Theory of Computation

Lecture 3

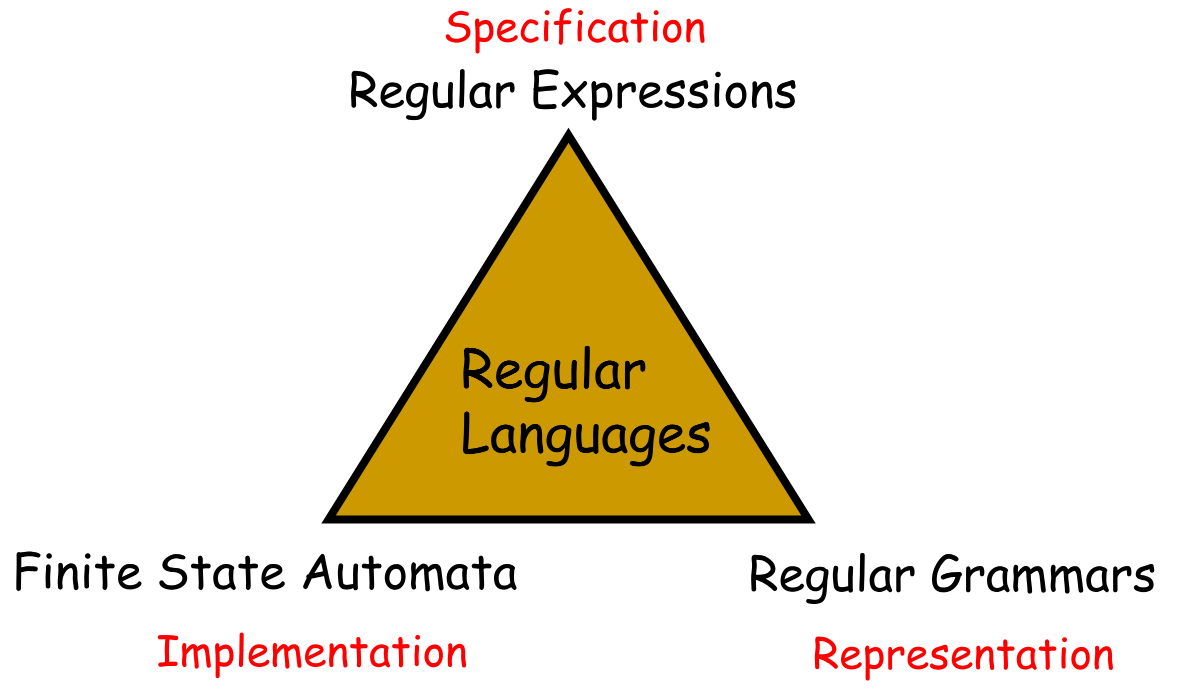
Finite State Automata (FSA)

Syllabus and Terminologies

- Regular Languages .. Regular Sets
  - REs (Regular Expressions)
  - FSMs (or FSA/FA) … Finite State Machines/Automata
    - DFA vs. NFA … Deterministic vs. Non-deterministic FSA
      - Comparison and conversion
    - Examples & Closure Operations
    - Pumping Lemma
- Context Free Languages
  - CFGs … Context Free Grammars
  - PDA … Push Down Automata
  - Parsing: CFG generating strings vs. PDA recognizing strings
- Turing Machine

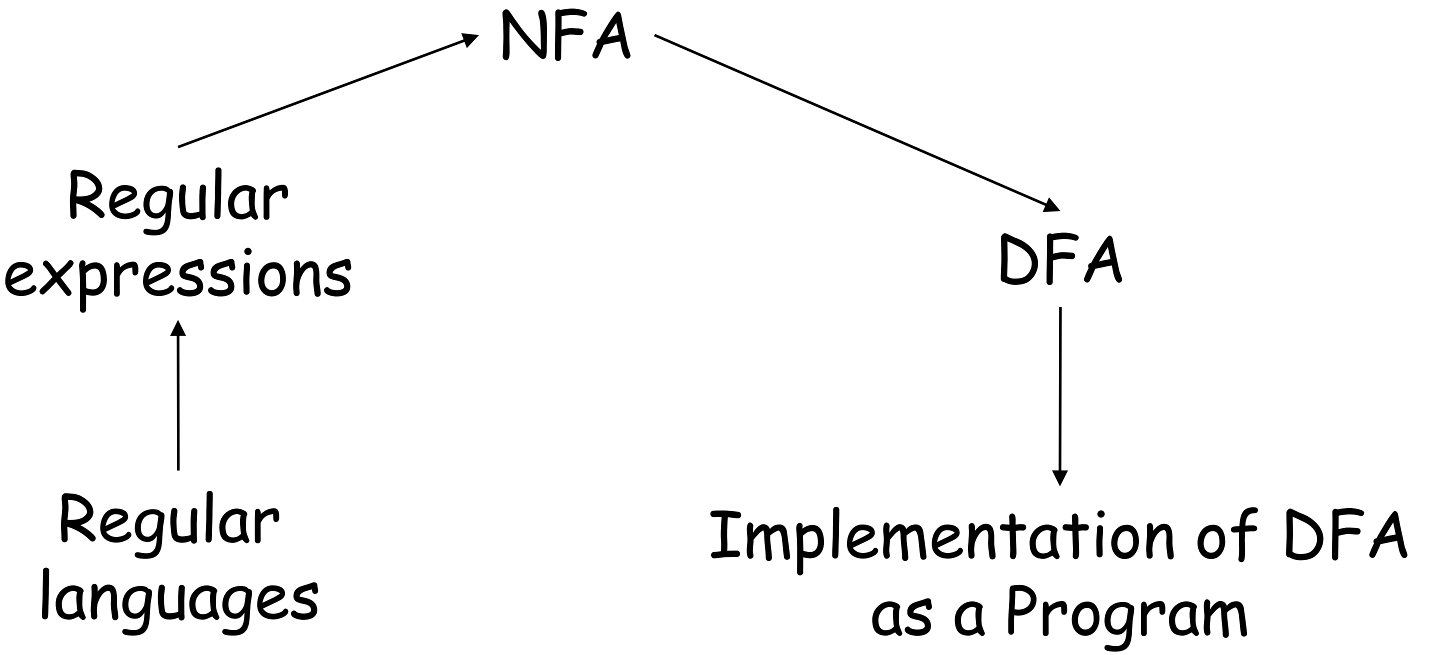
Three Views

- Three equivalent formal ways to approach *Regular Languages*



Regular Language to a Program

- Using Regular Language to generate a program that accepts this language!



What is Automata ?

What is Automata ?

- Automata is the plural of Automaton.
- Automaton is an abstract machine that represents a digital computer with some capabilities
- Thus, the simulation of running an automaton (over some input) to solve a problem gives a clue about the capabilities needed to solve this problem using a real computer.

Types of automata

Automata can be either:

- **Deterministic automata:**

  Each move is uniquely determined by the current configuration; if we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly.

- **Nondeterministic automata:**

  At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions.

Types of automata

Also, Automata can be either:

- **Accepter**: an automaton whose response is limited to a simple “yes” or “No”. Presented with input string, an accepter either accepts or rejects it.

- **Transducers:** a more general automaton that capable of producing strings of symbols as output.

Finite Automata (FA)

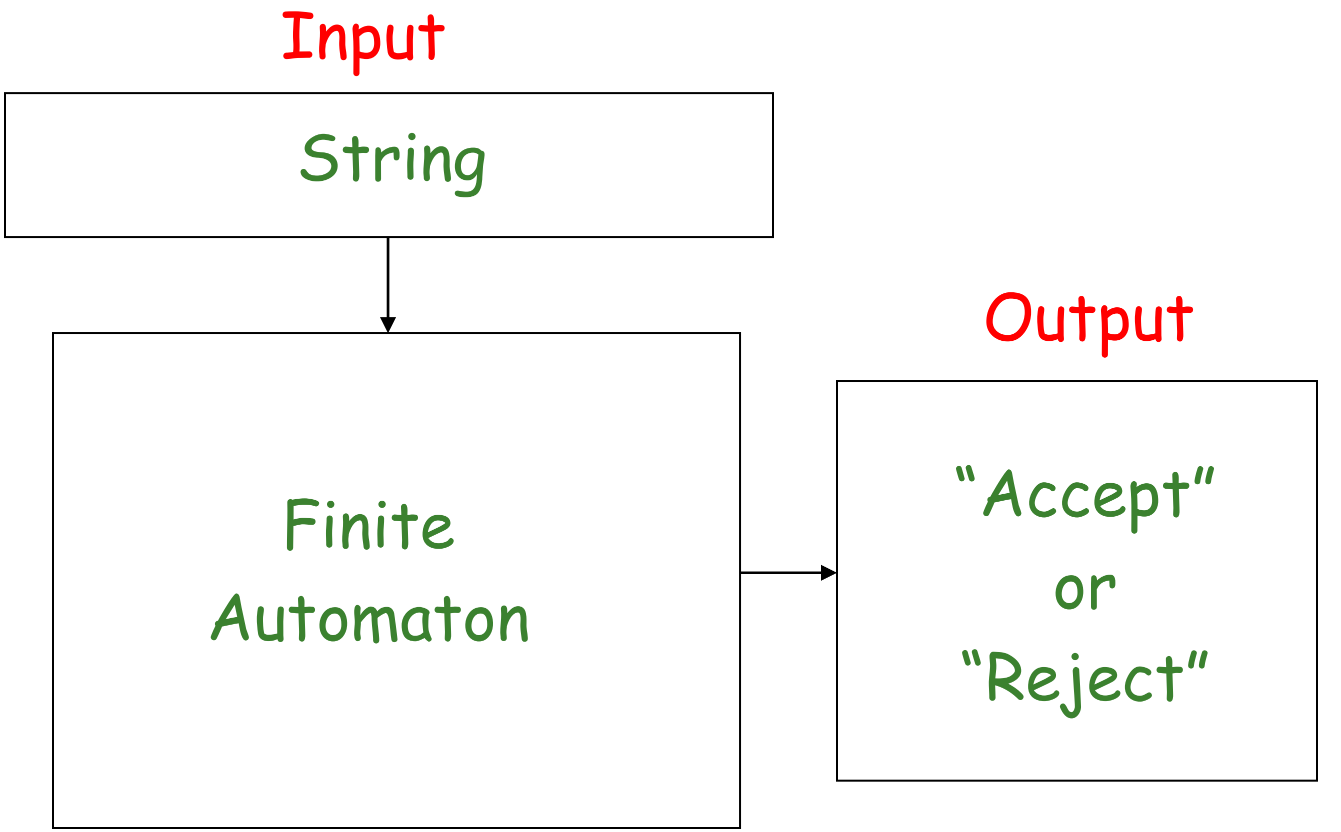
=

Finite State Automata (FSA)

=

Finite State Machines (FSM)

Finite Automaton



Why "finite"?

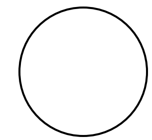
- FA have finite number of states.
- FA have limited memory.
  - It can't remember the processed symbols of the input string.

Representation of FA

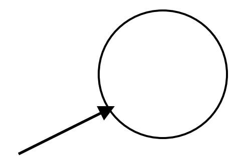
- FSA can be represented either by:
  - State Transition Diagram/Graph
  - State Transition Table

Finite Automata State Graphs

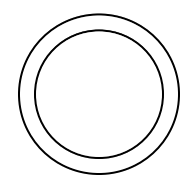
- A state



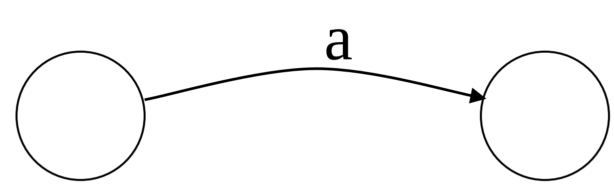
- The start state



- An accepting/final state

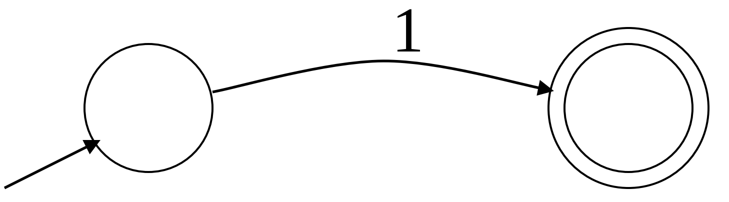


- A transition



A Simple Example

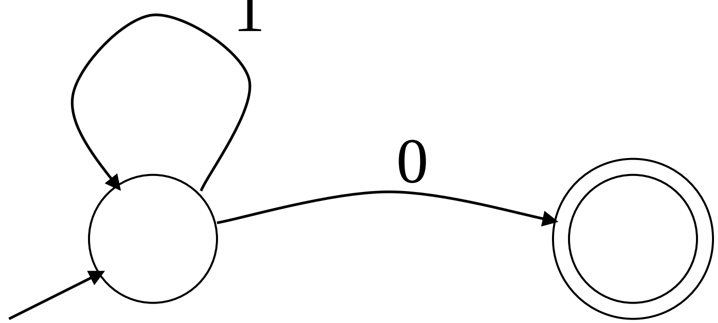
- A finite automaton that accepts only “1”



- A finite automaton accepts a string if we can follow transitions labeled with the characters in the string from the start to some accepting state

Another Simple Example

- A finite automaton accepting any number of 1's followed by a single 0
- Alphabet: {0,1}



111 ✗

1110 ✓

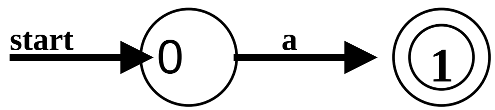
0 ✓

00 ✗

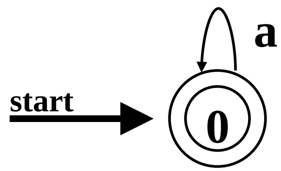
- Check that "1110" is accepted but "0110" is not
- What is the RE corresponding to this FA?

Simple notations

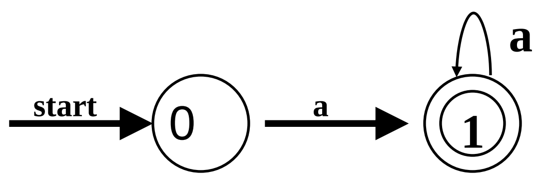
a



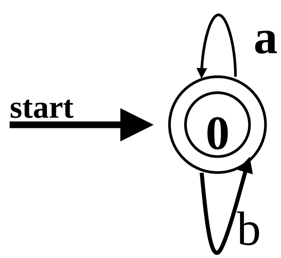
a*

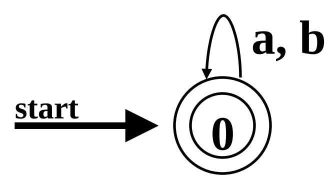


a+



(a+b)*

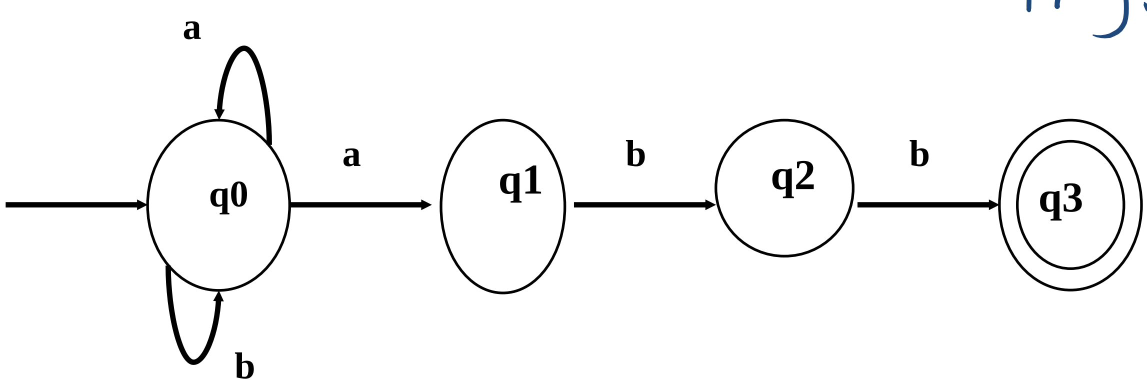




Another Example

- Example transition diagram to recognize $(a+b)^*abb$

try: aabb



possible paths?

scenarios:

① $q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} ?$ X

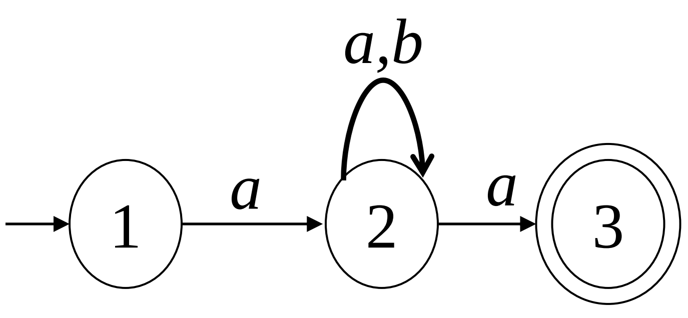
② $q_0 \xrightarrow{a} q_1 \xrightarrow{a} ?$ ×

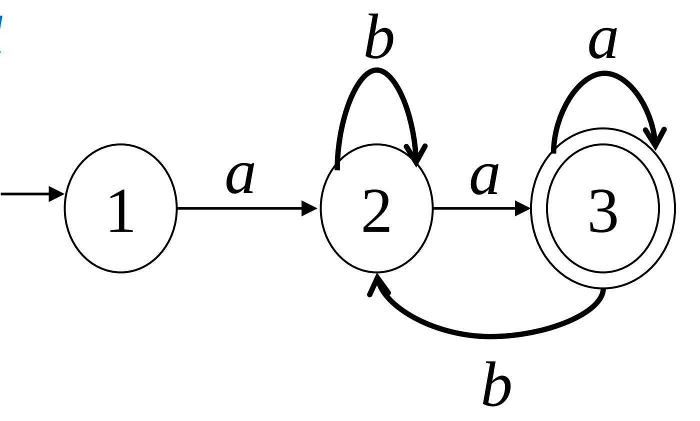
③ $q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{b} q_3$ ✓

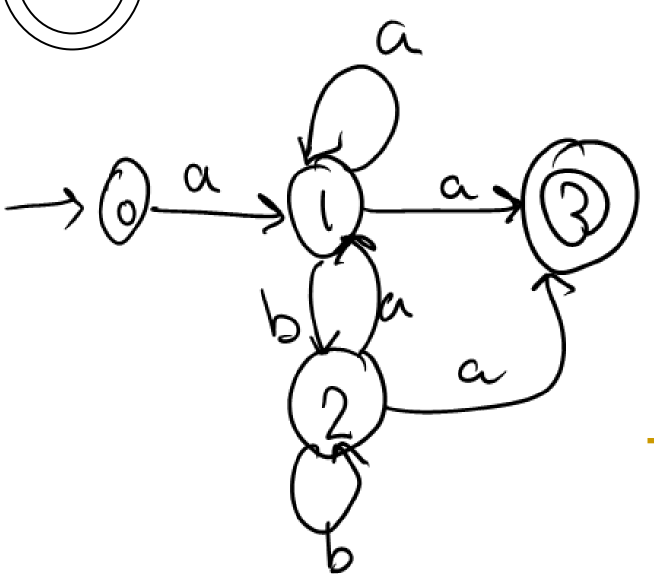
Practice

- Draw the transition diagram for recognizing the following regular expression. (3 possible answers!)

$$a(a+b)^{*}a$$

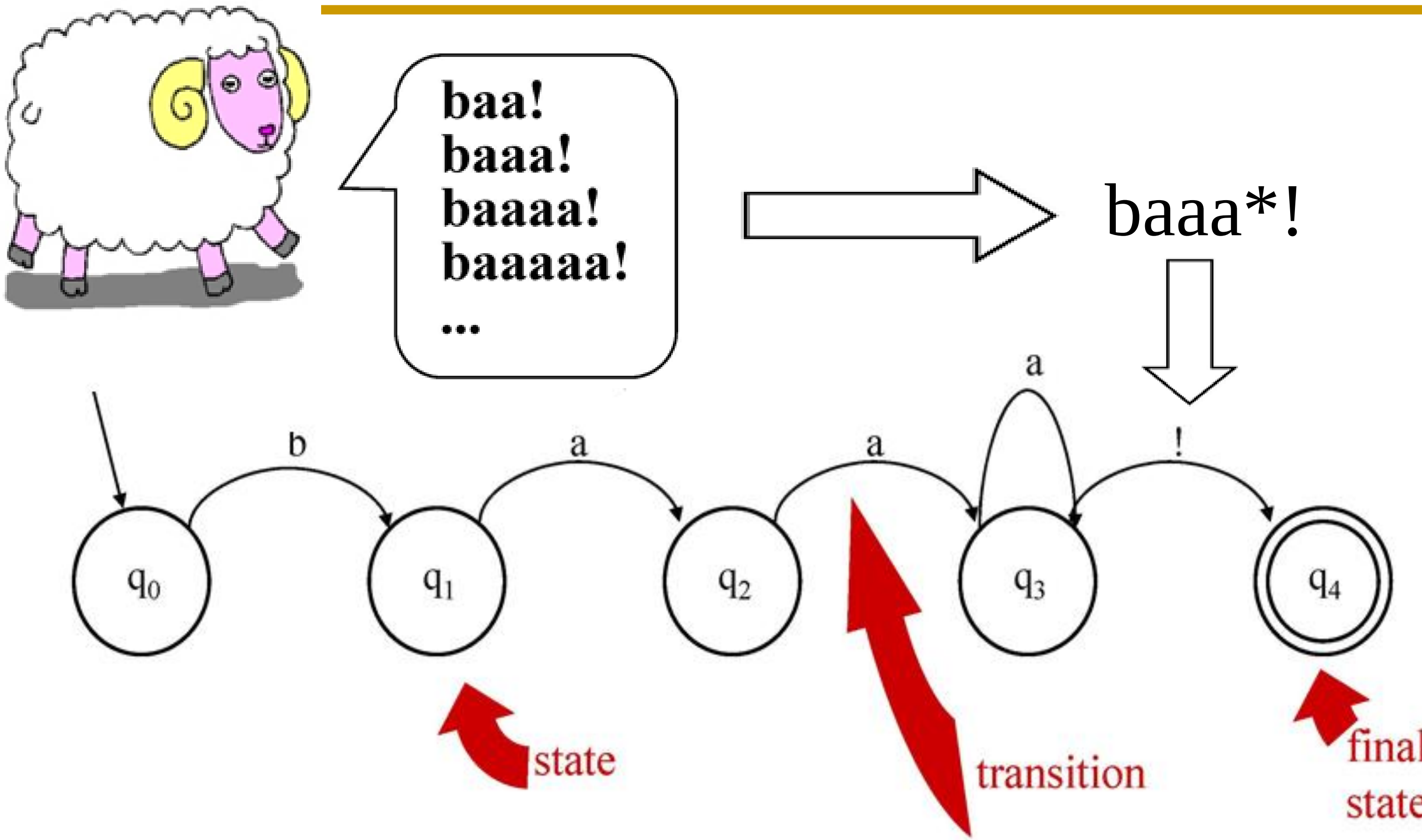






Try:
aababababba

Finite automata



FA as a Transition Diagram

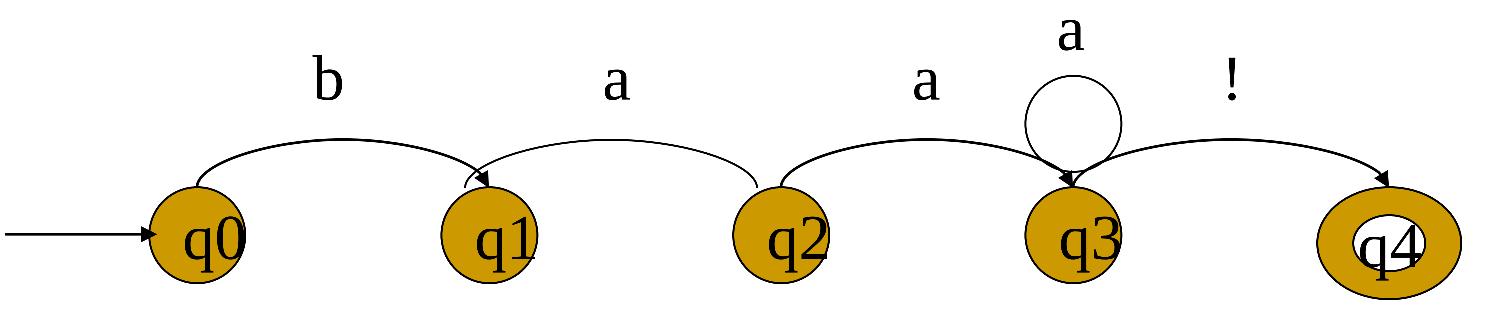
- FA can be *represented* using transition diagrams.
- A transition diagram has:
  - **States** represented by circles;
  - An **Alphabet** (Σ) represented by labels on edges;
  - **Transitions** represented by labeled directed edges between states. The label is the input symbol;
  - One **Start State** shown as having an arrow head;
  - One or more **Final State**(s) represented by double circles.

