Chapter 4
Finite Automata

by

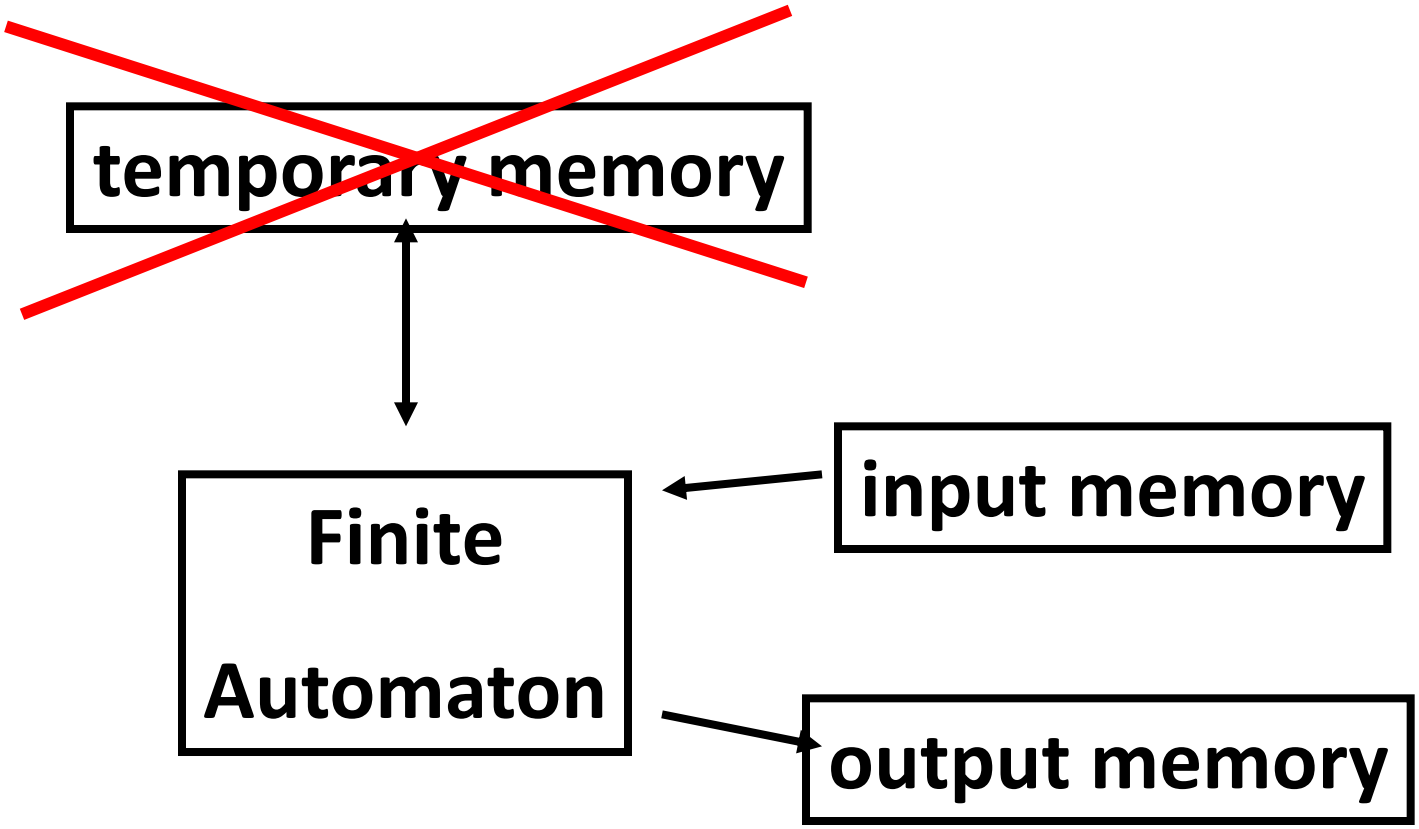
Dr Zalmiyah Zakaria

Models of Computation

Different Kinds of Automata

- Automata are distinguished by the temporary memory
  - **Finite Automata** : no temporary memory
  - **Pushdown Automata** : stack
  - **Turing Machines** : random access memory

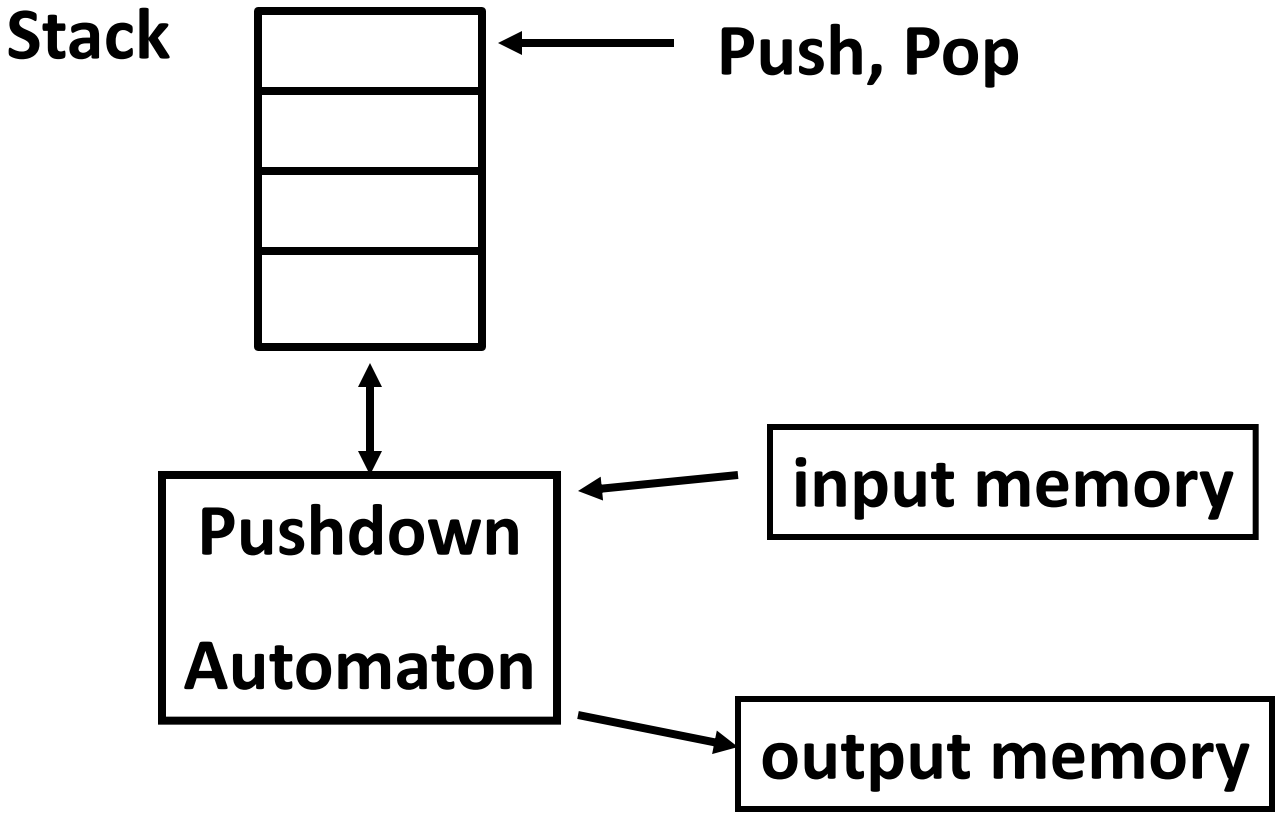
Finite Automaton



Example: Vending Machines

(small computing power)

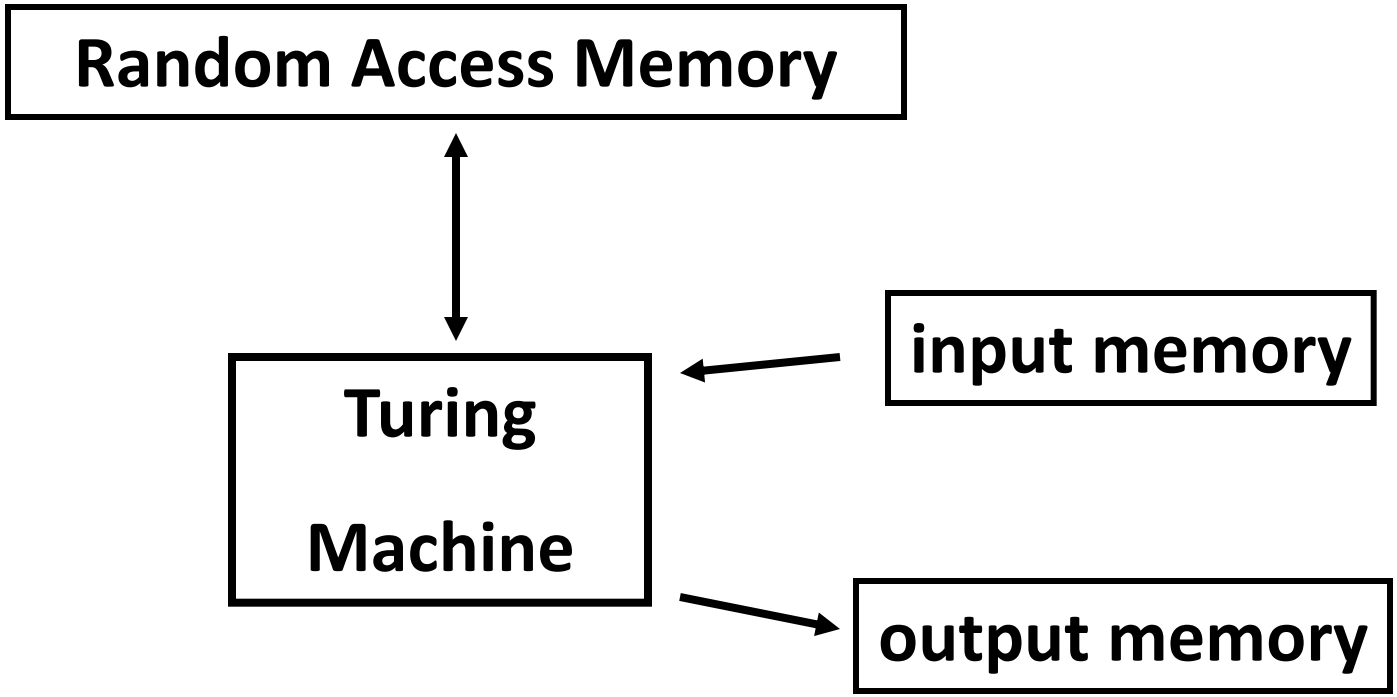
Pushdown Automaton



Example: Compilers for Programming Languages

(medium computing power)

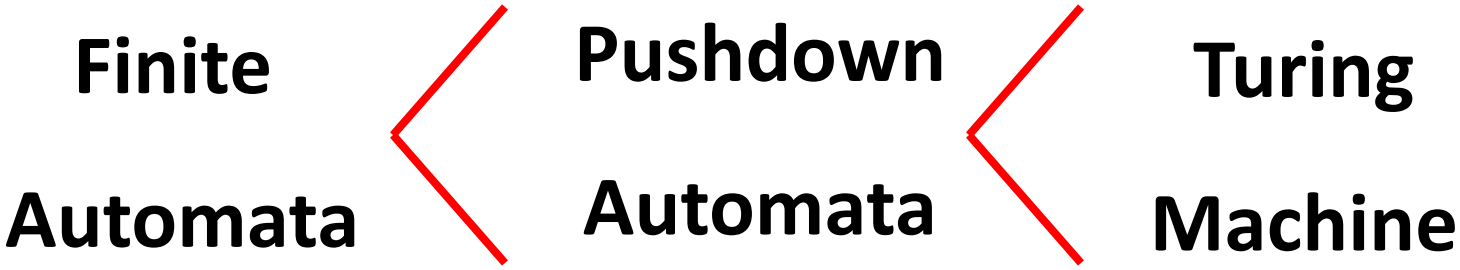
Turing Machine



Examples: Any Algorithm

(highest computing power)

Power of Automata



Less power → More power

Solve more computational problems

Finite Automata

http://www.youtube.com/watch?v=d7ZAcym3DUw

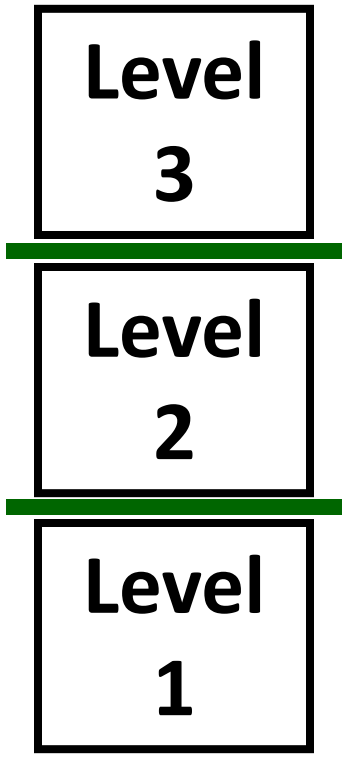
Finite Automata (FA)

- **Finite** - Every FA has a finite number of states
- **Automata** (singular automaton) means "machine"
- A simple class of **machines** with limited capabilities.
- Good models for computers with an extremely **limited amount** of **memory**.
- E.g., an automatic door : a computer with only a single bit of memory.

