Languages and Deterministic Finite Automata

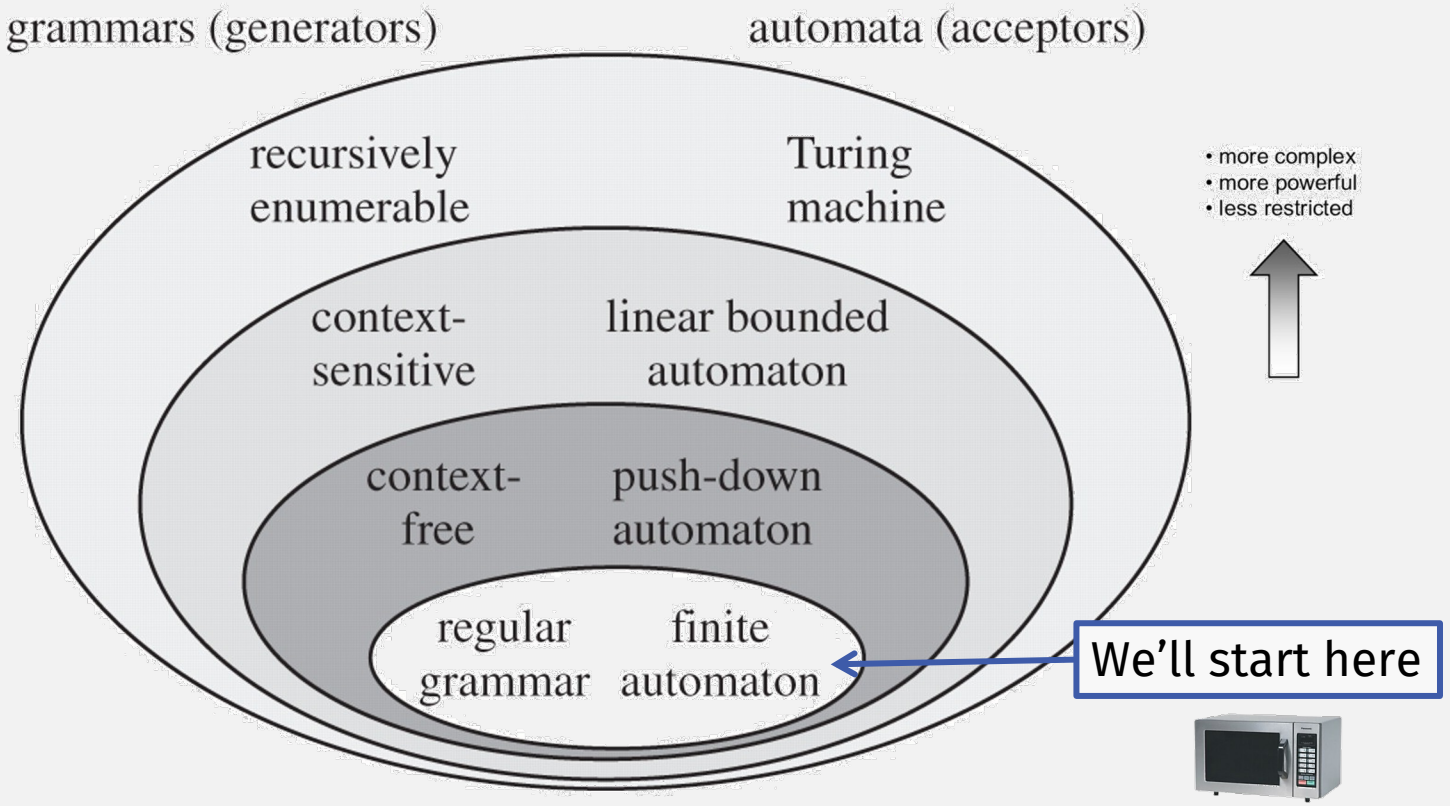
HW 0 Questions?

Last time: The Theory of Computation …

- Creates and studies mathematical models of computers

- In order to:
  - Make predictions about computer programs
  - Explore the limits of computation

Last time: Levels of Computational *Power*



Languages

Alphabets and Strings

An alphabet is a set of symbols

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

A Σ-string is a sequence of symbols from Σ
(omit the Σ when irrelevant or obv. in context)

Example Strings

a

ab

abba

aaabbbaabab

Decimal numbers alphabet $\Sigma = \{0,1,2,\ldots,9\}$

102345 567463386

String Operations

$w = a_1 a_2 \cdots a_n$ $\qquad$ $abba$

$v = b_1 b_2 \cdots b_m$ $\qquad$ $bbbaaa$

Concatenation

$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m$ $\qquad$ $abbabbbaaa$

String Length

$$w = a_1 a_2 \cdots a_n$$

- Length: $|w| = n$
- Examples: $|abba| = 4$, $|aa| = 2$, $|a| = 1$

Length of Concatenation

$$|uv| = |u| + |v|$$

- Example:

$$u = aab, \quad |u| = 3$$

$$v = abaab, \quad |v| = 5$$

$$|uv| = |aababaaab| = 8$$

$$|uv| = |u| + |v| = 3 + 5 = 8$$

Empty String

- A string with no letters is denoted: ε

- Observations: $|\varepsilon| = 0$

  $\varepsilon w = w\varepsilon = w$

  $\varepsilon abc = abc\varepsilon = ab\varepsilon c = a\varepsilon bc$

Substring

- Substring of string:
  - a subsequence of consecutive characters

- String $abbab$ Substring ab

  $abbab$ $abba$

  $abbab$ b

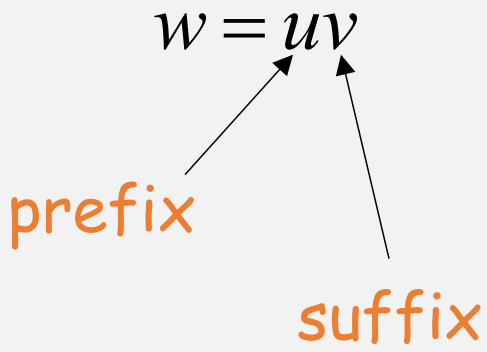
  $abbab$ $bbab$

Prefix and Suffix

$abbab$

- Prefixes Suffixes

Prefixes	Suffixes
ε	abbab
a	bbab
ab	bab
abb	ab
abba	b
abbab	ε



The *

- String: a sequence of symbols
- from some alphabet

- Σ*: the set of all sequences of symbols (i.e. strings) from the alphabet Σ

- Language: a set of strings
- Example strings: cat, dog, house
- A language: {cat, dog, house}

Languages

- A language over alphabet Σ
- is any subset of Σ^*
- Examples:

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

$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, \ldots\}$

Language: $\{\varepsilon\}$

Language: $\{babba, ab, baaaaaaa\}$

Language: $\{ab, \varepsilon, bbab, abab\}$

Languages are used to describe computation problems:

$$PRIMES = \{2,3,5,7,11,13,17,\ldots\}$$

$$EVEN = \{0,2,4,6,\ldots\}$$

Alphabet: $\Sigma = \{0,1,2,\ldots,9\}$

How we use languages to describe computation problems

- Think of the {(x1,y1),(x2,y2) ...} definition of a function
- Then every algorithm computes a function
- program ~ impl. of alg
- automaton ~ hardware impl

- (x,y) is a member of a set when f(x) = y. (Bools are even easier!)