Finite-State Automata

- **Definition:** A *finite-state automaton* is a 5-tuple $M=(S, I, f, S_0, F)$ where
  - S is a finite set of *states*
  - I is a finite *input alphabet*
  - $f:(S_i \times I) \rightarrow S_j$ is a *transition function* from a (***state* x *input)*** pair to another destination state
  - S_0 is the *initial state*
  - $F \subseteq S$ is a set of *final states*

Example

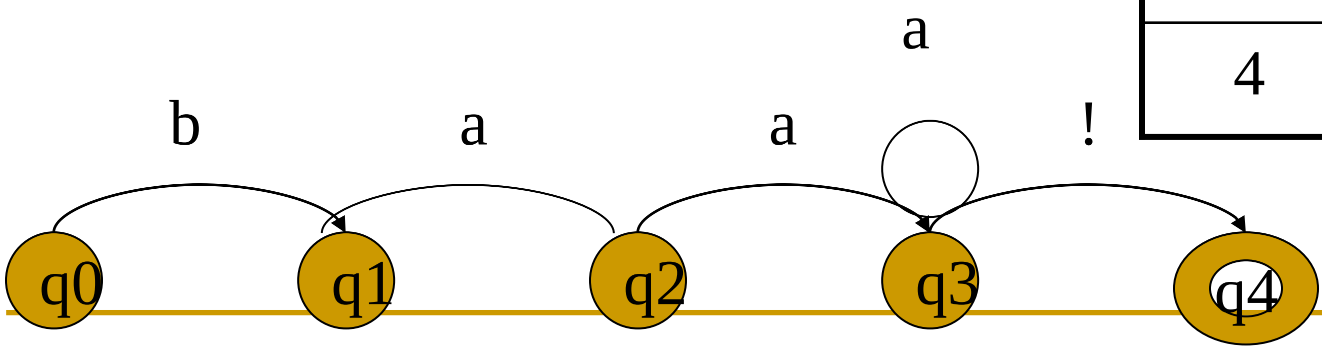
- FSA is a 5-tuple (Q, Σ, δ, q0, F) consisting of
  - Q: set of states {q0,q1,q2,q3,q4}
  - Σ: an alphabet of symbols {a,b,!}
  - δ(q,i): a transition function mapping Q x Σ to Q
  - q0: a start state in Q
  - F: a set of final states in Q {q4}



Another View: A State Transition Table for SheepTalk

- Remember that there are 2 ways to represent a FSM:
  - State Transition Diagram
  - State Transition Table

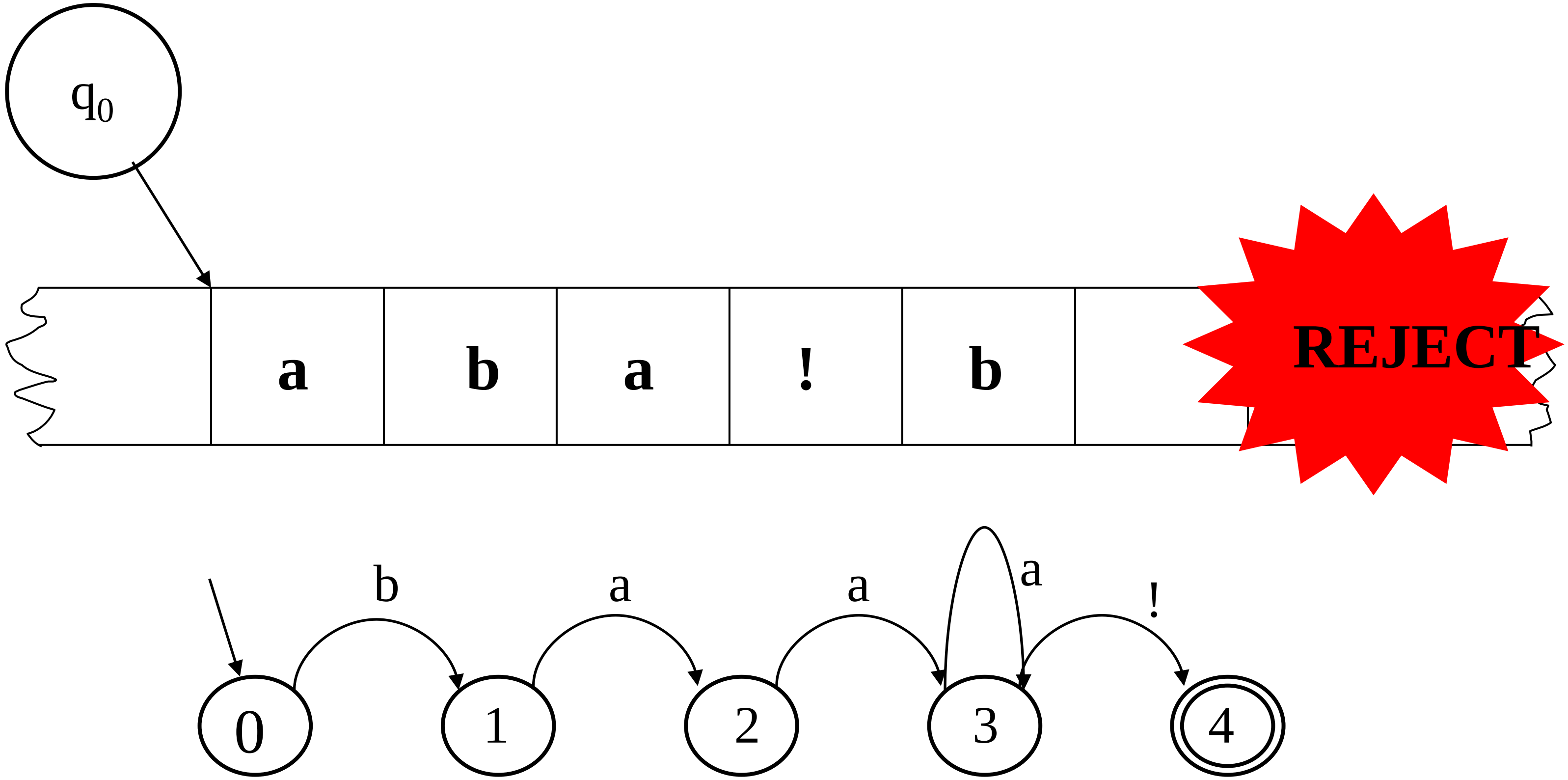
State	Input		
	b	a	!
0	1	-	-
1	-	2	-
2	-	3	-
3	-	3	4
4	-	-	-



FSA Recognition

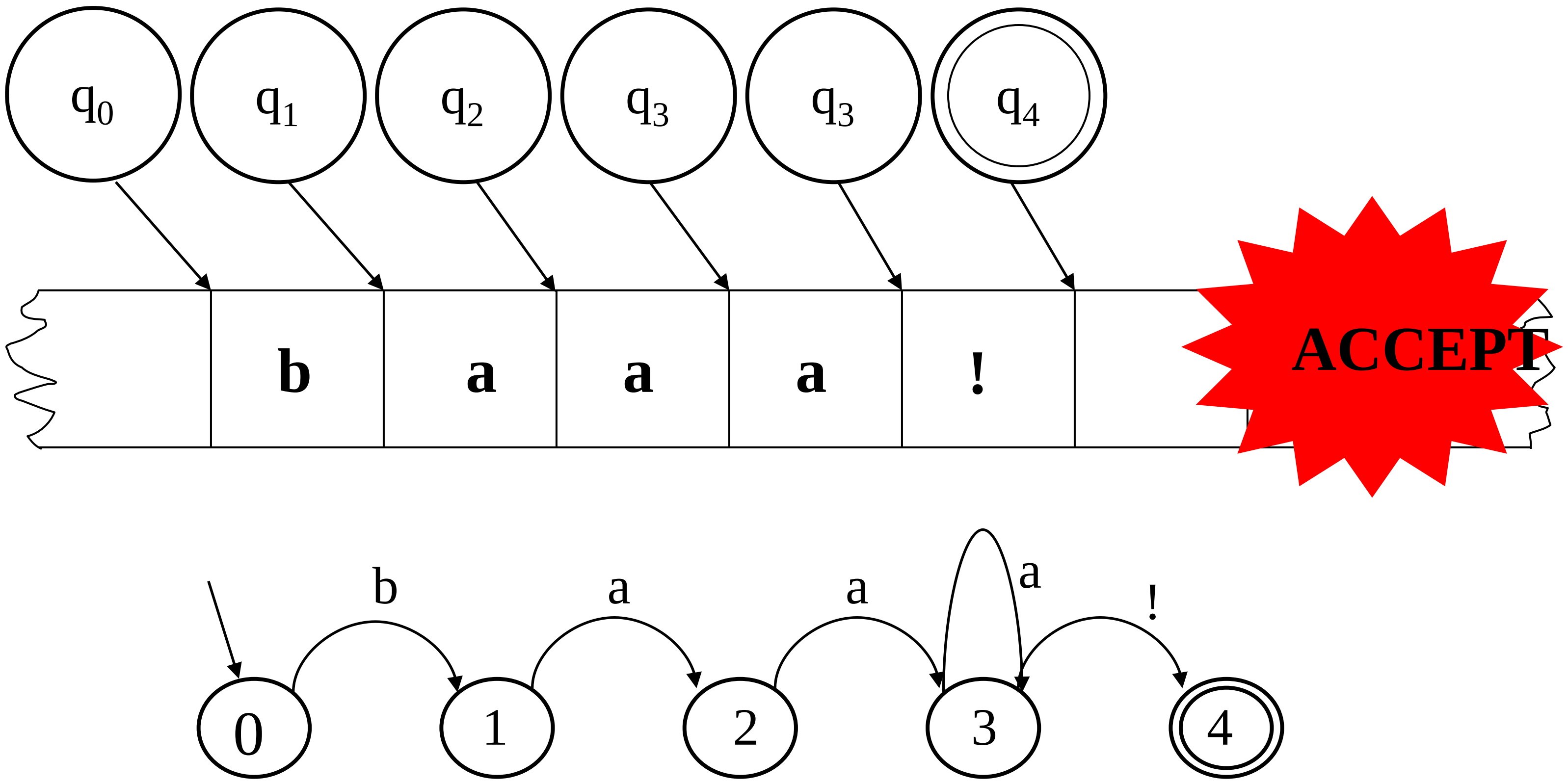
- Possible Goals
  - Determine whether a string should be *accepted* by a machine
  - Or… determine whether a string is in the language defined by the automaton
  - Or… determine whether a regular expression matches a string

Input Tape



Slide from Dorr/Monz

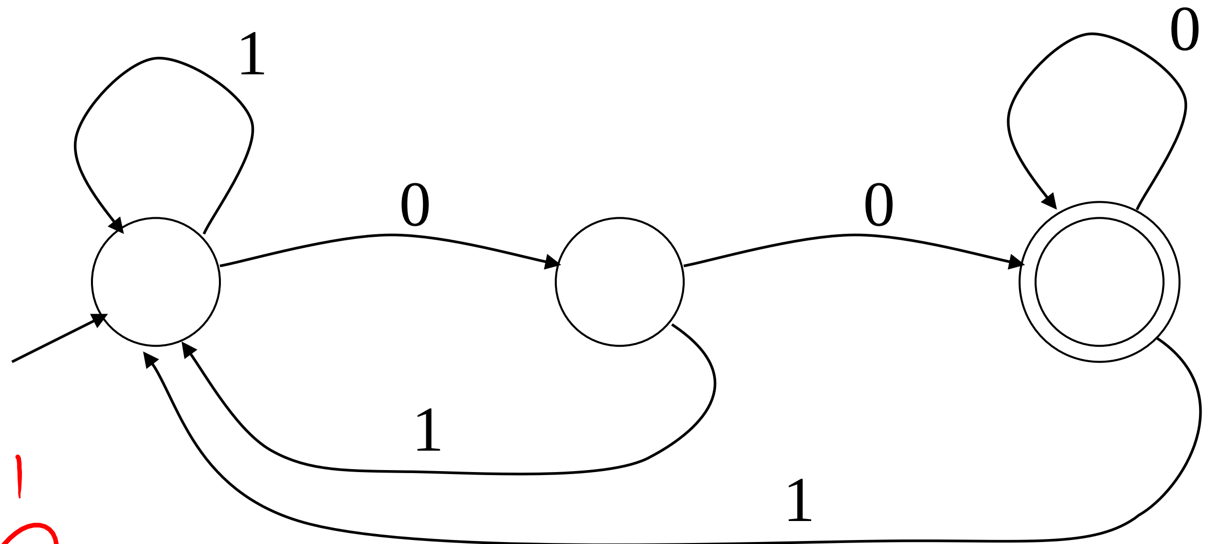
Input Tape

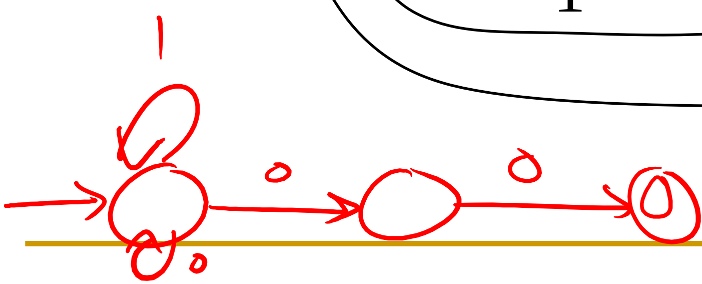


Slide from Dorr/Monz

Another Example

- Alphabet {0,1}
- What language does this machine recognize?

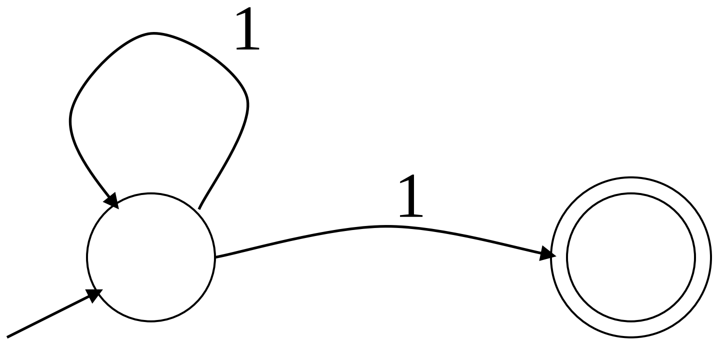




$(1+0)^* 00$

And Another Example

- Alphabet still { 0, 1 }



The operation of the automaton is not completely defined by the input

- On input “11” the automaton could be in either state

Language Recognition

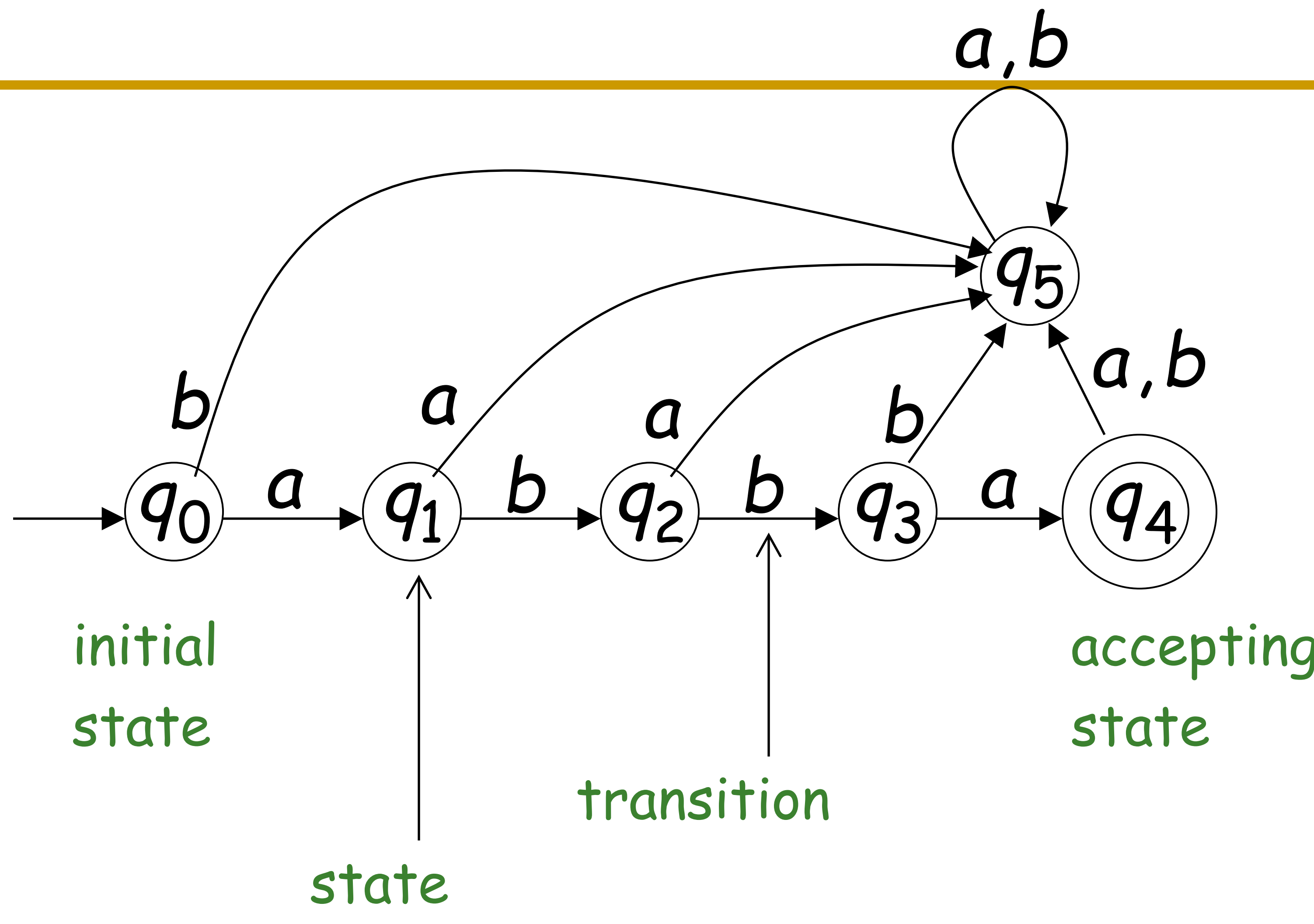
- **Definition:** A finite-state automaton *accepts* (or *recognizes*) a string x if $f(S_0, x) \in F$. That is, the finite state automaton ends up in a final state.
- **Definition:** The *language accepted* (or *recognized*) by a finite-state automaton M, denoted by $L(M)$, is the set of all strings recognized by M.
- **Definition:** Two finite-state automata are *equivalent* if they recognize the same language.

How FA can take decision?

