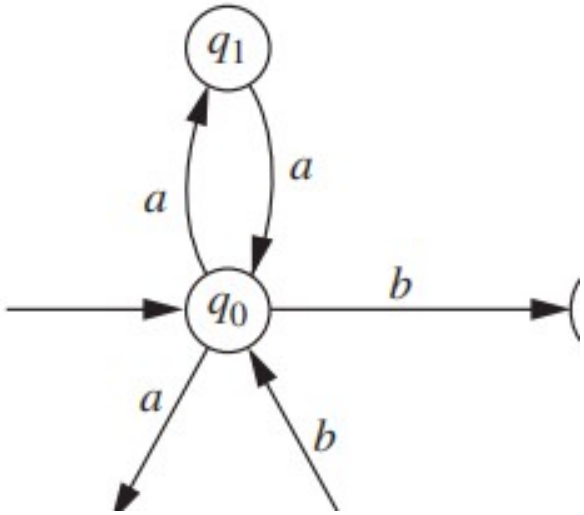
Language of an NFA

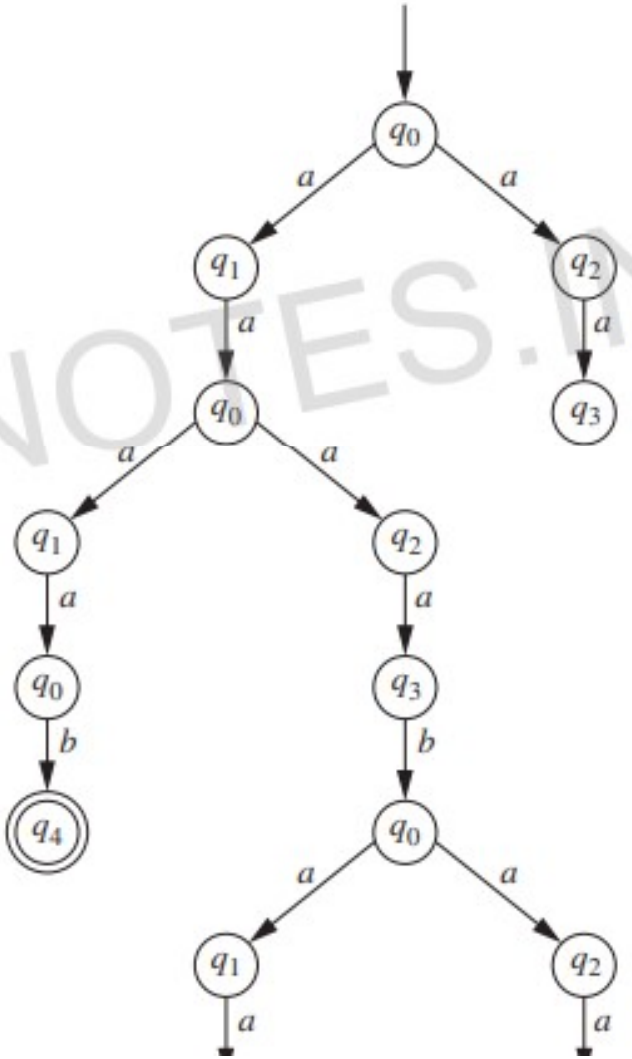
- An NFA accepts w if *there exists at least one* path from the start state to an accepting (or final) state that is labeled by w
- $L(N) = \{ w \mid \delta^*(q_0, w) \cap F \neq \Phi \}$

We say that a language L is accepted (or recognized) by NFA L(N) if L=L(N)

What is the language accepted by NFA?

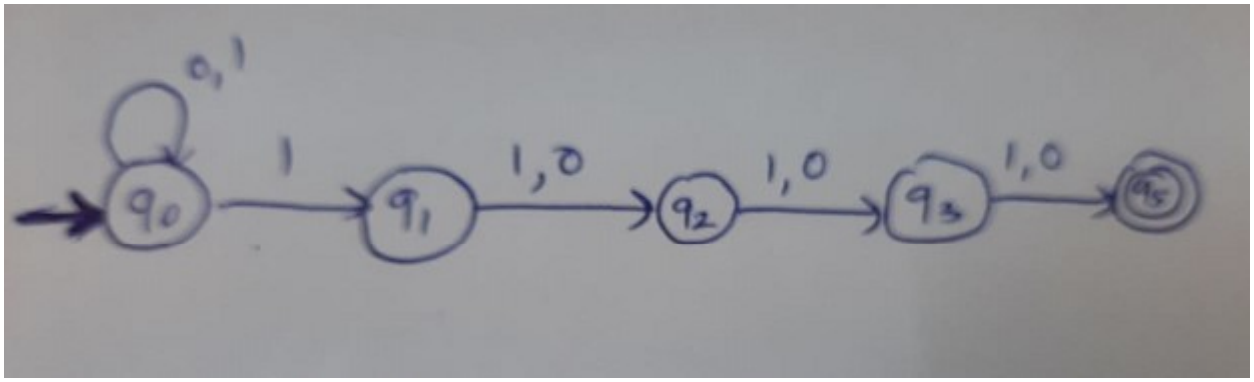
Input string: aaaabaab





L(M)={aa,aab}*{b}

What is the language accepted by NFA?



$L(M)=\{x \in \{0,1\}^* | x$ has 4^{th} symbol from the right end is 1$\}$

Differences: DFA vs. NFA

- **DFA**

1. All transitions are deterministic
   Each transition leads to exactly one state
2. For each state, transition on all possible symbols (alphabet) should be defined
3. Accepts input if the last state visited is in F
4. Sometimes harder to construct because of the number of states
5. Practical implementation is feasible

- **NFA**

1. Some transitions could be non-deterministic
   A transition could lead to a subset of states
2. Not all symbol transitions need to be defined explicitly (if undefined will go to an error state – this is just a design convenience, not to be confused with "non-determinism")
3. Accepts input if *one of* the last states is in F
4. Generally easier than a DFA to construct
5. Practical implementations limited but emerging

But, DFAs and NFAs are equivalent in their power to capture langauges !!

Equivalence of DFA & NFA

- Theorem:
  - A language L is accepted by a DFA *if and only if* it is accepted by an NFA.
- Proof:

Should be true for any L

If part:

- Prove by showing every NFA can be converted to an equivalent DFA (in the next few slides…)

2. Only-if part is trivial:
   - Every DFA is a special case of an NFA where each state has exactly one transition for every input symbol. Therefore, if L is accepted by a DFA, it is accepted by a corresponding NFA.

□

Proof for the if-part

- If-part: A language L is accepted by a DFA if it is accepted by an NFA
- rephrasing…
- Given any NFA N, we can construct a DFA D such that L(N)=L(D)

- How to convert an NFA into a DFA?
  - Observation: In an NFA, each transition maps to a *subset* of states
  - Idea: Represent:
    each "subset of NFA_states" ➔ a single "DFA_state"

Subset construction

NFA to DFA by subset construction

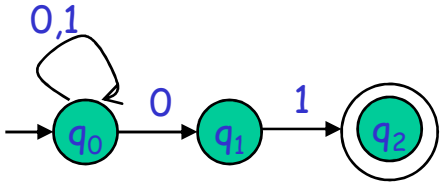
- Let $N = \{Q_N, \Sigma, \delta_N, q_0, F_N\}$
- Goal: Build $D=\{Q_D, \Sigma, \delta_D, \{q_0\}, F_D\}$ s.t. $L(D)=L(N)$
- Construction:
  1. Q_D= the set of states of DFA
     = all subsets of Q_N (i.e., power set)
  1. F_D=set of subsets S of Q_N s.t. $S \cap F_N \neq \Phi$
  2. δ_D: for each subset S of Q_N and for each input symbol a in Σ:
     - $\delta_D(S,a) = \bigcup_{p \text{ in } s} \delta_N(p,a)$

NFA to DFA construction: Example

- *L = {w | w ends in 01}*

Idea: To avoid enumerating all of power set, do "lazy creation of states"

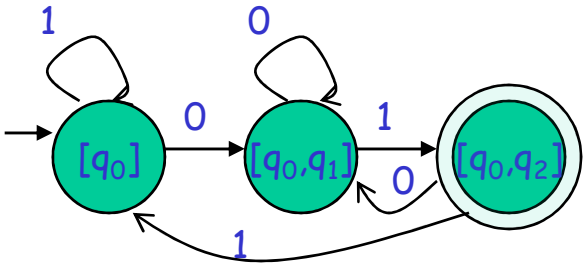
NFA:



δ_N	0	1
→ q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
*q_2	Ø	Ø

DFA:

δ_D
~~Ø~~
→ $[q_0]$
~~$[q_1]$~~
~~*$[q_2]$~~
$[q_0,q_1]$
*$[q_0,q_2]$
~~*$[q_1,q_2]$~~
~~*$[q_0,q_1,q_2]$~~



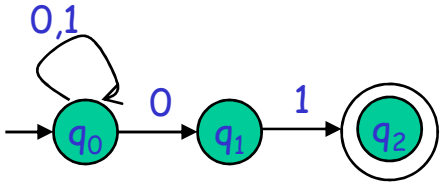
δ_D	0	1
→ $[q_0]$	$[q_0,q_1]$	$[q_0]$
$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_2]$
*$[q_0,q_2]$	$[q_0,q_1]$	$[q_0]$

0. Enumerate all possible subsets
1. Determine transitions
2. Retain only those states reachable from $\{q_0\}$

NFA to DFA: Repeating the example using *LAZY CREATION*

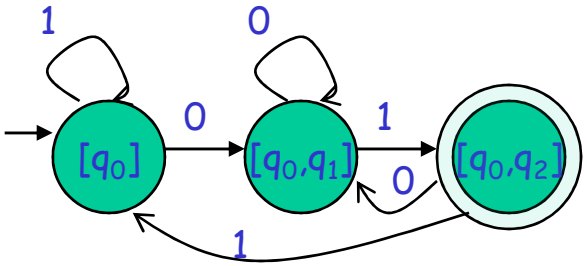
- *L = {w | w ends in 01}*

NFA:



δ_N	0	1
→ q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
$*q_2$	Ø	Ø

DFA:

