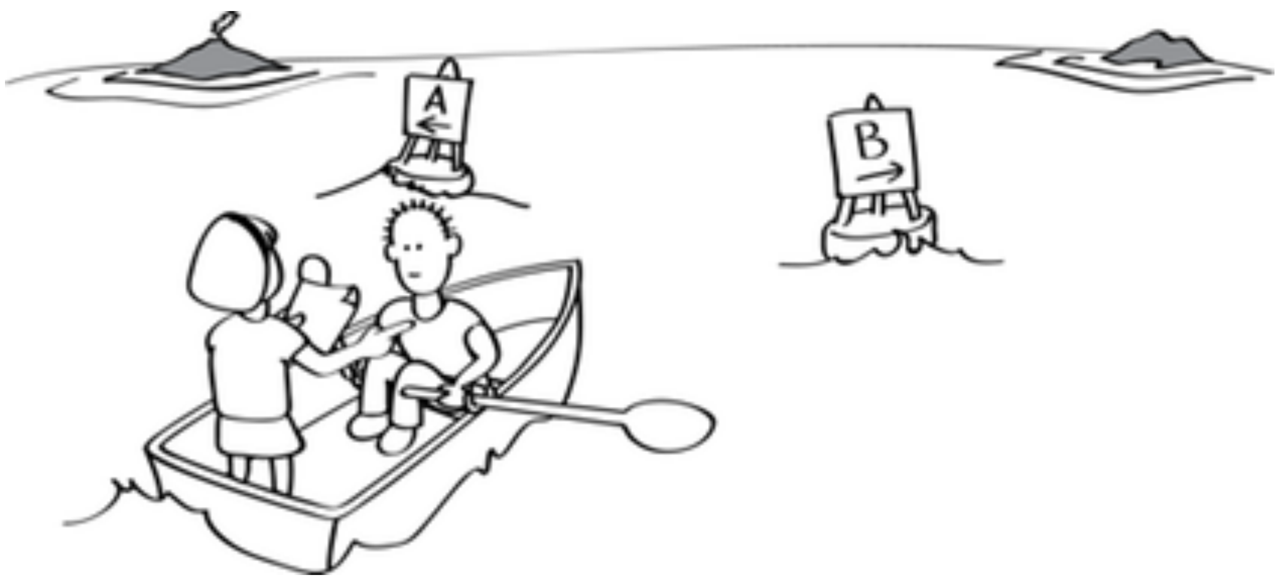
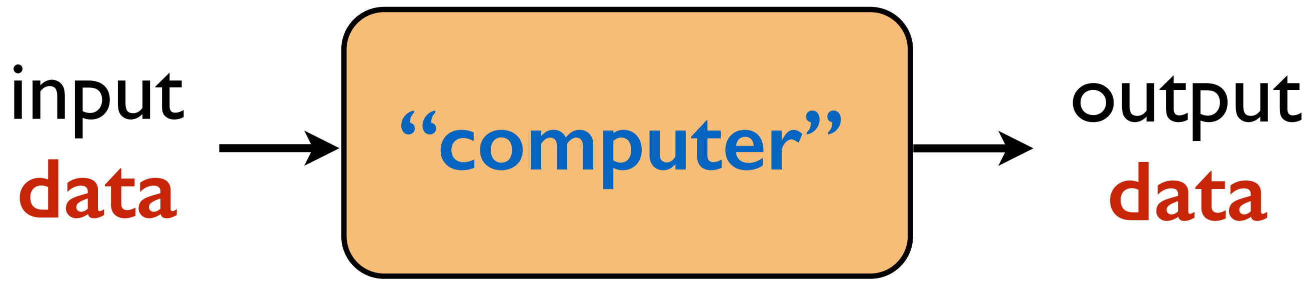
15-251
Great Theoretical Ideas in Computer Science

Lecture 3:
Deterministic Finite Automaton (DFA), Part 1



January 24th, 2017



Computation: manipulation of data.

How do we mathematically/formally represent data?

Representing information

Can encode/represent any kind of data
(*numbers*, *text*, *pairs of numbers*, *graphs*, *images*, *etc…*)
with a **finite length (binary) string**.

Representing information

$$\Sigma = \{0, 1\}$$

alphabet

symbols of the alphabet

Σ^* = the set of all finite length strings over Σ

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots\}$$

string of length 0 (empty string)

A subset $L \subseteq \Sigma^*$ is called a *language*.

Representing information

$$\Sigma = \{0, 1\}$$

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

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

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, g, h, i, j, k, \\ l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$$

Can use whichever is convenient.

What is a computational problem?

Definition: A *computational problem* is a function

$$f : \Sigma^* \to \Sigma^*.$$

Definition: A *decision problem* is a function

$$f : \Sigma^* \to \{0, 1\}.$$

No, Yes

False, True

Reject, Accept

What is a computational problem?