Examples

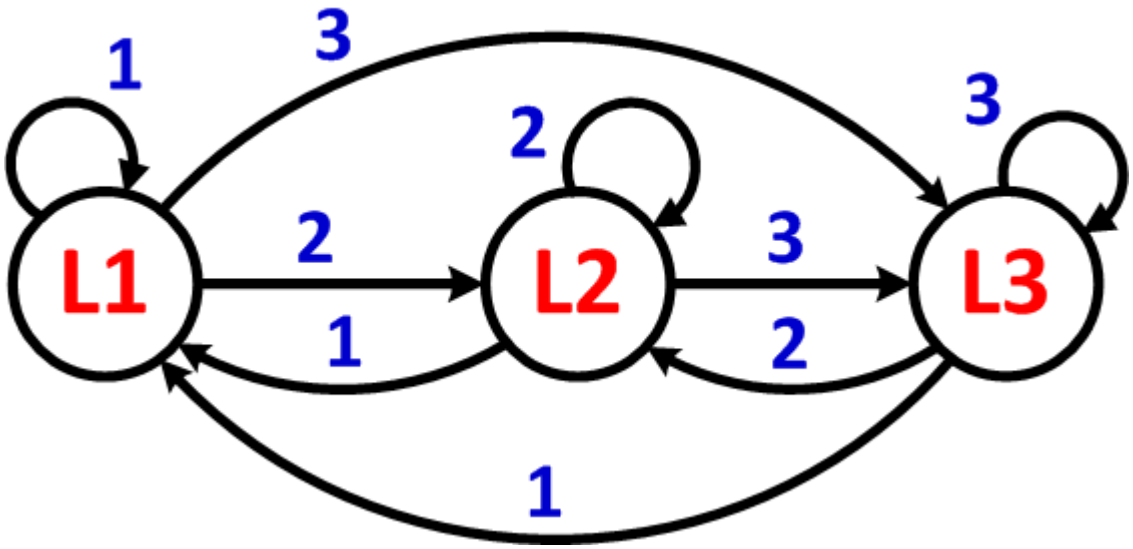
- **Elevator controller**
  - **state : *n*-th floor**
  - **input : input received from the buttons.**
- **Dishwashers**
- **Washing Machines**
- **Digital watches**
- **Calculators**

State Diagram



LIFT:

- move to level 1 when person push button 1
- move to level 2 when person push button 2
- move to level 3 when person push button 3



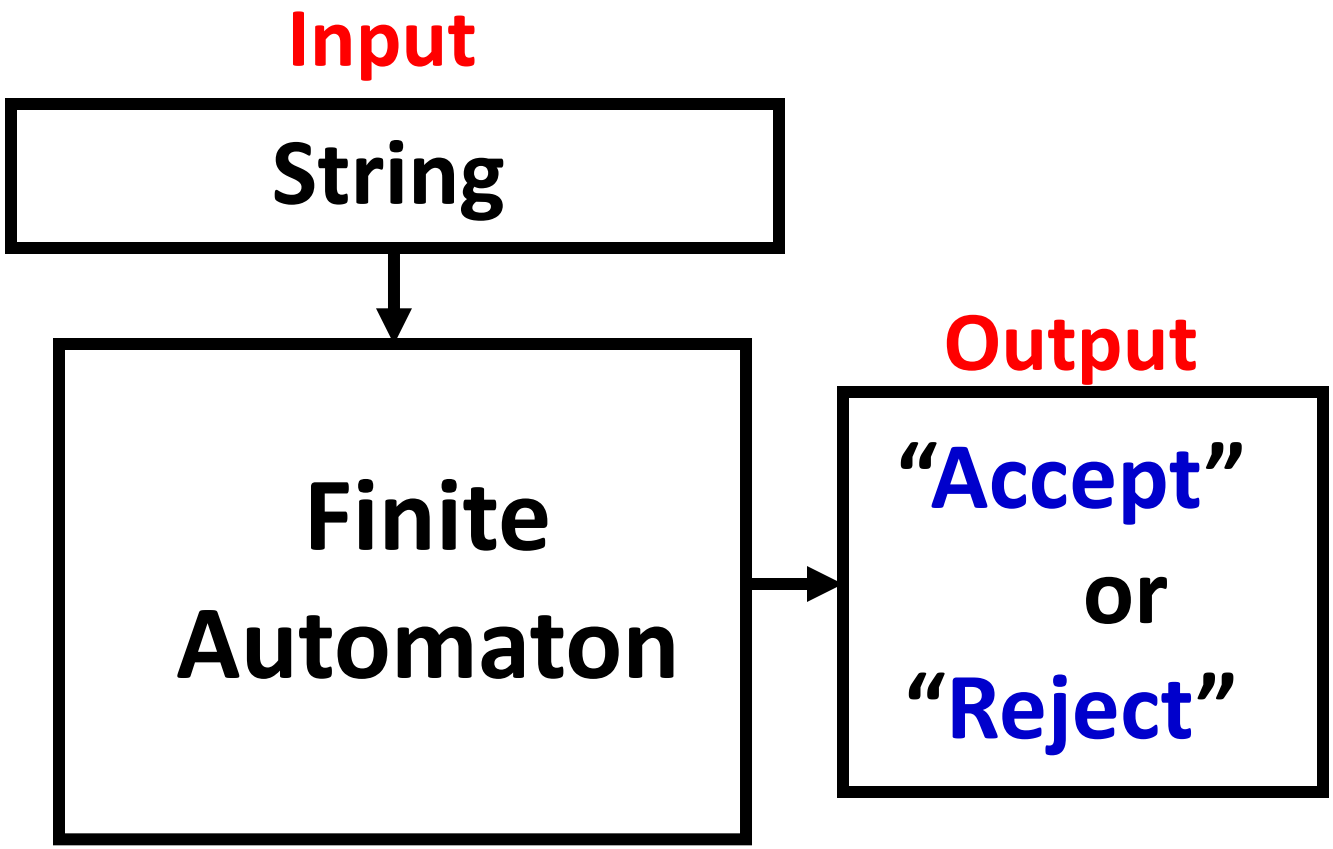
State Transition Table

States: `Level1, Level2, Level3`

Input: 1 someone push button 1
2 someone push button 2
3 someone push button 3

		INPUT		
		1	2	3
STATE	Level1	Level1	Level2	Level3
	Level2	Level1	Level2	Level3
	Level3	Level1	Level2	Level3

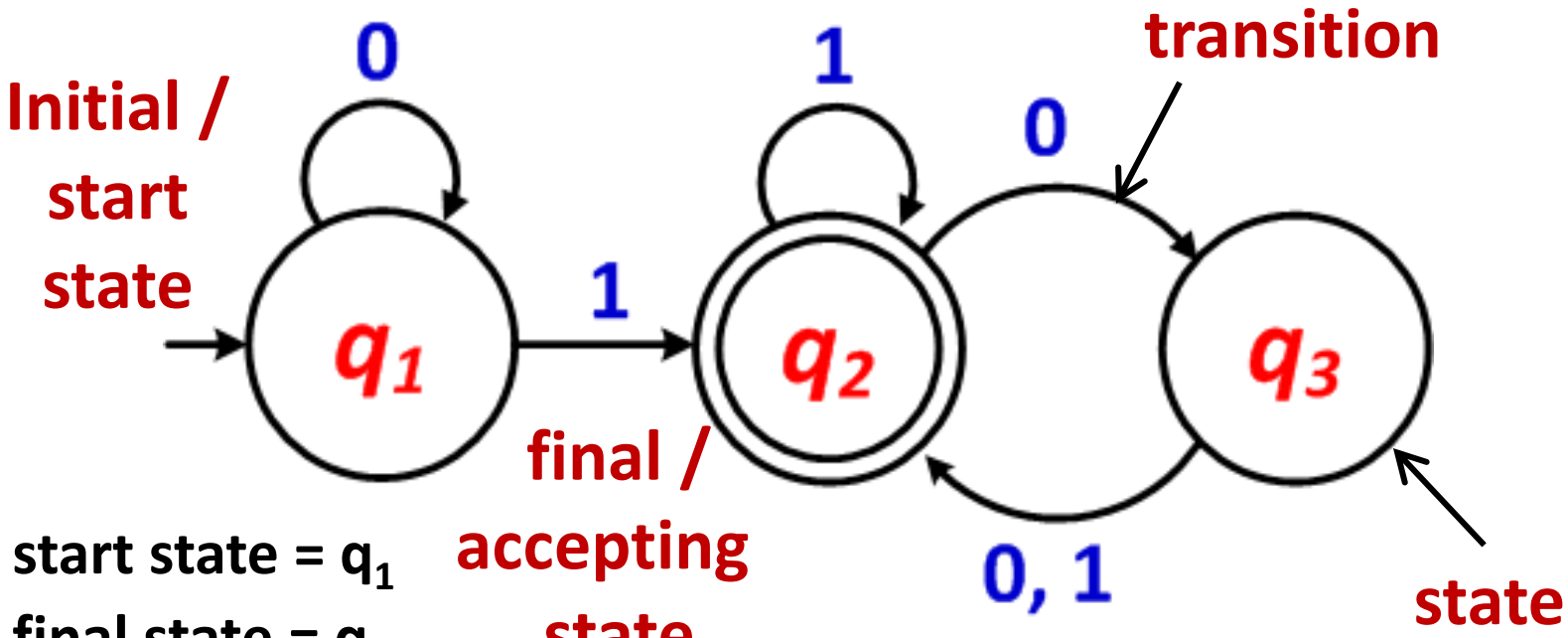
Finite Automaton



Definition of FA

- A finite automaton (FA) is a collection of 3 things:
  - A finite set of states, one of them will be the initial or start state, and some (maybe none) are final states.
  - An alphabet Σ of possible input letters.
  - A finite set of transition rules that tell for each state and for each input (alphabet) which state to go to next.

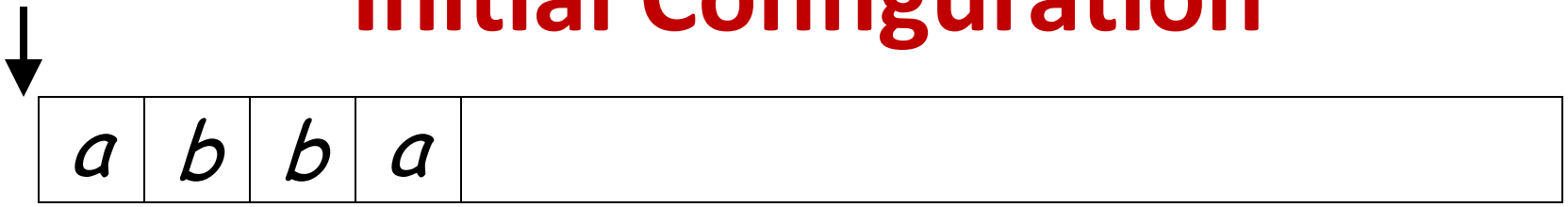
State Diagram



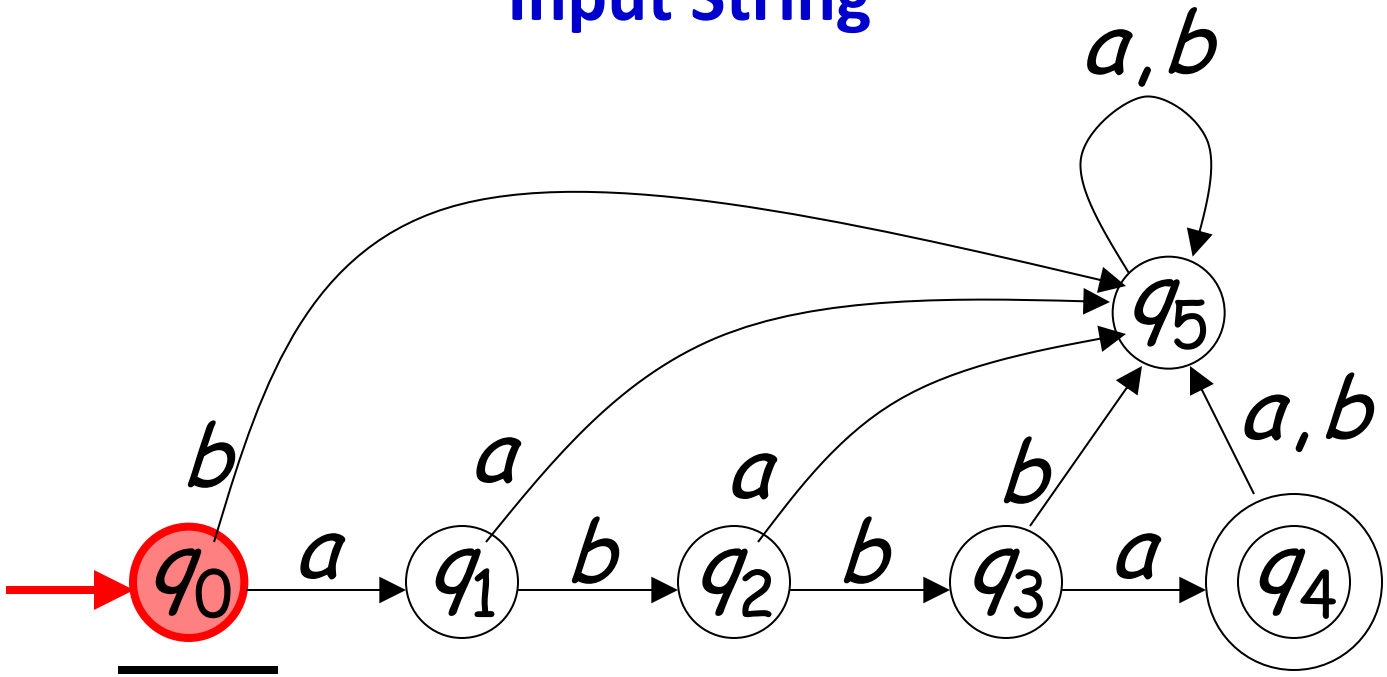
- start state = q_1
- final state = q_2
- transitions = each arrows
- alphabet = each labels

- When this automaton receives an input string such as 1101, it processes that string and produce output (Accept or Reject).