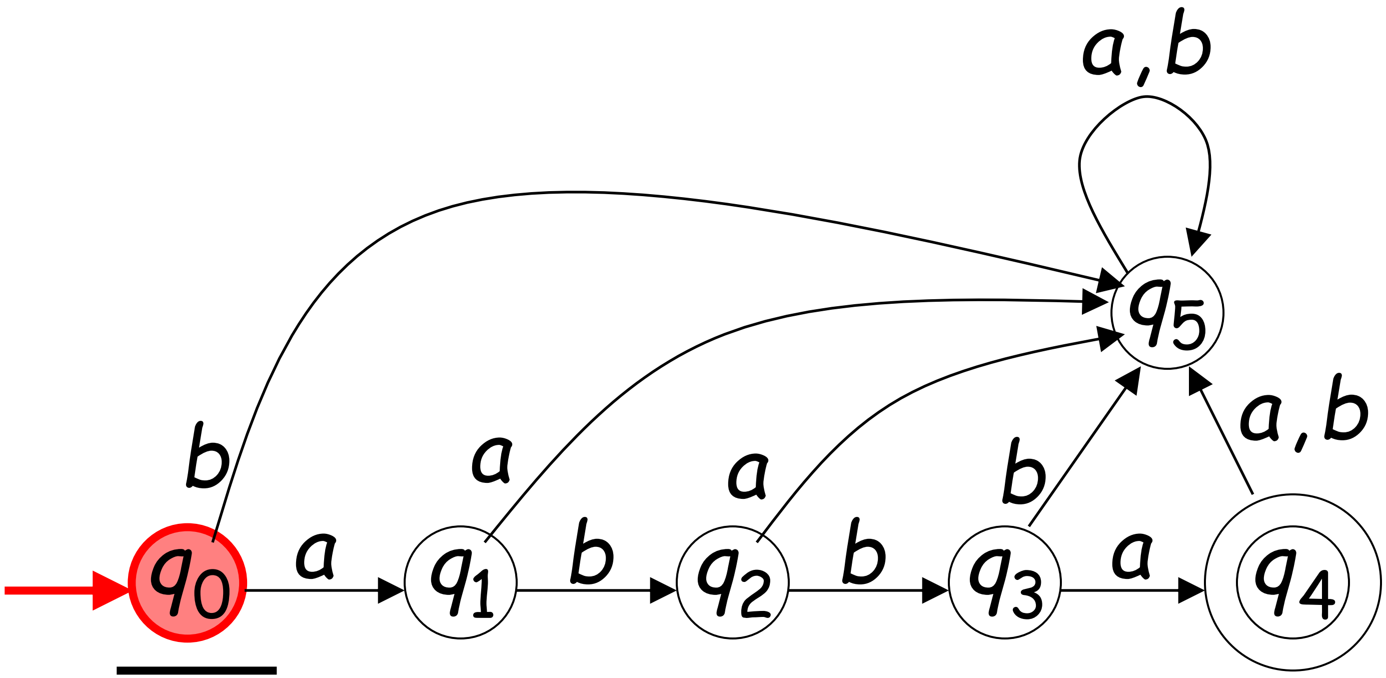
- It ***ACCEPTS*** *the input string if:*
  - If end of input, and an acceptance state is reached
- It ***REJECTS*** *the input string if:*
  - If no transition to a final state is possible (got stuck)
  - If tape still has input, even if a final state is reached

Transition Graph

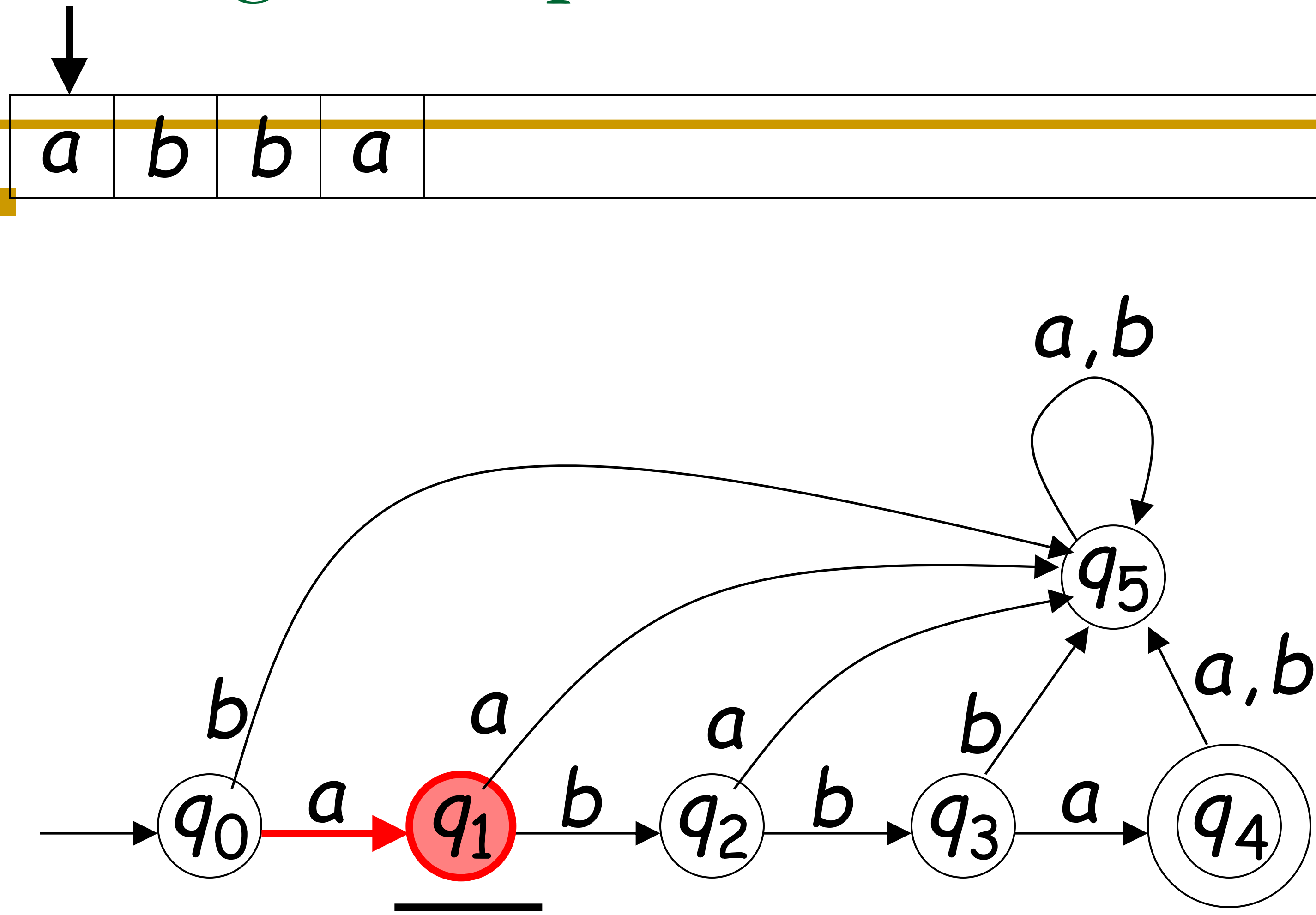
-

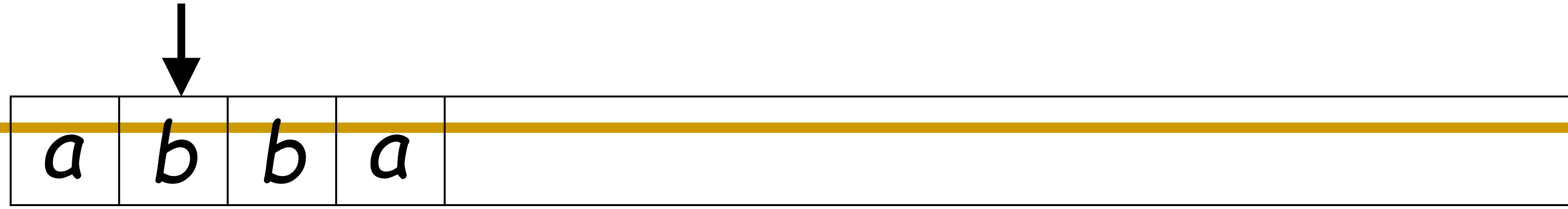


Initial Configuration

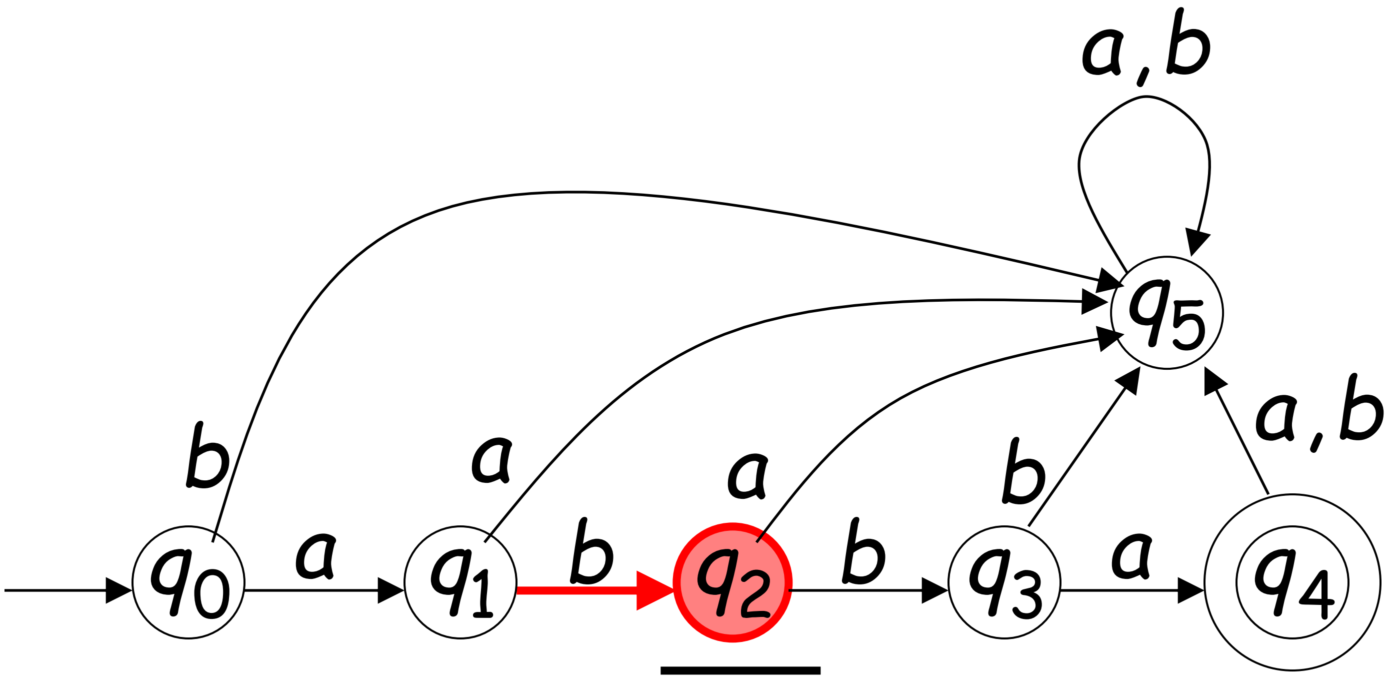


Reading the Input

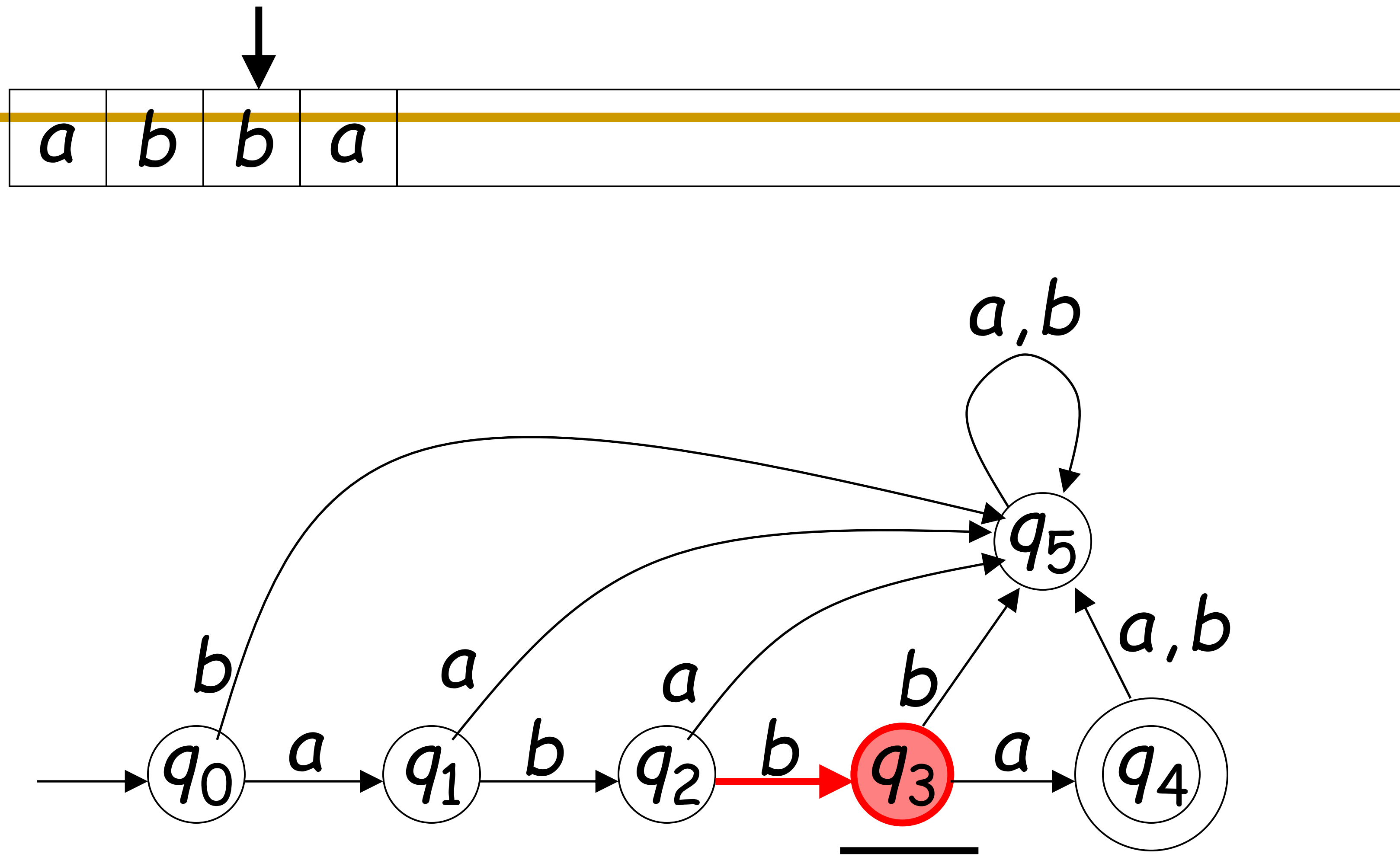


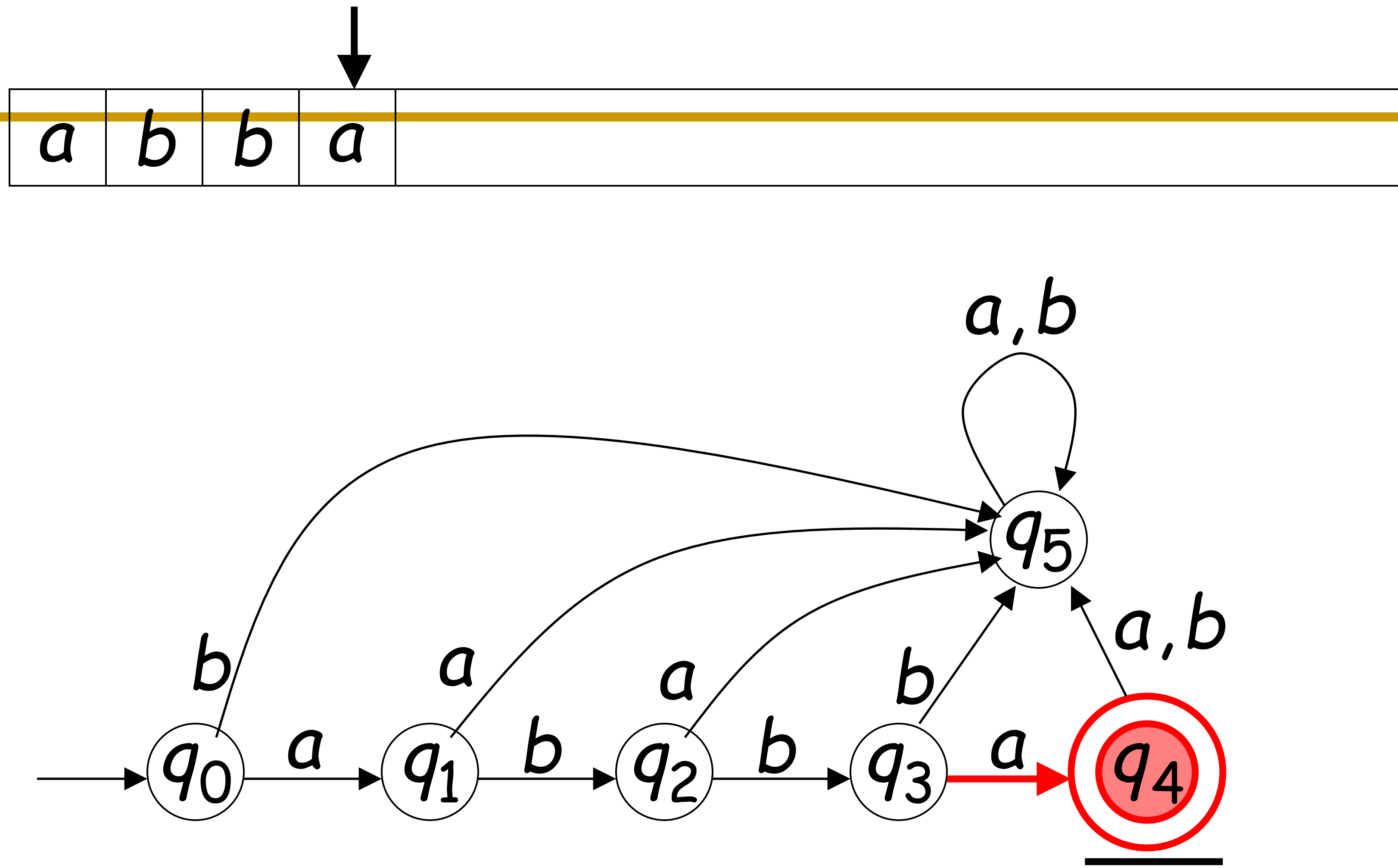


a
b
b
a



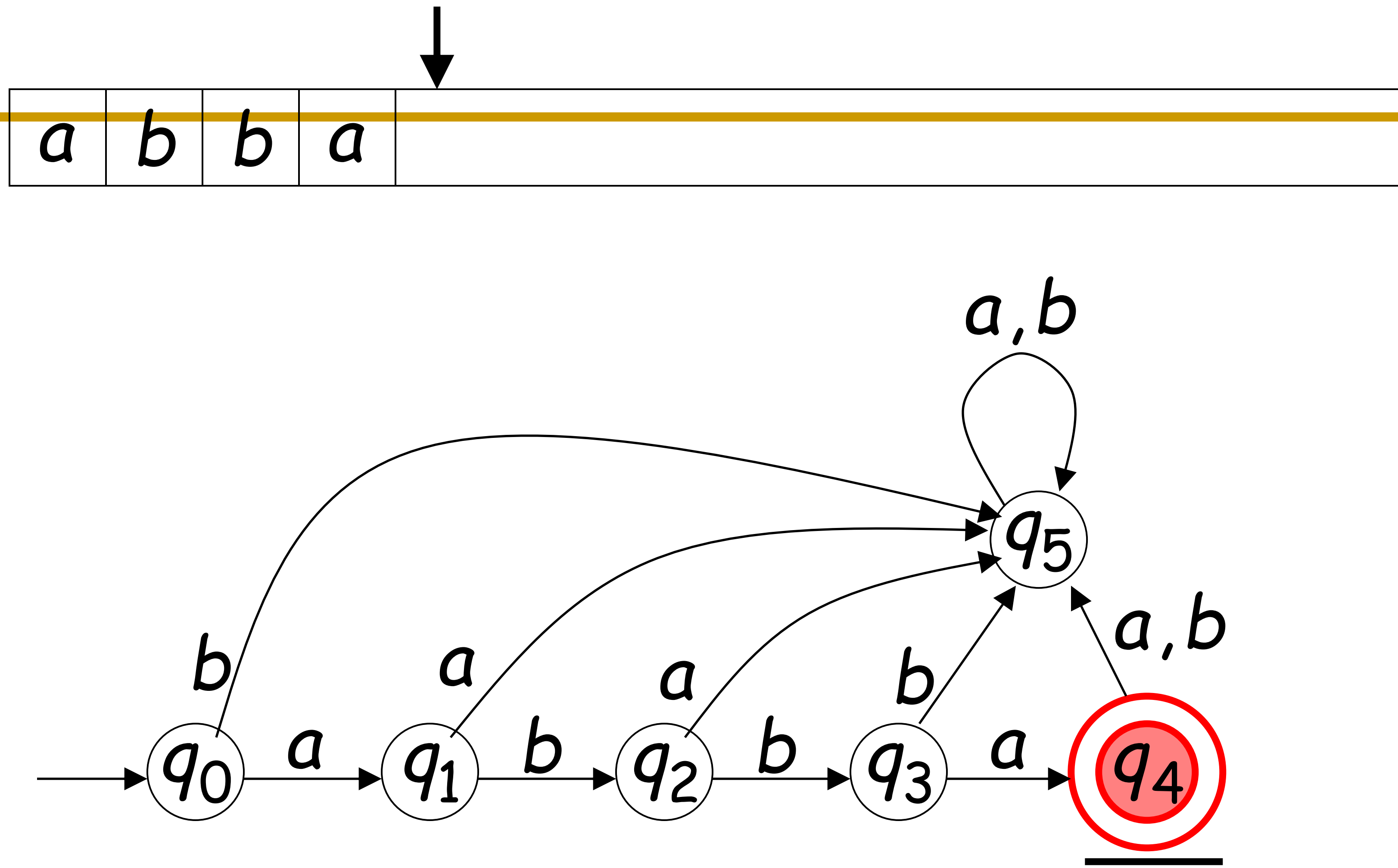
a,b
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4
q5


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



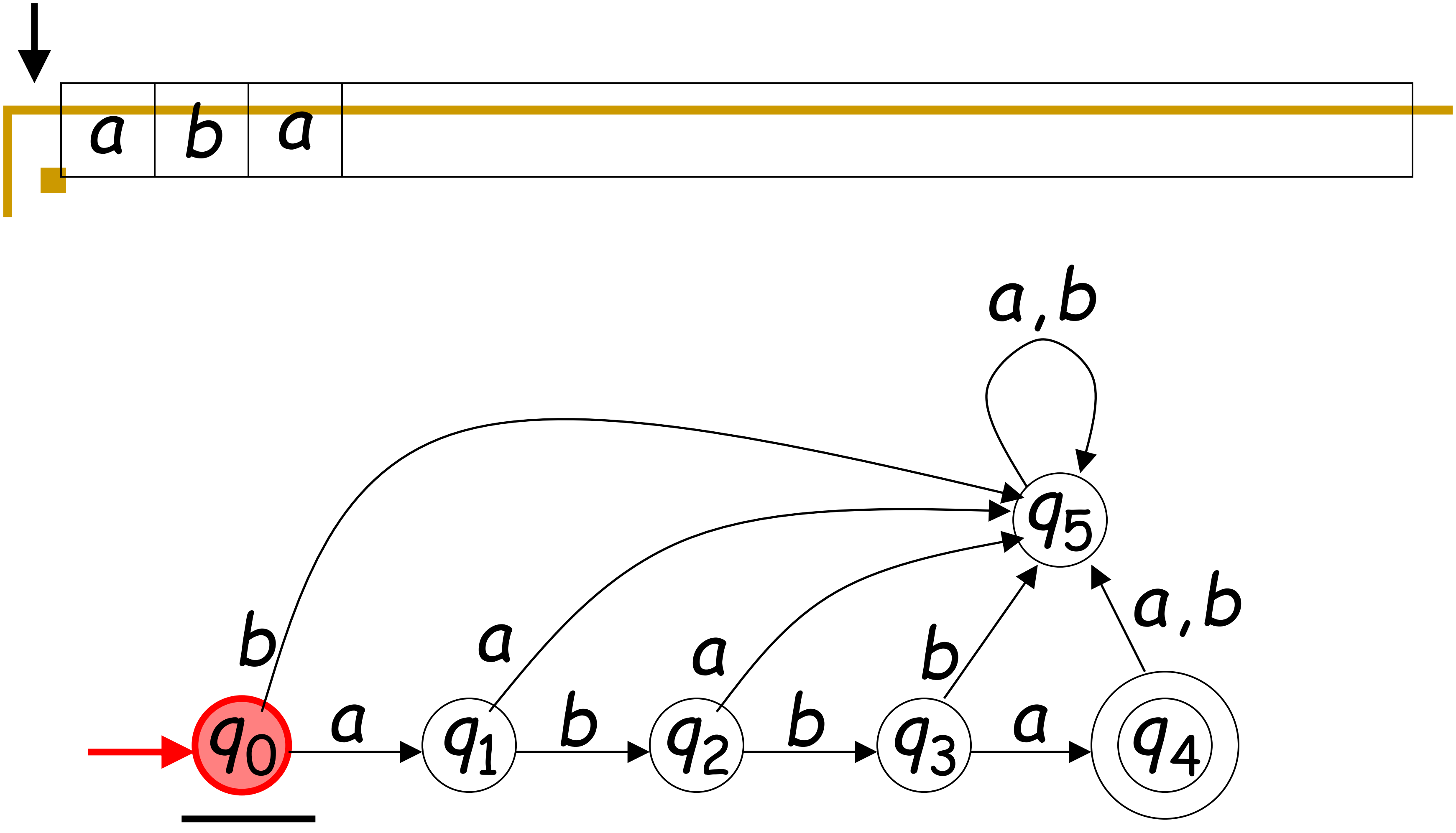
a
b
b
a
a, b
q5
b
a
a
b
a, b
q0
a
q1
b
q2
b
q3
a
q4