IMPORTANT

There is a one-to-one correspondence between decision problems and languages.

Instance	Solution
$\boxed{\epsilon}$	1
$\boxed{0}$	1
$\boxed{1}$	1
$\boxed{00}$	1
01	0
10	0
$\boxed{11}$	1
$\boxed{000}$	1
001	0
$\vdots$	$\vdots$

$$L \subseteq \Sigma^*$$

$$L = \{\epsilon, 0, 1, 00, 11, 000, \ldots\}$$

Our focus will be on languages!

(decision problems)

- Convenient restriction.
- Usually “without loss of generality”.

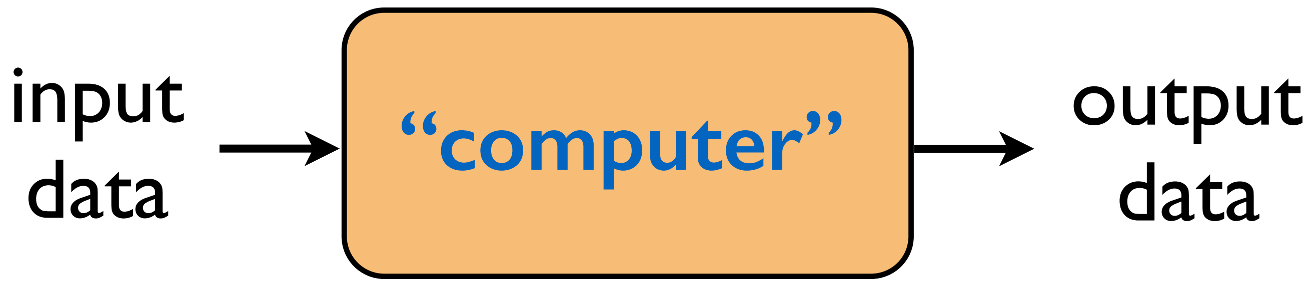
Integer factorization problem

Given as input a natural number **N**, output its prime factorization.

Integer factorization problem, decision version

Given as input natural numbers **N** and **k**,
does **N** have a factor (strictly) between **1** and **k**?

This Week and Next Week



What is computation?

What is an algorithm?

How can we mathematically define them?

This Week

Introducing deterministic finite automata (DFA)

- restricted model of computation
- very limited memory
  - reads input from left to right, and accepts or rejects.
    (one pass through the input)

Let's assume two things about our world

No universal machines exist.