- Question: every program computes some function:
  - So, is there, for every function, some program that computes it?

- Stay tuned to find out!

Squares

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

Language:

$$SQUARES = \{x \# y \ : x = 1^n, y = 1^m, m = n^2\}$$

$$11\#1111 \in SQUARES$$

$$111\#1111 \notin SQUARES$$

∅ is a set

{} is a set

ε is a string

Is a set a string?

Is a string a set?

What is {ε}?

- is it the same as ∅?
- is it the same as {}?
- is it the same as ε?

Pipes are an *overloaded operator*

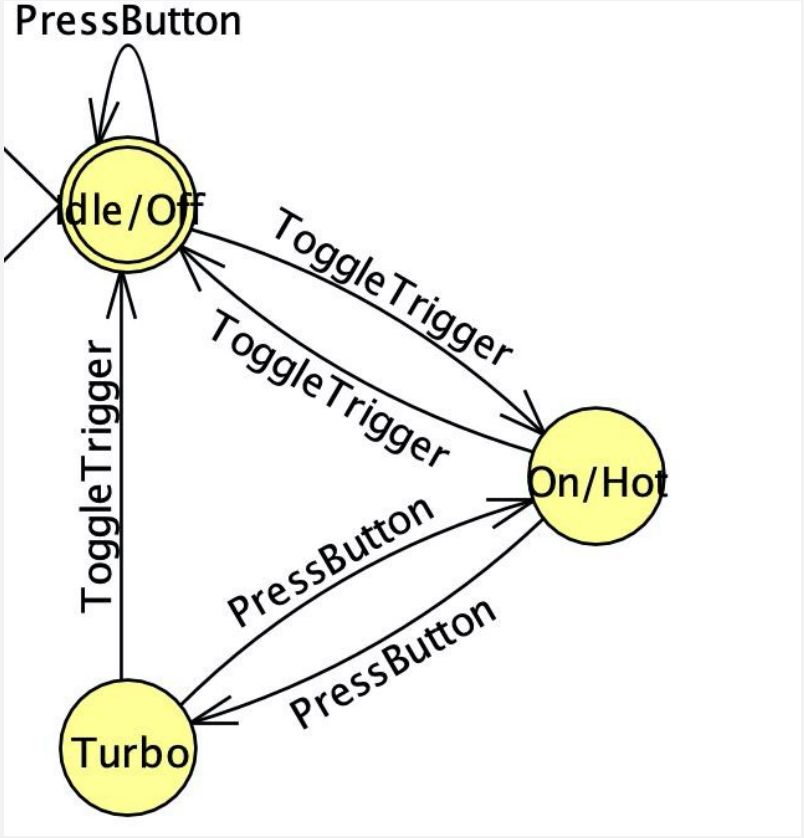
| yabba | = 5 (string length)

| { 1, 3, 5, 7 } | = 4 (set size)

Deterministic Finite State Automata

Finite Automata: A computational model for …

A Blow-dryer Finite Automaton



Finite Automata: Not Just for Microwaves

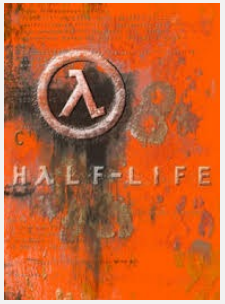
Finite Automata: a common programming pattern

State pattern

From Wikipedia, the free encyclopedia

The **state pattern** is a behavioral software design pattern that allows an object to alter its behavior when its internal state changes. This pattern is close to the concept of finite-state machines. The state pattern can be interpreted as a strategy pattern, which is able to switch a strategy through invocations of methods defined in the pattern's interface.

Finite Automata in Video Games



ValveSoftware / **halflife**

Code | Issues 1.6k | Pull requests 23 | Actions | Projects | Wiki

5d761709a3 — halflife / game_shared / bot / **simple_state_machine.h**

Alfred Reynolds initial seed of Half-Life 1 SDK

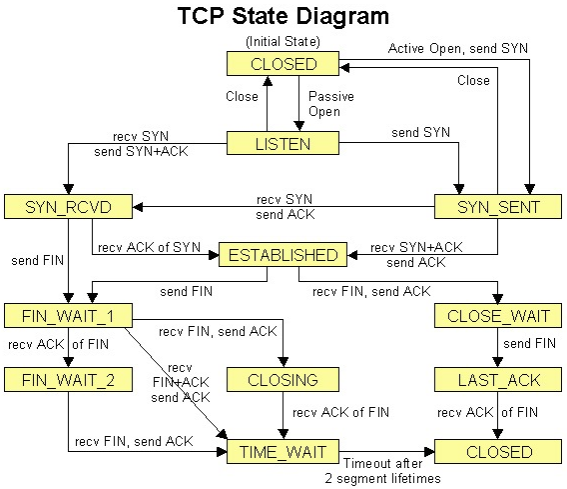
0 contributors

85 lines (67 sloc) | 2.15 KB

```
1  // simple_state_machine.h
2  // Simple finite state machine encapsulation
3  // Author: Michael S. Booth (mike@turtlerockstudios.com), November 2003
4
5  #ifndef _SIMPLE_STATE_MACHINE_H_
6  #define _SIMPLE_STATE_MACHINE_H_
7
8  //--------------------------------------------------------------------------------------------------------
9  /**
10  * Encapsulation of a finite-state-machine state
11  */
12 template < typename T >
13 class SimpleState
```