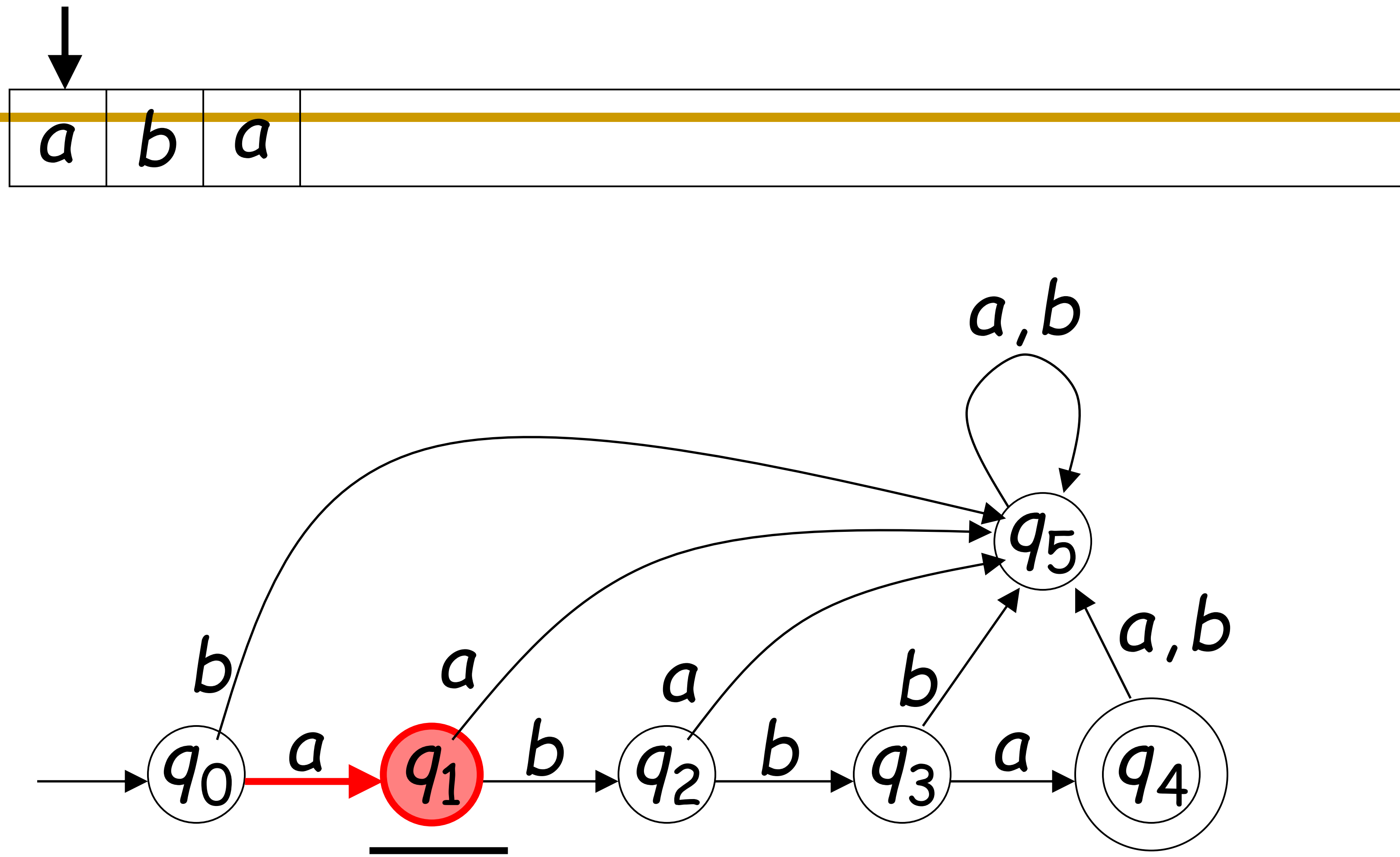
Input finished

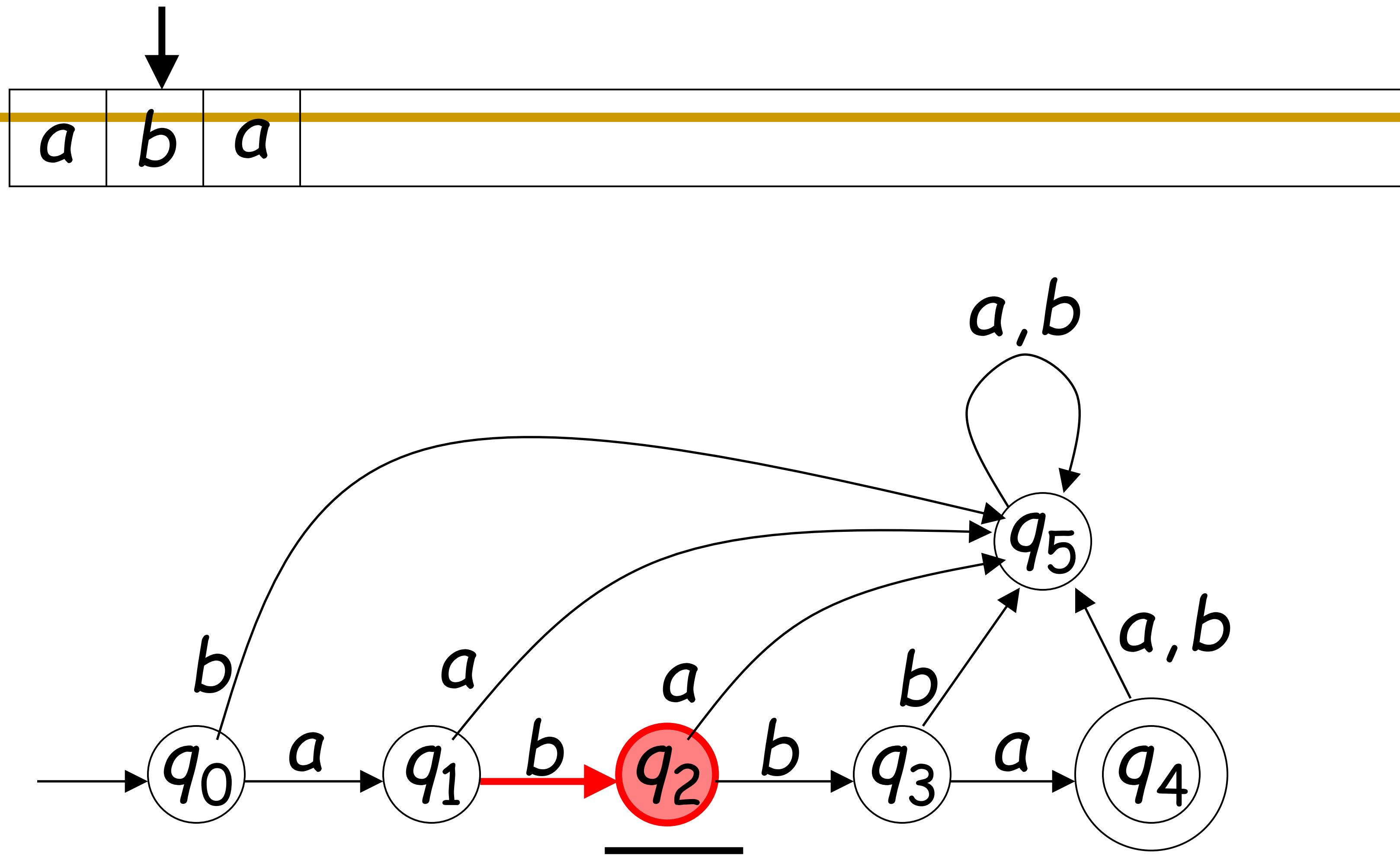


accept

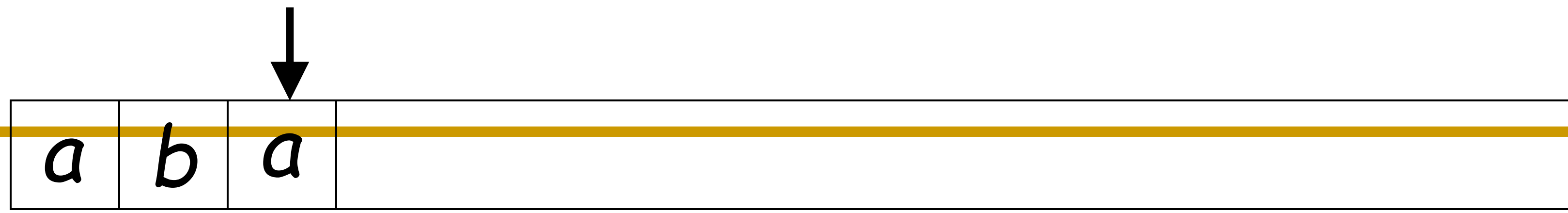
Rejection

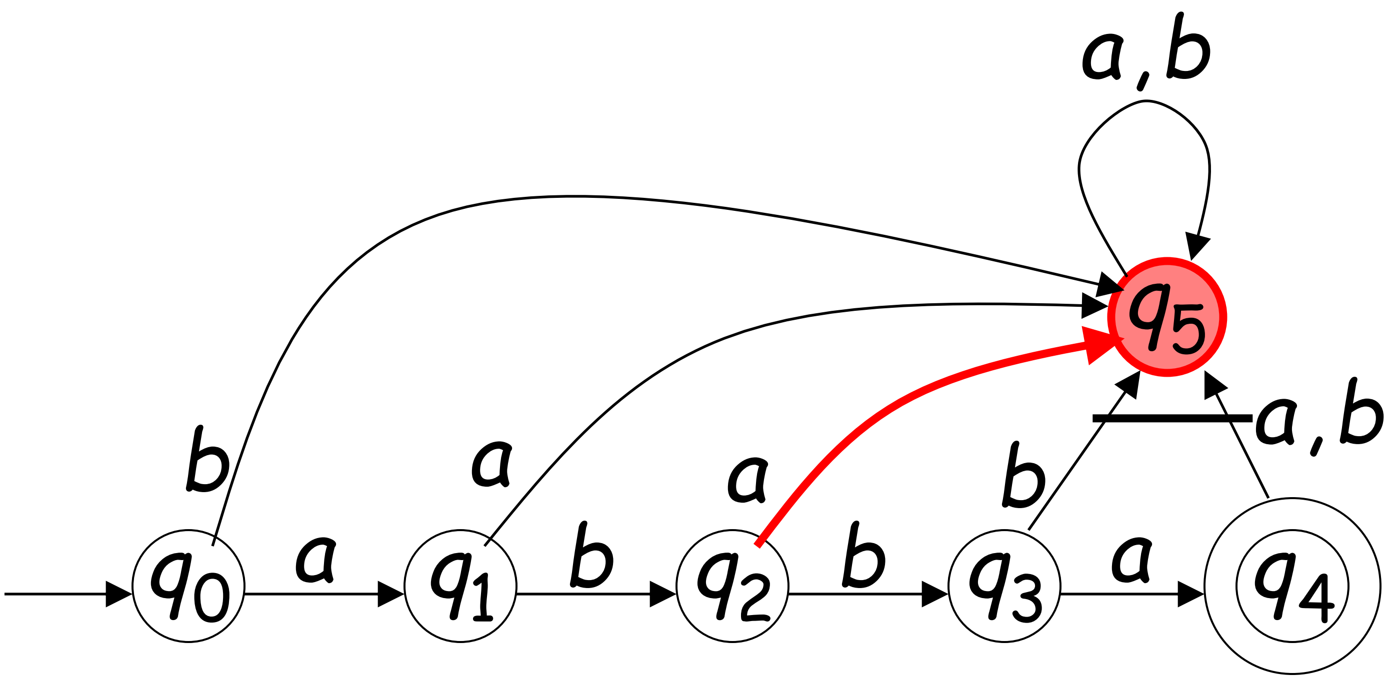



a
b
a
a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4

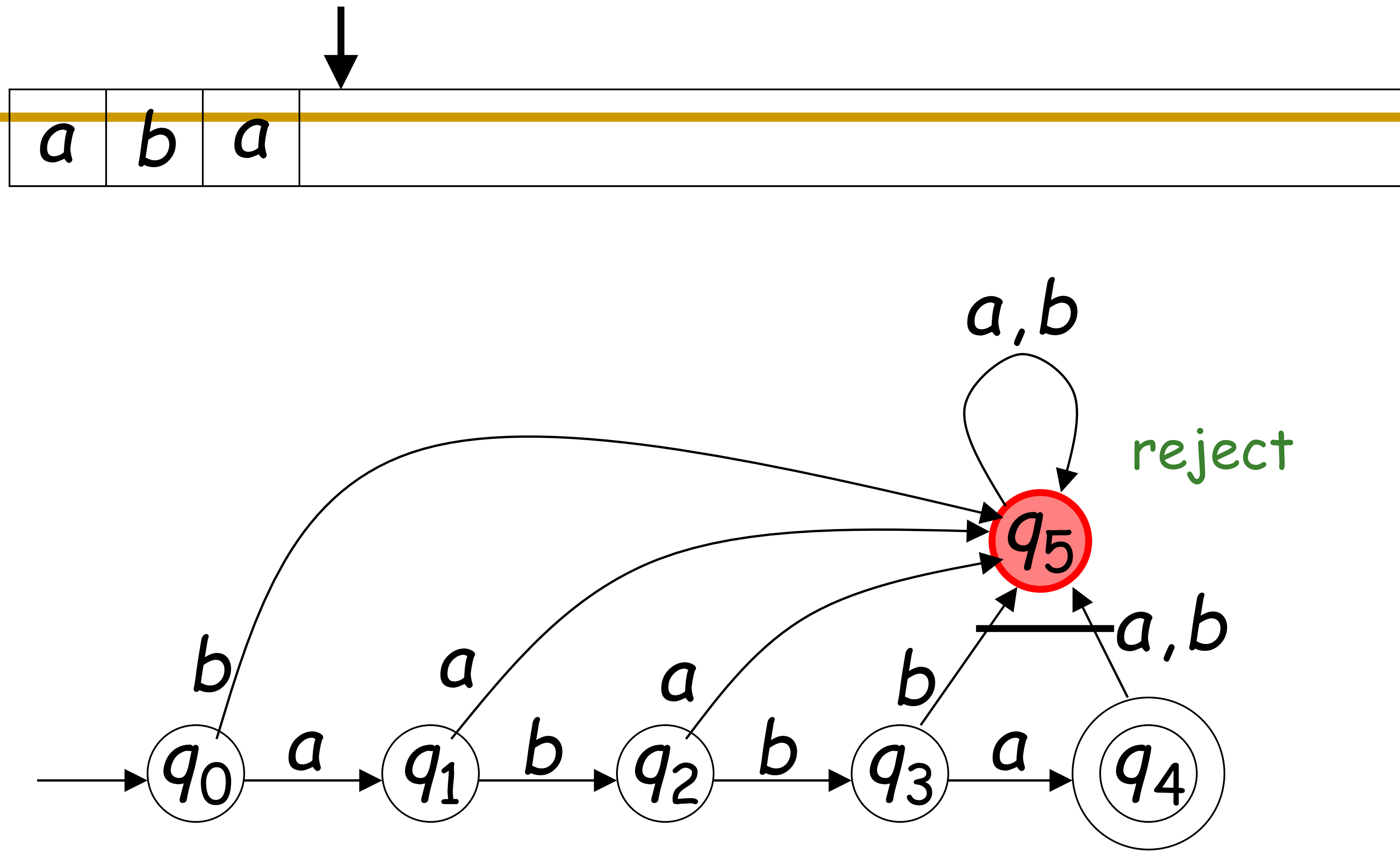


a
b
a
a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4