+

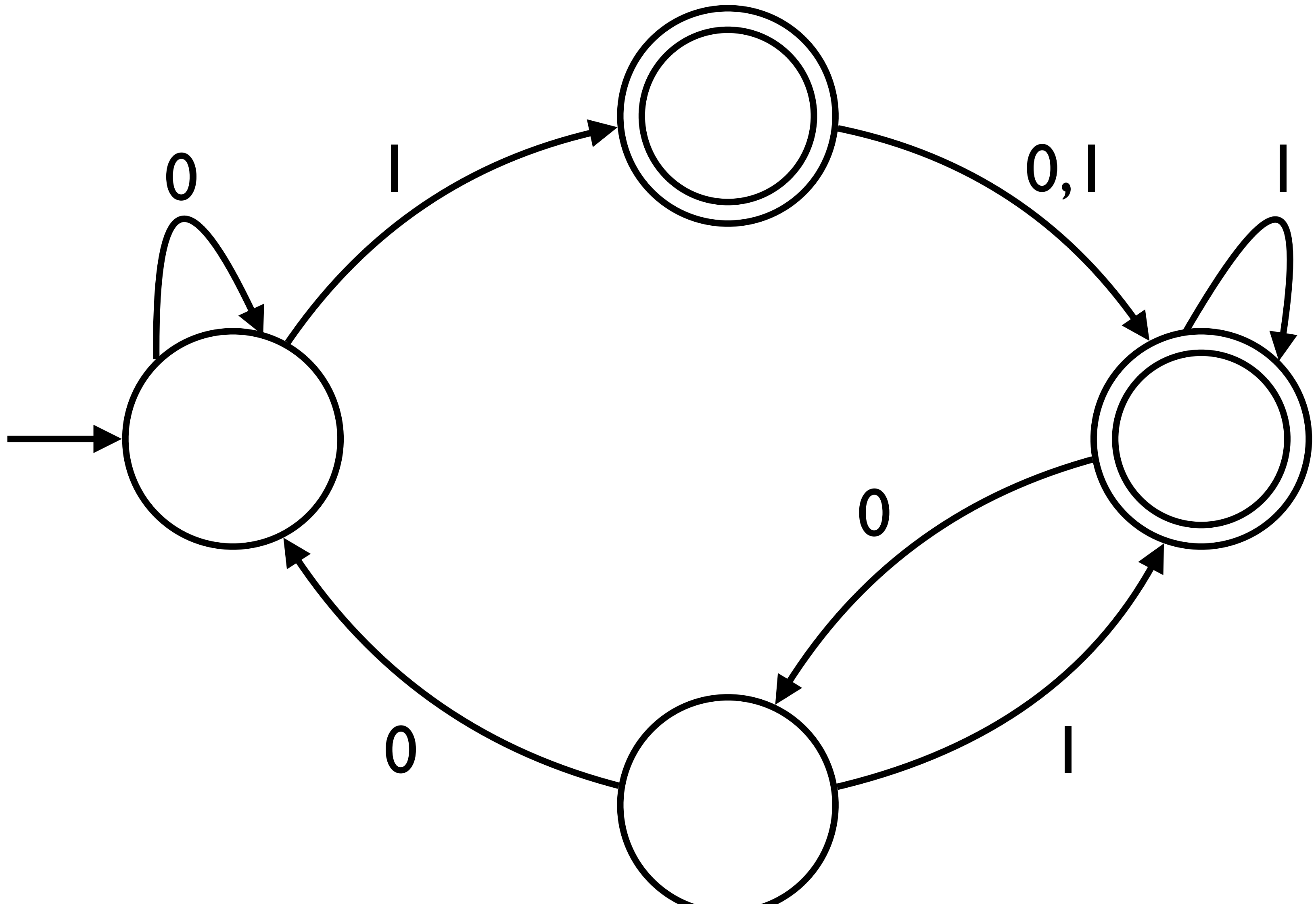
isPrime

Sorting

We only have machines to solve decision problems.