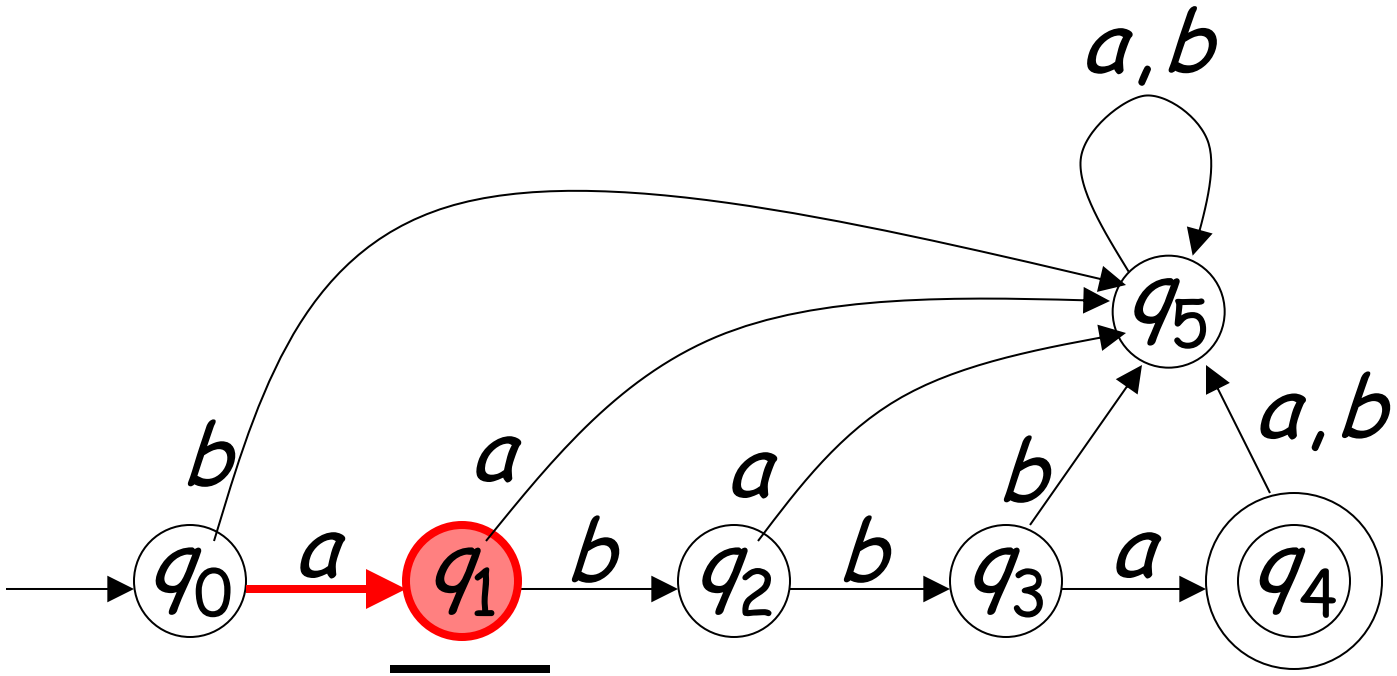
Initial Configuration

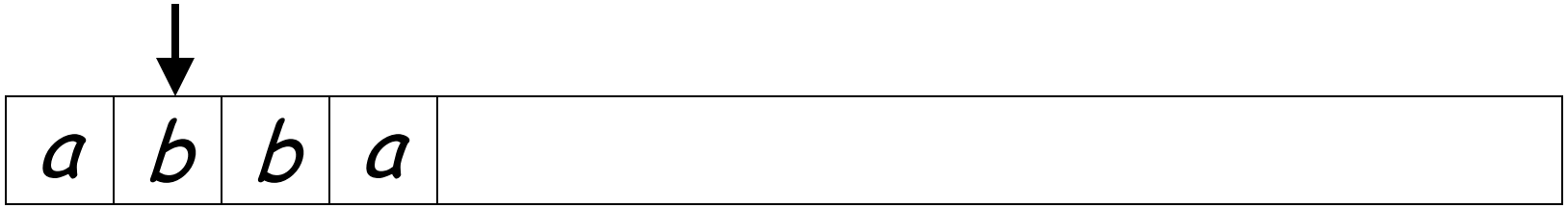


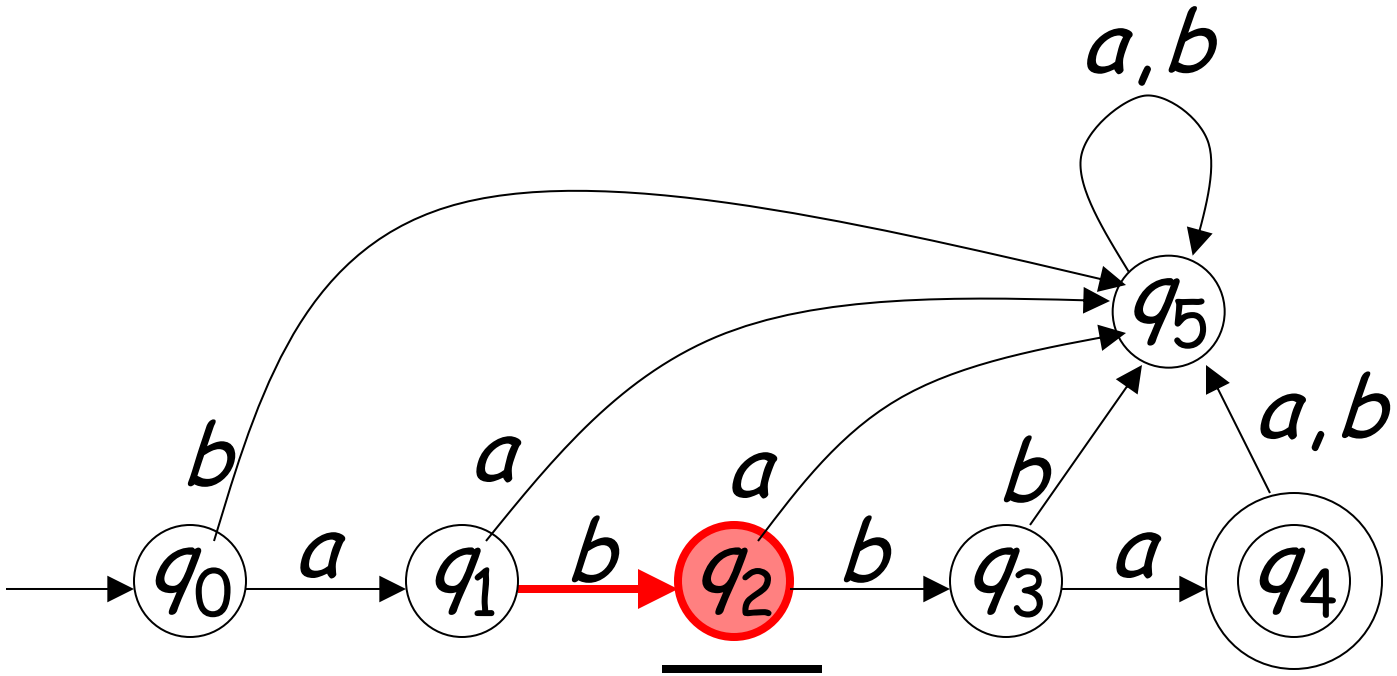
Input String

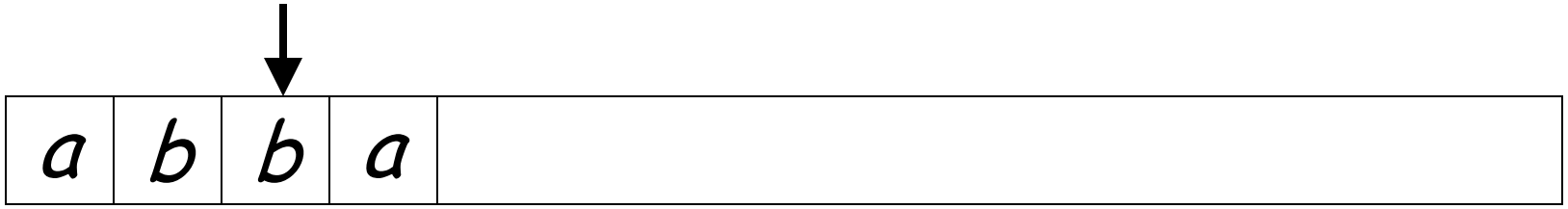


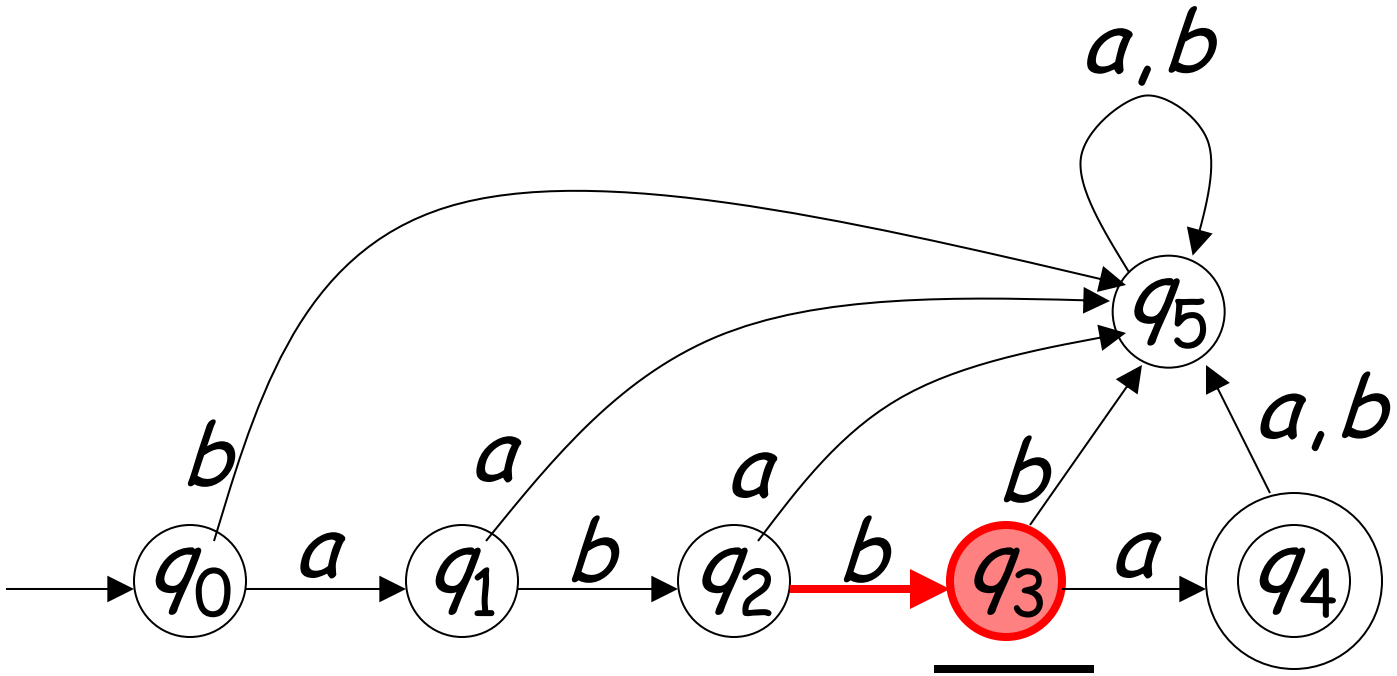
Reading the Input

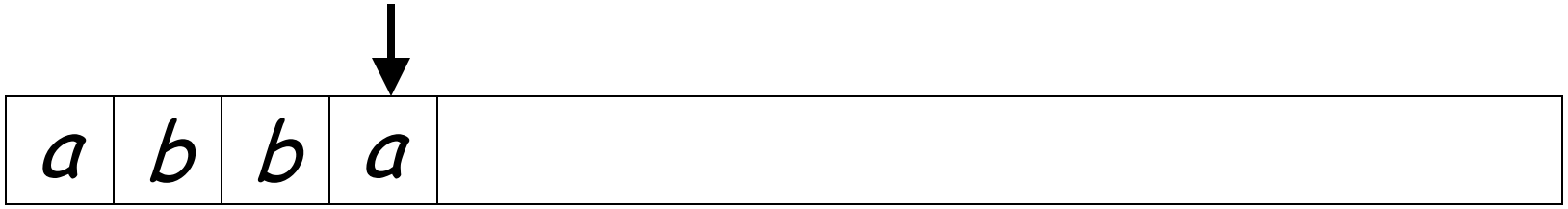


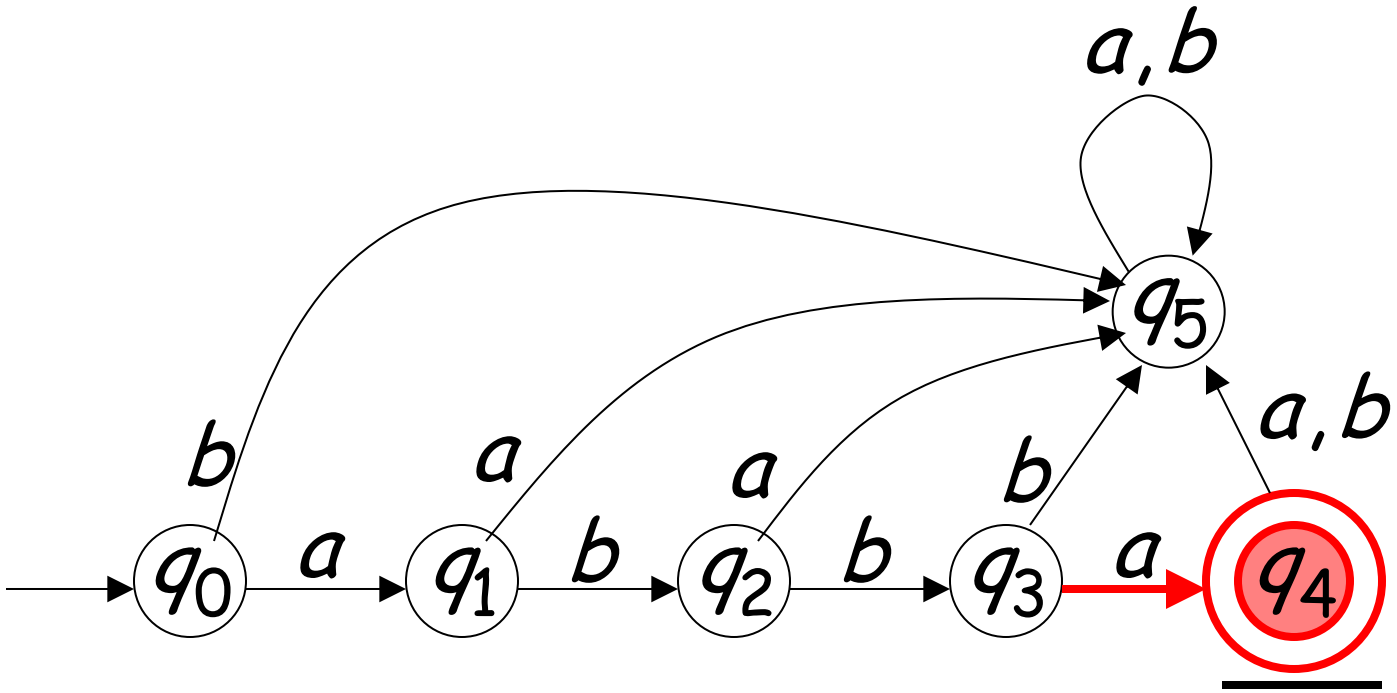

a
b
b
a


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

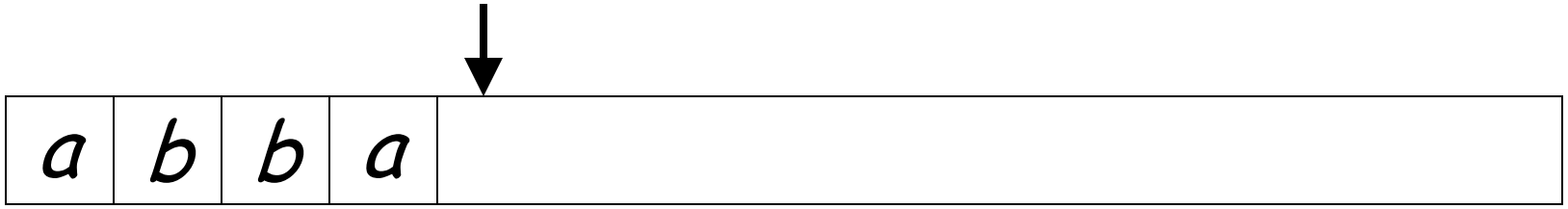

a
b
b
a

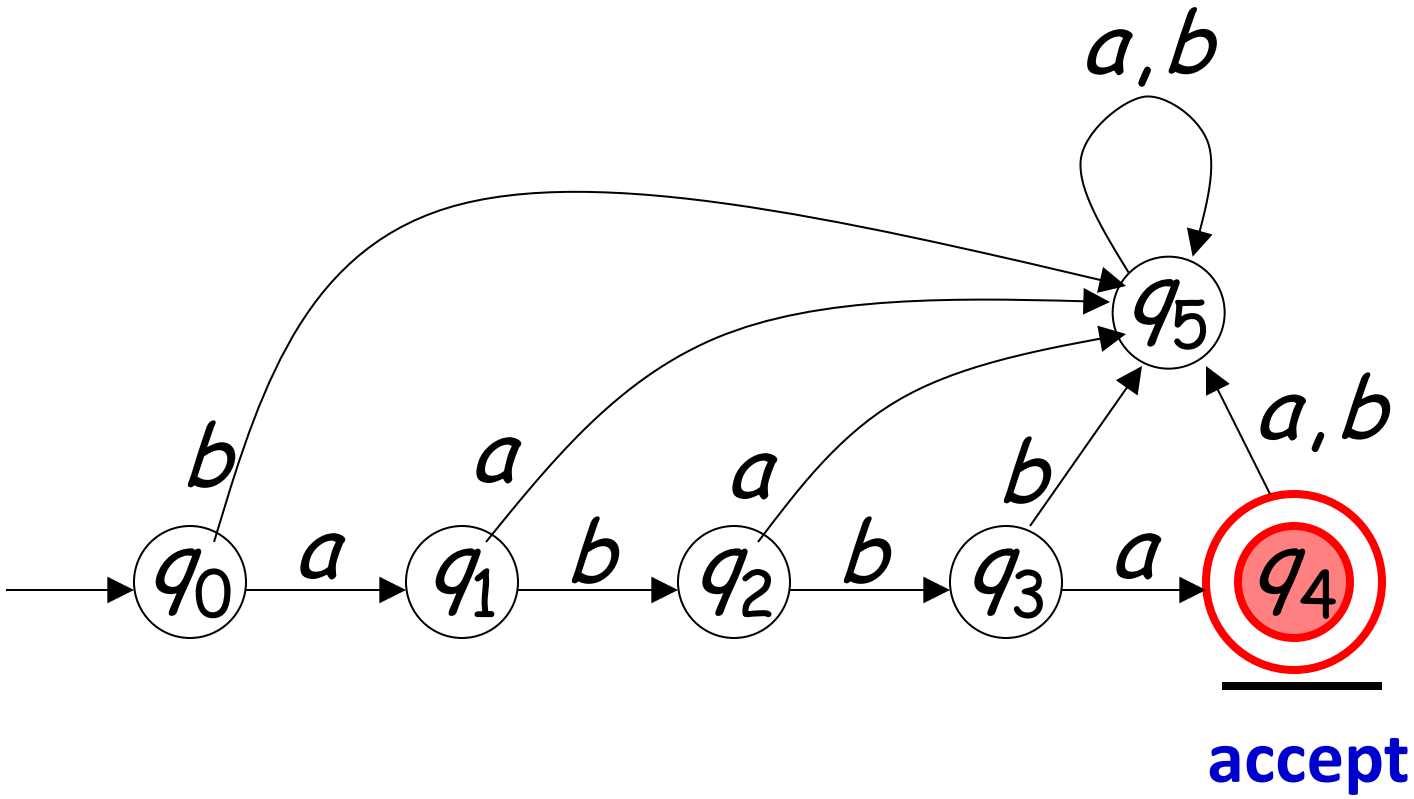

a, b
q5
b
a
a
b
a, b
q0
a
q1
b
q2
b
q3
a
q4


a
b
b
a


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