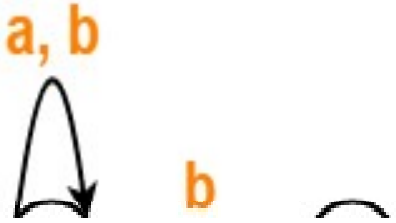


δ_D	0	1
→ $[q_0]$	$[q_0,q_1]$	$[q_0]$

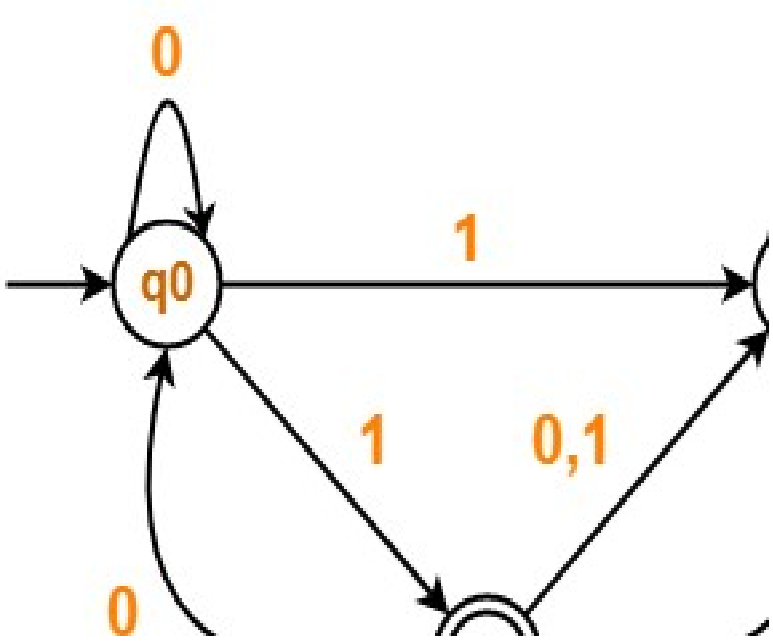
Main Idea:

Introduce states as you go (on a need basis)

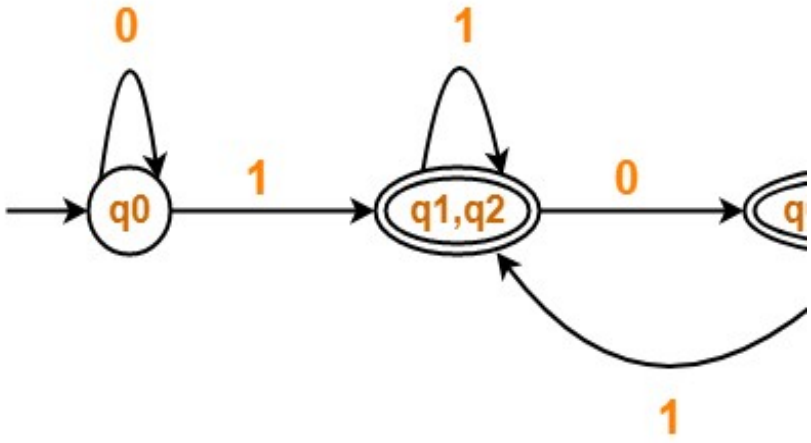
Ex. Convert the following Non-Deterministic Finite Automata (NFA) to Deterministic Finite Automata (DFA)-



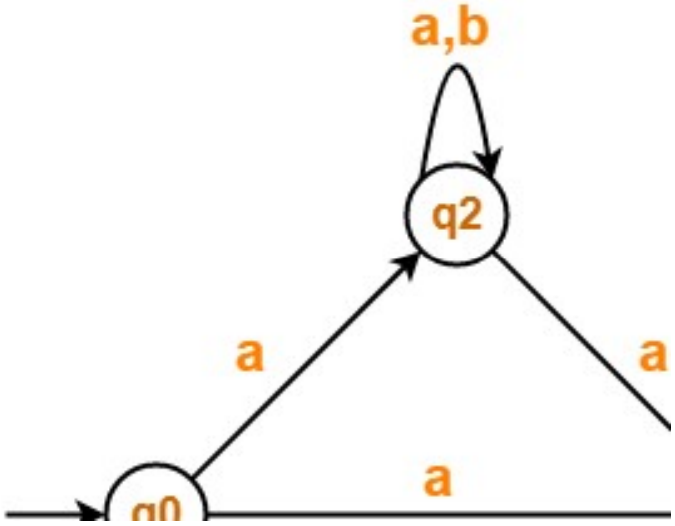
Ex Convert the following Non-Deterministic Finite Automata (NFA) to Deterministic Finite Automata (DFA)-

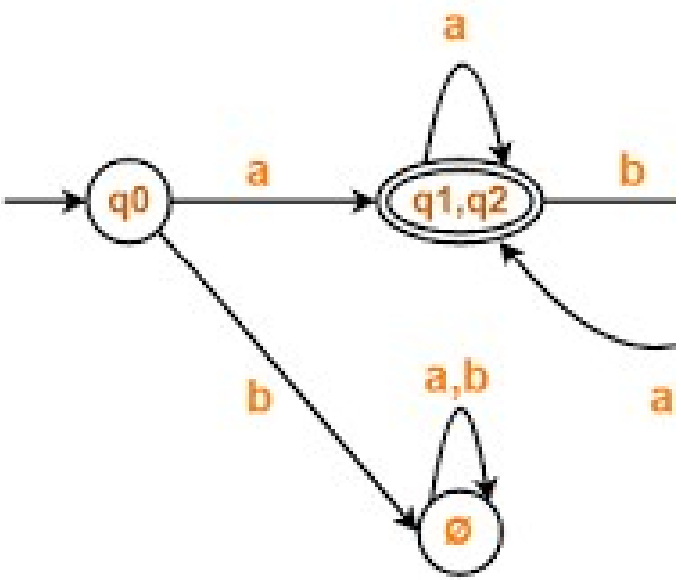


δ_D	0	1
$[q_0]$	$[q_0]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_0,q_1, q_2]$	$[q_1,q_2]$
$*[q_0, q_1,q_2]$	$[q_0,q_1, q_2]$	$[q_1,q_2]$



Ex Convert the following Non-Deterministic Finite Automata (NFA) to Deterministic Finite Automata (DFA)





δ_D	0	1
$[q_0]$	$[q_1, q_2]$	Φ
$*[q_1, q_2]$	$[q_1, q_2]$	$[q_2]$
$[q_2]$	$[q_1, q_2]$	$[q_2]$
Φ	Φ	Φ

Construct Deterministic Finite Automata (DFA) from the given Transition Table

Construct a deterministic finite automaton equivalen

$$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}), \delta, q_0$$

where δ is given by Table

TABLE State Table

State/Σ	a
$\rightarrow q_0$	q_0, q_1
q_1	q_2

State/Σ	a
$[q_0]$	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2, q_3]$
$[q_0, q_1, q_3]$	$[q_0, q_1, q_2]$

Construct Deterministic Finite Automata (DFA) from the given Transition Table

Find a deterministic acceptor equivalent to

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q$$

where δ is as given by Table

TABLE State Table

State/Σ	a
$\rightarrow q_0$	q_0, q_1

Correctness of subset construction

Theorem: If D is the DFA constructed from NFA N by subset construction, then L(D)=L(N)

- Proof:
  - Show that $\delta_D^*(\{q_0\},w) \equiv \delta_N^*(q_0,w\}$, for all w
  - Using induction on w's length:
    - Let w = xa
    - $\delta_D^*(\{q_0\},xa) \equiv \delta_D(\delta_N^*(q_0,x\}, a) \equiv \delta_N^*(q_0,w\}$

A few subtle properties of DFAs and NFAs

- The machine never really terminates.
  - It is always waiting for the next input symbol or making transitions.
- The machine decides when to consume the next symbol from the input and when to ignore it.
  - (but the machine can never skip a symbol)
- => A transition can happen even *without* really consuming an input symbol (think of consuming ε as a free token) – if this happens, then it becomes an ε-NFA (see next few slides).
- A single transition *cannot* consume more than one (non-ε) symbol.

FA with ε-Transitions

- We can allow explicit ε-transitions in finite automata
  - i.e., a transition from one state to another state without consuming any additional input symbol
  - Explicit ε-transitions between different states introduce non-determinism.
  - Makes it easier sometimes to construct NFAs
    - E.g. when constructing an NFA using RE

Definition:* ε *-NFAs are those NFAs with at least one explicit* ε*-transition defined.

- ε -NFAs have one more column in their transition table

NFA with epsilon transition

We extend the class of NFAs by allowing instantaneous ε transitions –

- *ε -NFAs add a convenient feature but (in a sense) they bring us nothing new. They do not extend the class of languages that can be represented.*
- *Both NFAs and ε -NFAs recognize exactly the same languages.*

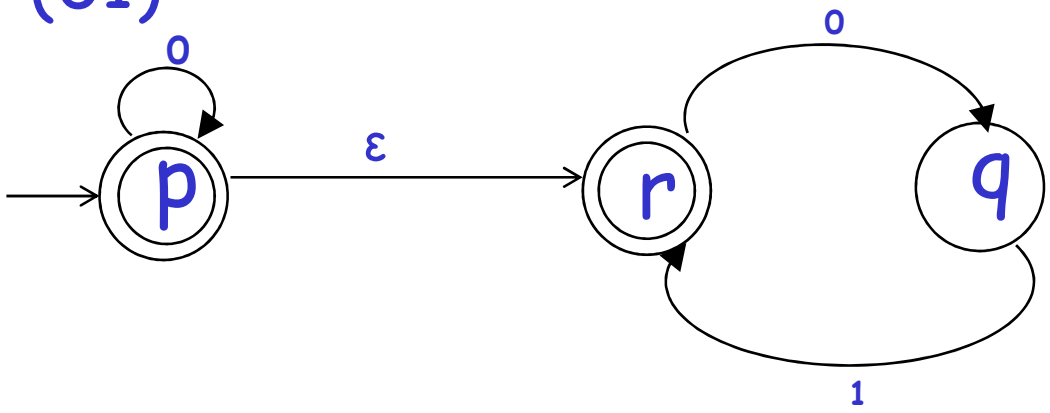
Example of an ε-NFA

Eg.1

RE = 0*(01)*

L_1 = 0*

L_2 =(01)*



E.g.2

RE = (aab)*(a+aba)*

ε-NFA

- An ε-NFA is also defined by the 5-tuple:
  - $\{Q, \Sigma, q_0, F, \delta\}$
- ε-NFA consists of:
  - Q ==> a finite set of states
  - Σ ==> a finite set of input symbols (alphabet)
  - q_0 ==> a start state
  - F ==> set of accepting states
  - δ ==> a transition function, which is a mapping between $Q \times \{\Sigma \cup \{\varepsilon\}\}$ ==> subset of Q (2^Q)

Extension of δ for NFA

- Let $M = \{Q, \Sigma, q_0, F, \delta\}$ be an NFA
- The function $\delta^* : Q \times (\Sigma^*) \Longrightarrow 2^Q$ is defined as