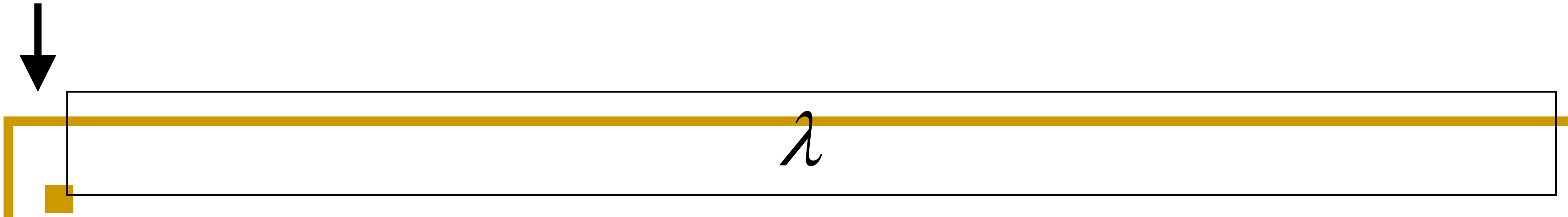

a
b
a

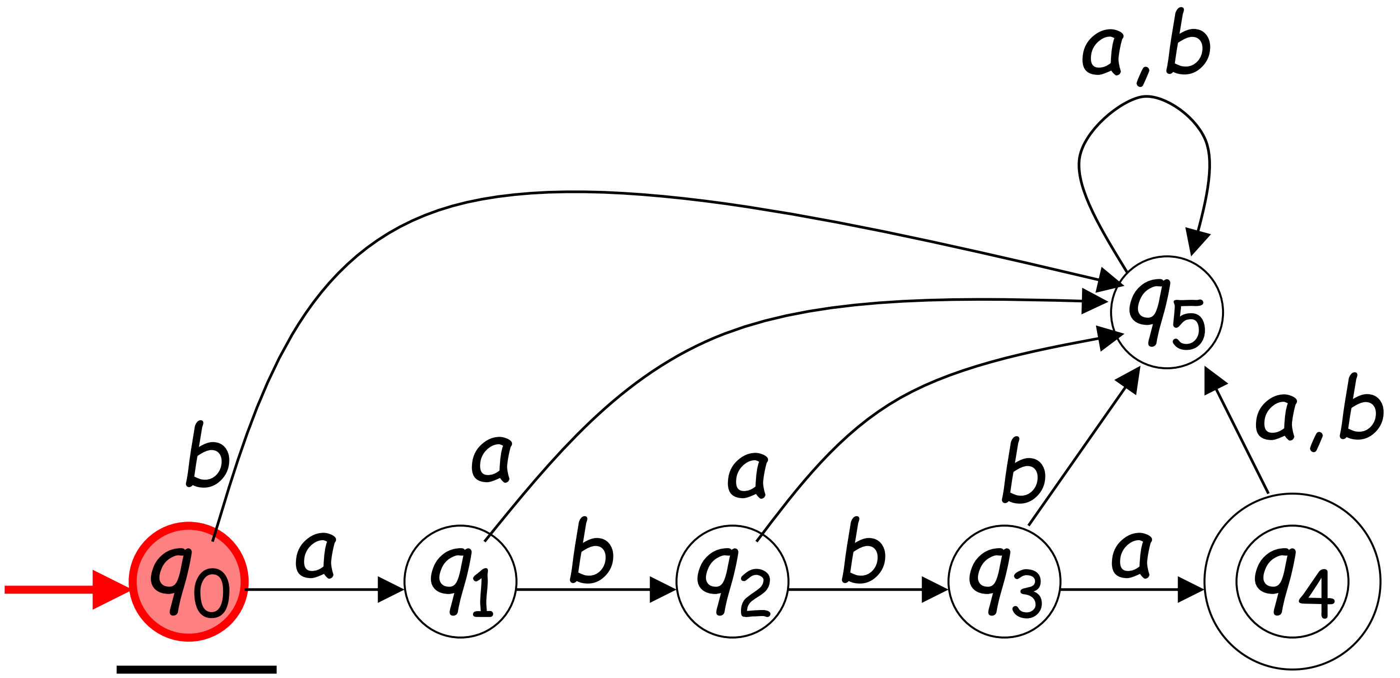

a,b
b
a
a
b
a,b
a
b
b
a
q0
q1
q2
q3
q4
q5

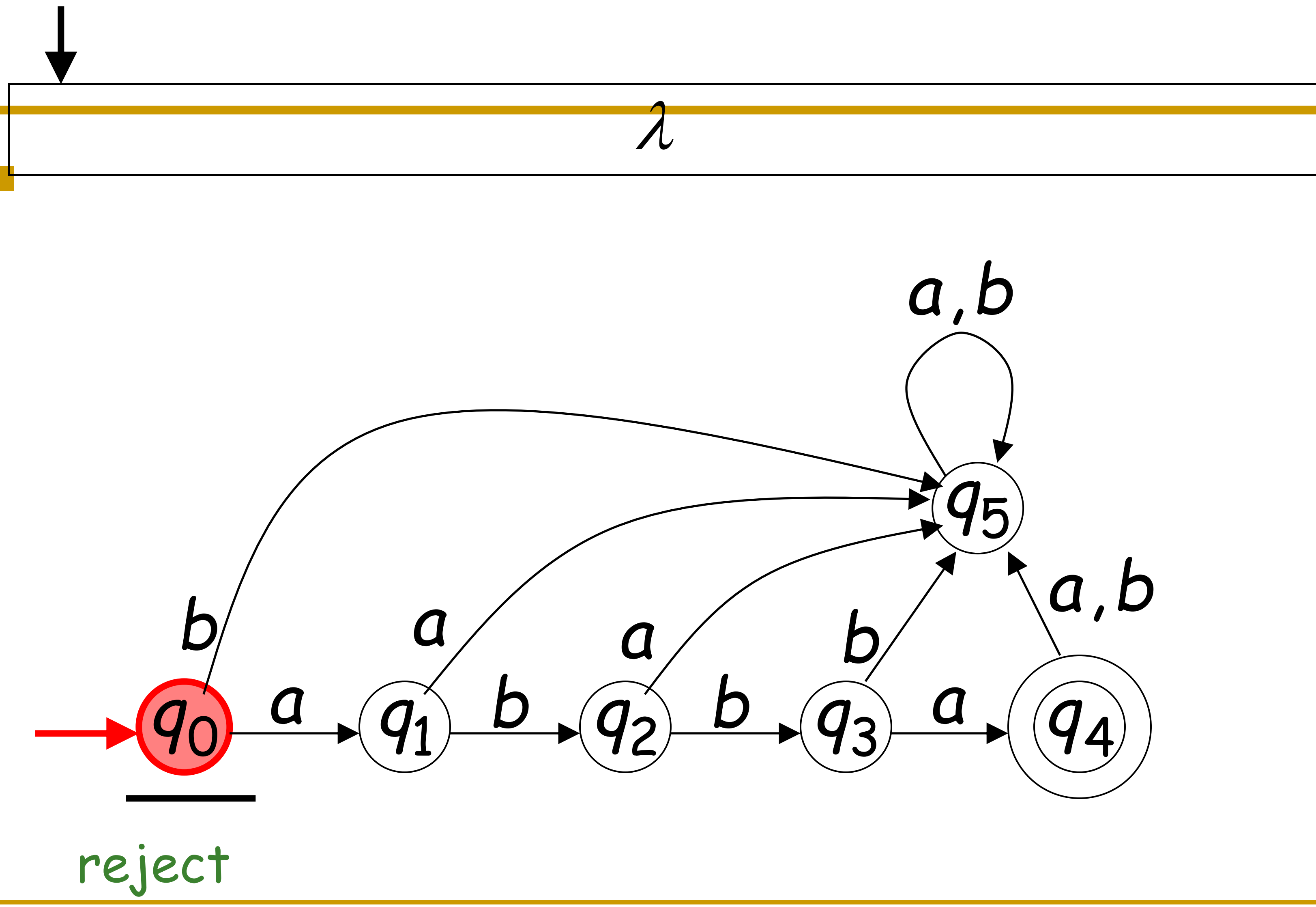
Input finished



Another Rejection



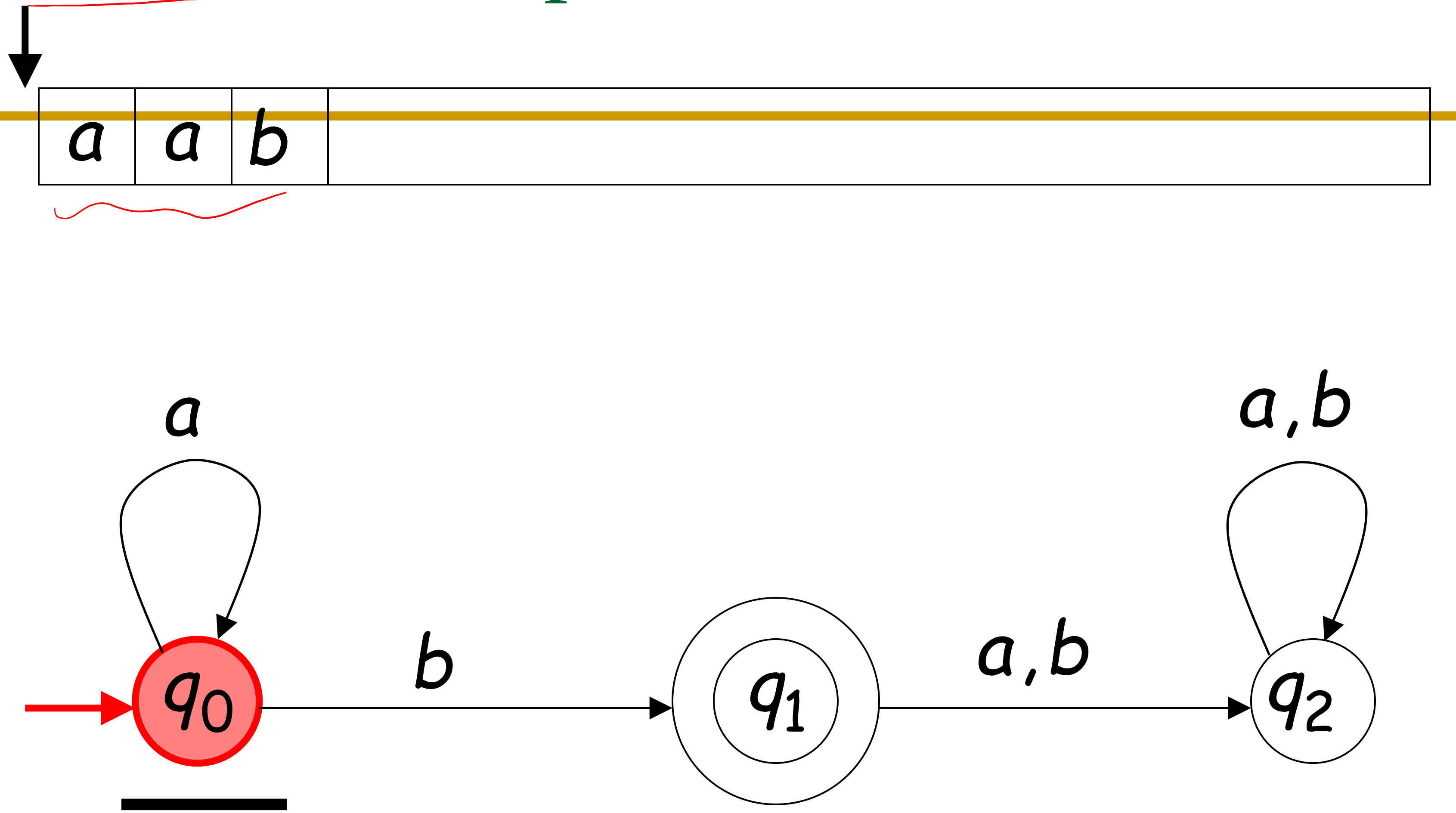


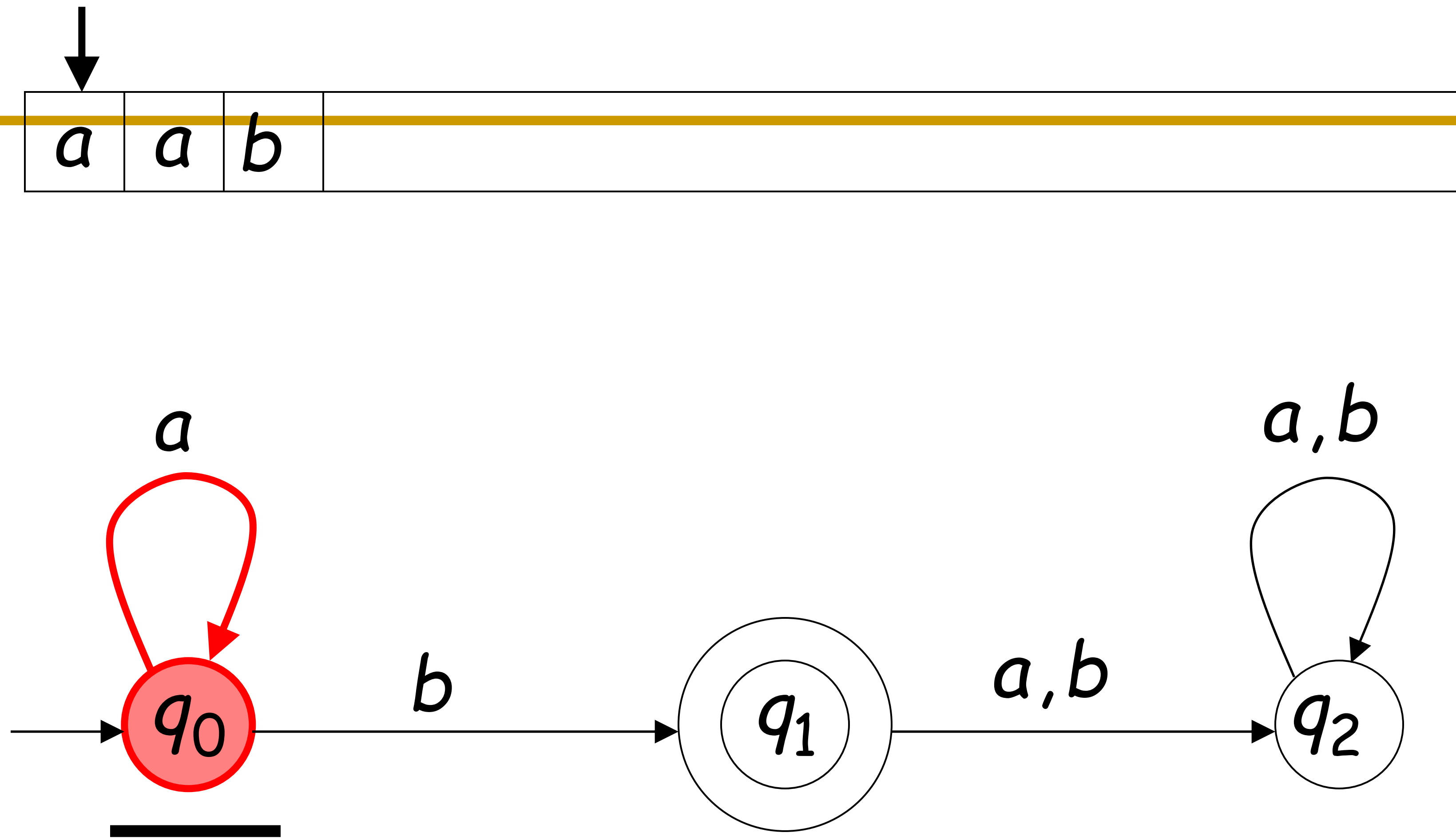


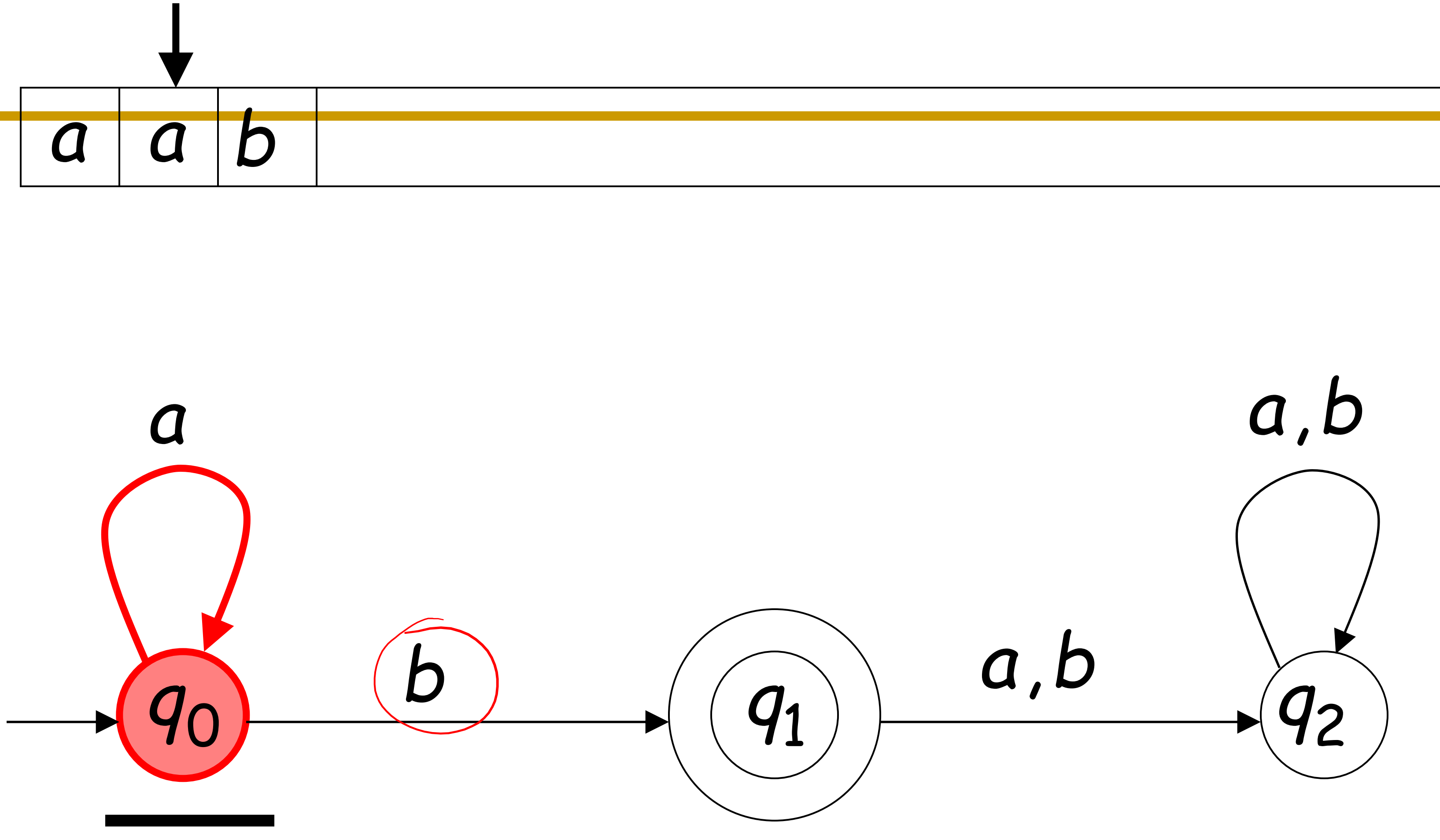
λ
a, b
b
a
a
b
a, b
a
b
b
a
q0
q1
q2
q3
q4
q5
reject