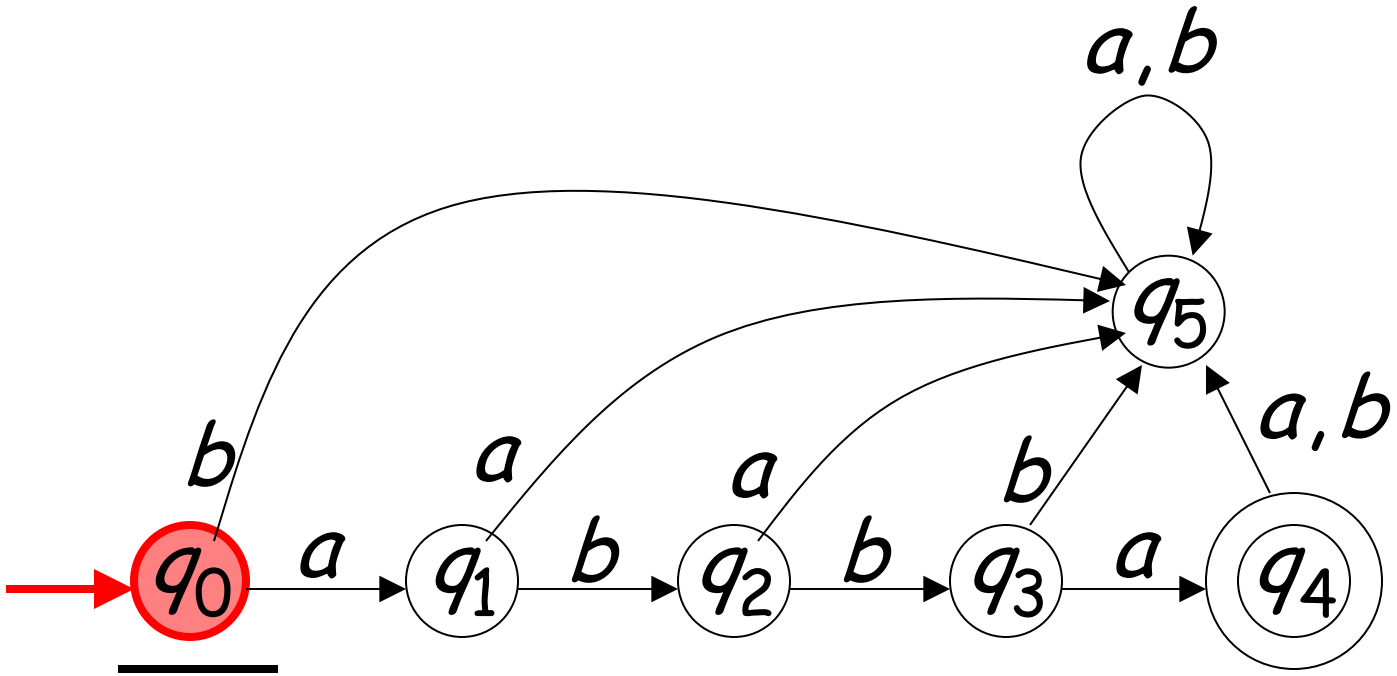
Input finished

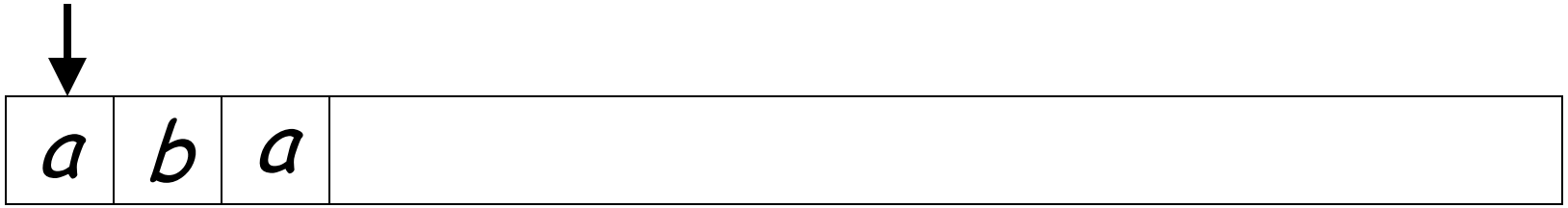


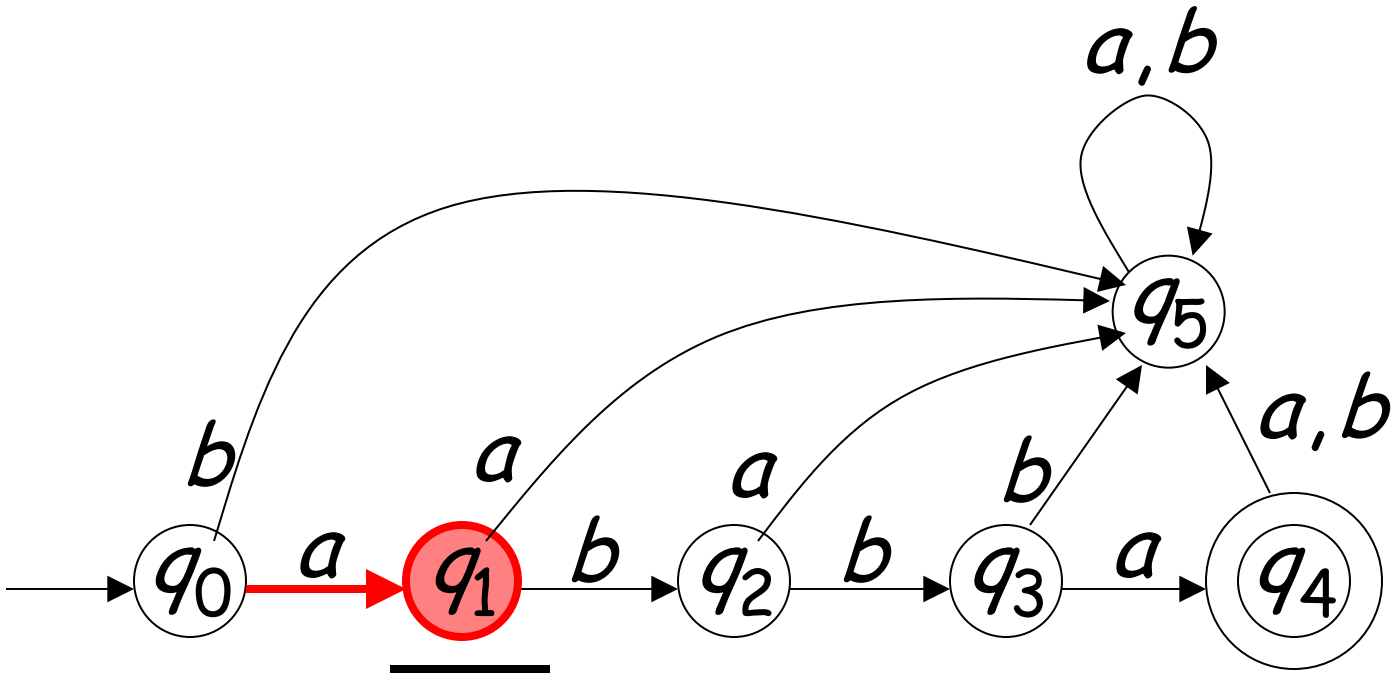


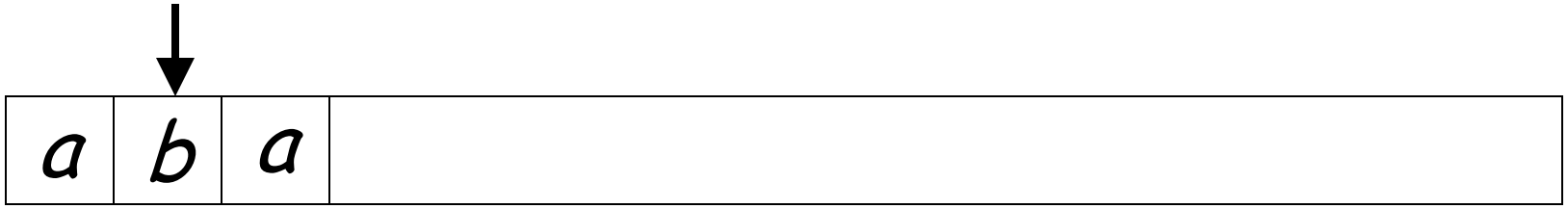
accept

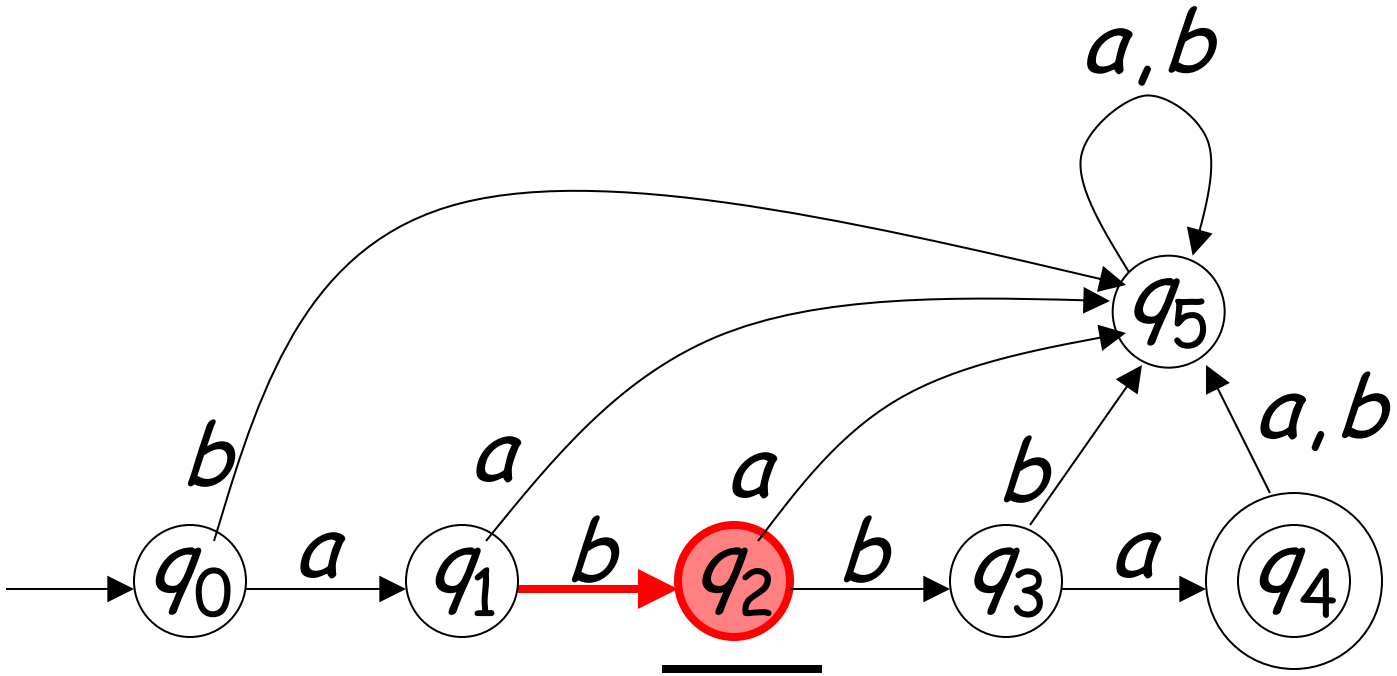
Rejection

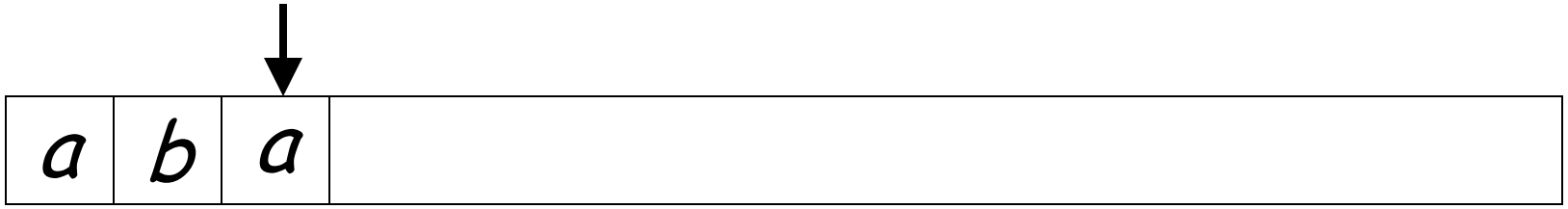


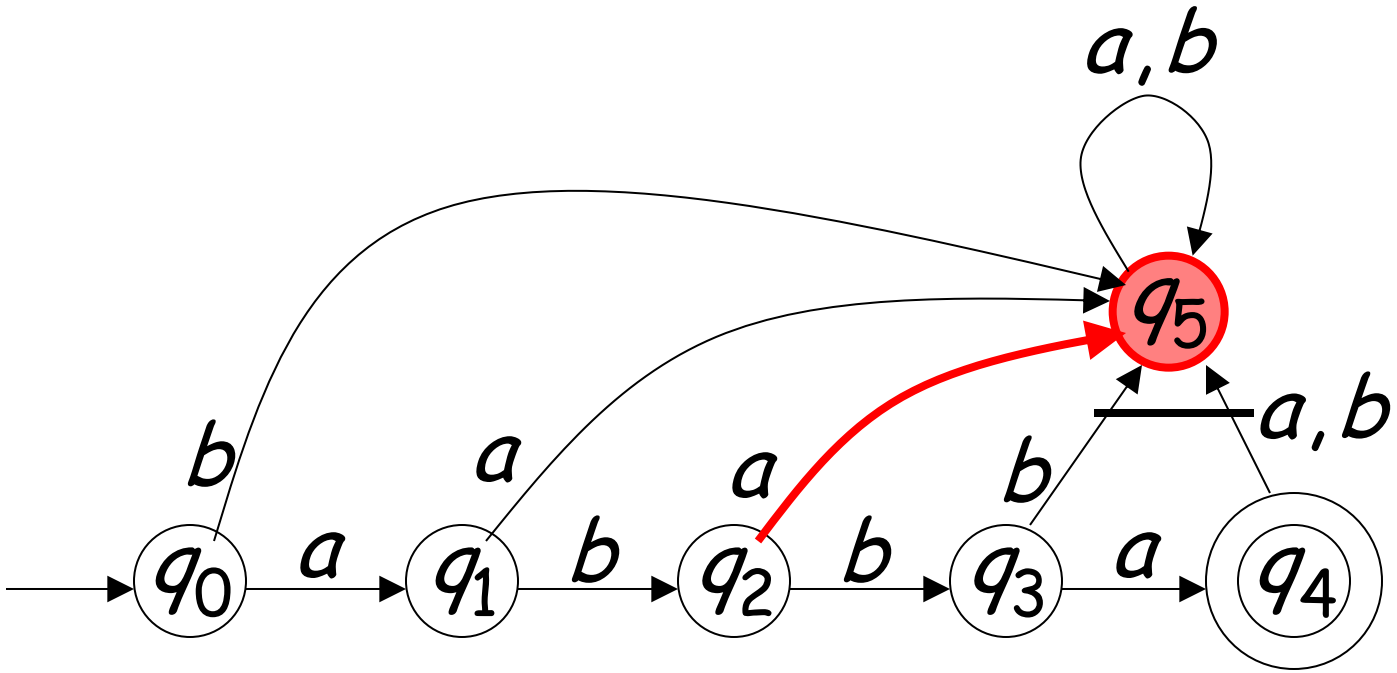

a
b
a


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

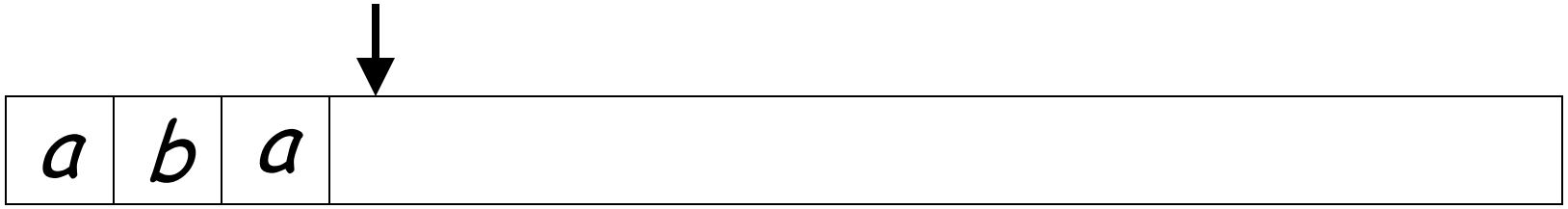

a
b
a

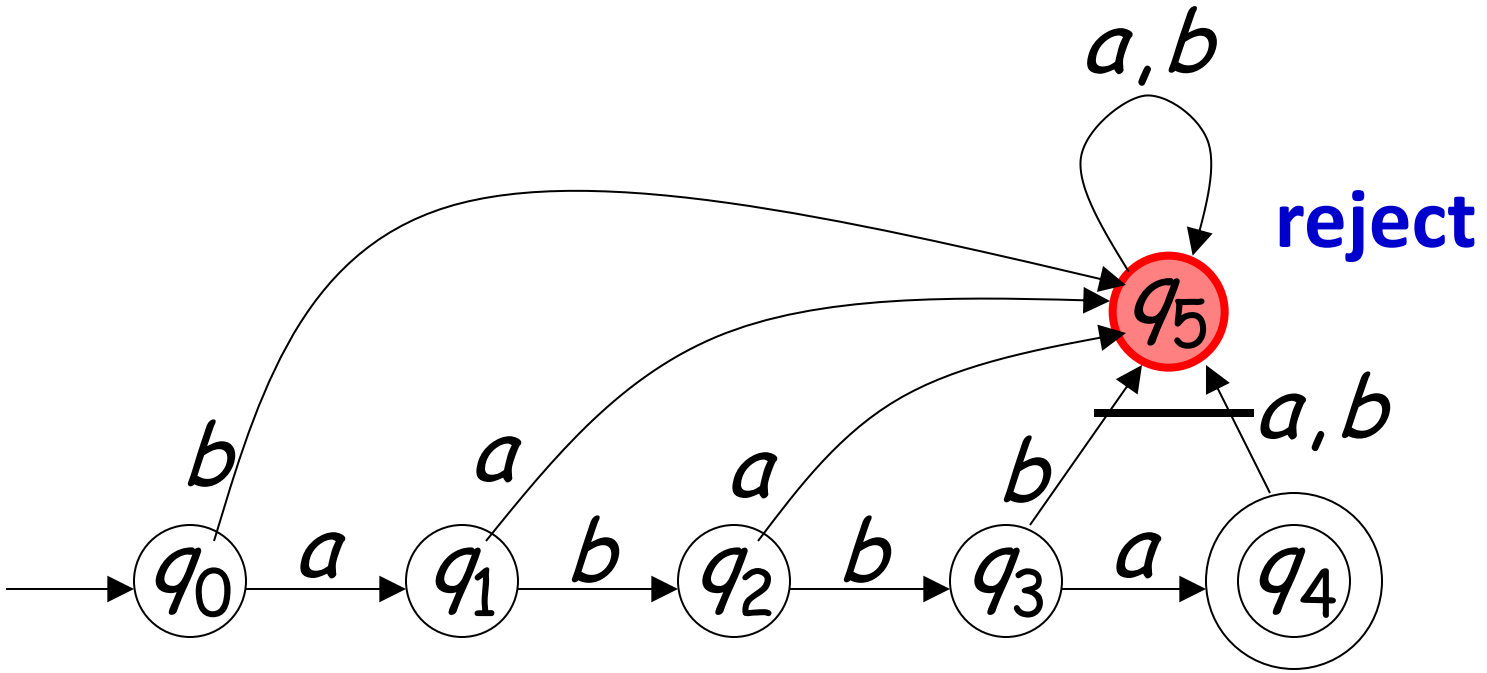

a, b
q5
b
a
a
b
a, b
q0
a
q1
b
q2
b
q3
a
q4


a
b
a

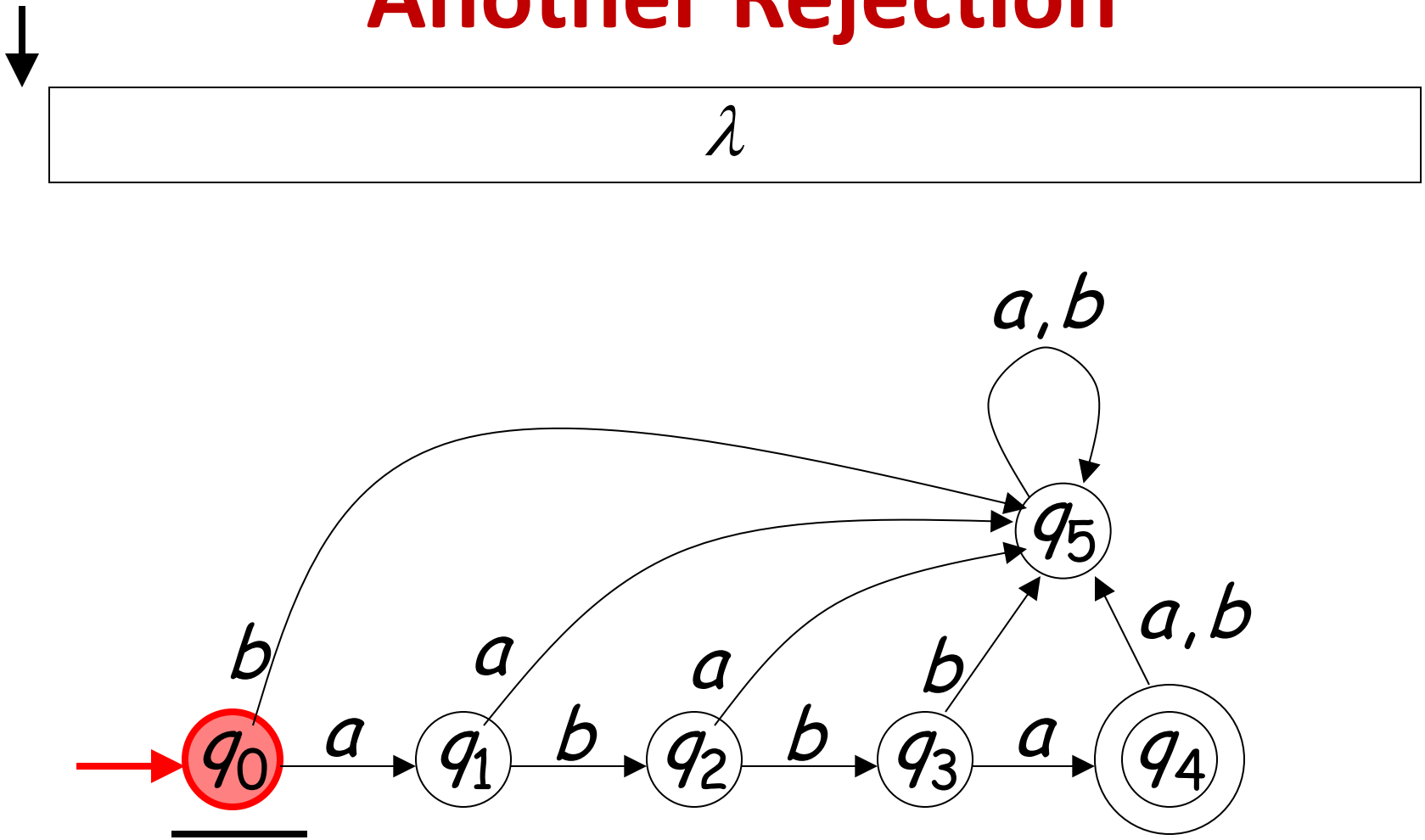