- <u>Basis:</u> for any $q \in Q$, $\delta^*(q, \varepsilon) = \{q\}$

- <u>Induction:</u>
  - Let $\delta^*(q_0, w) = \{p_1, p_2 \ldots, p_k\}$
  - $\delta(p_i, a) = S_i$ for $i = 1, 2 \ldots, k$

  - Then, $\delta^*(q_0, wa) = S_1 \cup S_2 \cup \ldots \cup S_k$

    or $\delta^*(q_0, wa) = \bigcup_{s \in \delta^*(q_0, w)} \delta(s, a)$

Extension of δ for ε-NFA

- Let $M = \{Q, \Sigma, q_0, F, \delta\}$ be an NFA
- The function $\delta^* : Q \times (\Sigma^* \cup \{\varepsilon\}) \Longrightarrow 2^Q$ is defined as

- Basis: for any $q \in Q$, $\delta^*(q,\varepsilon) = \varepsilon\_closure\{q\}$

- Induction:
  - Let $\delta^*(q_0,w) = \{p_1,p_2\ldots,p_k\}$
  - $\delta(p_i,a) = S_i$ for $i=1,2\ldots,k$

  - Then, $\delta^*(q_0,wa) = S_1 \cup S_2 \cup \ldots \cup S_k$

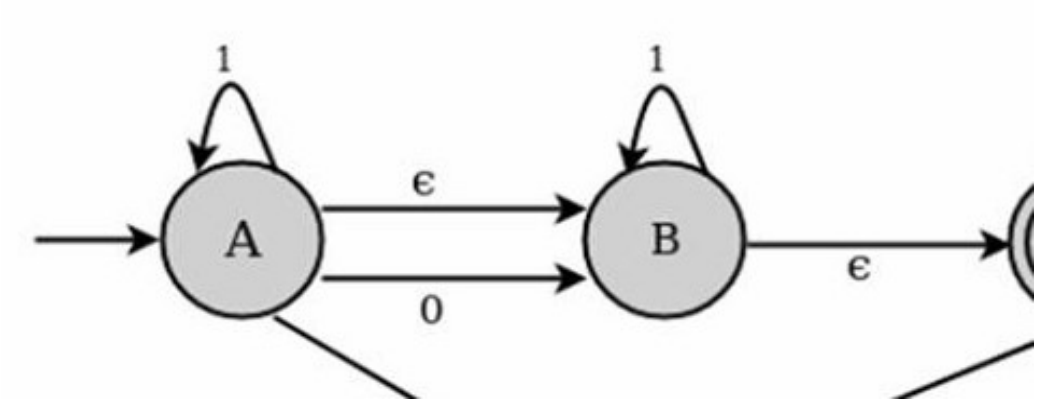
    or $\delta^*(q_0,wa) = \varepsilon\_closure\left(\bigcup_{s \in \delta^*(q_0,w)} \delta(s,a)\right)$

NFA with epsilon transition

Epsilon (ε) - closure

- *Epsilon closure for a given state X is a set of states which can be reached from the states X with only (null) or ε moves including the state X itself.*

- *In other words, ε -closure for a state can be obtained by union operation of the ε -closure of the states which can be reached from X with a single ε move in a recursive manner.*



For the given example, ε closure are as follow

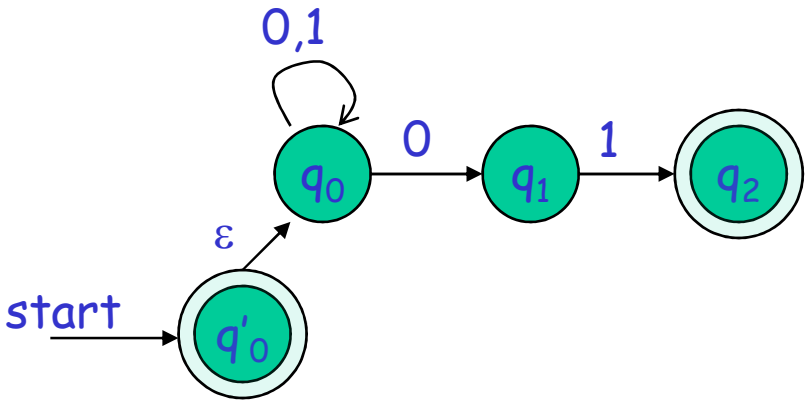
- ε-closure(A)= {A, B,C}
- ε-closure(B)= {B,C}
- ε-closure(C)={C}

Language of an ε-NFA

- An ε-NFA accepts w if *there exists at least one* path from the start state to an accepting (or final) state that is labeled by w
- $L(N) = \{ w \mid \delta^*(q_0, w) \cap F \neq \Phi \}$

Example of an ε-NFA

L = {w | w is empty, or if non-empty will end in 01}



δ_E	0	1	ε	ε_closer
→ *q'_0	Ø	Ø	$\{q_0\}$	$\{q'_0, q_0\}$
q_0	$\{q_0, q_1\}$	$\{q_0\}$	Ø	$\{q_0\}$
q_1	Ø	$\{q_2\}$	Ø	$\{q_1\}$
*q_2	Ø	Ø	Ø	$\{q_2\}$

ECLOSE(q'_0)
ECLOSE(q_0)
ECLOSE(q_1)
ECLOSE(q_2)

- ε-closure of a state q, ***ECLOSE(q)***, is the set of all states (including itself) that can be *reached* from q by repeatedly making an arbitrary number of ε-transitions.

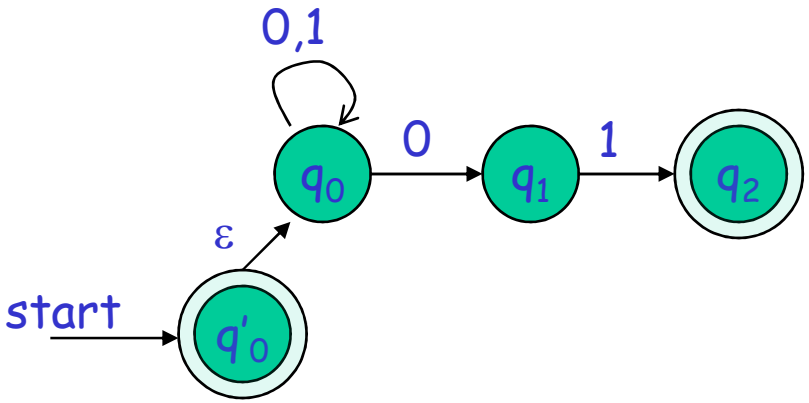
To simulate any transition:

Step 1) Go to all immediate destination states.

Step 2) From there go to all their ε-closure states as well.

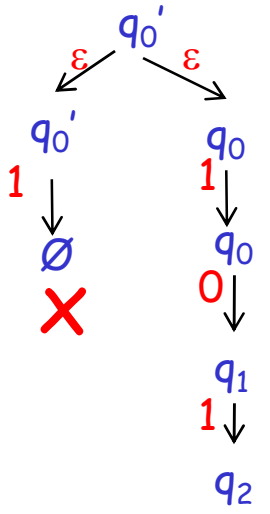
Example of an ε-NFA

L = {w | w is empty, or if non-empty will end in 01}



δ_E	0	1	ε	
→ *q'_0	Ø	Ø	$\{q'_0, q_0\}$	ECLOSE(q'_0)
q_0	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0\}$	ECLOSE(q_0)
q_1	Ø	$\{q_2\}$	$\{q_1\}$	
*q_2	Ø	Ø	$\{q_2\}$	

Simulate for w=101:

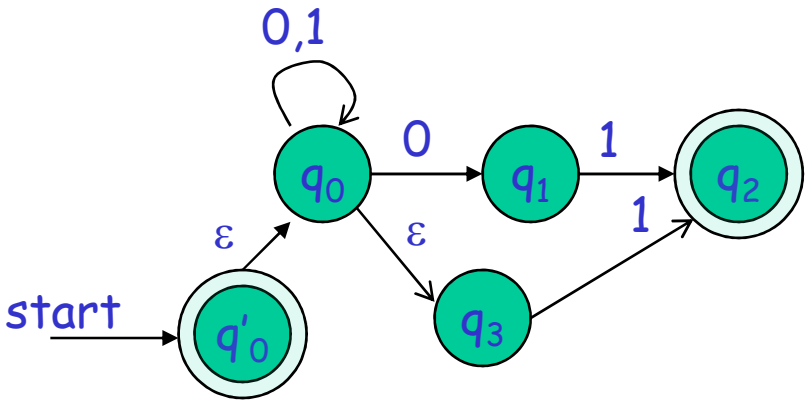


To simulate any transition:

Step 1) Go to all immediate destination states.

Step 2) From there go to all their ε-closure states as well.

Example of another ε-NFA

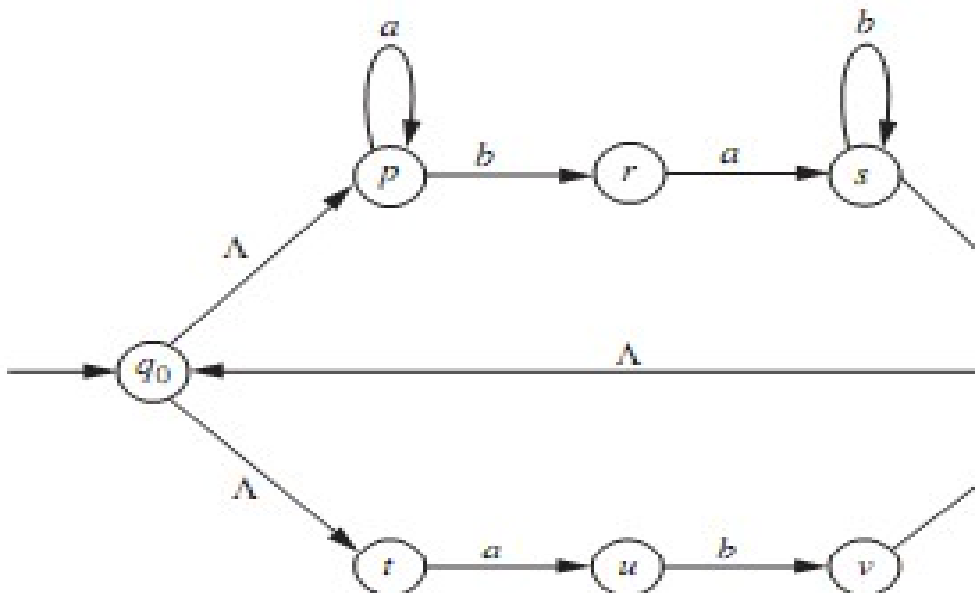


Simulate for w=101:

?