Stateful Protocols, State Machines

- TCP
- BGP



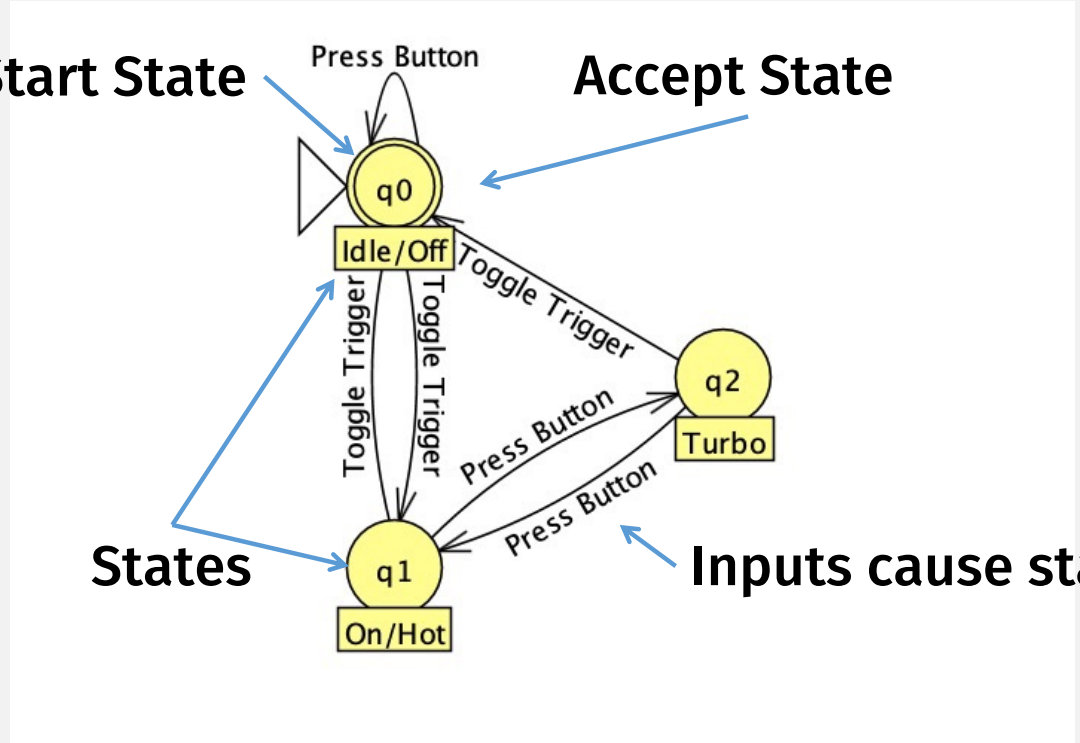
- Stateless Protocols
  - HTTP (but can be _simulated_ with a FSM)

A Finite Automata is a Computer!

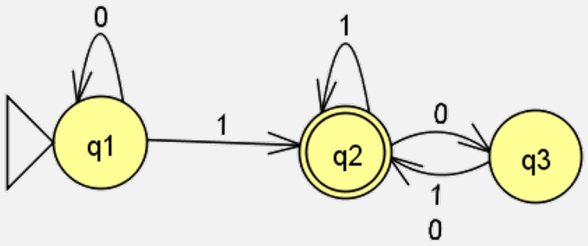
- A very limited computer with finite memory
  - Memory = states

- In this class, we'll formally study automata as:
  - State diagrams
  - Formal mathematical model
  - Code simulations of the mathematical model

A Blow-dryer Finite Automata



Finite Automata state diagram



Finite Automata: The Formal Definition

5 components

DEFINITION 1.5

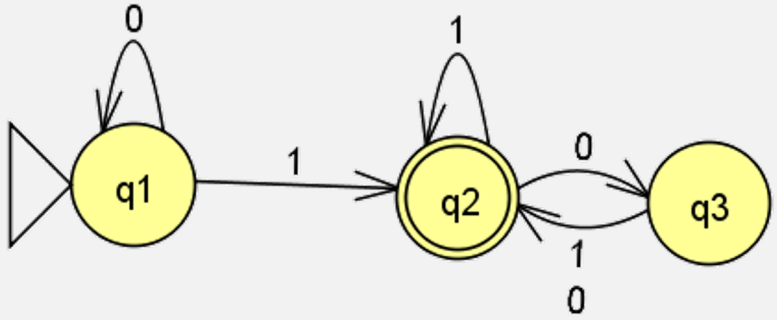
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

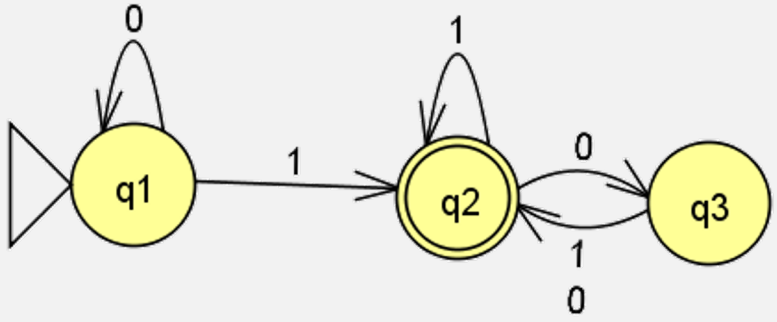


Example: as state diagram

DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



Example: as state diagram

Example: as formal description

$M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

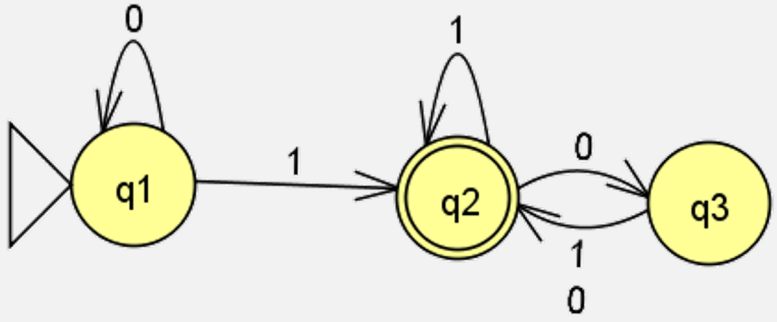
4. q_1 is the start state, and
5. $F = \{q_2\}$.

Braces = Set notation (no duplicates)

DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



Example: as state diagram

Example: as formal description

$M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$, ← **Possible inputs**
3. δ is described as

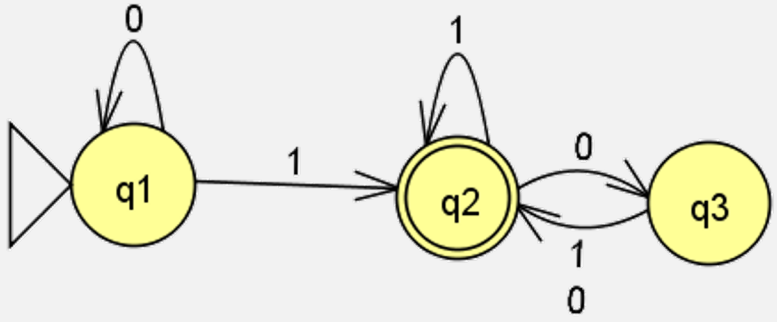
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



Example: as state diagram

Example: as formal description

$M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

"And this is next input symbol"

"If in this state"

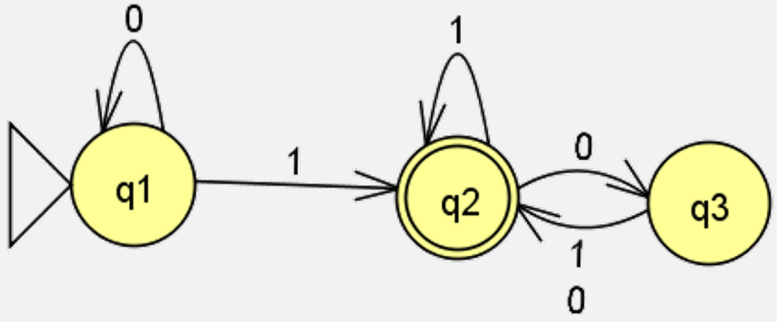
"Then go to this state"

4. q_1 is the start state, and
5. $F = \{q_2\}$.

DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



Example: as state diagram

Example: as formal description

$M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

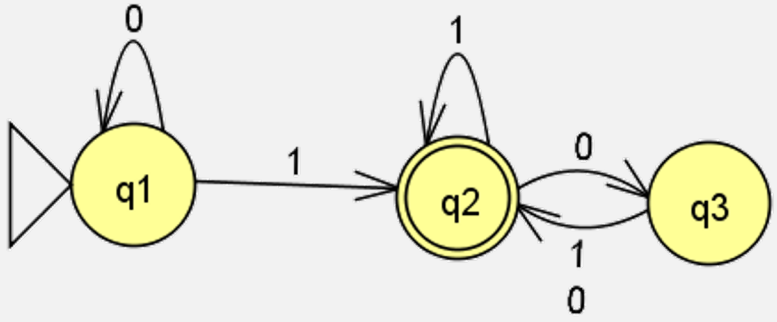
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



Example: as state diagram

Example: as formal description

$M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and
5. $F = \{q_2\}$.

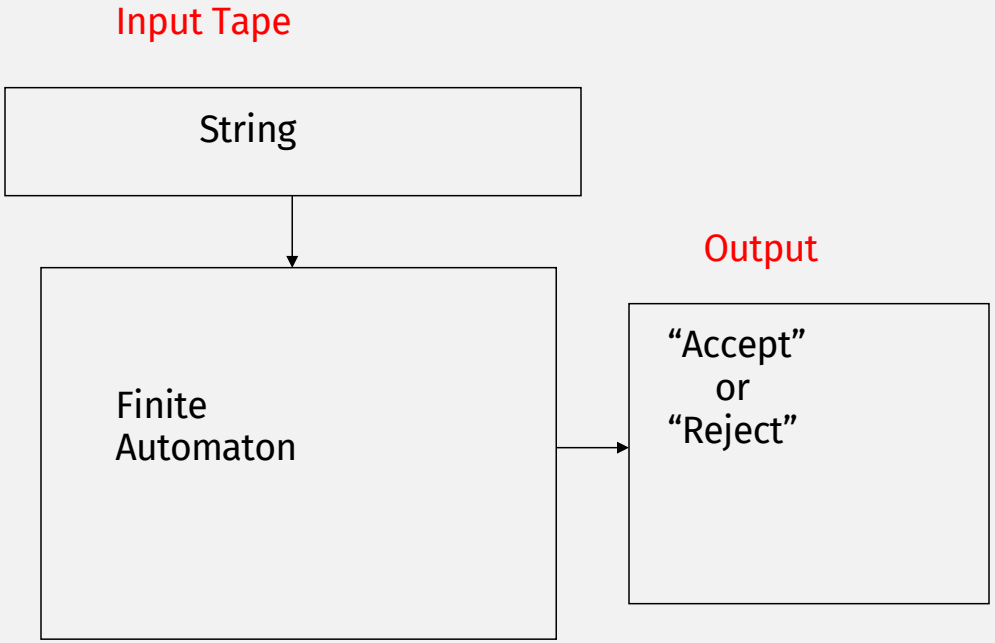
Precise Terminology is Important

- An automaton *accepts* a **string**
- An automaton *recognizes* a **language**

Do FSAs perform some of what we think of as computation?

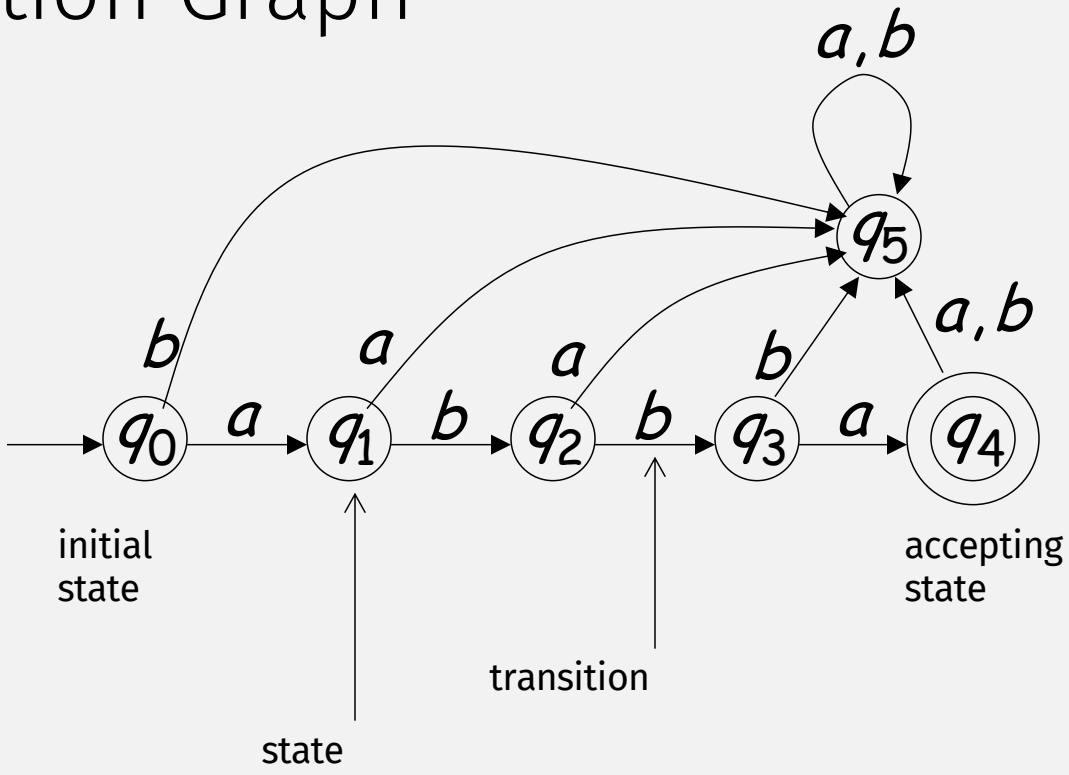
What parts of computation are missing from the behavior of FSA

Deterministic Finite Automaton (DFA)