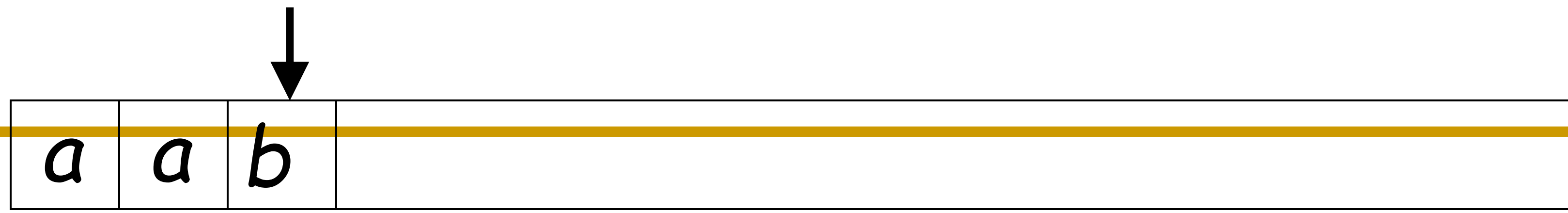
Part 2

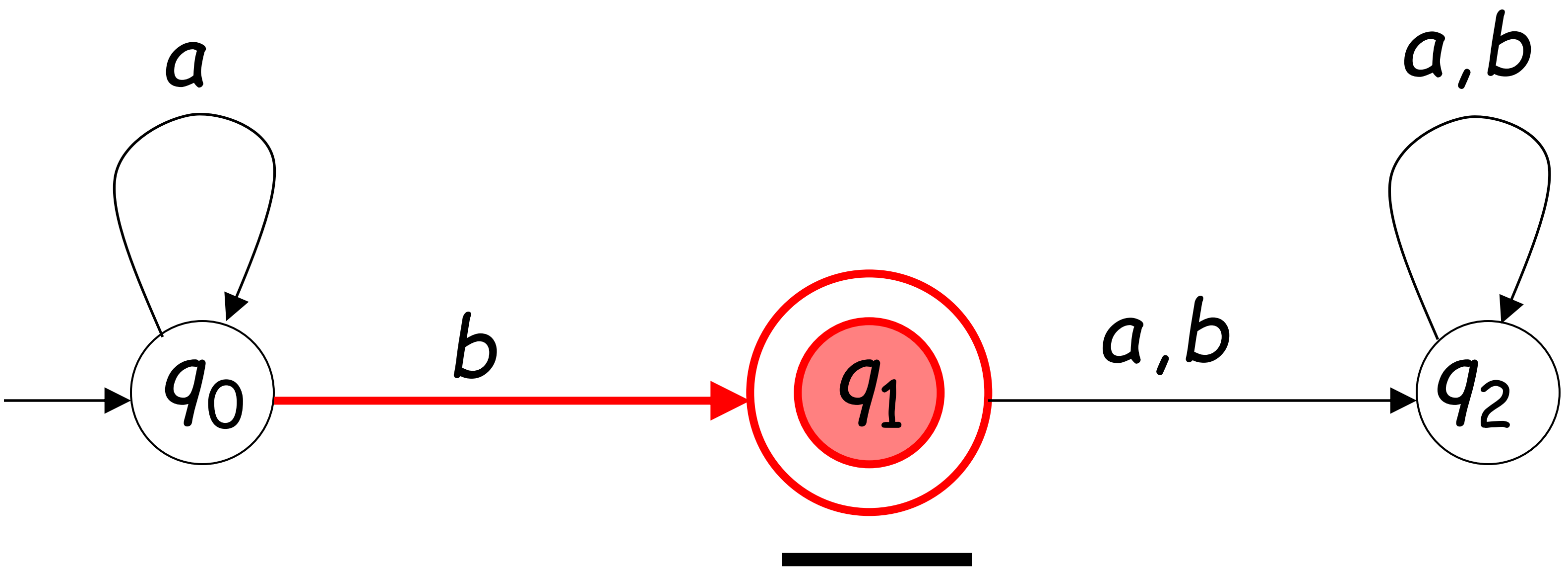
Another Example



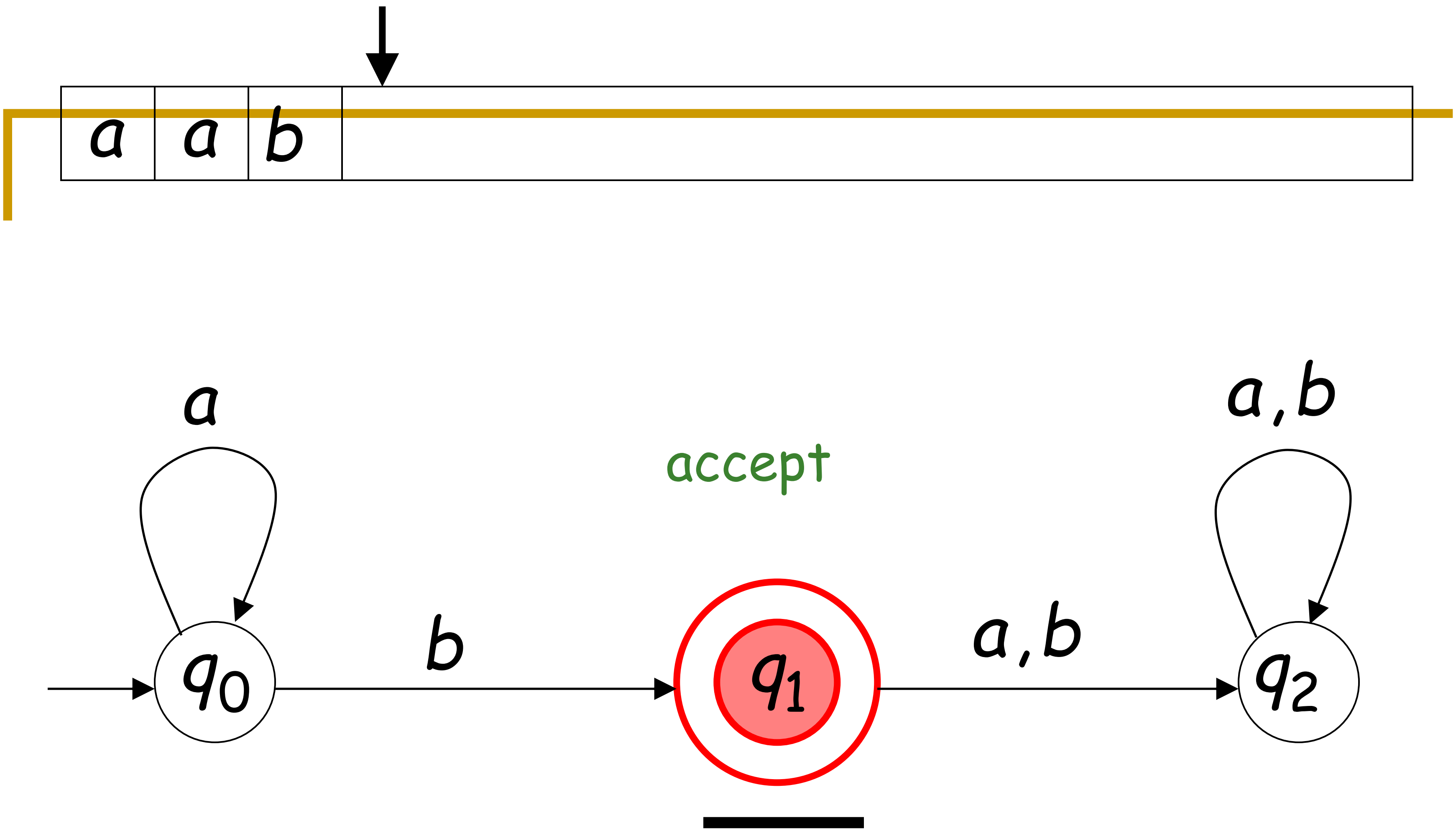

a
a
b
a
b
a,b
a,b
q_0
q_1
q_2


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

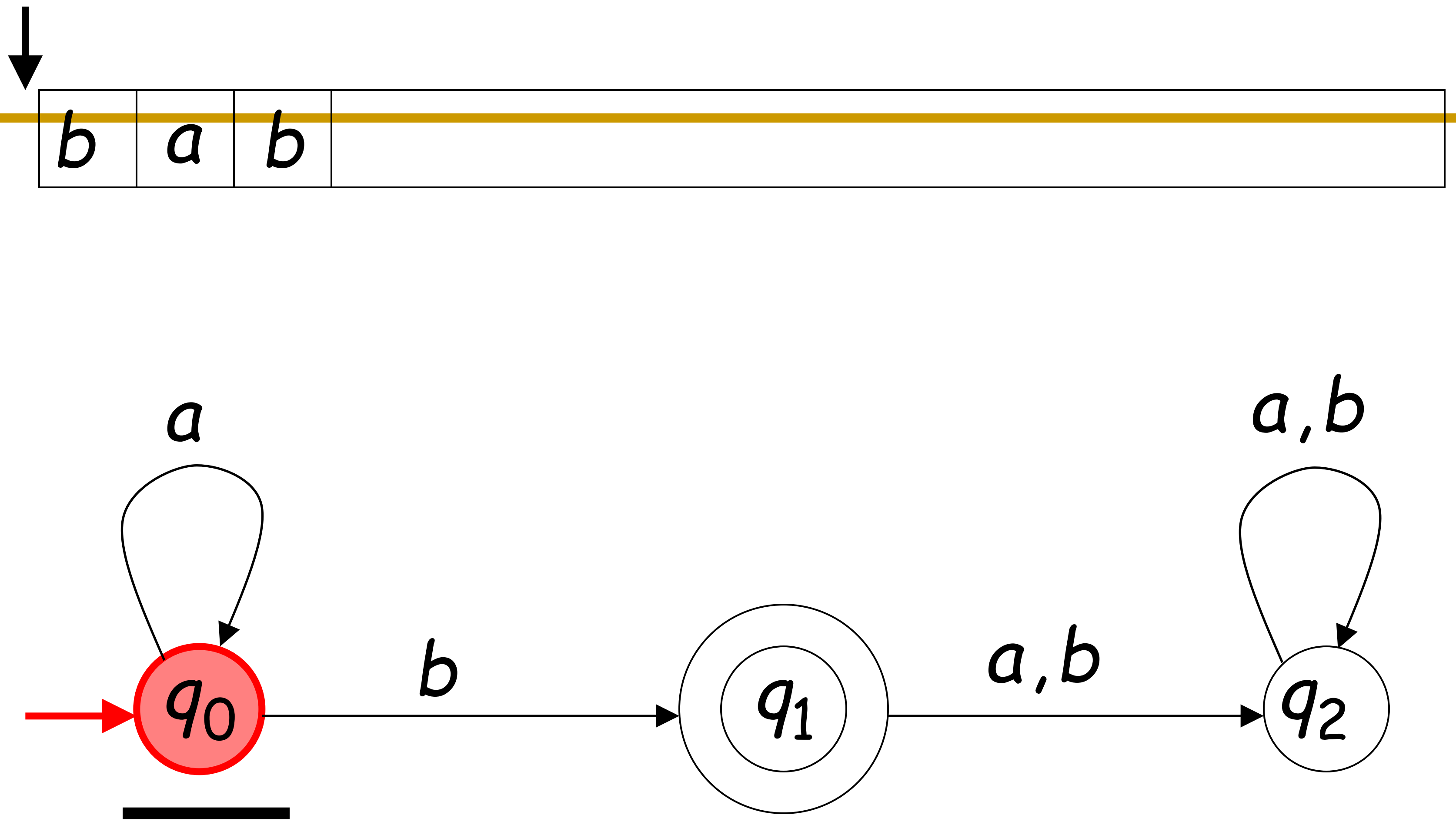

a
a
b

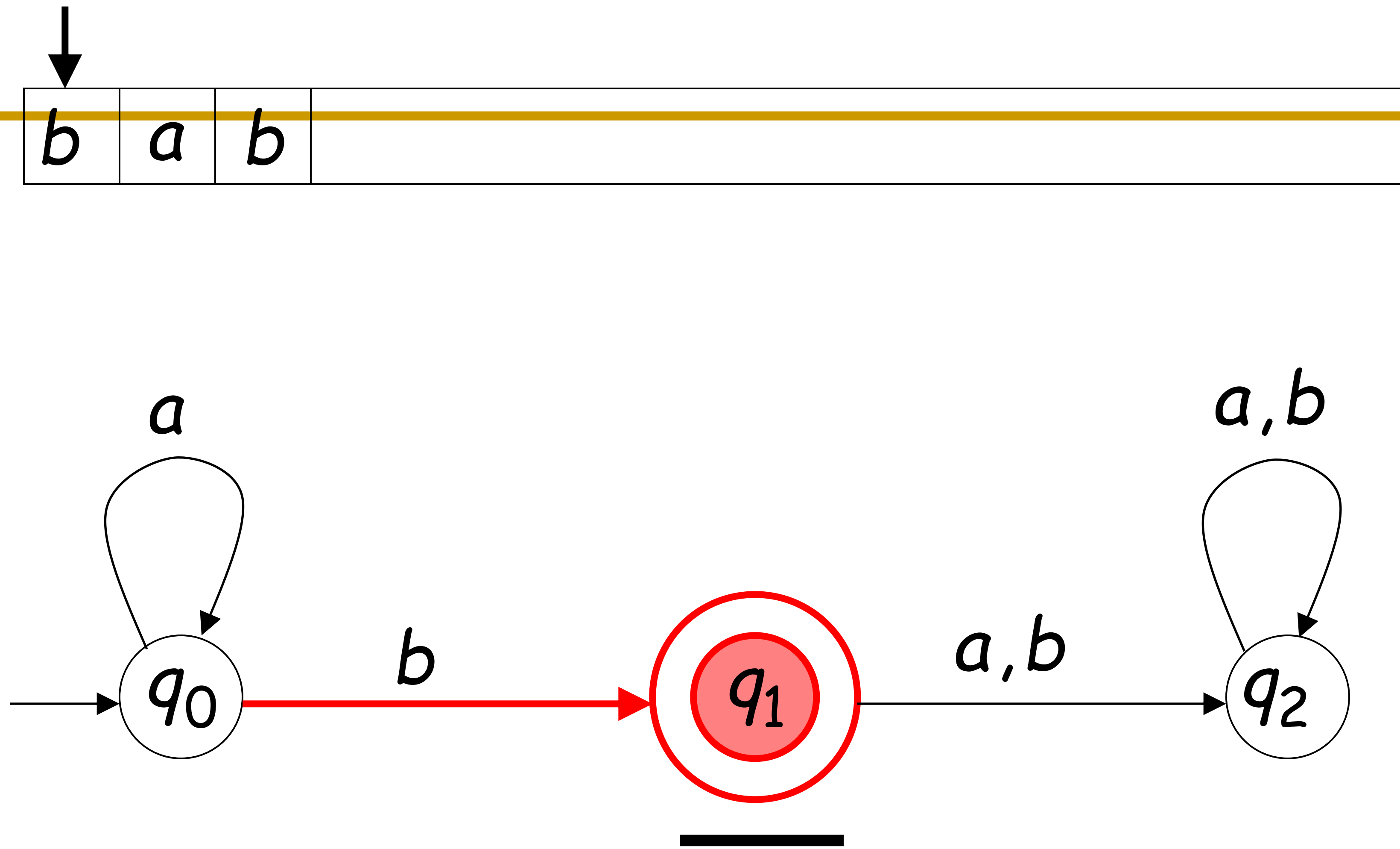

a
a,b
b
a,b
q_0
q_1
q_2

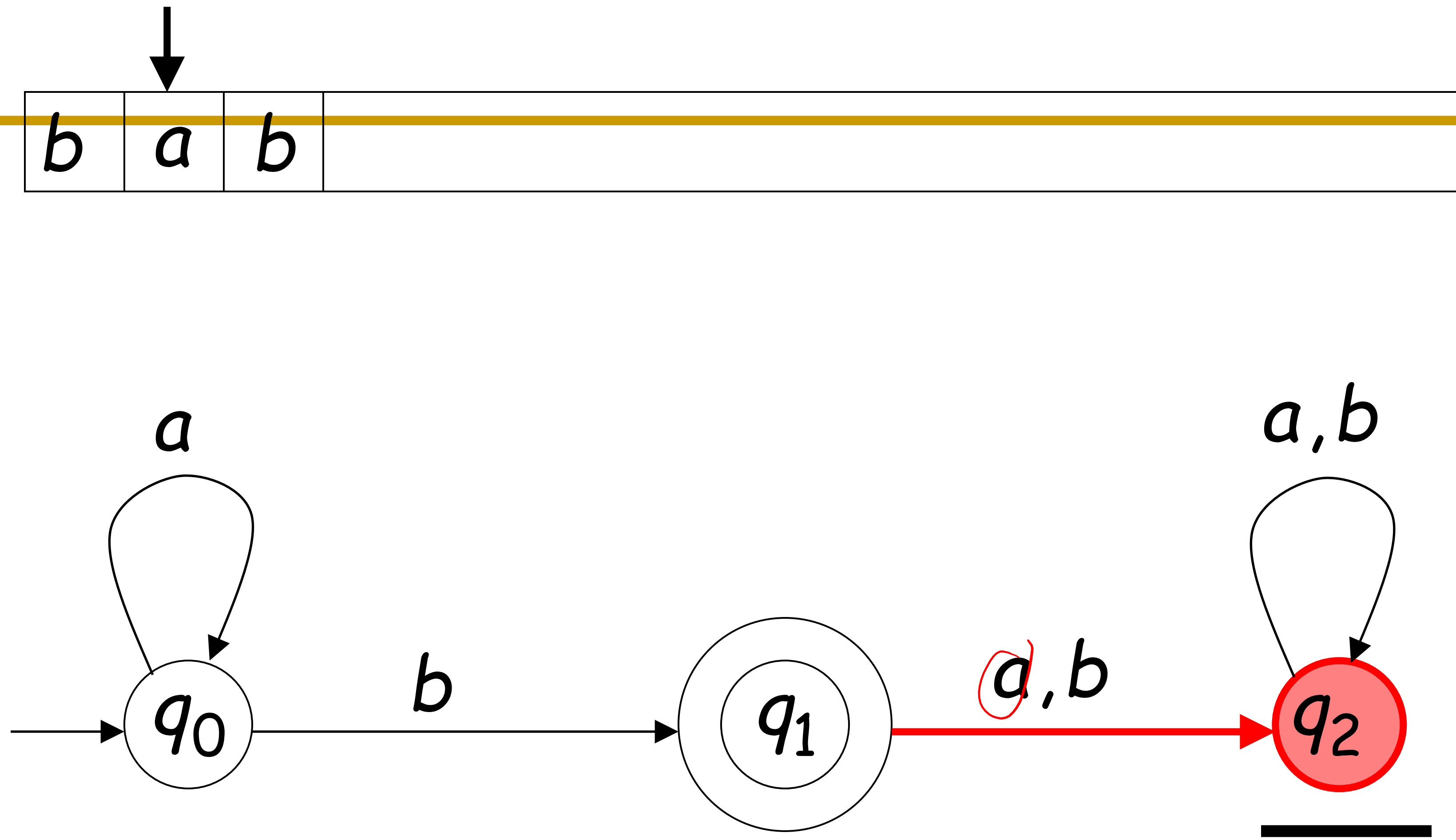
Input finished

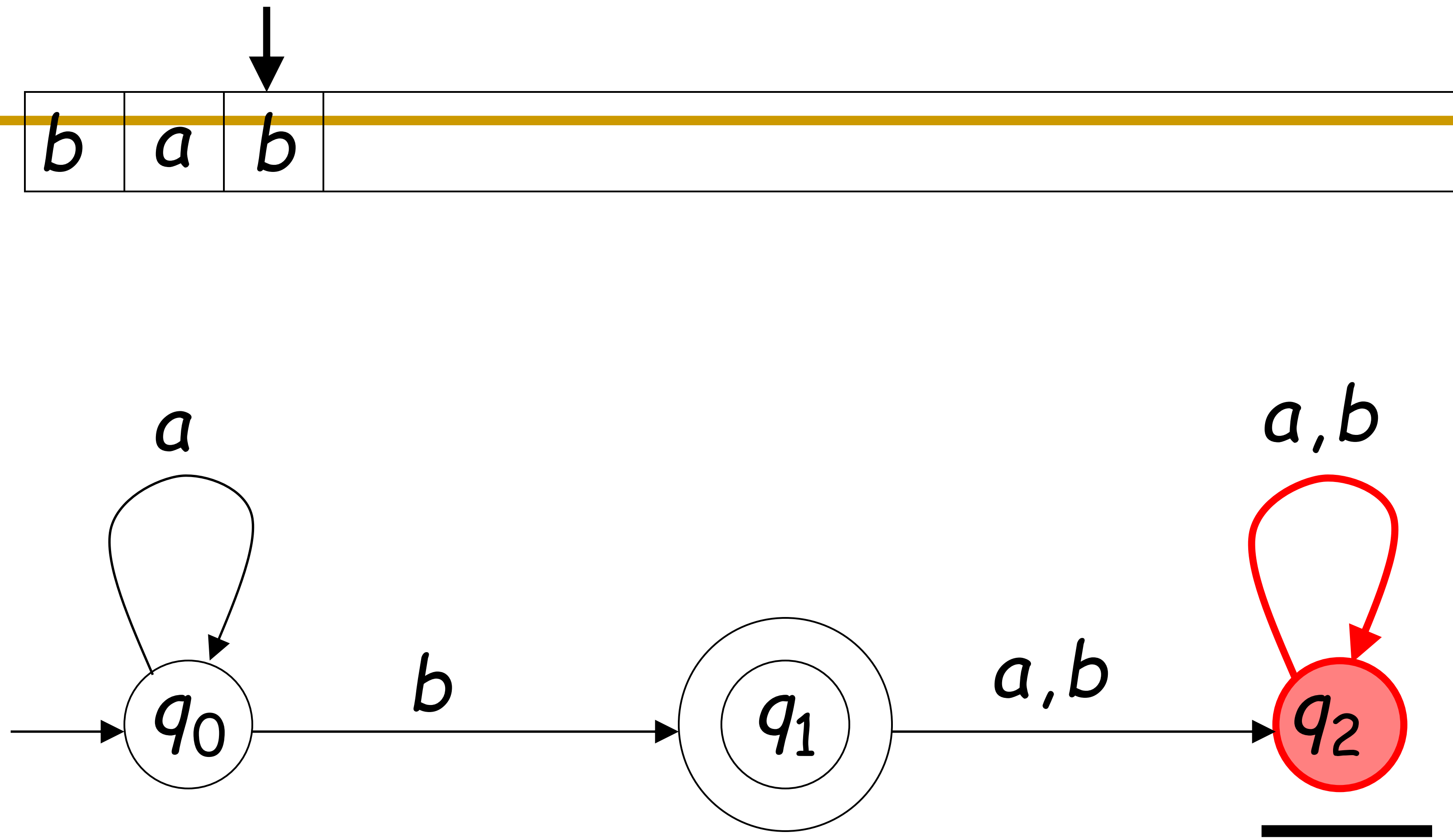


Rejection Example

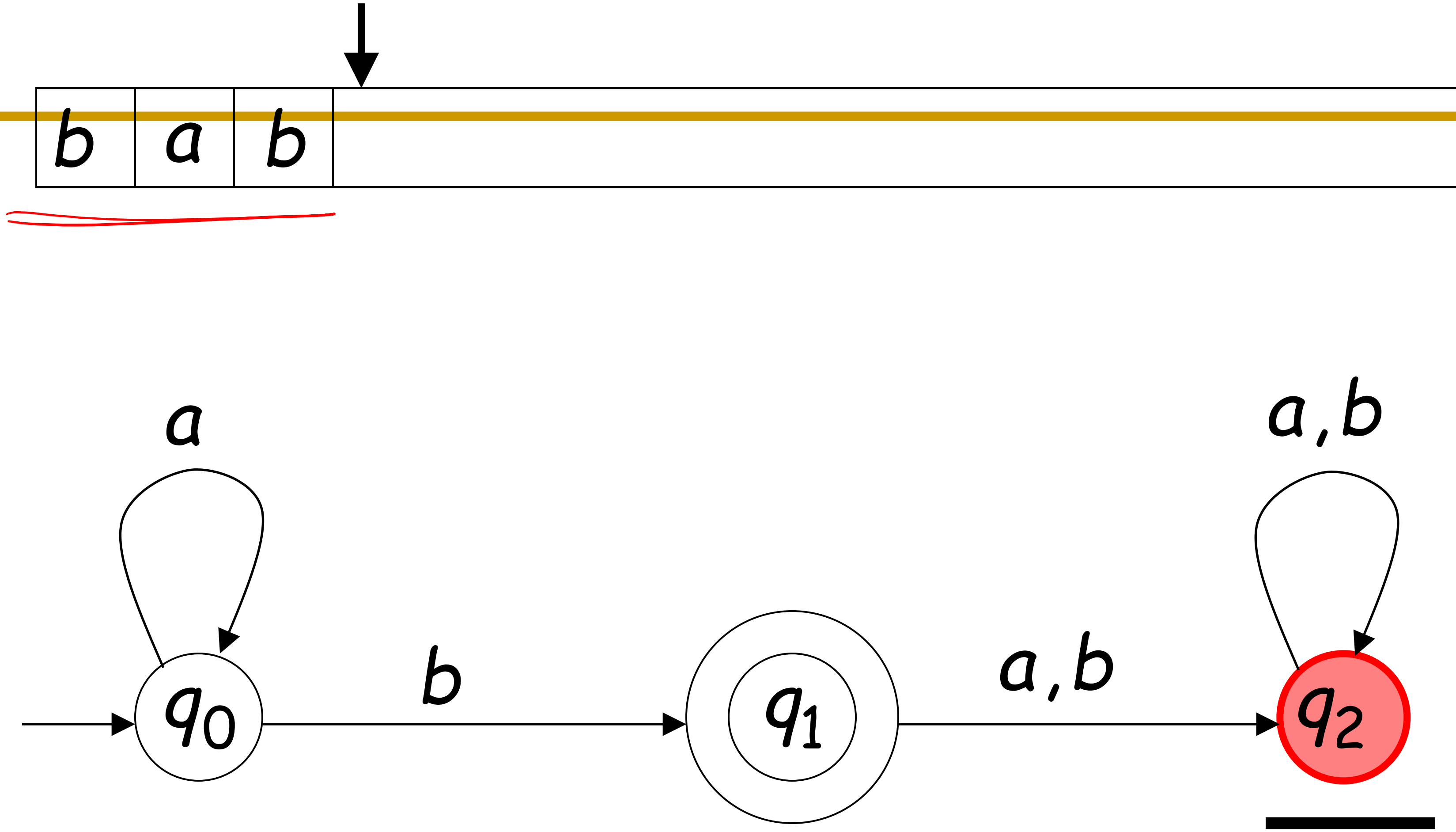



b
a
b
a
a,b
b
q_0
q_1
a,b
q_2


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

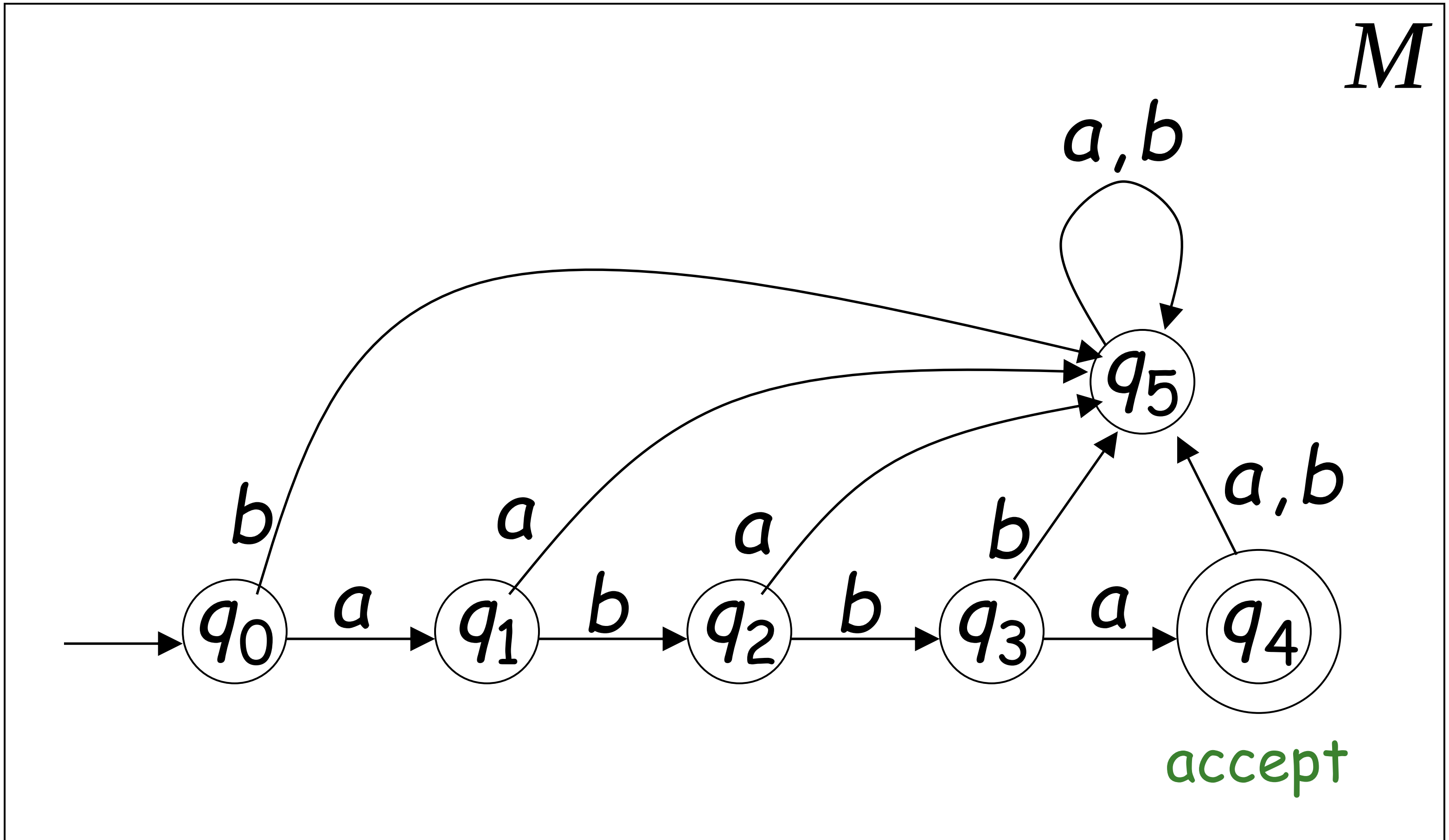

b
a
b
a
b
a,b
a,b
q_0
q_1
q_2

Input finished



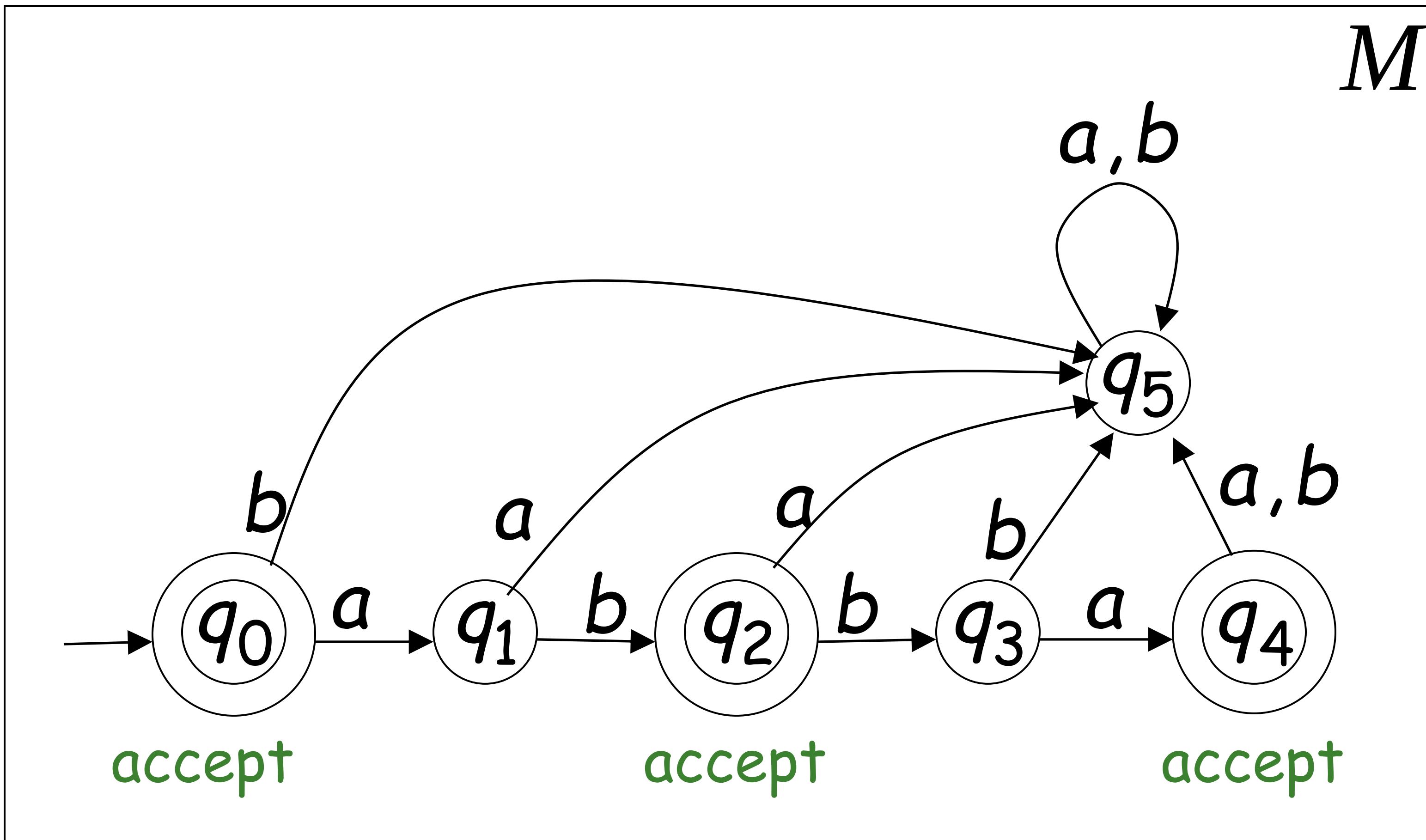
More Examples …

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



More Examples …

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



Finite Automata

- A **finite automaton** is a recognizer that takes a string, and answers “yes” if the string matches a pattern of a specified language, and “no” otherwise.