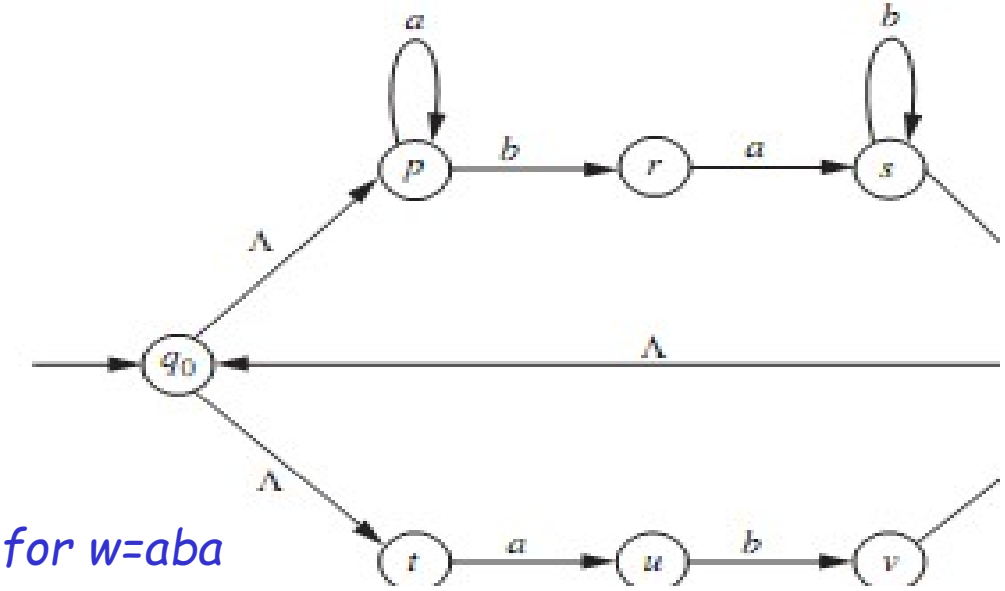
δ_E	0	1	ε	ε_closer
→ *q'_0	Ø	Ø	$\{q_0\}$	$\{q'_0,q_0,q_3\}$
q_0	$\{q_0,q_1\}$	$\{q_0\}$	$\{q_3\}$	$\{q_0,q_3\}$
q_1	Ø	$\{q_2\}$	Ø	$\{q_1\}$
*q_2	Ø	Ø	Ø	$\{q_2\}$
q_3	Ø	$\{q_2\}$	Ø	$\{q_3\}$

Example



- Calculate ε –*closure(s)*

  = {s, w, q_0,p, t}



Calculate $\delta^*(q_0,w)$ for w=aba

$$\delta^*(q_0, \Lambda) = \Lambda(\{q_0\})$$
$$= \{q_0, p, t\}$$

$$\delta^*(q_0, a) = \Lambda\left(\bigcup\{\delta(k, a) \mid k \in \delta^*(q_0, \Lambda)\}\right)$$
$$= \Lambda(\delta(q_0, a) \cup \delta(p, a) \cup \delta(t, a))$$
$$= \Lambda(\emptyset \cup \{p\} \cup \{u\})$$
$$= \Lambda(\{p, u\})$$
$$= \{p, u\}$$

$$\delta^*(q_0, ab) = \Lambda\left(\bigcup\{\delta(k, b) \mid k \in \{p, u\}\}\right)$$
$$= \Lambda(\delta(p, b) \cup \delta(u, b))$$
$$= \Lambda(\{r, v\})$$
$$= \{r, v, w, q_0, p, t\}$$

$$\delta^*(q_0, aba) = \Lambda\left(\bigcup\{\delta(k, a) \mid k \in \{r, v, w, q_0, p, t\}\}\right)$$
$$= \Lambda(\delta(r, a) \cup \delta(v, a) \cup \delta(w, a) \cup \delta(q_0, a) \cup \delta(p,$$
$$= \Lambda(\{s\} \cup \{v\} \cup \emptyset \cup \emptyset \cup \{p\} \cup \{u\})$$
$$= \Lambda(\{s, v, p, u\})$$
$$= \{s, v, p, u, w, q_0, t\}$$

Equivalency of DFA, NFA, ε-NFA

- Theorem: A language L is accepted by some ε-NFA if and only if L is accepted by some DFA

- Theorem: A language L is accepted by some ε-NFA then there is an NFA that also accepts L.

- Implication:
  - DFA ≡ NFA ≡ ε-NFA
  - (all accept Regular Languages)

Eliminating ε-transitions

Theorem: For every language $L \subseteq \Sigma^*$ accepted by an ε-NFA $E = \{Q, \Sigma, \delta_E, q_0, F_E\}$, there is an NFA N with no ε-transitions that also accepts L.

Let $E = \{Q, \Sigma, \delta_E, q_0, F_E\}$ be an ε-NFA

Goal: To build NFA $N = \{Q, \Sigma, \delta_N, q_0, F_N\}$ s.t. $L(N) = L(E)$

Construction:

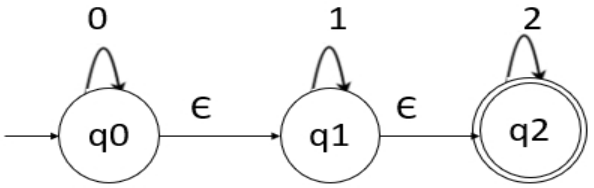
for every $q \in Q$, $\delta_N(q, \varepsilon) = \Phi$ and for every $q \in Q$ and every $a \in \Sigma$,

$$\delta_N(q, a) = \delta_E^*(q, a)$$

And we define

$$F_N = F_E \cup \{q_0\} \quad \text{if } \varepsilon \in L.$$
$$= F_E \quad \text{if not.}$$

Convert the given NFA with epsilon to NFA without epsilon.



Step:1

ε-closure(q0) = {q0,q1,q2}

ε-closure(q1) = {q1,q2}

ε-closure(q2) = {q2}

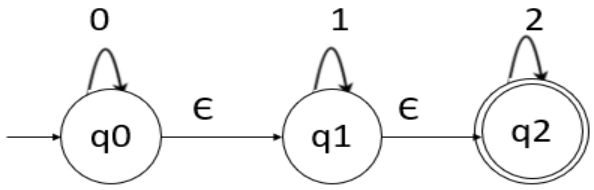
δ'(q0, 2) = ε-closure(δ(δ*(q0, ε),2)
= ε-closure(δ(q0,q1,q2), 2
= ε-closure(δ(q0, 2) ∪ δ(q
= ε-closure(Φ ∪ Φ∪ q2)
= ε-closure(q2)
= {q2}

Step:2 Now we will obtain δ' transitions for each state on each input symbol as shown below

δ'(q0, 0) = ε-closure(δ(δ*(q0, ε),0))
= ε-closure(δ(ε-closure(q0),0))
= ε-closure(δ(q0,q1,q2), 0))
= ε-closure(δ(q0, 0) ∪ δ(q1, 0) ∪ δ(q2, 0))
= ε-closure(q0 ∪ Φ ∪ Φ)
= ε-closure(q0)
= {q0,q1, q2}

δ'(q0, 1) = ε-closure(δ(δ*(q0, ε),1))
= ε-closure(δ(q0,q1,q2), 1))
= ε-closure(δ(q0, 1) ∪ δ(q1, 1) ∪ δ(q2, 1))
= ε-closure(Φ ∪q1 ∪ Φ)
= ε-closure(q1)
= {q1, q2}

Convert the given NFA with epsilon to NFA without epsilon.



Step:1

ε-closure(q0) = {q0,q1,q2}

ε-closure(q1) = {q1,q2}

ε-closure(q2) = {q2}

δ'(q1, 0) = ε-closure(δ(δ*(q1, ε),0))
= ε-closure(δ(q1,q2), 0))
= ε-closure(δ(q1, 0) U δ(q2, 0))
= ε-closure(Φ ∪ Φ)
= ε-closure(Φ)
= Φ

δ'(q1,1) = ε-closure(δ(δ*(q1, ε),1))
= ε-closure(δ(q1,q2), 1))
= ε-closure(δ(q1, 1) U δ(q2, 1))
= ε-closure(q1 ∪ Φ)
= ε-closure(q1)
= {q1,q2}

δ'(q1, 2) = ε-closure(δ(δ*(q1, ε),2))
= ε-closure(δ(q1,q2), 2))
= ε-closure(δ(q1, 2) U δ(q2, 2))
= ε-closure(Φ ∪ q2)
= ε-closure(q2)

δ'(q2, 0) = ε-closure(δ(δ*(q2, ε),0))
= ε-closure(δ(q2), 0))
= ε-closure(δ(q2, 0))
= ε-closure(Φ)
= Φ

δ'(q2, 1) = ε-closure(δ(δ*(q2, ε),1))
= ε-closure(δ(q2), 1)
= ε-closure(δ(q2, 1))
= ε-closure(Φ)
= Φ

δ'(q2, 2) = ε-closure(δ(δ*(q2, ε),))
= ε-closure(δ(q2), 2))
= ε-closure(δ(q2, 2))
= ε-closure(q2)
= {q2}

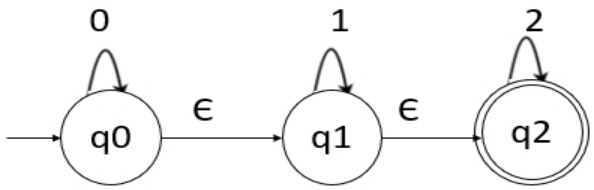
Step:3 Now, we will summarize all the computed δ' transitions as given below −

δ'(q0,0)={q0,q1,q2}
δ'(q0,1)={q1,q2}
δ'(q0,2)={q2}

δ'(q1,0)= { Φ }
δ'(q1,1)={q1,q2}
δ'(q1,2)={q2}

δ'(q2,0)={ Φ }
δ'(q2,1)={ Φ }
δ'(q2,2)={q2}

Convert the given NFA with epsilon to NFA without epsilon.