a, b
q5
b
a
a
b
a, b
q0
a
q1
b
q2
b
q3
a
q4

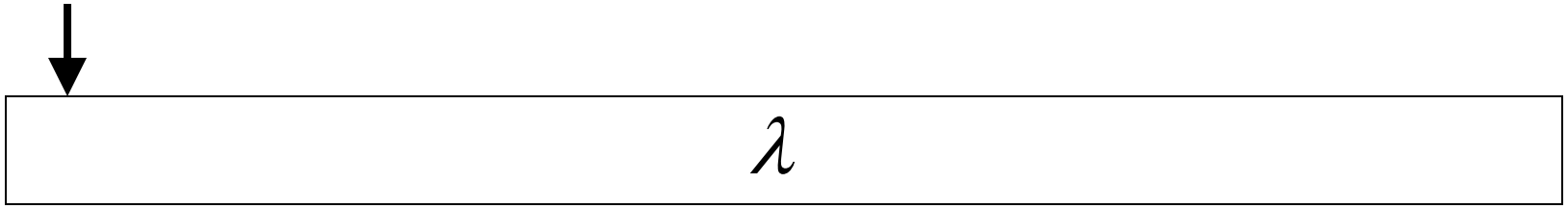
Input finished

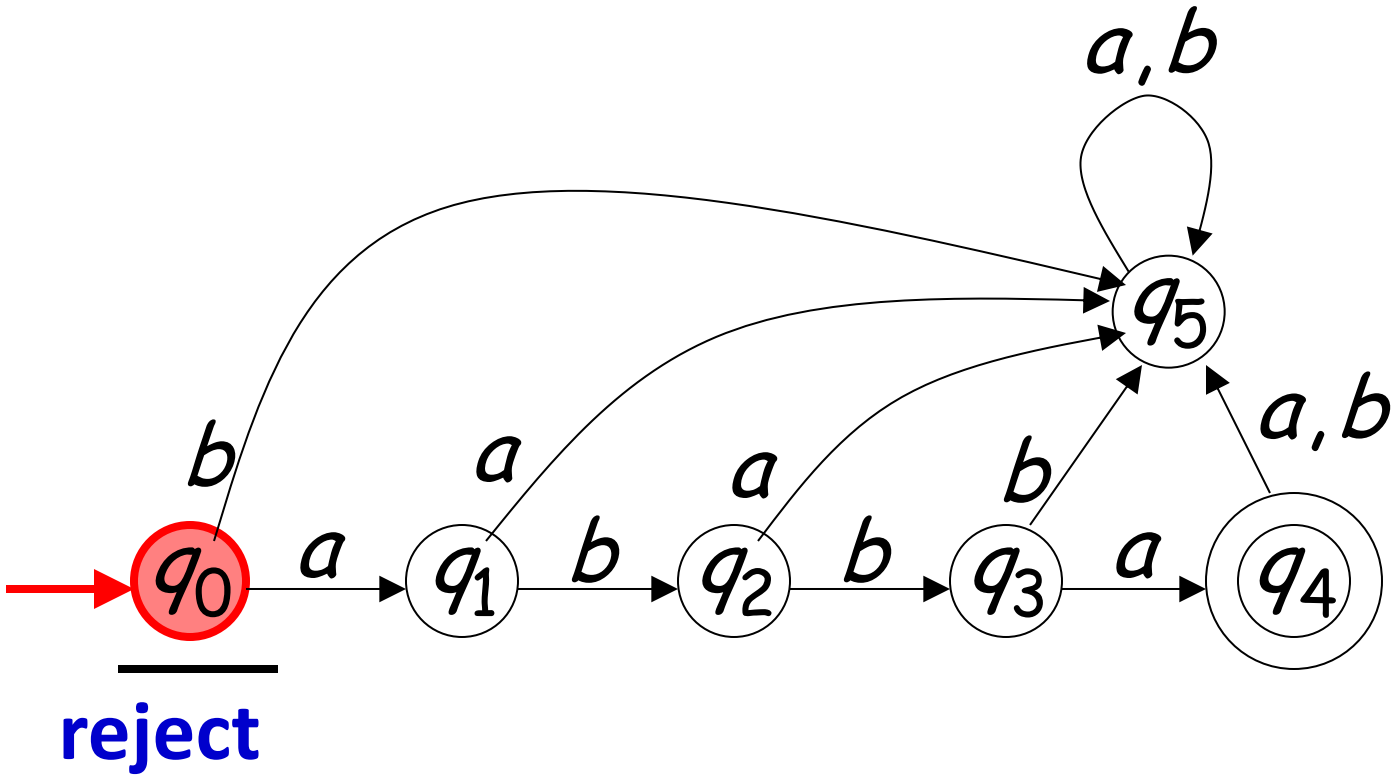




Another Rejection




λ

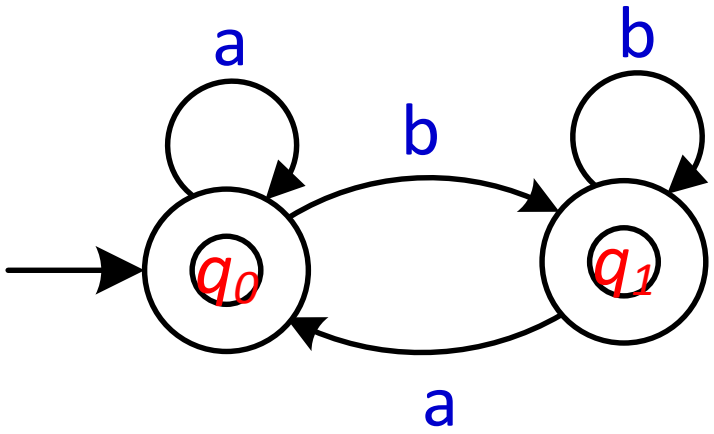

a, b
q5
b
a
a
b
a, b
q0
a
q1
b
q2
b
q3
a
q4
reject

Formal Definition

- A finite automaton is a 5-tuple $(Q, \sum, \delta, q_0, F)$ where
  - Q is a finite set of states,
  - $\sum$ is a finite set of alphabets,
  - $\delta : Q \times \sum \rightarrow Q$ is the transition function,
  - $q_0 \in Q$ is the start state, and
  - $F \subseteq Q$ is the set of accept states (final states)