- Two kinds:
  - Nondeterministic finite automaton (NFA)
    - no restriction on the labels of their edges
  - Deterministic finite automaton (DFA)
    - exactly one edge with a distinguished symbol goes out of each state

- Both NFA and DFA have the same capability

- We may use NFA or DFA as lexical analyzer of a compiler

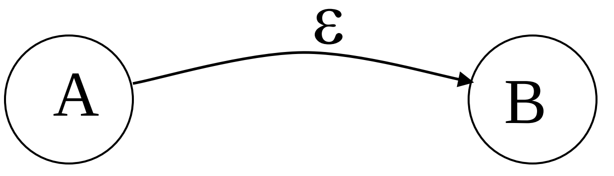
Finally

We have

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

Epsilon Moves in NFA!

- Another kind of transition: ε-moves



- Machine can move from state A to state B without reading input

Deterministic and Nondeterministic Automata

- Deterministic Finite Automata (DFA)
  - One transition per input per state
  - No ε-moves
- Nondeterministic Finite Automata (NFA)
  - Can have multiple transitions for one input in a given state
  - Can have ε-moves

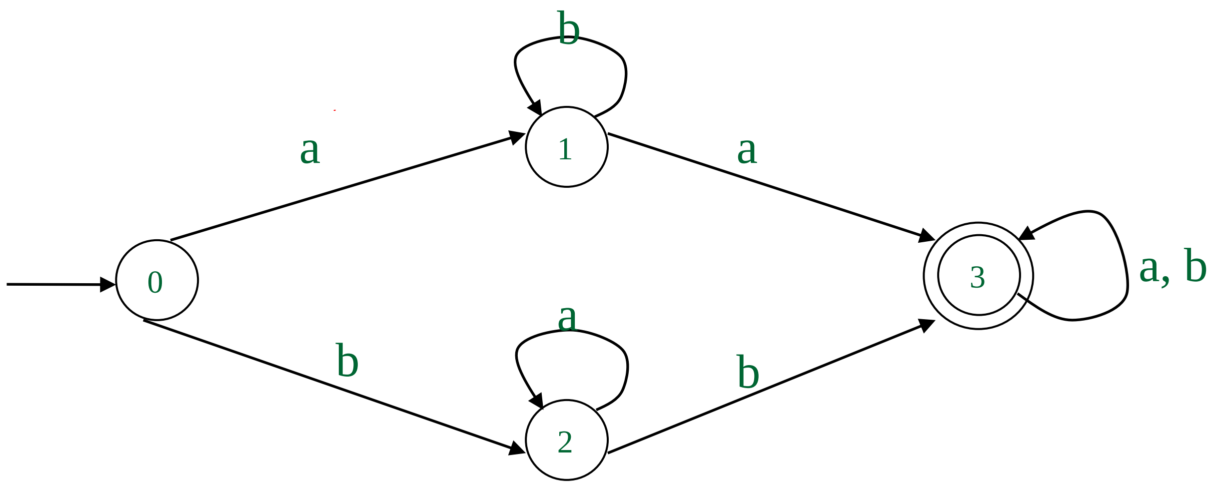
Difference in the formal definition

- **Definition:** A *finite-state automaton* is a 5-tuple $M=(S, I, f, S_0, F)$ where
  - S is a finite set of *states*
  - I is a finite *input alphabet*
  - $f:(S_i \mathrm{x} I) \rightarrow S_j$ is a *transition function* from a (***state* x *input)*** pair to another destination state
  - S_0 is the *initial state*
  - $F \subseteq S$ is a set of *final states*

- **DFA: exactly one transition for each distinct (state X symbol) pair**
- **NFA: no restriction:**
  - **A state can have no transition for an input symbol**
  - **A state can have one ore more transition(s) for an input symbol**
  - **A state can have ε-move transition**

Deterministic Finite Automata (DFA)

- Example:



The above automaton accepts the strings aba, baaabab. However it rejects any string of the form ab^n.

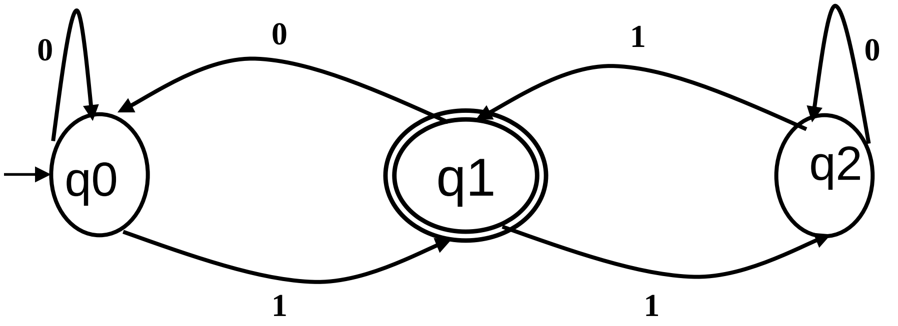
DFA

Example

- $M = (\{q0, q1, q2\}, \{0,1\}, \delta, q0, \{q1\})$,
- where δ is

$\delta(q0, 0) = q0$ $\delta(q0, 1) = q1$ $\delta(q1, 0) = q0$

$\delta(q1, 1) = q2$ $\delta(q2, 0) = q2$ $\delta(q2, 1) = q1$



The non-Deterministic Finite Automata (NFA)

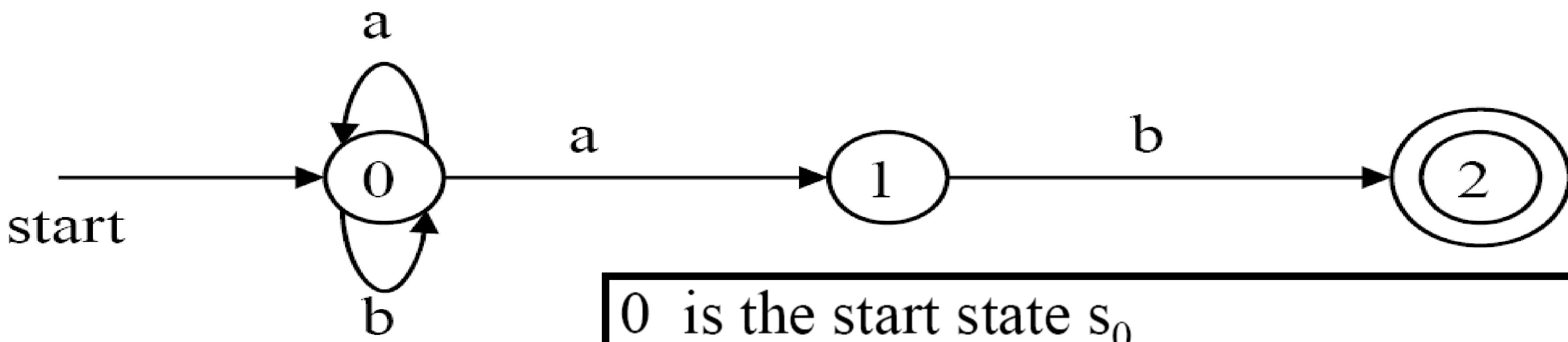
- **A nondeterministic finite automaton is defined as quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, F are defined as for dfa but $\delta : Q \times (\Sigma \cup \lambda) \rightarrow 2^Q$ and δ is a relation.**
- **There are three major differences between dfa and nfa:**
  - **The range of δ is in the power set 2^Q, so that its value defines a set of all possible states that can be reach by the transition**
  - **We allow λ as the second argument, this means that nfa can make a transition without consuming any input symbol**
  - **The value of δ may be empty, meaning that there is no transition defined for this specific situation.**

DFA VS. NFA

DFA	NFA
DFA stands for Deterministic Finite Automata.	NFA stands for Nondeterministic Finite Automata.
For each symbolic representation of the alphabet, there is only one state transition in DFA.	No need to specify how does the NFA react according to some symbol.
DFA cannot use Empty String transition.	NFA can use Empty String transition.
DFA can be understood as one machine.	NFA can be understood as multiple little machines computing at the same time.

In DFA, the next possible state is distinctly set.	**In NFA, each pair of state and input symbol can have many possible next states.**
DFA is more difficult to construct.	NFA is easier to construct.
DFA rejects the string in case it terminates in a state that is different from the accepting state.	NFA rejects the string in the event of all branches dying or refusing the string.
Time needed for executing an input string is less.	Time needed for executing an input string is more.
All DFA are NFA.	Not all NFA are DFA.
DFA requires more space.	NFA requires less space then DFA.
Dead state may be required.	Dead state is not required.
δ: Q$\times\Sigma$ -> Q i.e. next possible state belongs to Q.	δ: Q$\times\Sigma$ -> 2^Q i.e. next possible state belongs to power set of Q.

NFA Example (1)



0 is the start state s_0

{2} is the set of final states F

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

$S = \{0,1,2\}$

Transition Function:

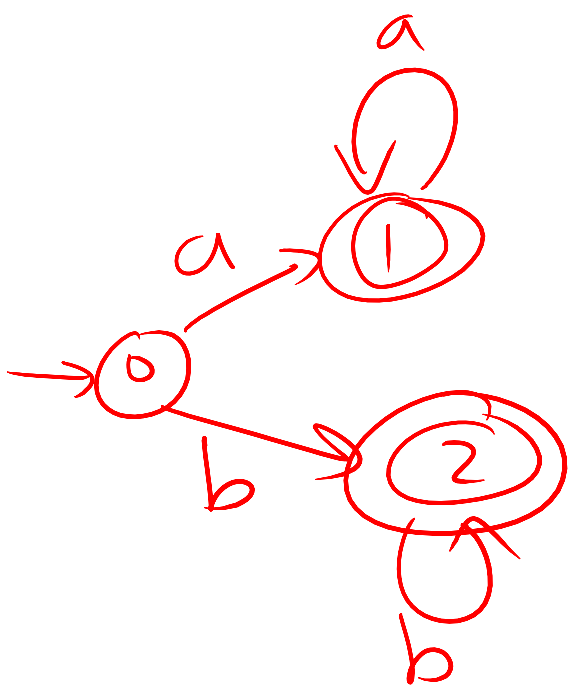
	a	b
0	{0,1}	{0}
1	–	{2}
2	_	_

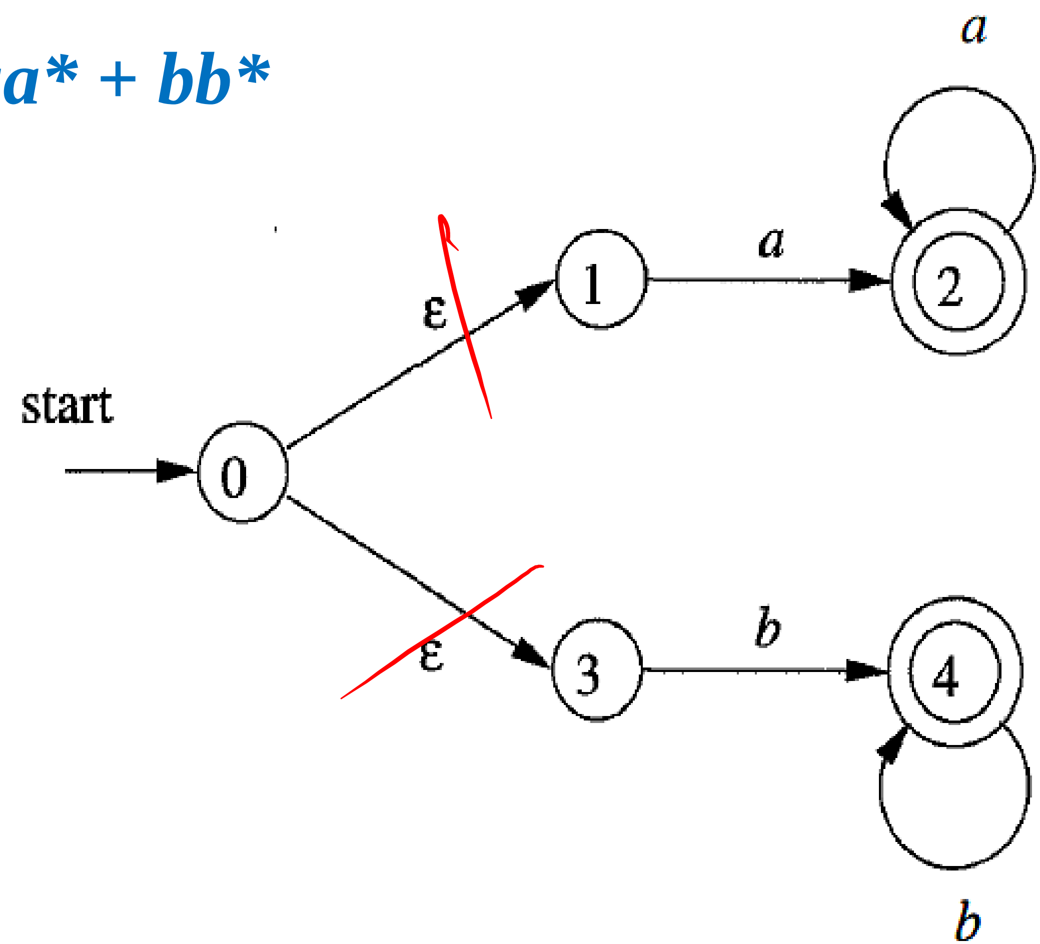
The language recognized by this NFA is ??

$(a+b)^*ab$

NFA Example (2)

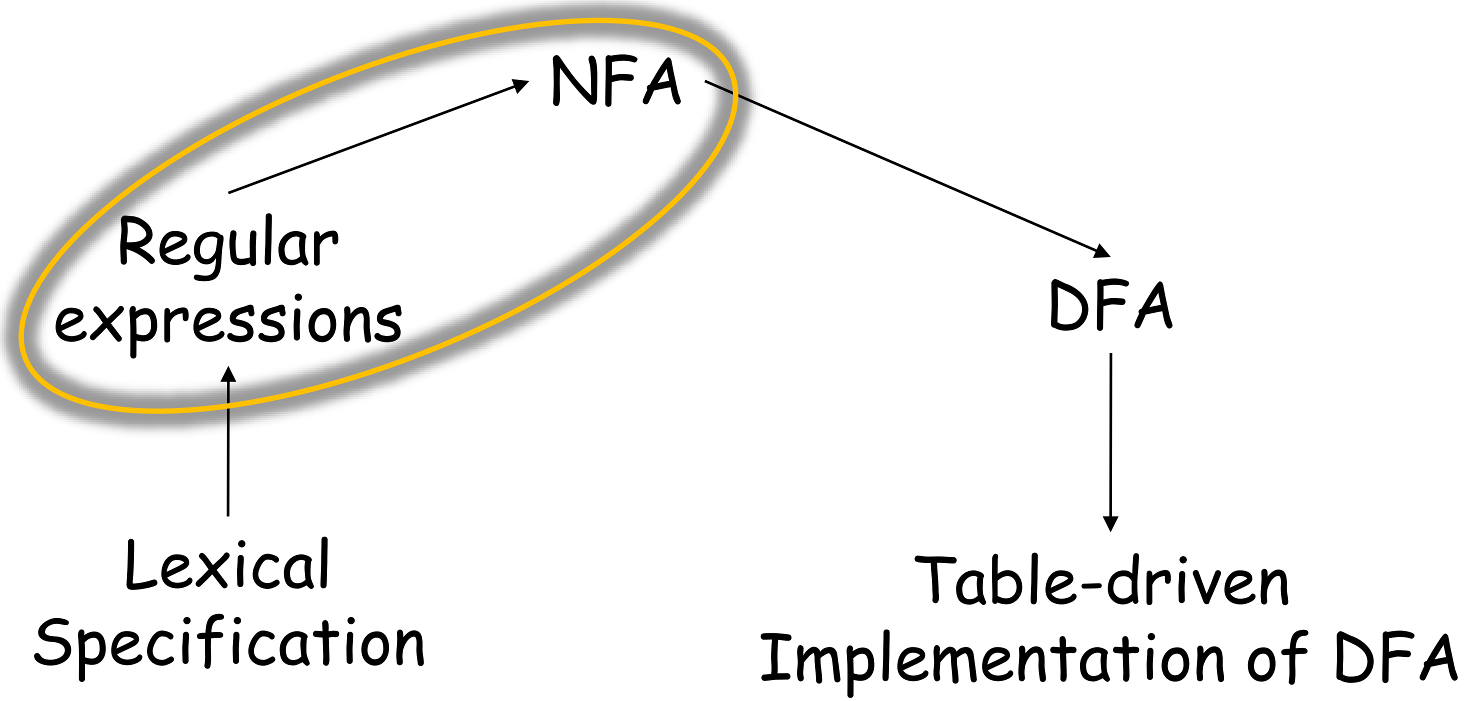
NFA accepting ***aa* + bb****





Regular Expressions to Finite Automata

- High-level sketch



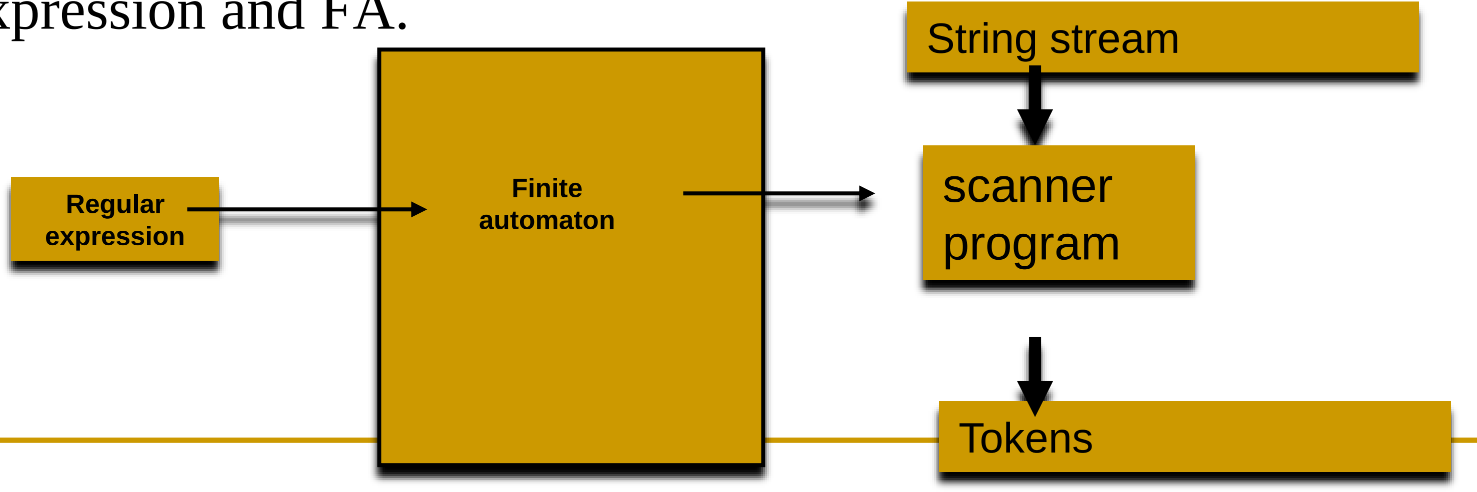
Regular Expression and FA

Every language that can be defined by a regular expression can also be defined by a finite automaton.

So we will see how to convert Regular expression to a FA, then how to convert FA to Regular expression.