Step:1

ε-closure(q0) = {q0,q1,q2}

ε-closure(q1) = {q1,q2}

ε-closure(q2) = {q2}

Step:3 Now, we will summarize all the computed δ' transitions as given below –

δ'(q0,0)={q0,q1,q2}

δ'(q0,1)={q1,q2}

δ'(q0,2)={q2}

δ'(q1,0)= { Φ }

δ'(q1,1)={q1,q2}

δ'(q1,2)={q2}

δ'(q2,0)={ Φ }

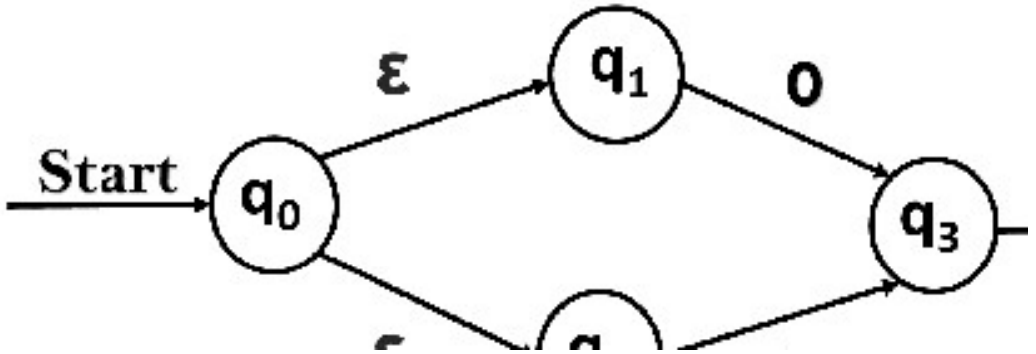
δ'(q2,1)={ Φ }

δ'(q2,2)={q2}

Step:4-The transition table for **NFA without epsilon** is given below-

States\inputs	0	1	2
q0	{q0,q1,q2}	{q1,q2}	{q2}
q1	Φ	{q1,q2}	{q2}
q2	Φ	Φ	{q2}

Convert the given NFA with epsilon to DFA.



Eliminating ε-transitions

Let $E = \{Q_E, \Sigma, \delta_E, q_0, F_E\}$ be an ε-NFA

Goal: To build DFA $D=\{Q_D, \Sigma, \delta_D, \{q_D\}, F_D\}$ s.t. $L(D)=L(E)$

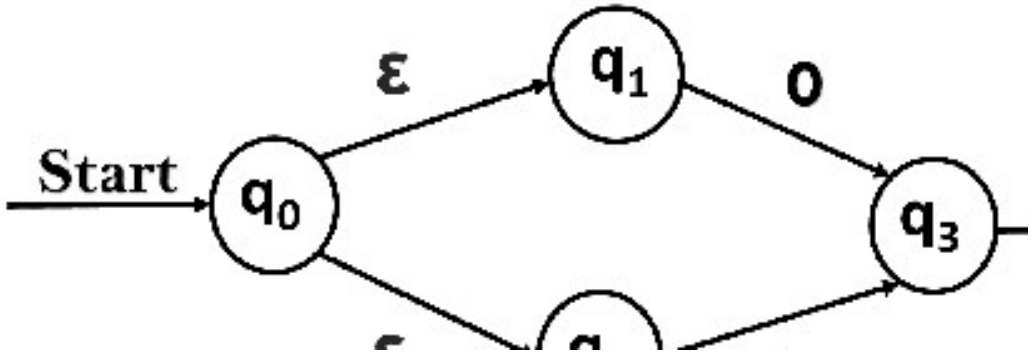
Construction:

1. Q_D= all reachable subsets of Q_E factoring in ε-closures
2. q_D = ECLOSE(q_0)
3. F_D=subsets S in Q_D s.t. $S \cap F_E \neq \Phi$
4. δ_D: for each subset S of Q_E and for each input symbol $a \in \Sigma$:
   - Let $r = \bigcup_{p \text{ in } s} \delta_E(p,a)$ // go to destination states
   - $\delta_D(S,a) = \bigcup_{r \text{ in } R} \text{ECLOSE}(r)$ // from there, take a union of all their ε-closures

ε-NFA to DFA

- Steps for converting NFA with ε to DFA:
- **Step 1**: We will take the ε-closure for the starting state of NFA as a starting state of DFA.
- **Step 2**: Find the states for each input symbol that can be traversed from the present. That means the union of transition value and their closures for each state of NFA present in the current state of DFA.
- **Step 3**: If we found a new state, take it as current state and repeat step 2.
- **Step 4**: Repeat Step 2 and Step 3 until there is no new state present in the transition table of DFA.
- **Step 5**: Mark the states of DFA as a final state which contains the final state of NFA.

Convert the given NFA with epsilon to DFA



ε-closure {q0} = {q0, q1, q2}
ε-closure {q1} = {q1}
ε-closure {q2} = {q2}
ε-closure {q3} = {q3}
ε-closure {q4} = {q4}

Step:1 find ε-closure of initial state

ε-closure {q0} = {q0, q1, q2} **let say it as *A***

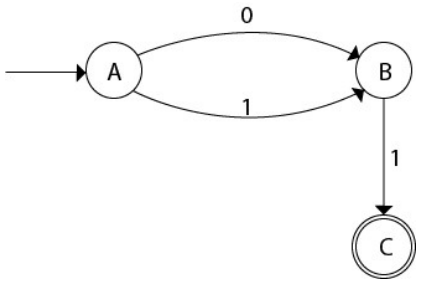
Step: 2

δ'(*A*, 0) = ε-closure {δ((q0, q1, q2), 0) }
= ε-closure {δ(q0, 0) ∪ δ(q1, 0) ∪ δ(q2, 0)}
= ε-closure {q3}
= {q3} **call it as state B.**

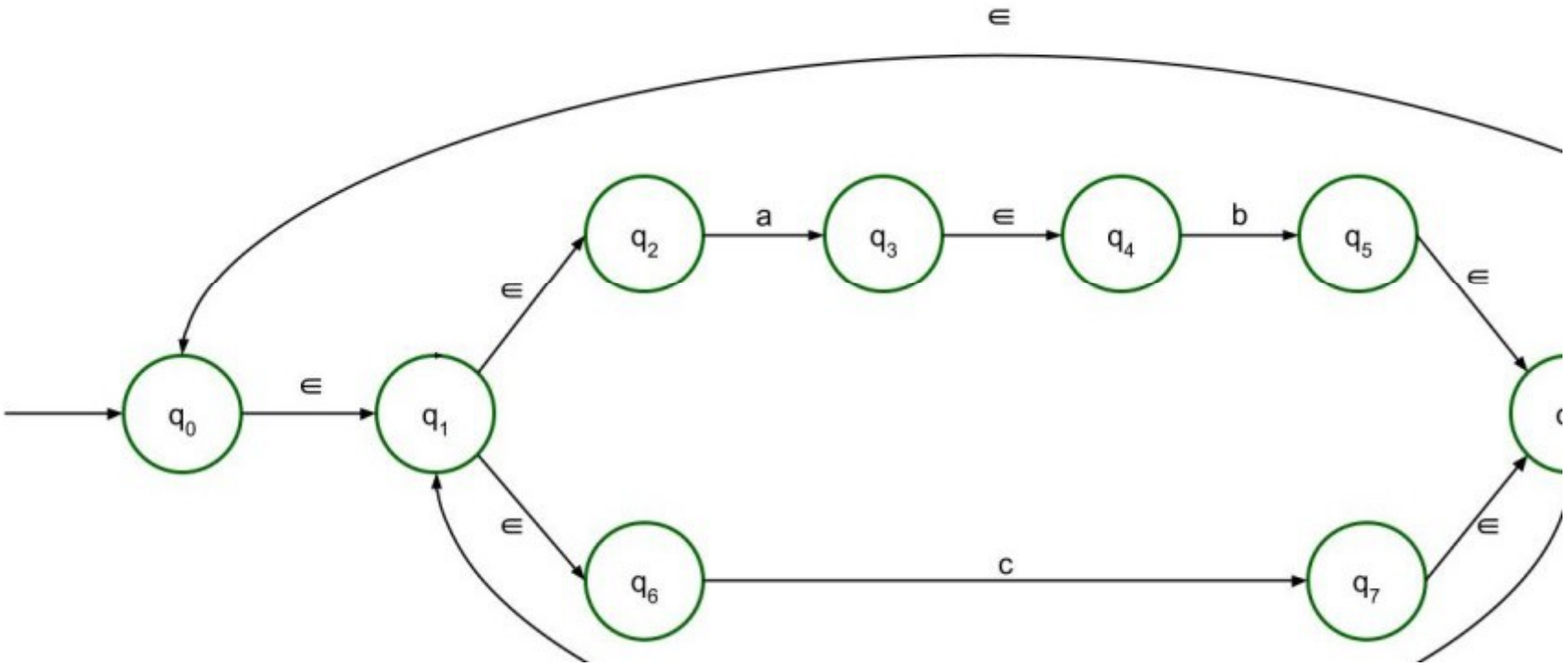
δ'(*A*, 1) = ε-closure {δ((q0, q1, q2), 1) }
= ε-closure {δ((q0, 1) ∪ δ(q1, 1) ∪ δ(q2, 1)}
= ε-closure {q3}
= {q3} = B.

δ'(B, 0) = ε-closure {δ(q3, 0) }
= ϕ
δ'(B, 1) = ε-closure {δ(q3, 1) }
= ε-closure {q4}
= {q4} **i.e. state *C***

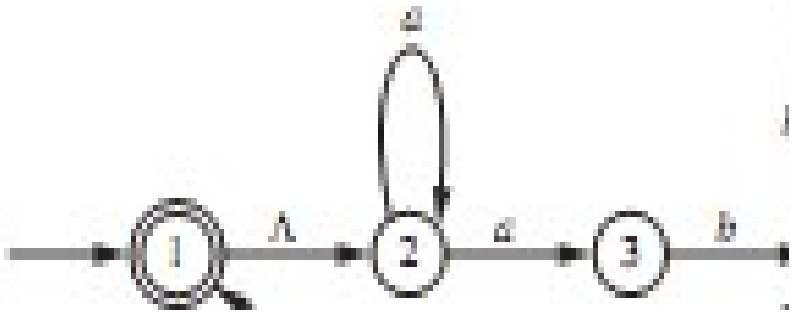
δ'(*C*, 0) = ε-closure {δ(q4, 0) }
= ϕ
δ'(*C*, 1) = ε-closure {δ(q4, 1) }
= ϕ