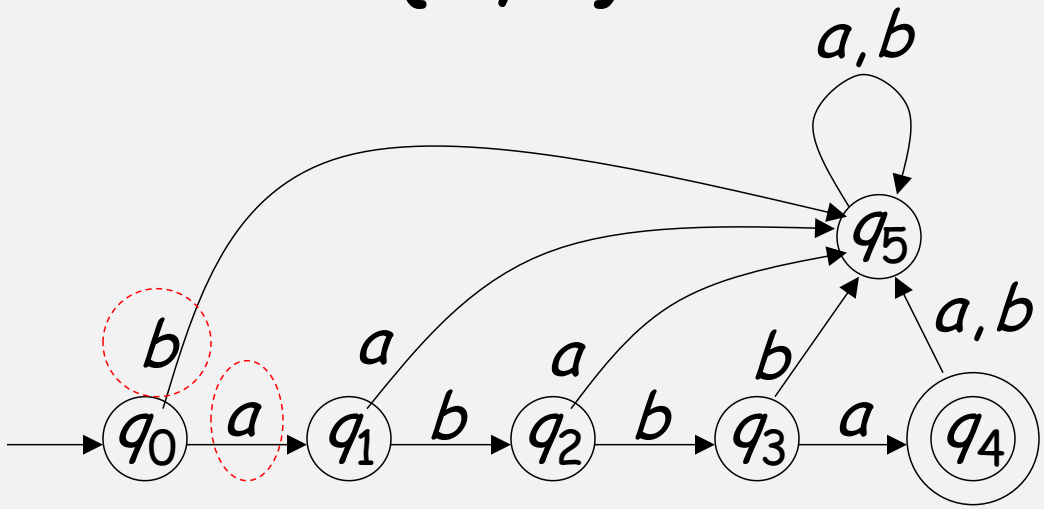
- 

Transition Graph

•

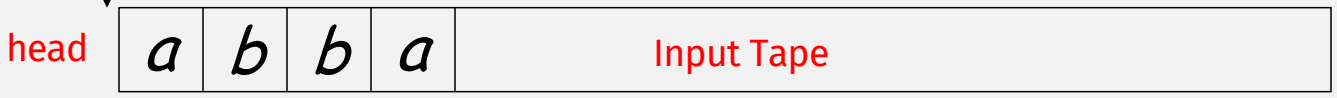


Alphabet $\Sigma = \{a, b\}$

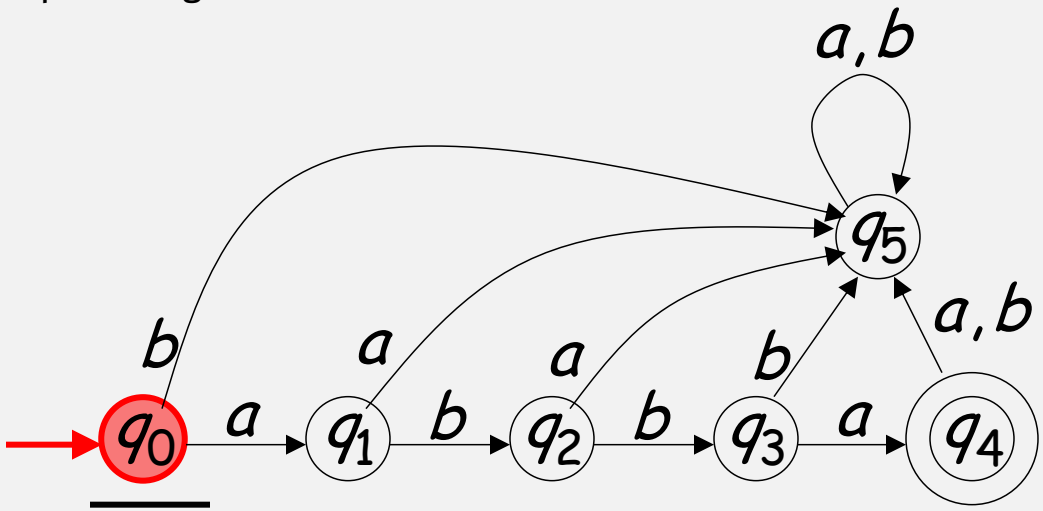


For every state, there is a transition
for every symbol in the alphabet

Initial Configuration

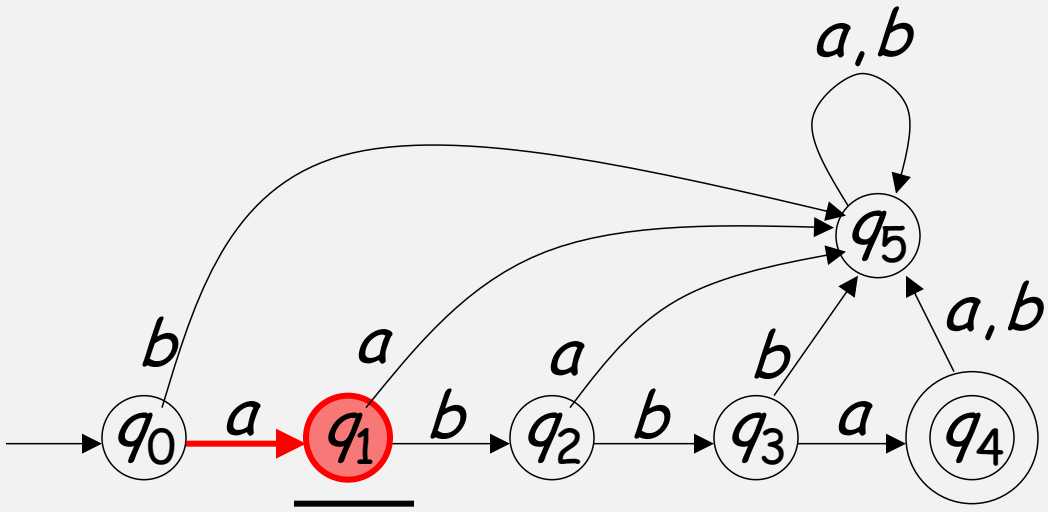


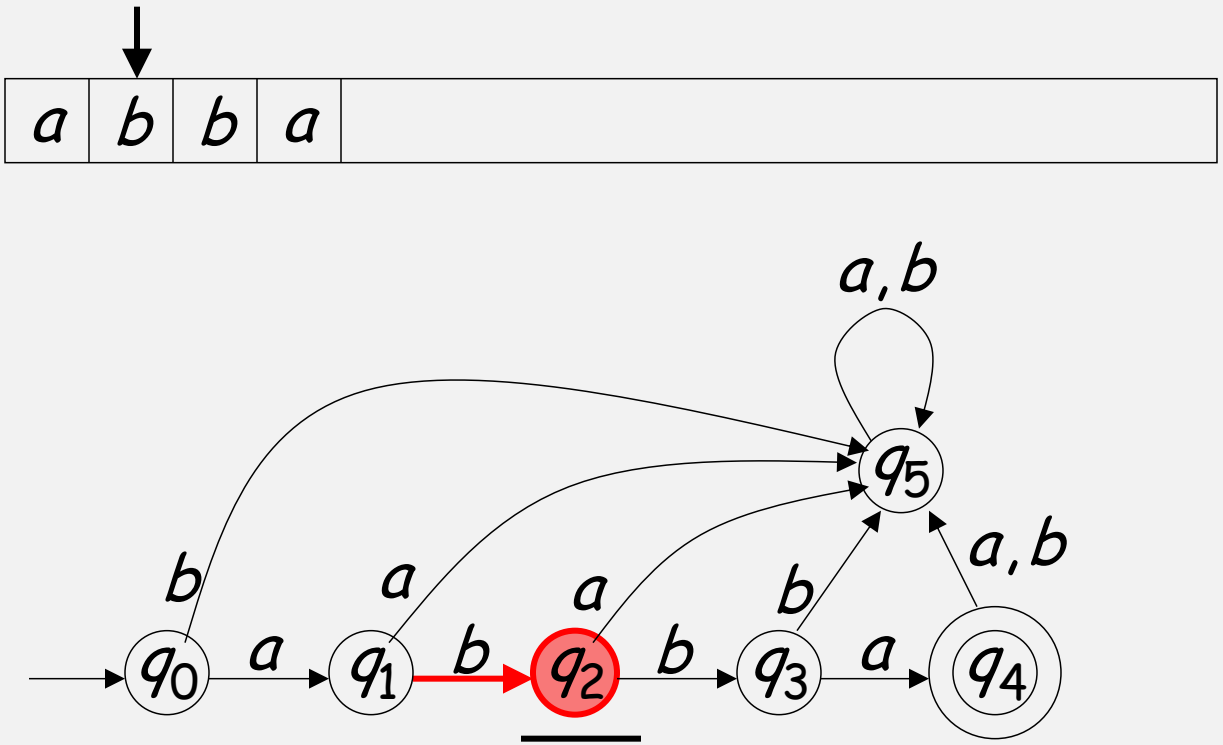