Note: δ - delta

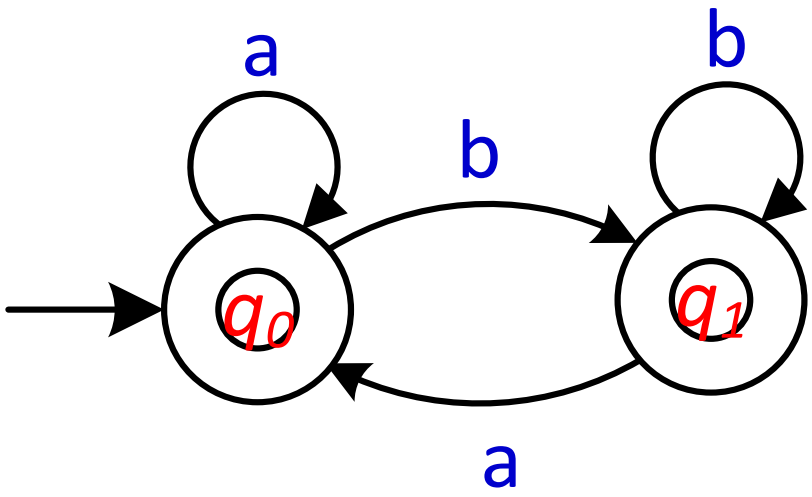
Example :
Finite Automaton M_1



- $M_1 = (Q, \Sigma, \delta, q_1, F)$, where
  - $Q = \{q_0, q_1\}$,
  - $\Sigma = \{a, b\}$,
  - δ is described as
  - q_0 is the start state, and
  - $F = \{q_0, q_1\}$.

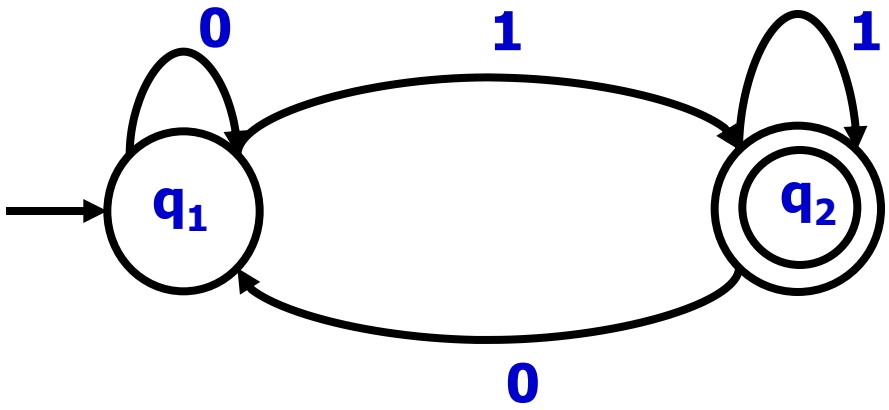
	a	*b*
q_0	q_0	q_1
q_1	q_0	q_1

Example :
Finite Automaton M_1



$$L(M_1) = (a + b)^*$$

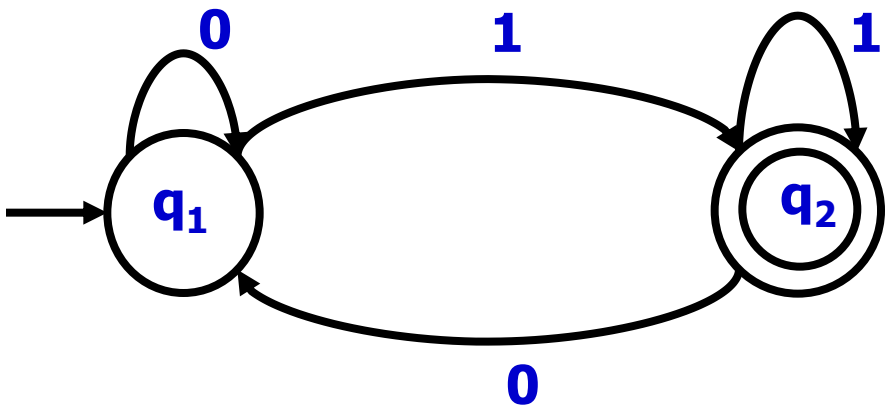
Example :
Finite Automaton M_2



- $M_2 = (Q, \Sigma, \delta, q_0, F)$, where
  - Q =
  - Σ =
  - δ is described as
  - q_1 is the start state, and
  - $F = \{\ \}$.

	0	1
q_1		
q_2		

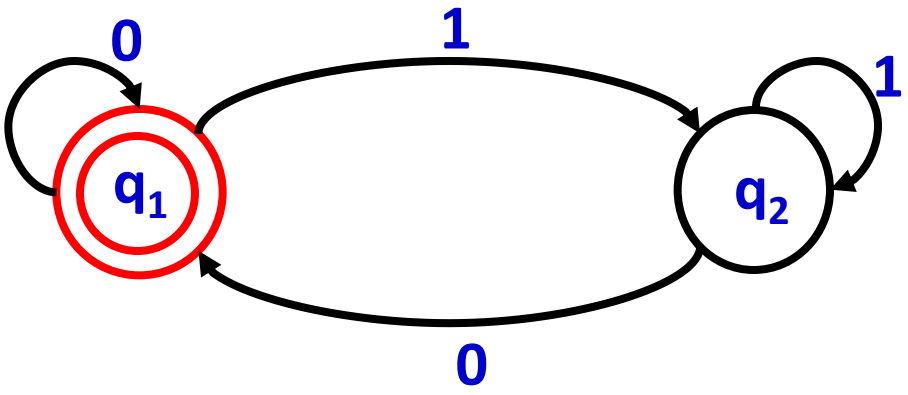
Example :
Finite Automaton M_2



What is the language of M_2?

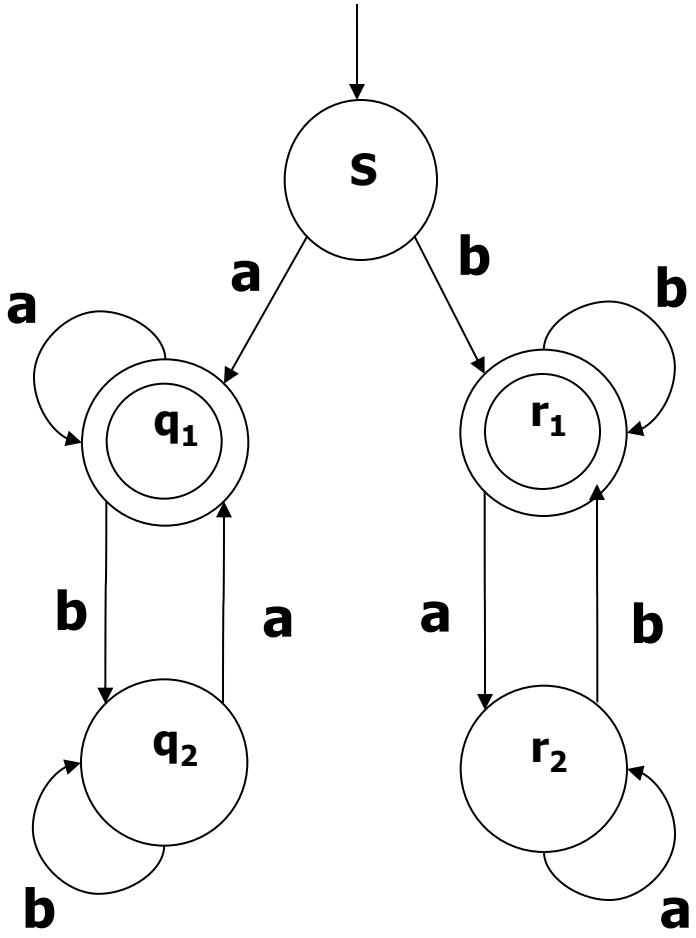
$L(M_2) = \{w \mid w$ ends in a 1$\}$

Empty String ε



- If the start state is also a final state, what string does it automatically accept ?
- $L(M_3)$ = { *w* | *w* is the empty string ε or ends in a 0}

Example :
Finite Automaton M_4



- $M_4 = (Q, \Sigma, \delta, S, F)$, where
  - Q =
  - Σ =
  - δ is described as

r_1	a	b
S		
q_1		
q_2		
r_1		
r_2		

  - S is the start state, and
  - F = { }.

$L(M_4)$ = strings that start and end with the same symbol.

\# a(a + bb*a)* + b(b + aa*b)*

FA Design : Example

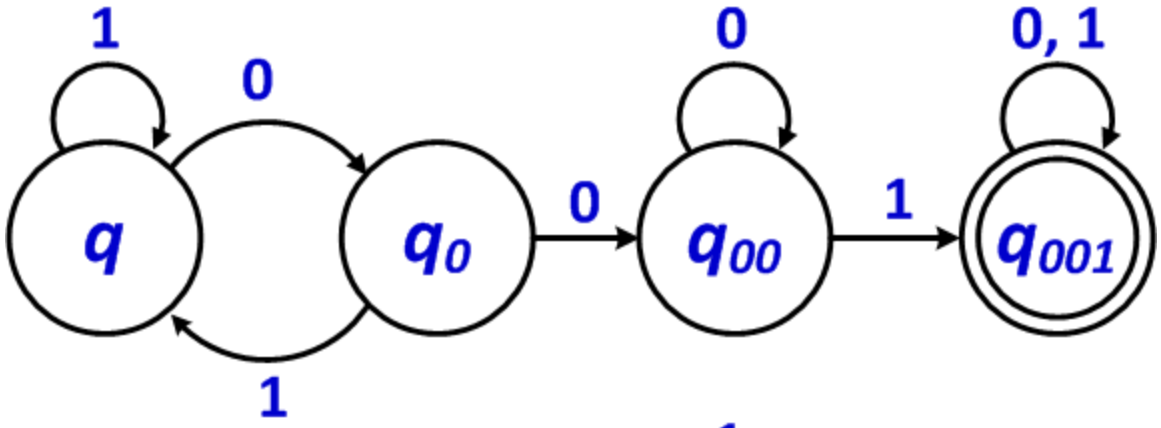
- **Question**: Design an FA to accept the language of all strings over {0, 1} that contain the string 001 as a substring, e.g.: accept 0010, 1001, 001, 111001111, but not 11, 000.
- Thinking of… for every input, skip over all 1s. If we read 0, note that we may have just seen the first of the 3 symbols 001. Then if we see a 1, go back to skipping over 1s. Then … proceed … to seek for another two 0s ...

Example : Strings with substring 001

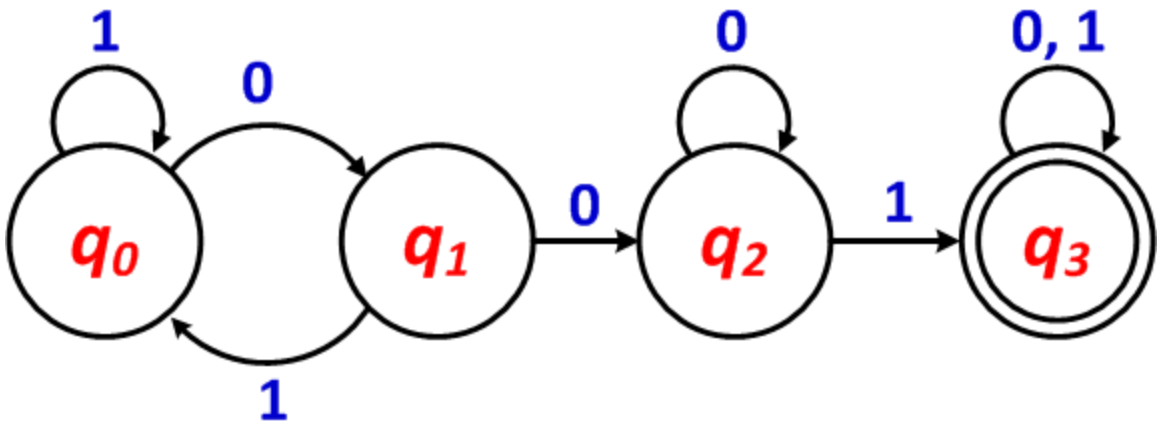
- So we can conclude there are four possibilities:
  - Haven't just seen any symbols of 001
  - Have just seen a 0
  - Have just seen 00
  - Have seen 001
- Assign the states q, q_0, q_{00}, q_{001} to these possibilities. We can design the transition by observing that from q reading a 1, we stay in q, but reading a 0 we move to q_0. In q_0, reading a 1 we return to q, but reading a 0 we move to q_{00}. In q_{00}, reading a 1 we move to q_{001}, but reading a 0 leaves us in q_{00}. finally, in q_{001} reading a 0 or a 1 leaves us in q_{001}.

Example : Strings with substring 001

- The start state is q, and the only accept state is q_{001}.



or



Computation of FA, $\vdash_M$

- Way to describe the computation of a DFA
- What state the DFA is currently in, and what string is left to process
  - $(q_0, 0010)$ Machine is in state q_0, has 0010 left to process
  - $(q_1, 010)$ Machine is in state q_1, has 010 left to process
  - (q_3, ε) Machine is in state q_3 at the end of the computation (accept iff $q_3 \in F$)
- Binary relation $\vdash_M$: What machine M yields in one step
  - $(q_0, 0010) \vdash_M (q_1, 010)$

Accepted @ Rejected Strings

- To trace all computations that process the string 0010.
- $[q_0 , 0010] \vdash [q_1 , 010]$
  $\vdash [q_2 , 10]$
  $\vdash [q_3 , 0]$
  $\vdash [q_3 , \lambda]$
- Stop at q_3 (final state) and all alphabet is traced (string is empty);
- Hence, string 0010 is accepted by machine.

Determinism

- So far, every step of a computation follows in a unique way from the preceding step.
- When the machine is in a given state and reads the next input symbol, we know what the next state will be – it is called **deterministic** computation
- Deterministic Finite Automata **(DFA)**

Why DFA?