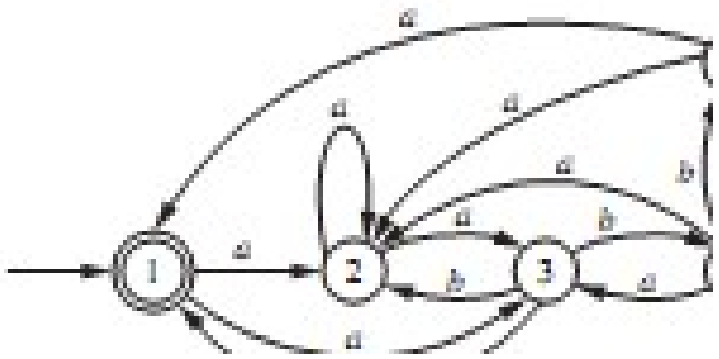
Example:Convert ε-NFA to DFA



Example: ε-NFA ➔ DFA



ε-NFA



NFA

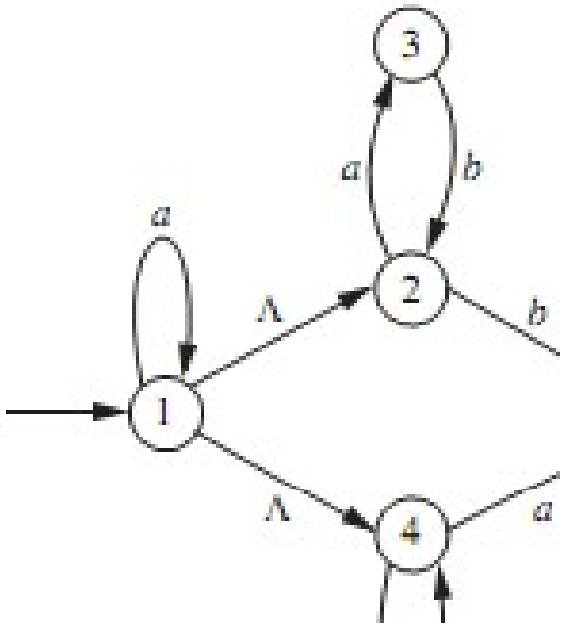
q	δ(q,a)	δ(q,b)	δ(q, ε)	δ*(q,a)	δ*(q,b)
1	ϕ	ϕ	{2}	{2,3}	ϕ
2	{2,3}	ϕ	ϕ	{2,3}	ϕ
3	ϕ	{4}	ϕ	ϕ	{1,2,4}
4	ϕ	{5}	{1}	{2,3}	{5}
5	{4}	ϕ	ϕ	{1,2,4}	ϕ

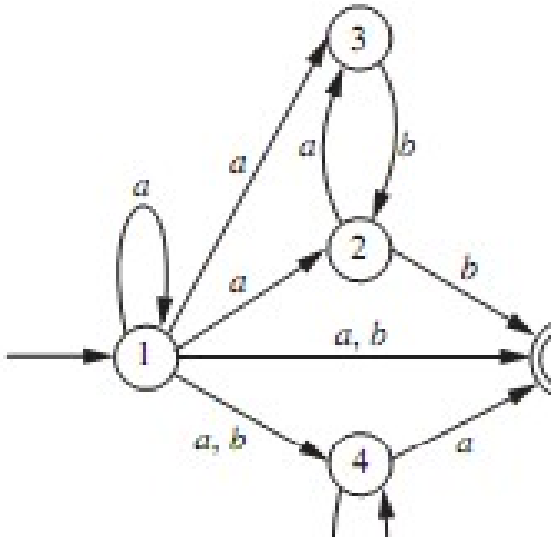
Example: NFA ➔ DFA

q	δ*(q,a)	δ*(q,b)
1	{2,3}	ϕ
2	{2,3}	ϕ
3	ϕ	{1,2,4}
4	{2,3}	{5}
5	{1,2,4}	ϕ

q	δ(q,a)	δ(q,b)
{1}	{2,3}	ϕ
{2,3}	{2,3}	{1,2,4}
ϕ	ϕ	ϕ
{1,2,4}	{2,3}	{5}
{5}	{1,2,4}	ϕ

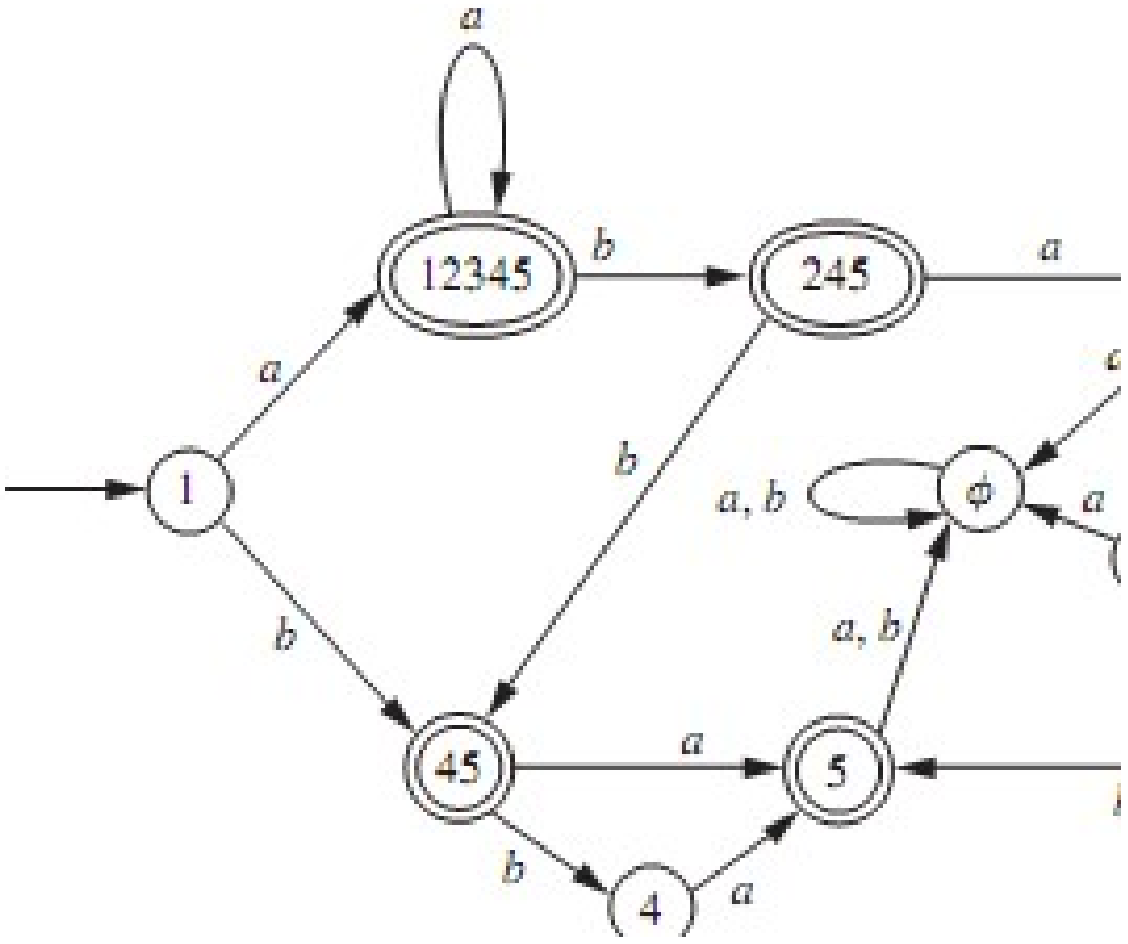
Example: ε-NFA ➔ NFA ➔ DFA



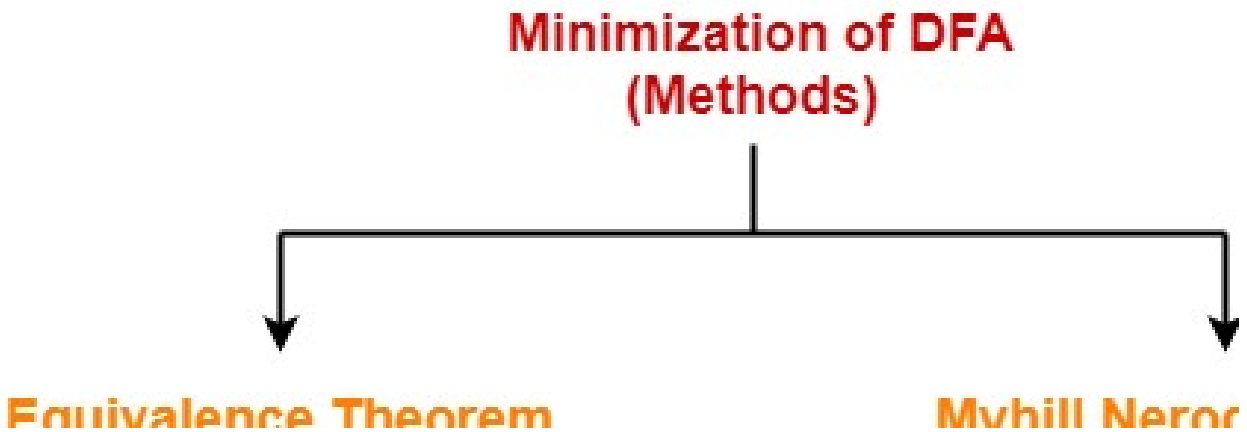


q	$\delta(q, a)$	$\delta(q, b)$	$\delta(q, \Lambda)$	$\delta^*(q, a)$
1	{1}	∅	{2, 4}	{1, 2, 3, 4, 5}
2	{3}	{5}	∅	{3}
3	∅	{2}	∅	∅
4	{5}	{4}	∅	{5}

DFA



Minimization of DFA

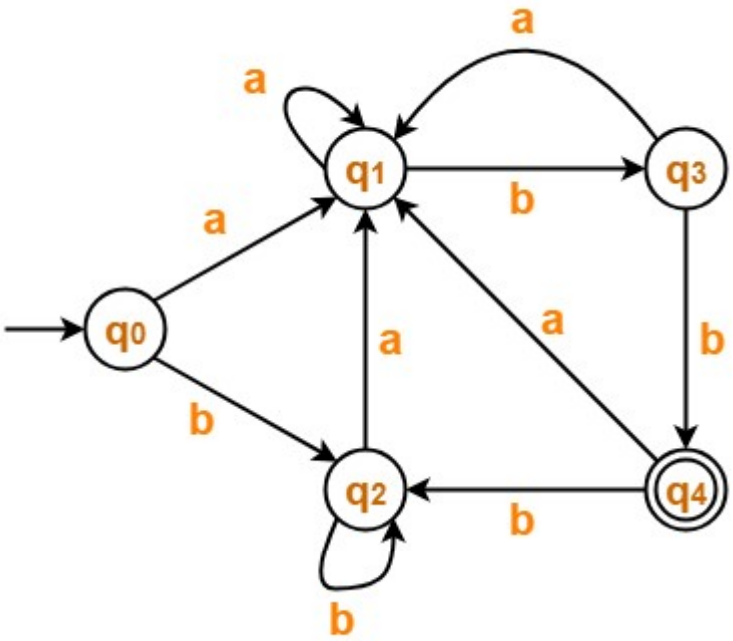


Minimization of DFA using Equivalence Theorem

- **Step 1**: We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called P_0.
- **Step 2**: Initialize k = 1
- **Step 3**: Find P_k by partitioning the different sets of P_{k-1}. In each set of P_{k-1}, we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in P_k.
- **Step 4**: Stop when $P_k = P_{k-1}$ (No change in partition)
- **Step 5**: All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in P_k.