- Input String

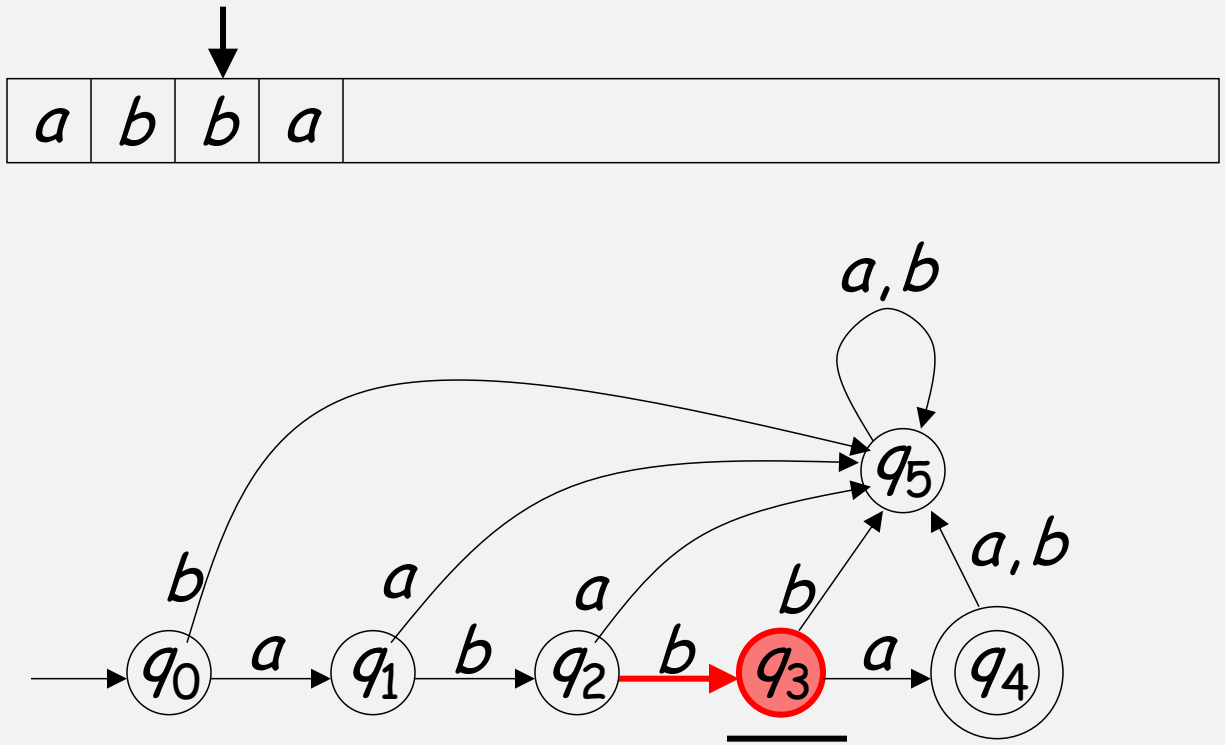


Initial state

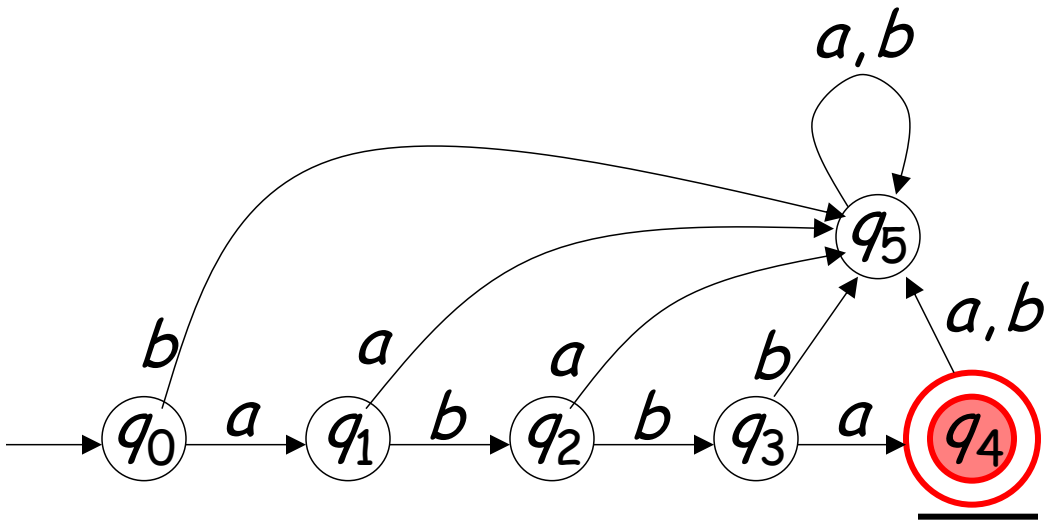
Scanning the Input




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

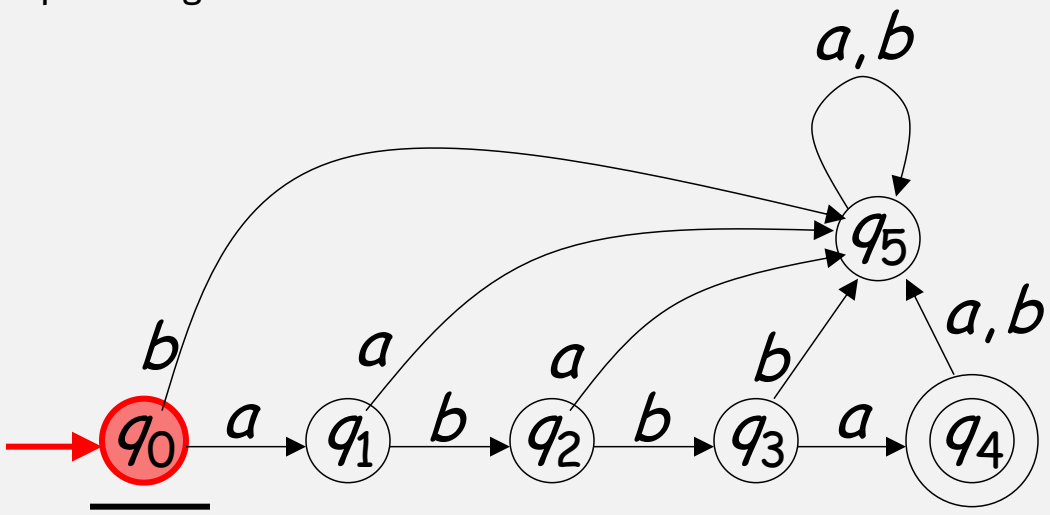
Input finished

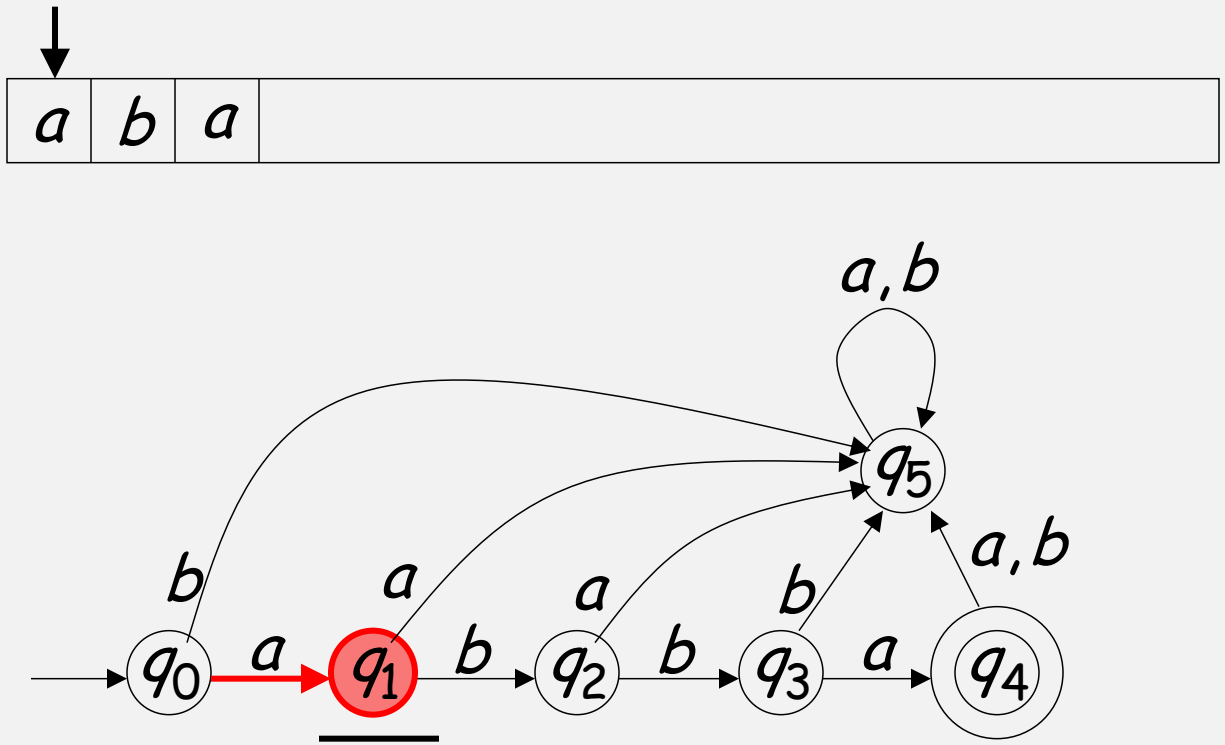


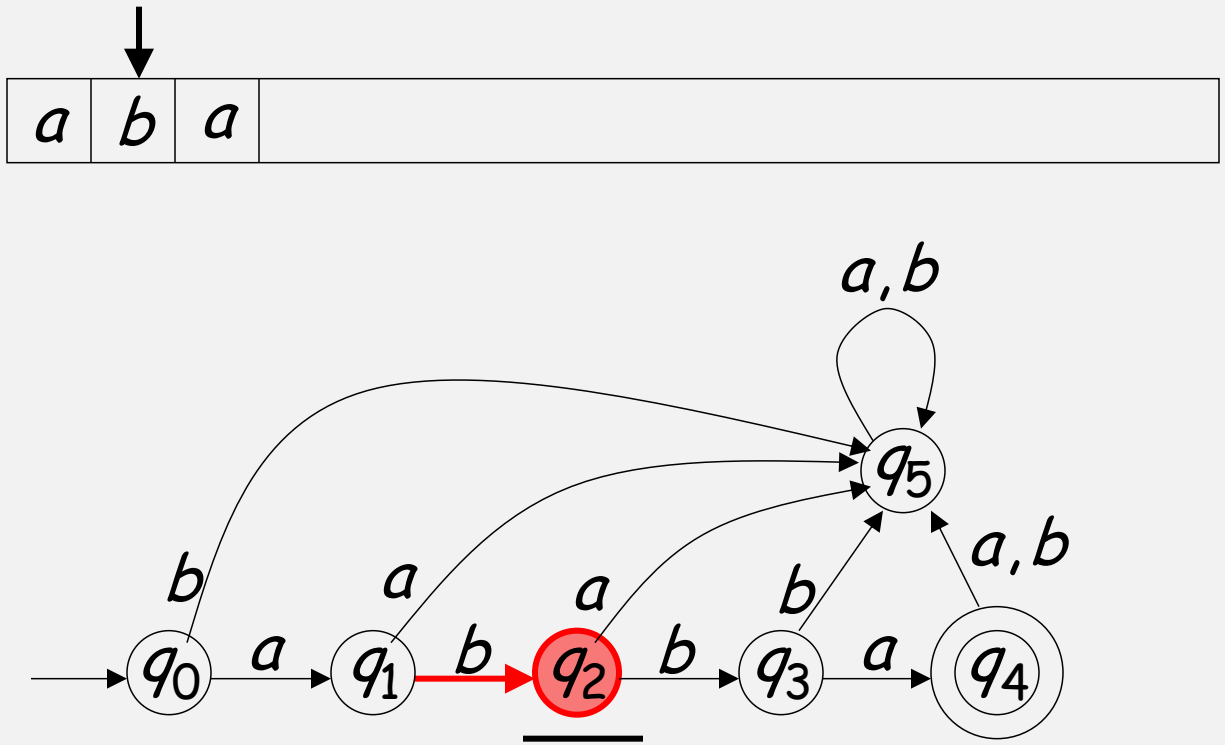
accept