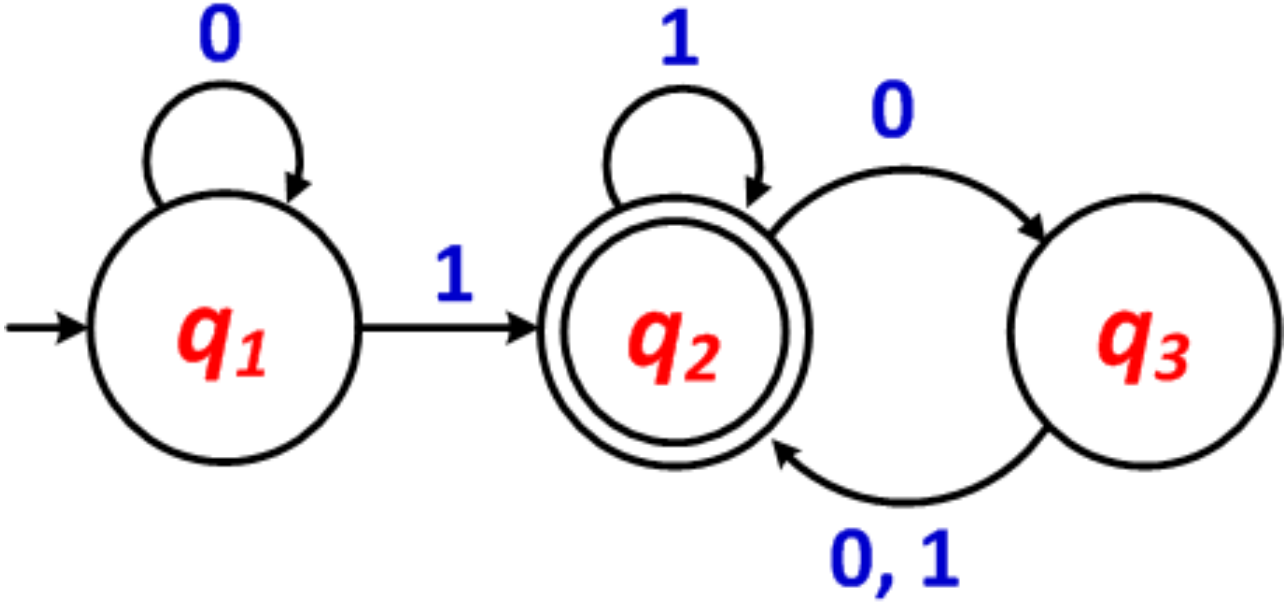
- Why are these machines called "Deterministic Finite Automata"
- **Deterministic** - Each transition is completely determined by the current state and next input symbol. That is, for each state / symbol pair, there is exactly *one state that is transitioned to.*

Nondeterminism

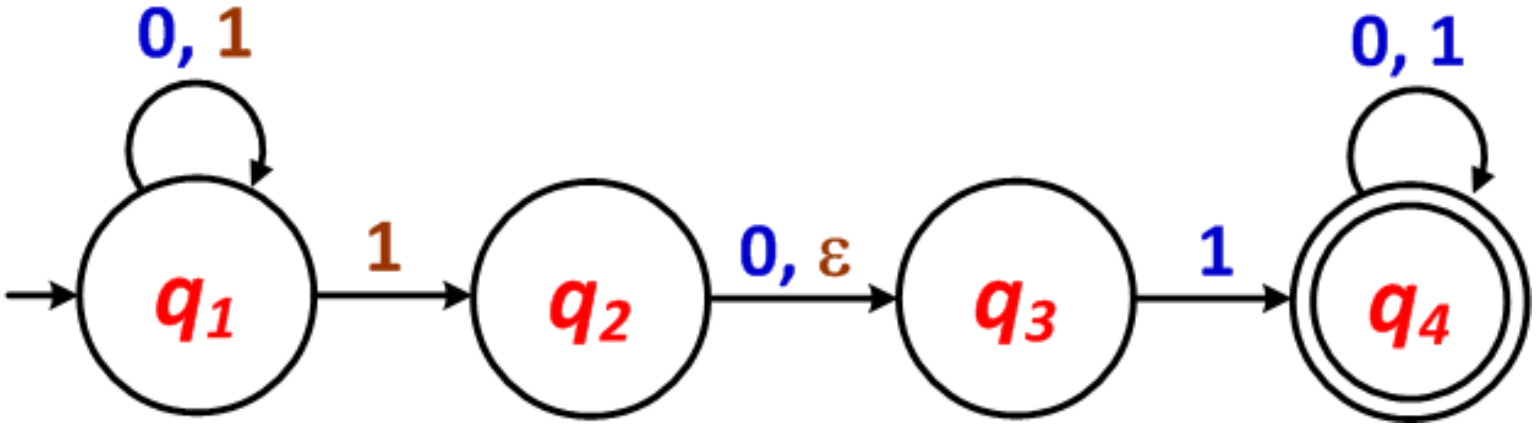
- In a **nondeterministic** machine, **several choices** may exist for the next state at any point.
- Nondeterministic Finite Automata **(NFA)**
- **Nondeterminism** is a generalization of the determinism, so every DFA is automatically a NFA.

Example of DFA vs. NFA

DFA



NFA



Differences between DFA & NFA

- Every state of DFA always has exactly one exiting transition arrow for each symbol in the alphabet while the NFA may has zero, one or more.
- In a DFA, labels on the transition arrows are from the alphabet while NFA can have an arrow with the label ε.

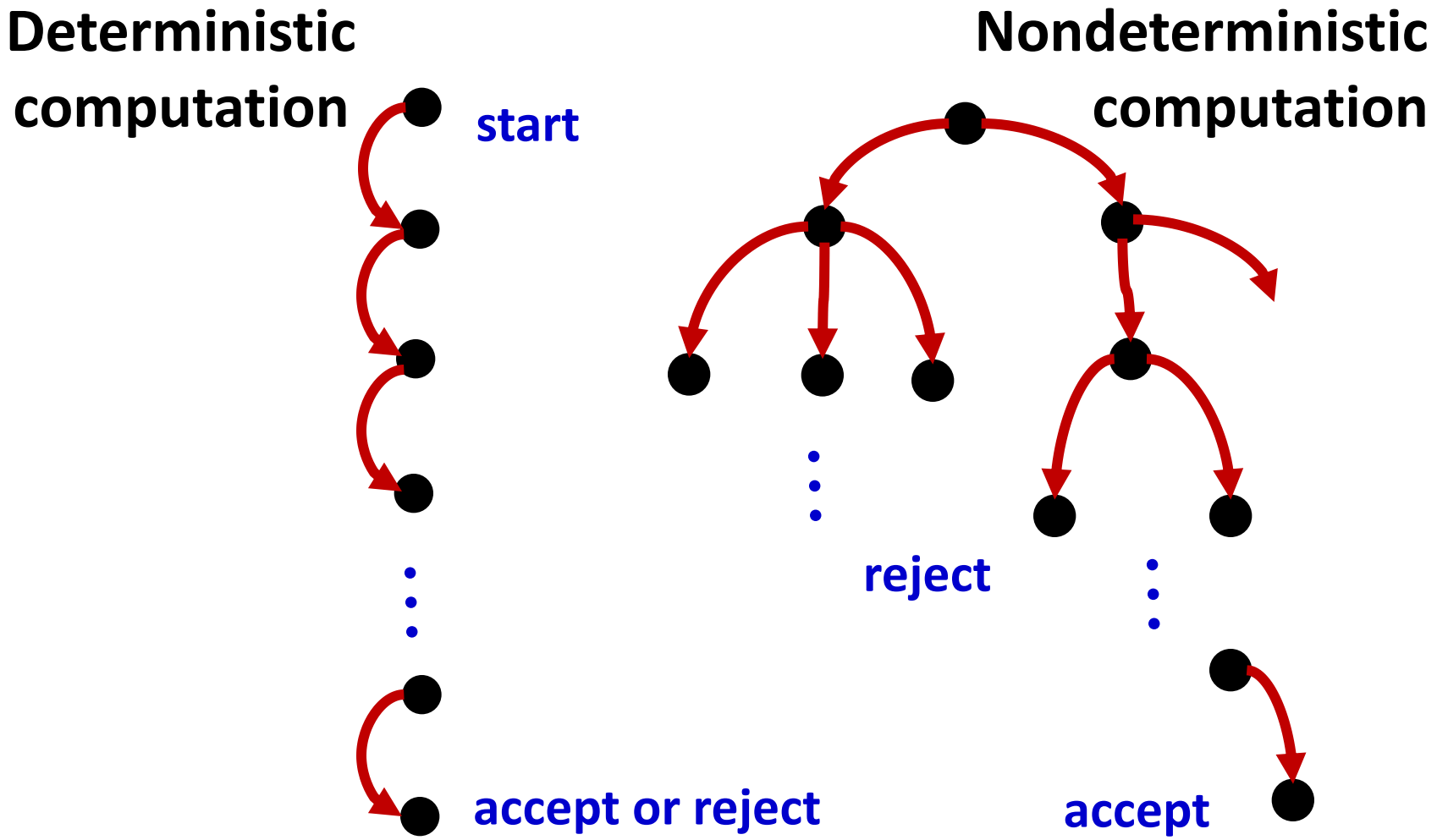
How does the NFA work?

- When we are at a state with multiple choices to proceed (including ε symbol), the machine splits into multiple copies of itself and follow all the possibilities in parallel.
- Each copy of the machine takes one of possible ways to proceed and continuous as before. If there are subsequent choices, the machine splits again.
- If the next input symbol doesn't appear on any of the arrows exiting the state occupied by a copy of the machine, that copy dies.
- If any one of these copies is in an accept state at the end of the input, the NFA accepts the input string.

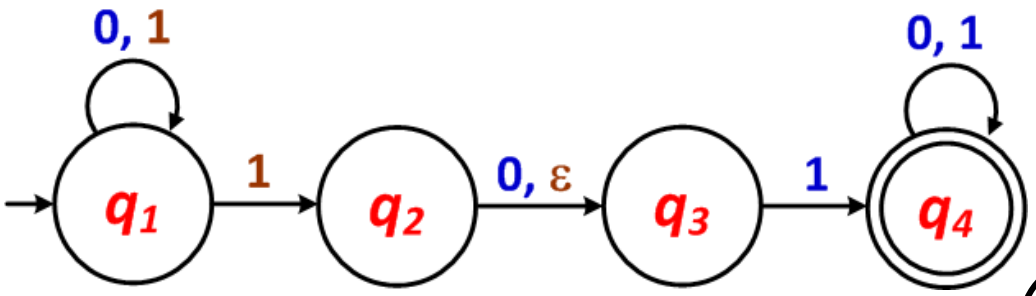
Tree of possibilities

- Think of a nondeterministic computation as a tree of possibilities
- The root of the tree corresponds to the start of the computation.
- Every branch point in the tree corresponds to a point in the computation at which the machine has multiple choices.
- The machine accepts if at least one of the computation branches ends in the an accept state.

Tree of possibilities



Example: 010110

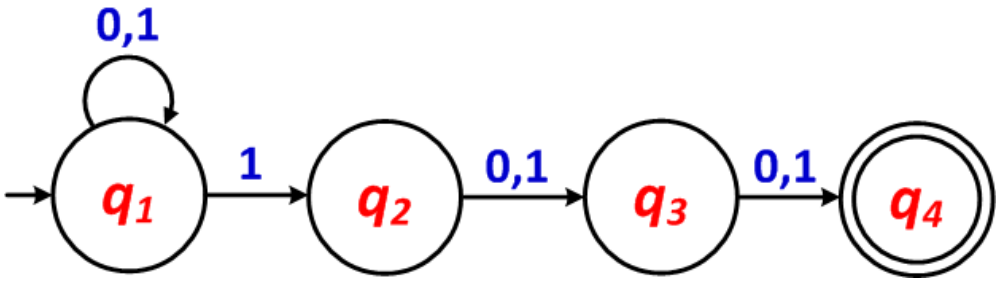


Properties of NFA

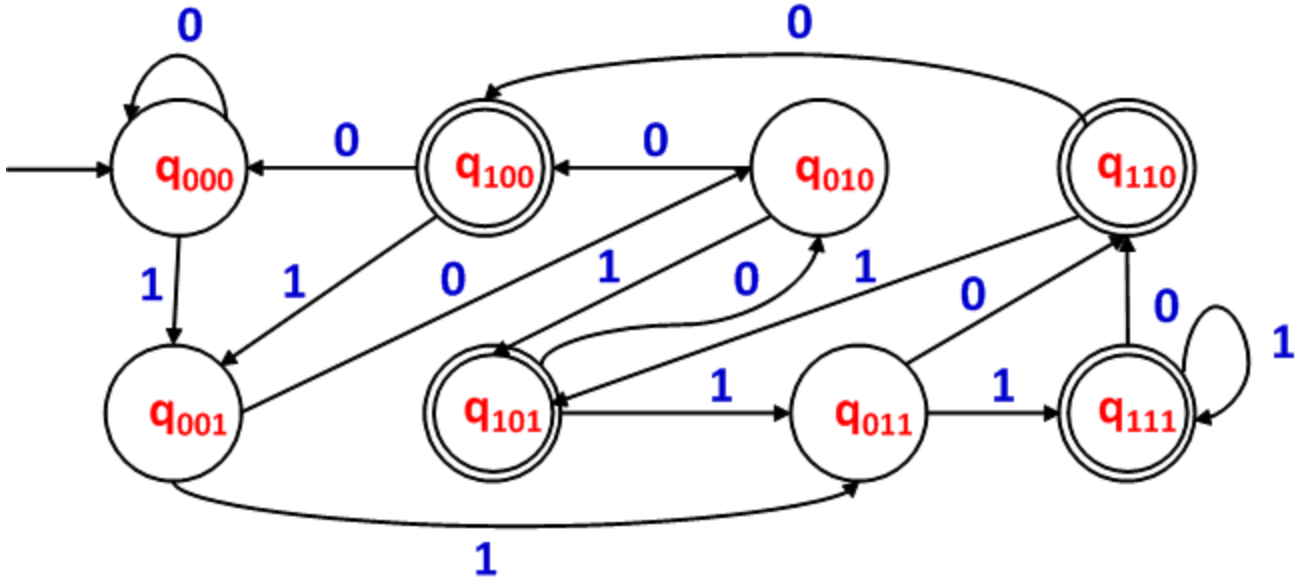
- Every NFA can be converted into an equivalent DFA.
- Constructing NFAs is sometimes easier than directly construction DFAs.
- NFA may be much smaller than it DFA counterpart.
- NFA's functioning may be easier to understand.
- Good introduction to non-determinism in more powerful computational models because FA are especially easy to understand.

Example: Converting NFA into DFA

NFA: recognizes language which contains 1 in the third position from the end



Equivalent DFA:



Formal definition of NFA

- A NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta : Q \times \Sigma_\varepsilon \rightarrow P(Q)$ is the transition function,
  - q_0 is the start state, and
  - $F \subseteq Q$ is the set of accept states.