Two states (qi, qj) are distinguishable in partition P_k if for any input symbol a, δ (qi, a) and δ (qj, a) are in different sets in partition P_{k-1}.

Minimization of DFA



	a	b
->q0	q1	q2
q1	q1	q3
q2	q1	q2
q3	q1	q4
*q4	q1	q2

$P_0 = \{ q_0 , q_1 , q_2 , q_3 \} \{ q_4 \}$

$P_1 = \{ q_0 , q_1 , q_2 \} \{ q_3 \} \{ q_4 \}$

$P_2 = \{ q_0 , q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$

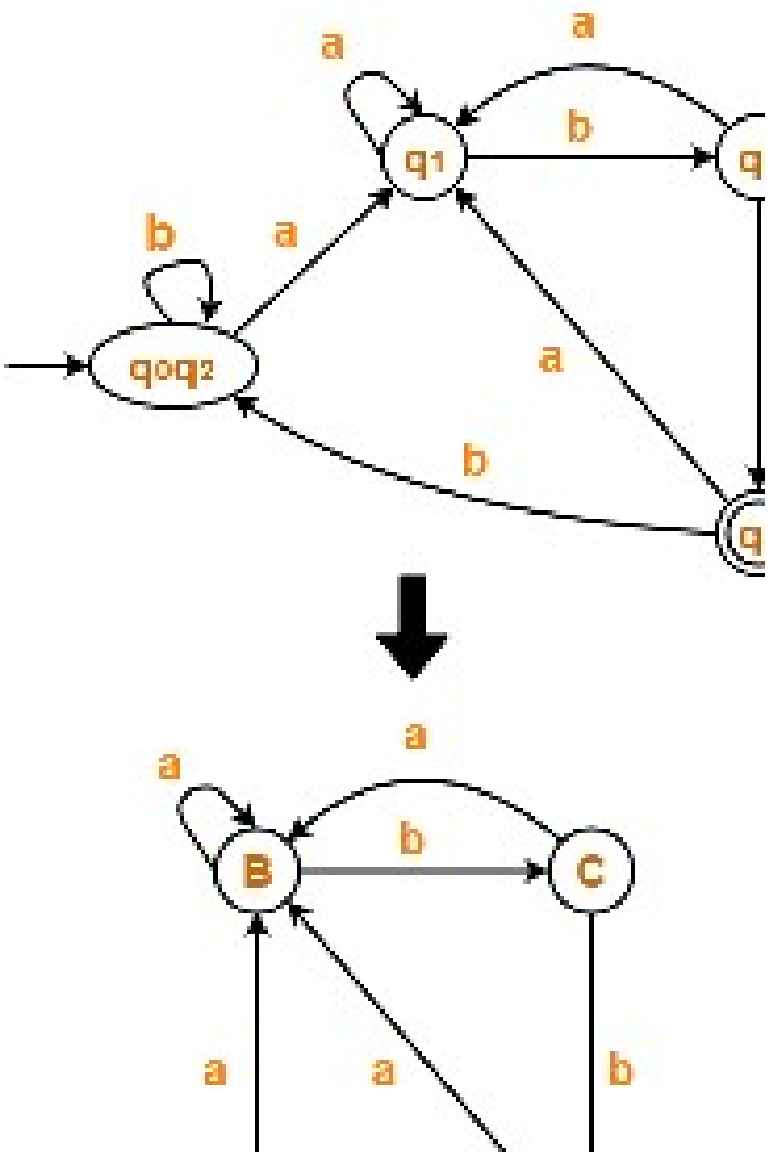
$P_3 = \{ q_0 , q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$

Since $P_3 = P_2$, so we stop.

From P_3, we infer that states q_0 and q_2 are equivalent and can be merged together.

Minimization of DFA

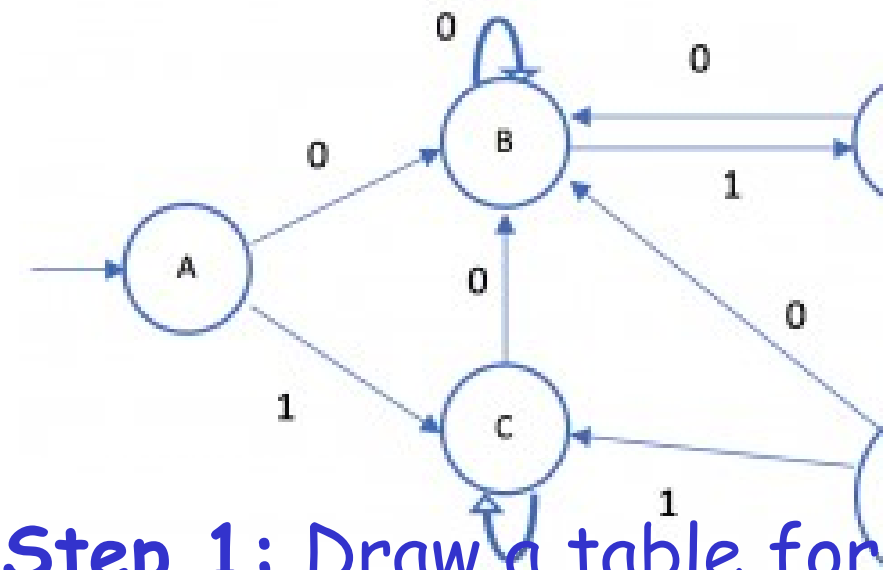
	a	b
->q0	q1	q2
q1	q1	q3
q2	q1	q2
q3	q1	q4
*q4	q1	q2



Minimization of DFA – Table Filling Method

- Draw a table for all pairs of states (P,Q)
- Mark all pairs where $P \in F$ and $Q \notin F$.
- *If there are any unmarked pairs (P,Q) such that* [δ(P,x), δ(Q,x)] is marked, then mark [P,Q], where 'x' is an input symbol. Repeat this until no more markings can be made.
- Combine all the unmarked pairs and make them a single state in the minimized DFA.

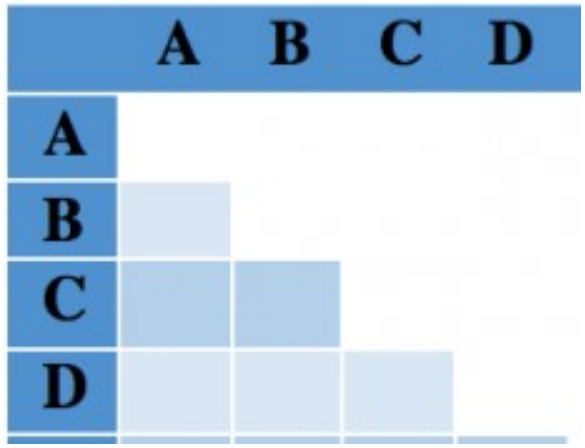
Example



Step 1: Draw a table for all pairs of states (P, Q)

	A	B	C	D
A				
B				
C				
D				

- Step 2: Mark all pairs where $P \in F$ and $Q \notin F$.



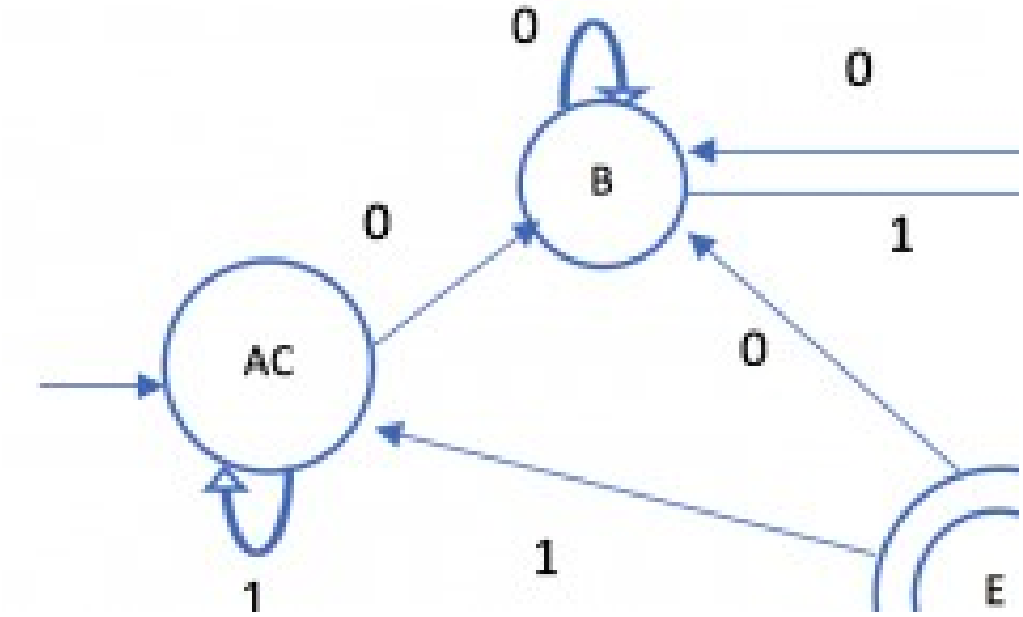
- Step 3: If there are any Unmarked pairs (P, Q) such that [δ(P, x),δ(Q, x)] is marked, then mark [P, Q] where 'x' is an input symbol. Repeat this until no more marking can be made.