A Rejection Case

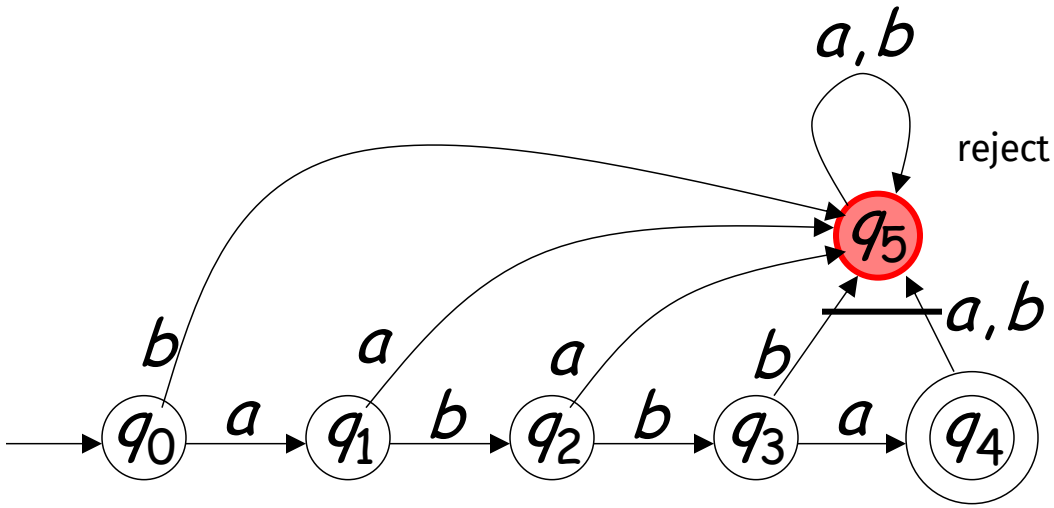
Input String



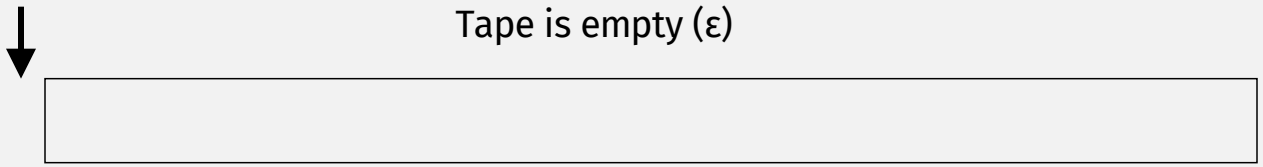

a b a
a, b
q5
b
a
a
b
a, b
q0
a
q1
b
q2
b
q3
a
q4


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

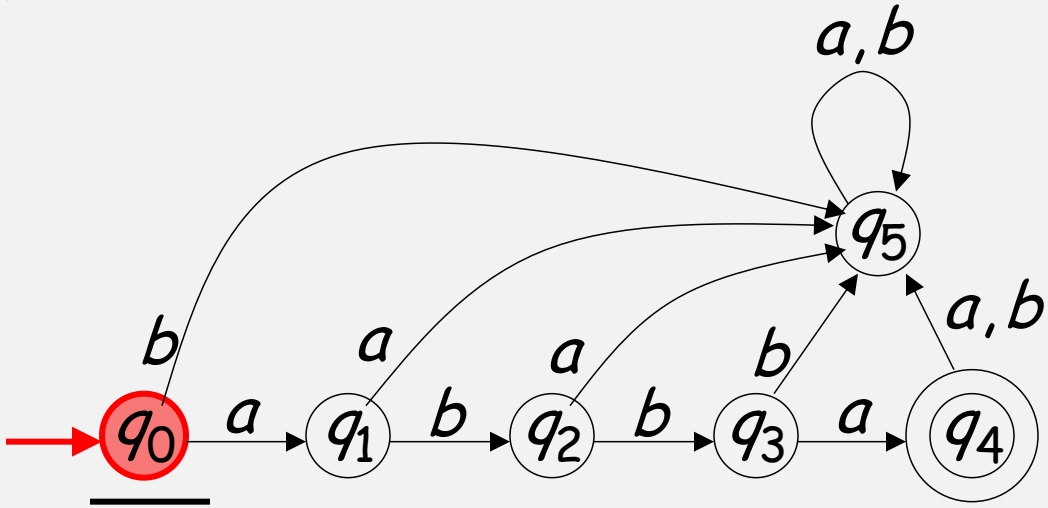
Input finished



Another Rejection Case

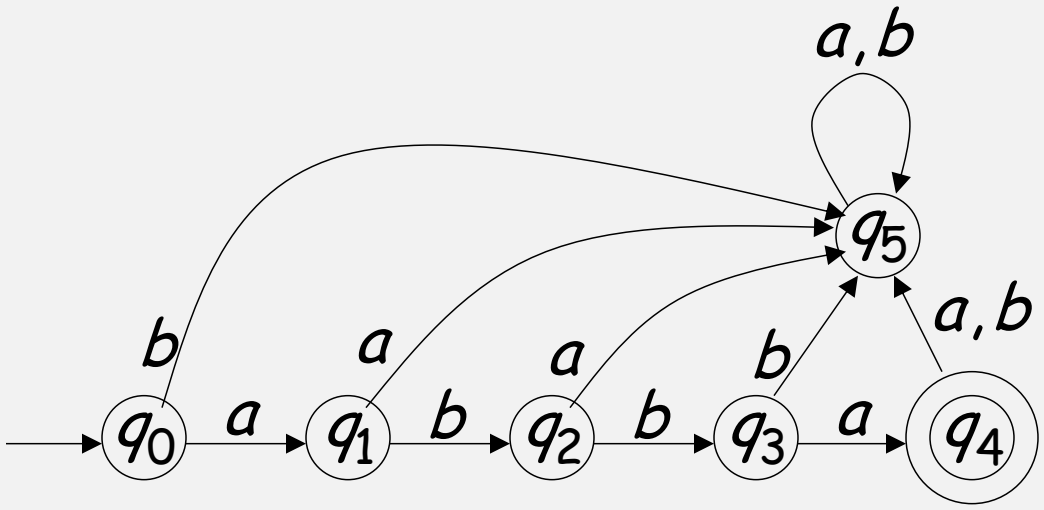


- Input Finished



Language Accepted:

$$L = \{abba\}$$

- 

To accept a string:

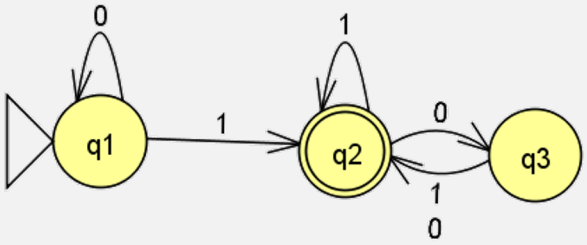
all the input string is scanned
and the last state is accepting

To reject a string:

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

“Running” an FSM “Program” (JFLAP demo)

- **FSM:**



- **Program**: “`1101`”

See “Enrichment” for Links!