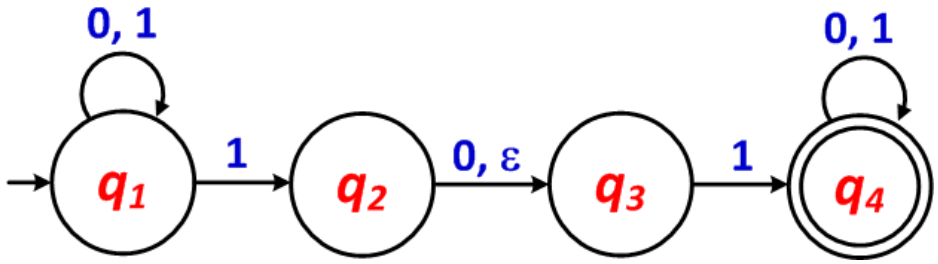
Notation:

P(Q) is called power set of Q (a collection of all subsets of Q).

and $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

Example: Formal definition of NFA

NFA M_1:

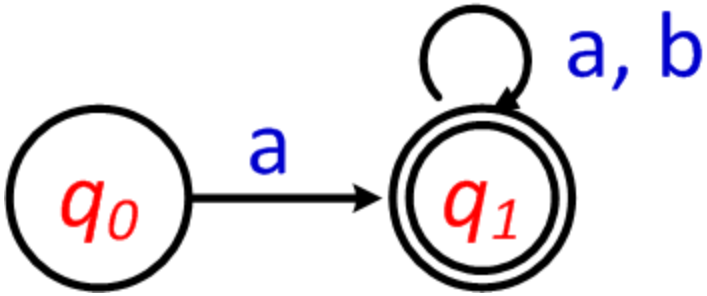


- Formal definition of N_1 is $(Q, \Sigma, \delta, q_0, F)$, where
  - $Q = \{q_1, q_2, q_3, q_4\}$
  - $\Sigma = \{0,1\}$
  - δ is given as
  - q_0 is the start state, and
  - $F = \{q_4\}$

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	∅
q_2	$\{q_3\}$	∅	$\{q_3\}$
q_3	∅	$\{q_4\}$	∅
q_4	$\{q_4\}$	$\{q_4\}$	∅

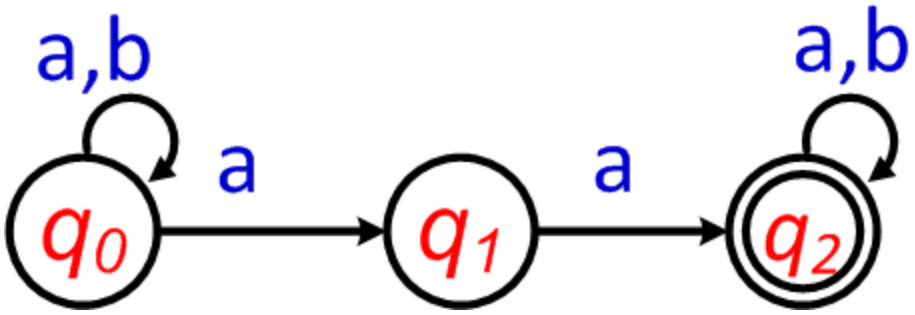
Example of NFA

- L = {w ∈ {a, b} : w starts with a}
- What is the RE for the language?



- What happen if the 1st input is *b*?

Example of NFA



- L = {w ∈ {a, b} : w contains the substring aa}
- What is the RE for the language?
- What happen if the 1st input is *a*?
- Does the machine accept *abaa*?

(q0, abaa) → (q0, baa) → (q0,aa) → (q0, a) → (q0, e) reject
(q0, a) → (q1, e) reject
(q0, baa) → (q1, baa) reject
(q0,aa) → (q1, a) → (q2, e) accept

Regular Languages

Definition:

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

Observation:

- All languages accepted by FAs form the family of regular languages

Regular Languages

- **Examples of regular languages:**

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

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

{all strings with prefix ab}

{all strings without substring 001}

There exist automata that accept these languages.

Non-Regular Languages

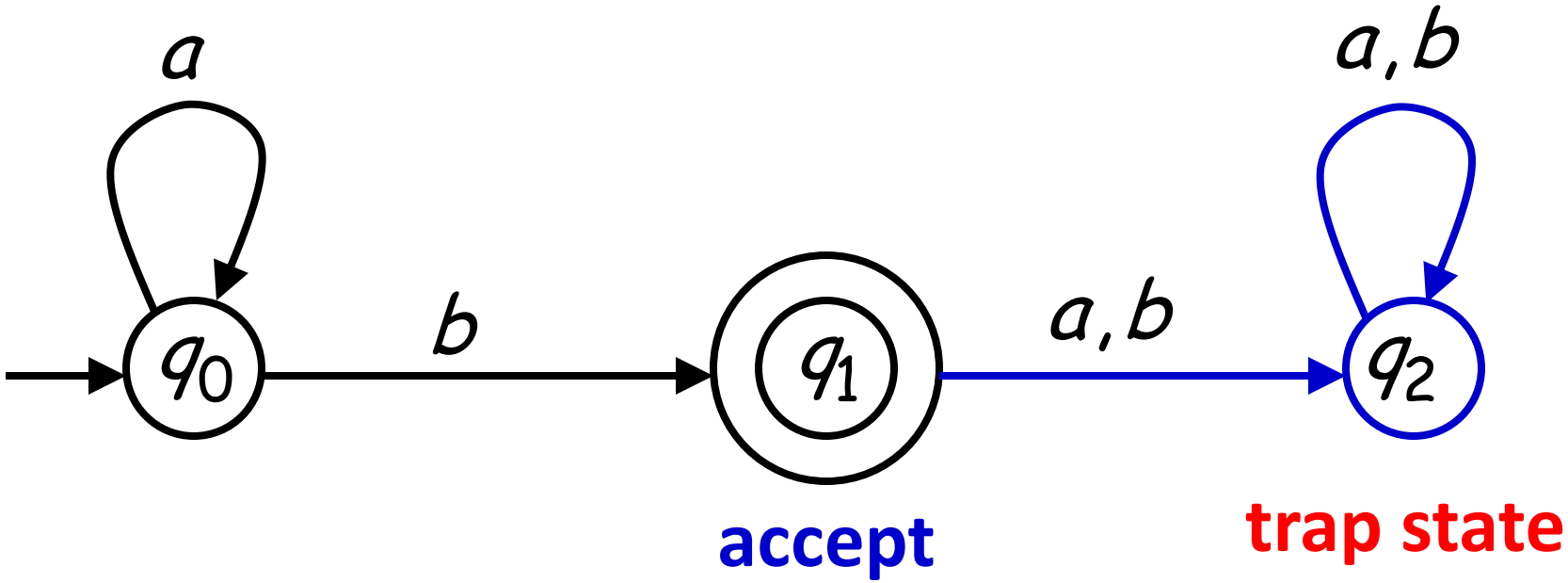
- There exist languages which are <u>not</u> regular :
- Example :

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

There is no FA that accepts such a language.

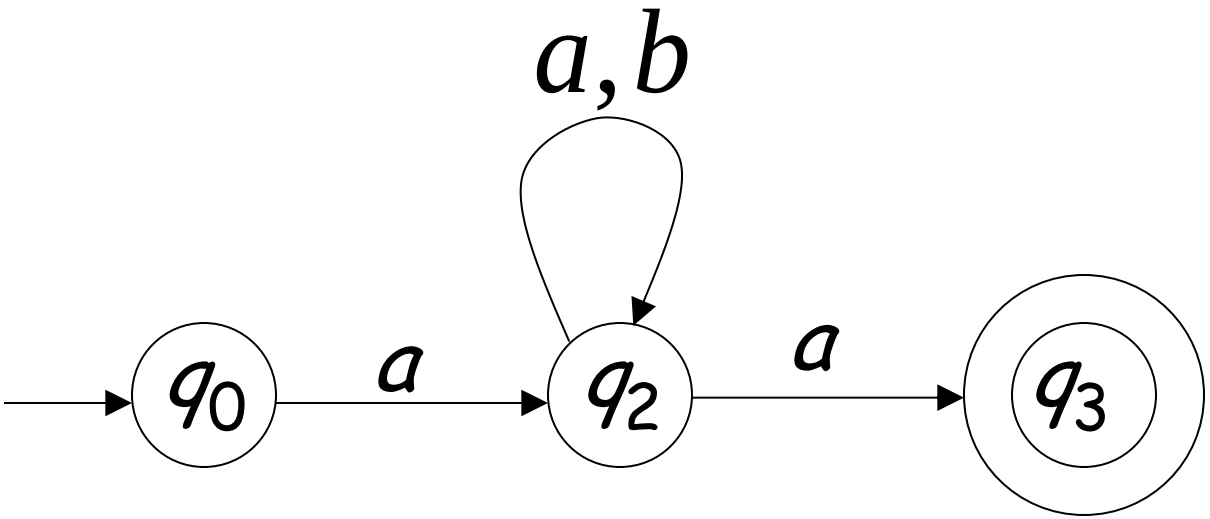
Example of DFA

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



Example (NFA or DFA ?)

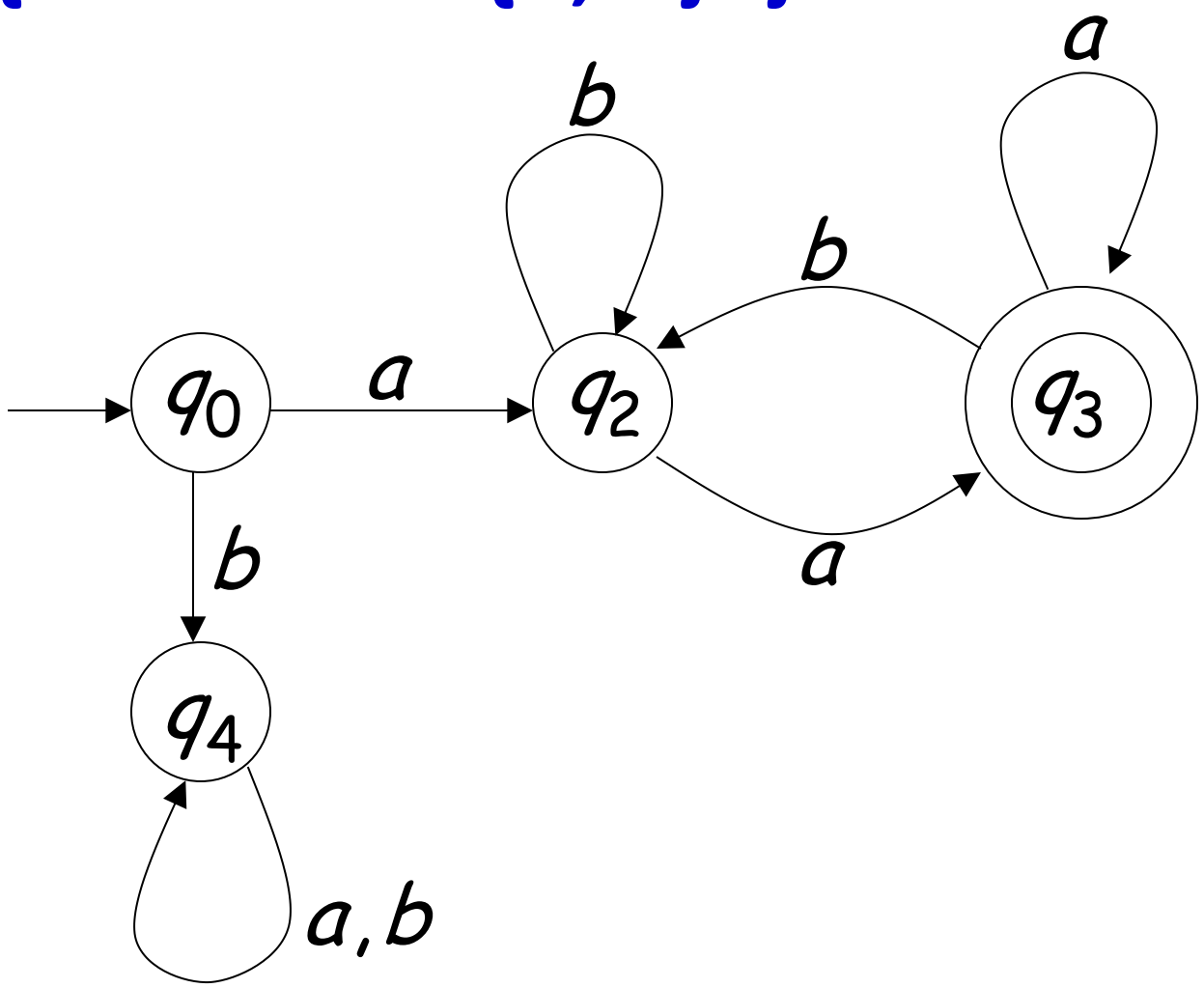
- **Create a FA for *L(M) = {awa : w ∈ {a, b}*}***



- **What happen if the 1st input is *b*?**
- **What happen if the input are *bab*, *abab*, … ?**

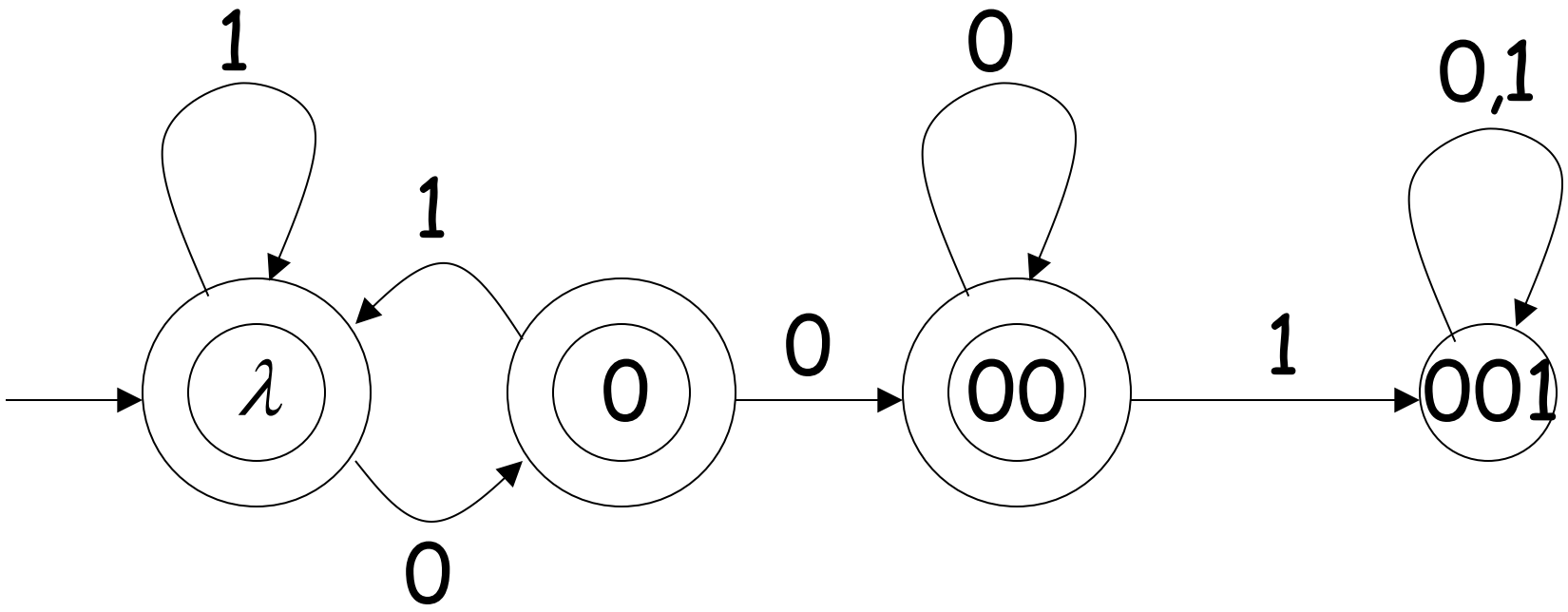
Example of DFA

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



Example

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



Example

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