-

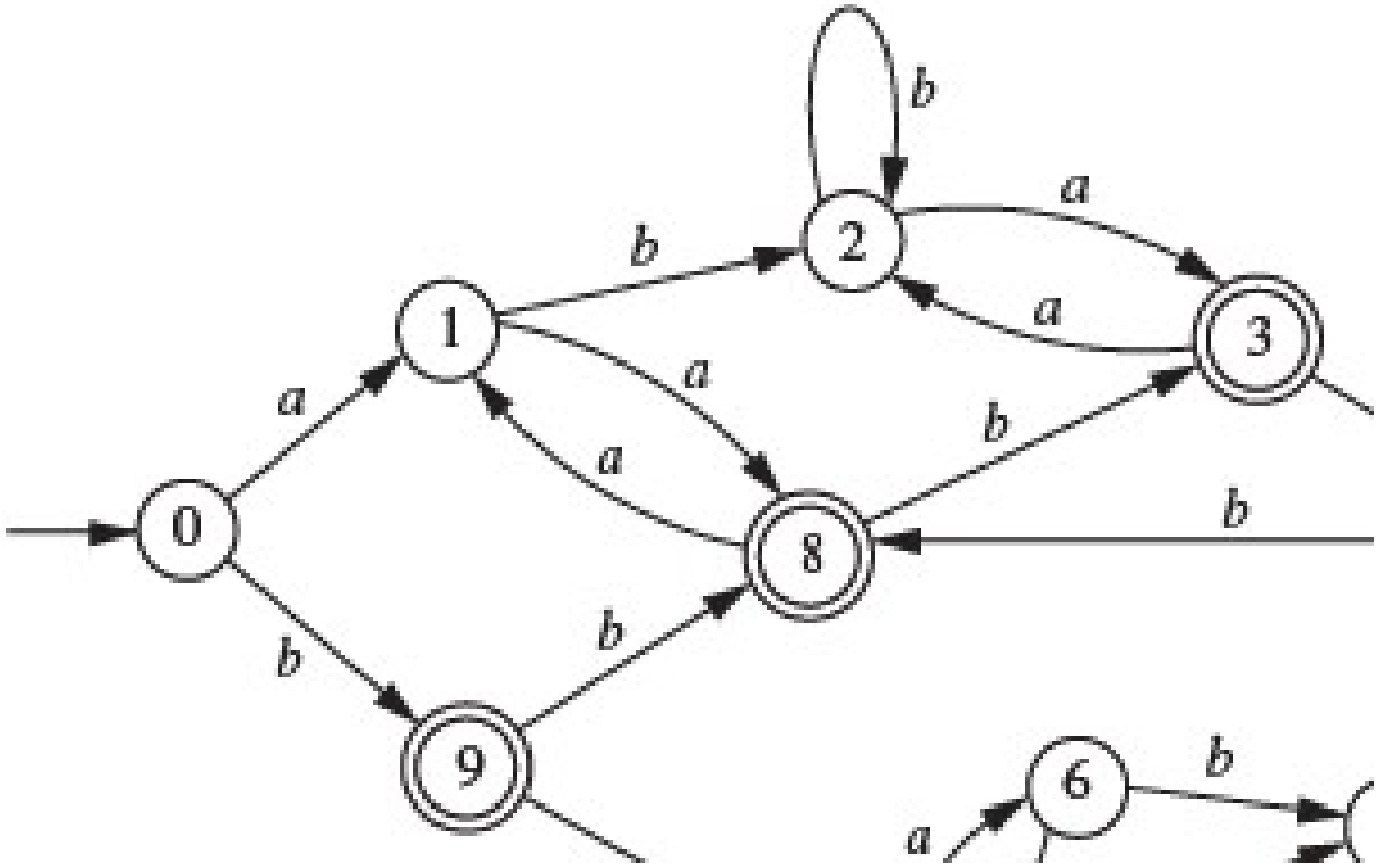
	A	B	C	D	E
A					
B	+				
C		+			
D	+	+	+		
E	+	+	+	+	

(B, A) For input 0 (B,0) = B (A, 0) = B	(B, A) For input 1 (B,1) = D (A,1) = C	(C, A) For input 0 (C,0) = B (A,0) = B	(C, A (C,1) (A,1)
(C, B) For input 0 (C,0) = B (B,0) = B	(C, B) For input 1 (C,1) = C (B,1) = D	(D, A) For input 0 (D,0) = B (A,0) = B	(D, A (D,1) (A,1)

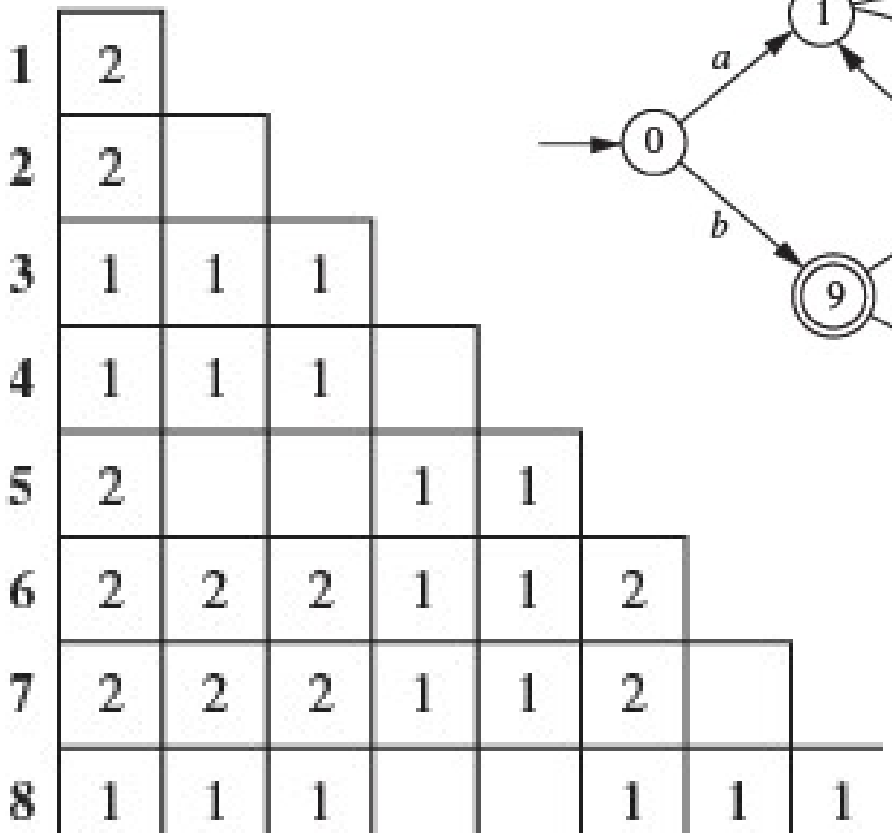
- **Step 4 :(A, C), B, D, E**



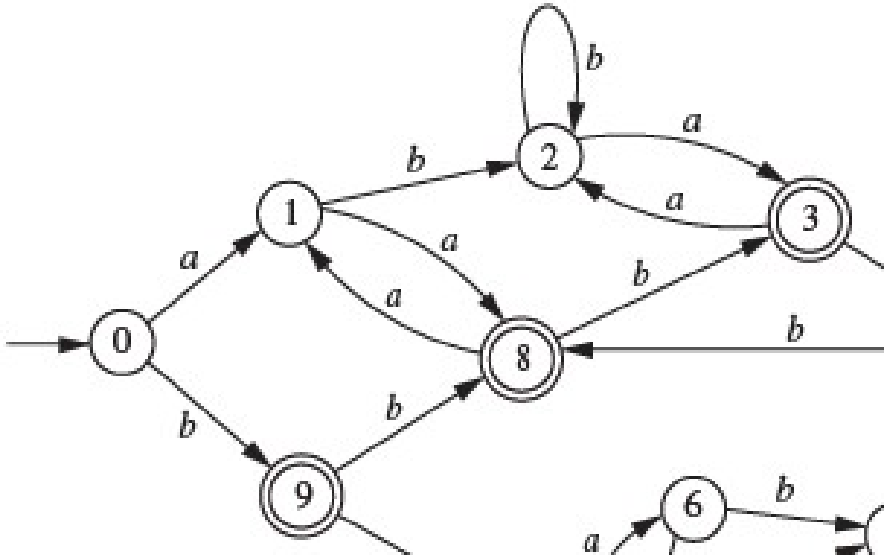
Example



Example

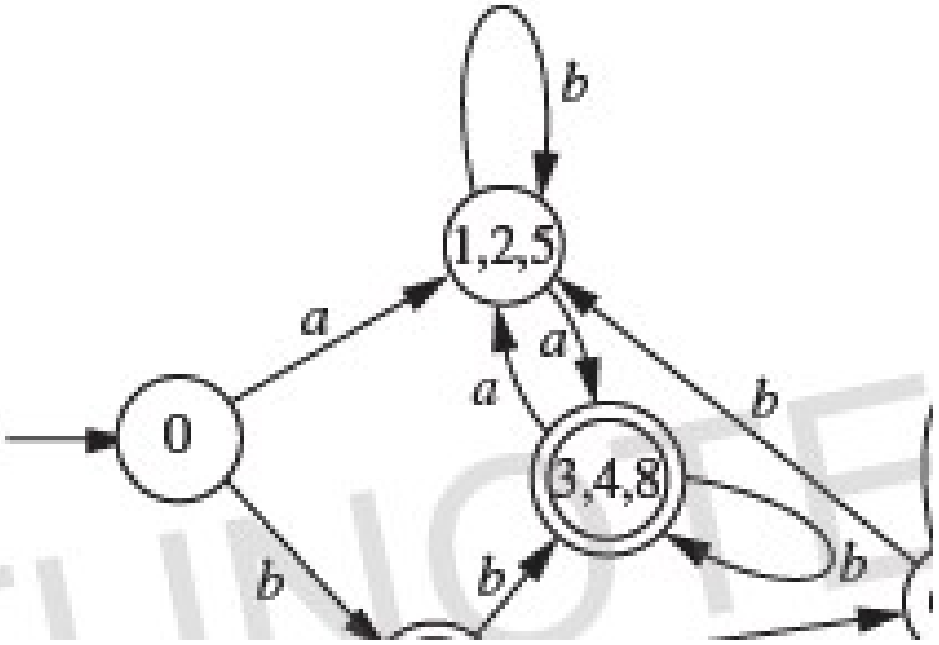


1	2							
2	2							
3	1	1	1					
4	1	1	1					
5	2			1	1			
6	2	2	2	1	1	2		
7	2	2	2	1	1	2		
8	1	1	1			1	1	1



Example

1	2							
2	2							
3	1	1	1					
4	1	1	1					
5	2			1	1			
6	2	2	2	1	1	2		
7	2	2	2	1	1	2		
8	1	1	1			1	1	1



Summary

- DFA
  - Definition
  - Transition diagrams & tables
- Regular language
- NFA
  - Definition
  - Transition diagrams & tables
- DFA vs. NFA
- NFA to DFA conversion using subset construction
- Equivalency of DFA & NFA
- Removal of redundant states and including dead states
- ε-transitions in NFA
- DFA Minimization