RE and Finite State Automaton (FA)

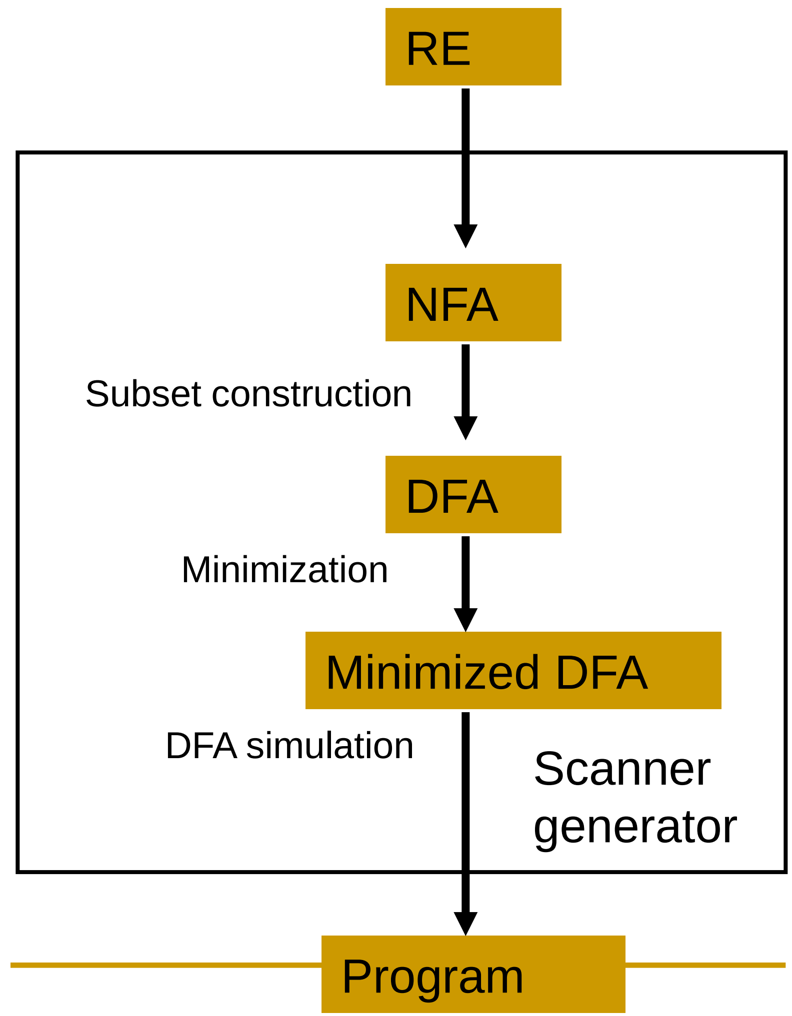
- Regular expression is a declarative way to describe the tokens
  - It describes *what* is a token, but not *how* to recognize the token.
- FA is used to describe *how* the token is recognized
  - FA is easy to be simulated by computer programs;
- There is a 1-1 correspondence between FA and regular expression
  - Scanner generator (such as lex) bridges the gap between regular expression and FA.



Inside scanner generator

Main components of scanner generation (e.g., Lex)

- Convert a regular expression to a non-deterministic finite automaton (NFA)
- Convert the NFA to a determinstic finite automaton (DFA)
- Improve the DFA to minimize the number of states
- Generate a program in C or some other language to "simulate" the DFA



NFA

Non-deterministic Finite Automata (FA)

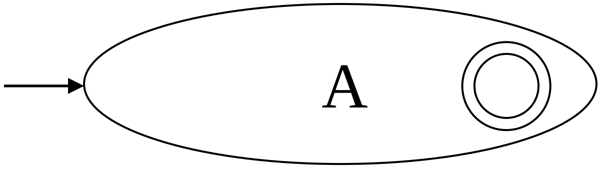
- NFA (Non-deterministic Finite Automaton) is a 5-tuple $(S, \Sigma, \delta, S0, F)$:
  - S: a set of states;
  - Σ: the symbols of the input alphabet;
  - δ : a set of transition functions;
    - move(state, symbol) → a set of states
  - S0: $s0 \in S$, the start state;
  - F: $F \subseteq S$, a set of final or accepting states.
- Non-deterministic -- a state and symbol pair can be mapped to a set of states.
- Finite—the number of states is finite.

Nondeterministic Finite Automaton (NFA)

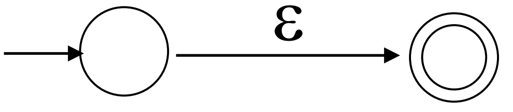
- NFA can be represented by a transition graph
- Accepts a string x, if and only if there is a path from the starting state to one of accepting states such that edge labels along this path spell out x.
- Remarks
  - The same symbol can label edges from one state to several different states
  - An edge may be labeled by ε, the empty string

Regular Expressions to NFA (1)

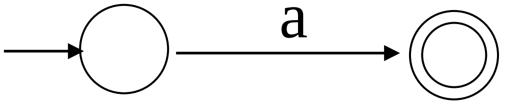
- For each kind of rexp, define an NFA
  - Notation: NFA for rexp A



- For ε

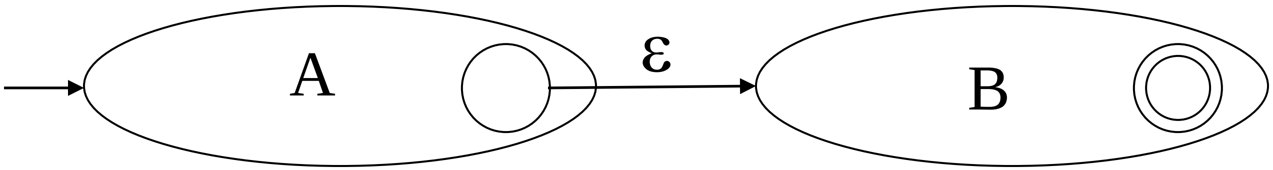


- For input a

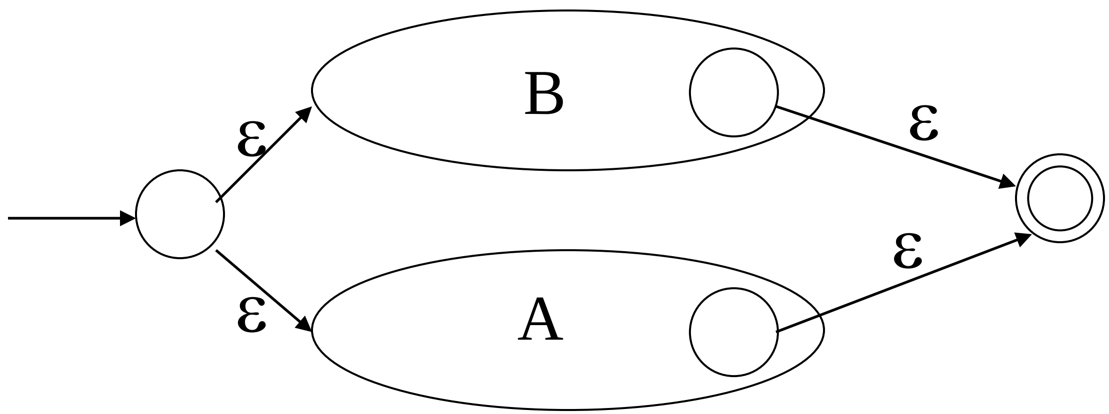


Regular Expressions to NFA (2)

- For AB



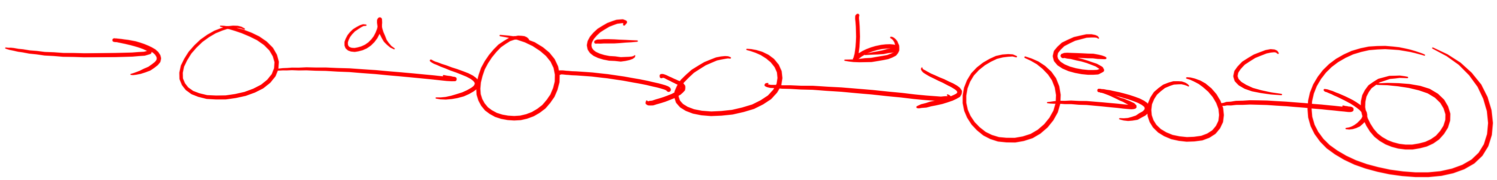
- For A | B



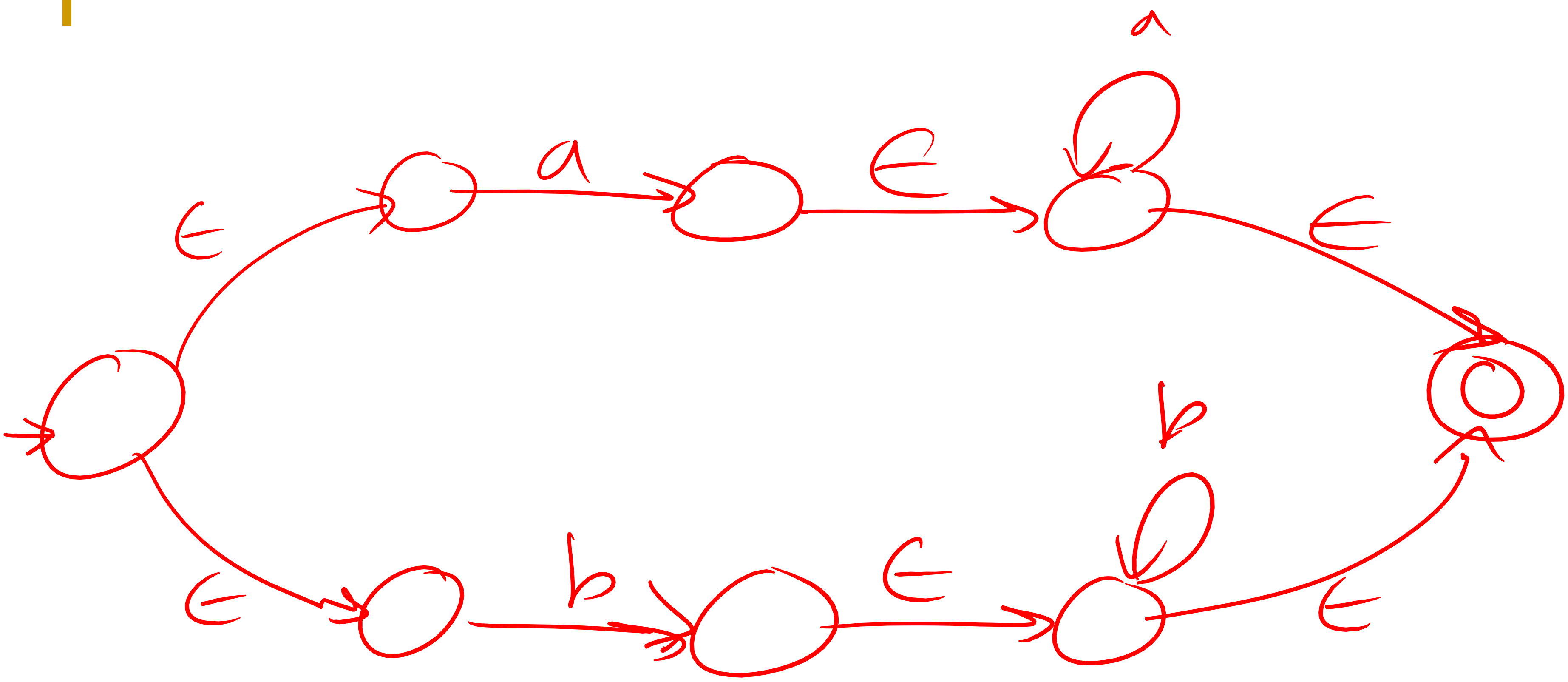
Regular Expressions to NFA (2)

- RE = abc

abc

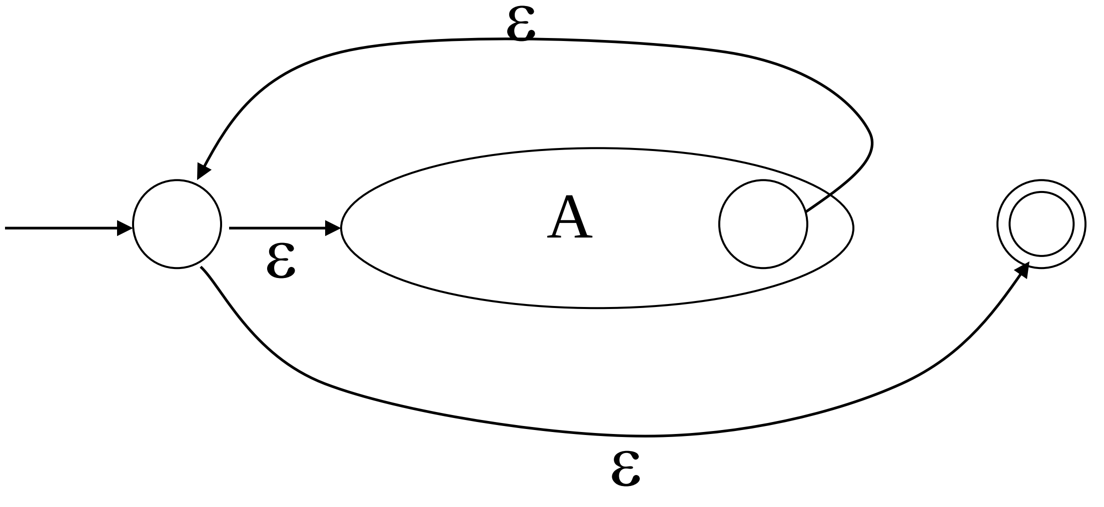


$aa^* + bb^*$



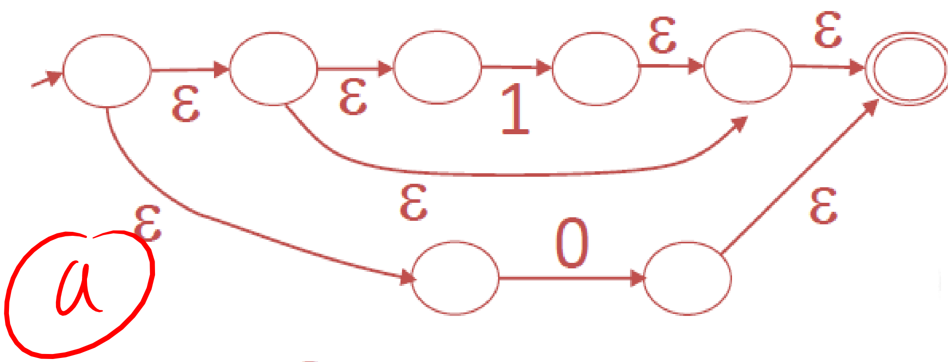
Regular Expressions to NFA (3)

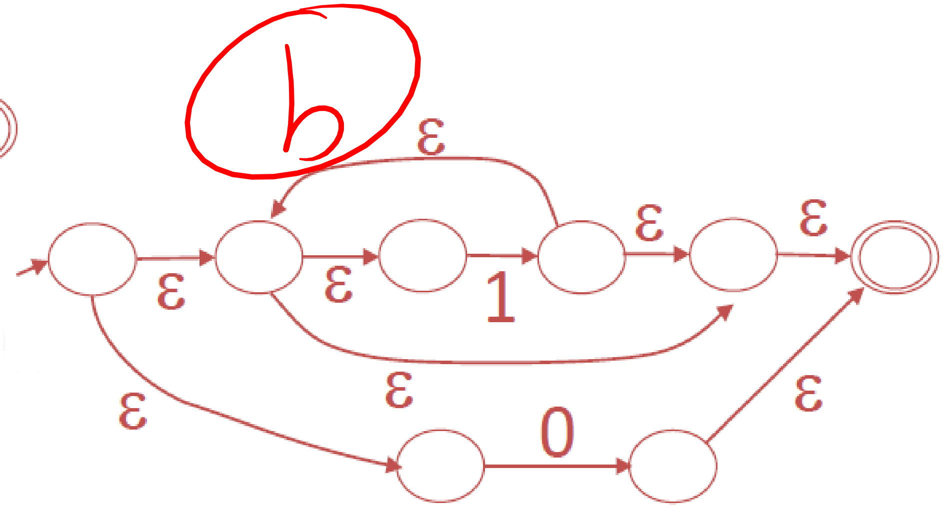
- For A*

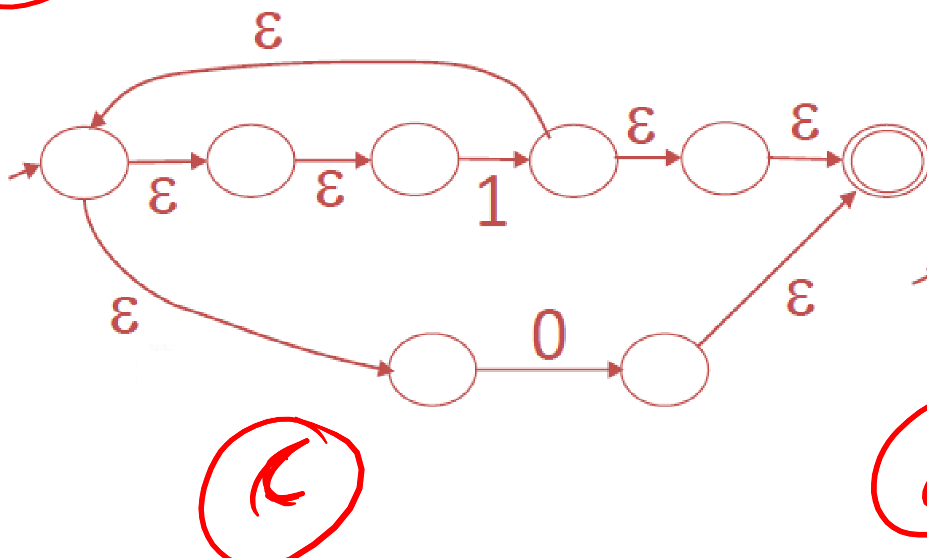


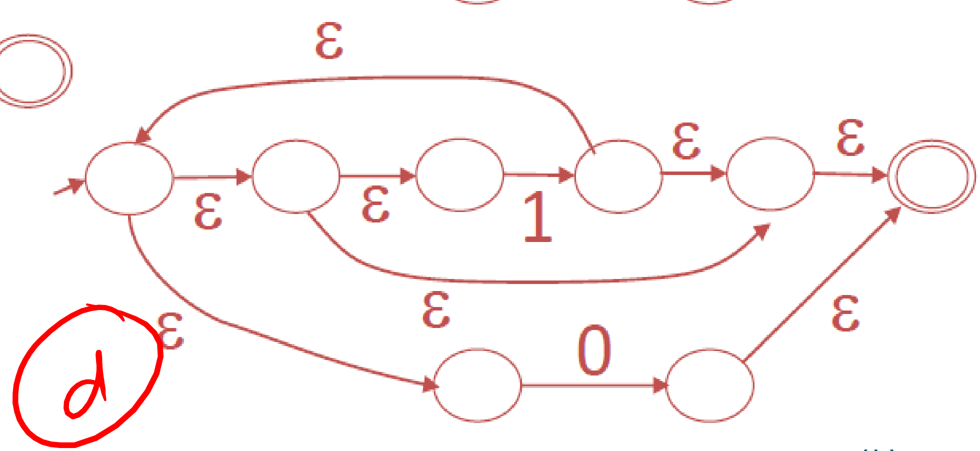
EXAMPLE OF (RE → NFA) CONVERSION

- Choose the NFA that accepts the following regular expression: 1* + 0



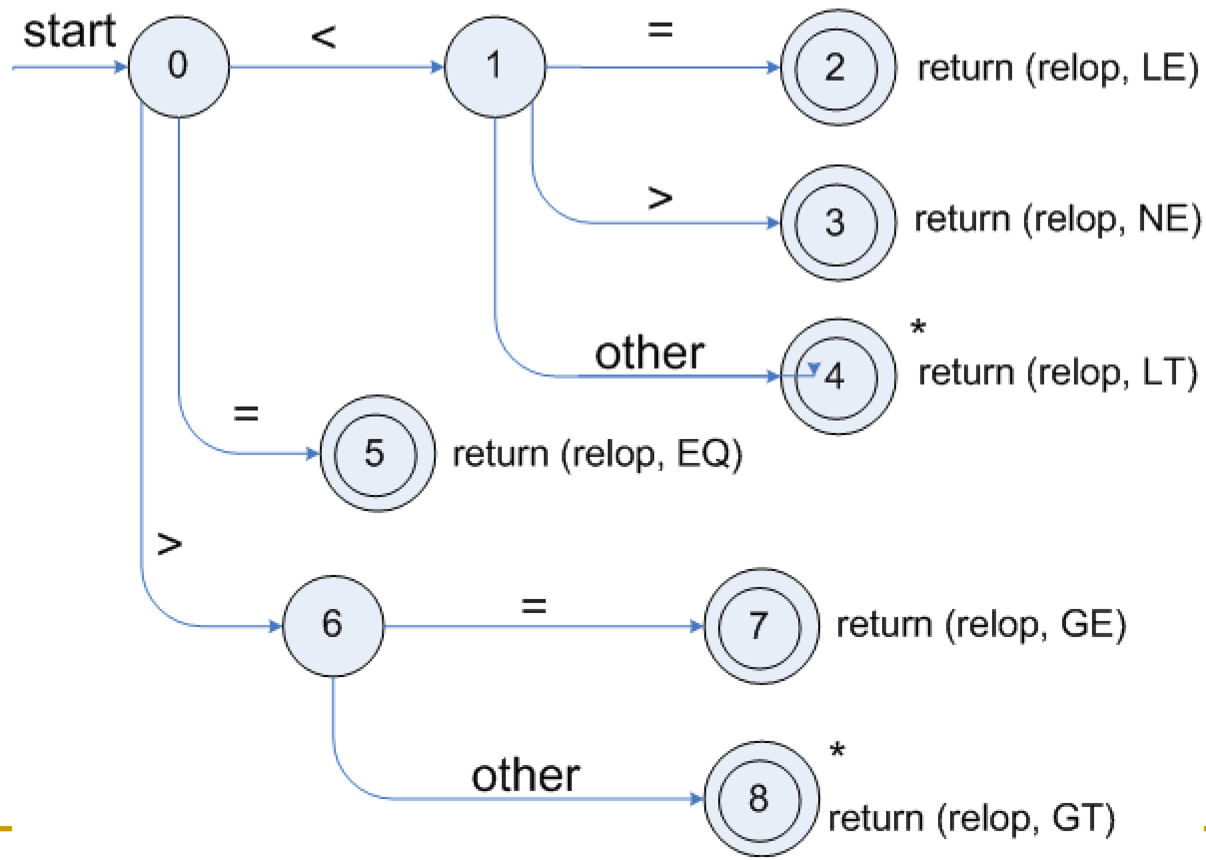






Transition diagrams

Transition Diagram for "***relop*** → < | > |<= | >= | = | <>"



Transition diagrams

A transition diagram for valid identifiers