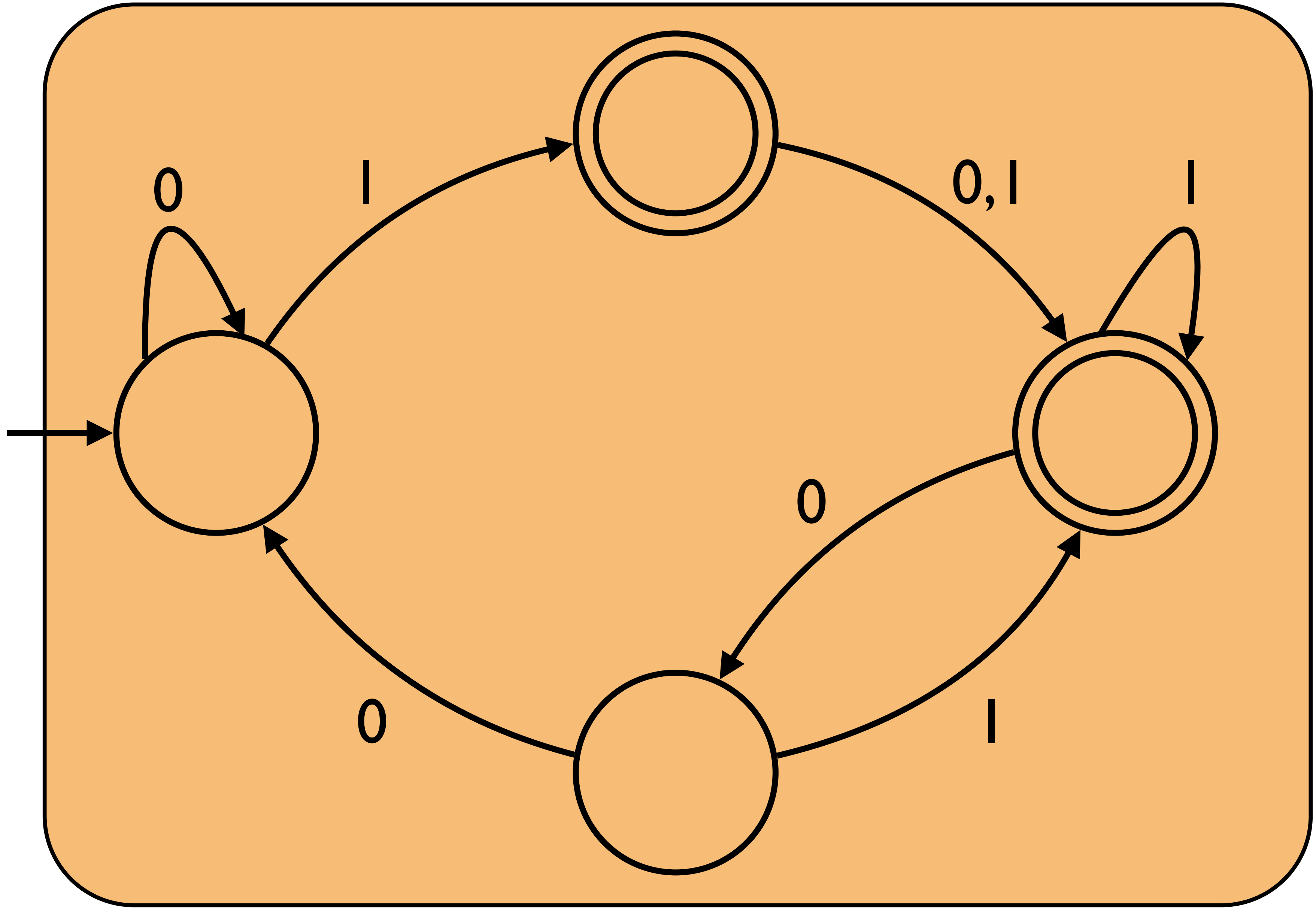
State diagram of a DFA

$\Sigma = \{0, 1\}$



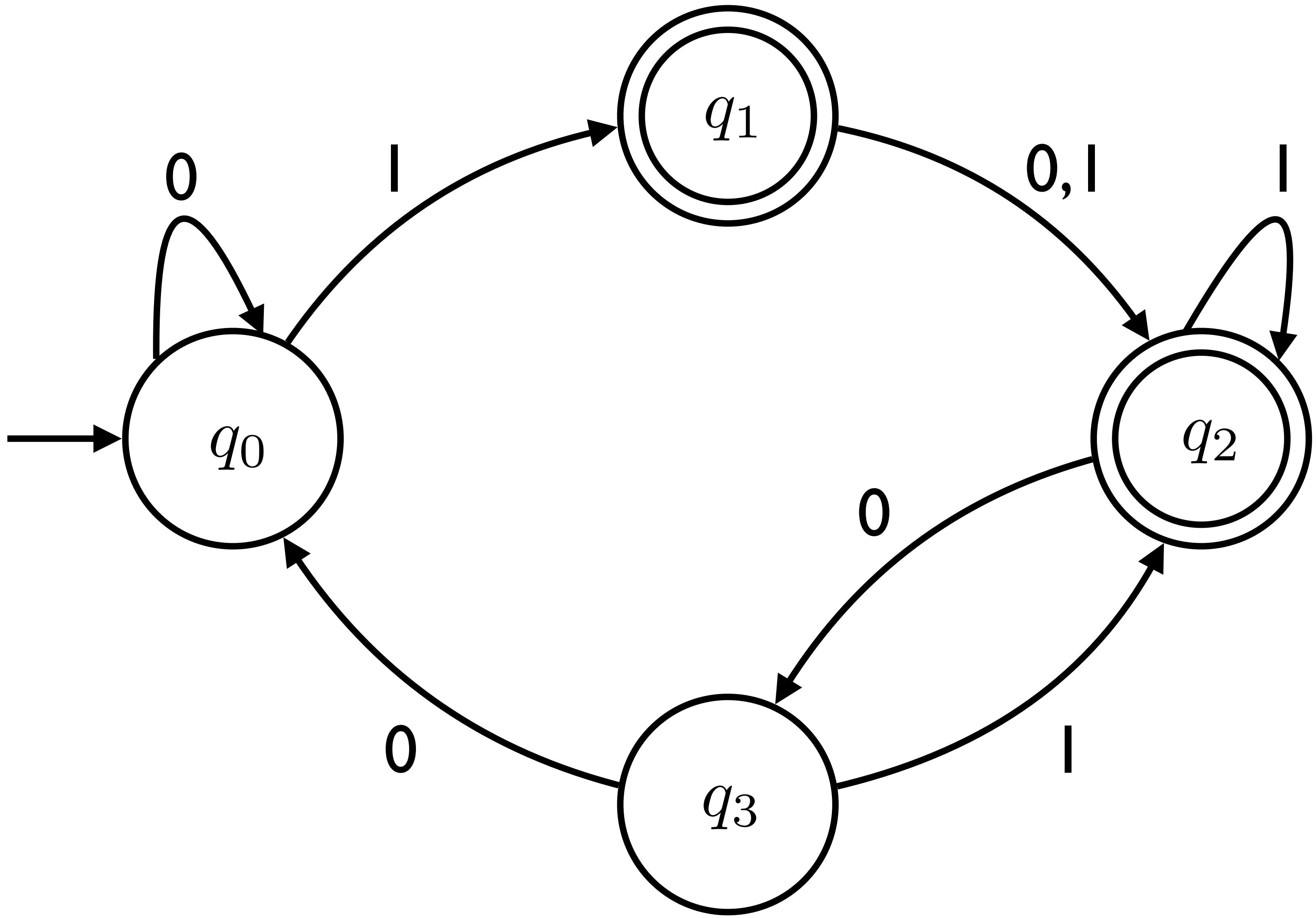
State diagram of a DFA

$\Sigma = \{0, 1\}$



State diagram of a DFA

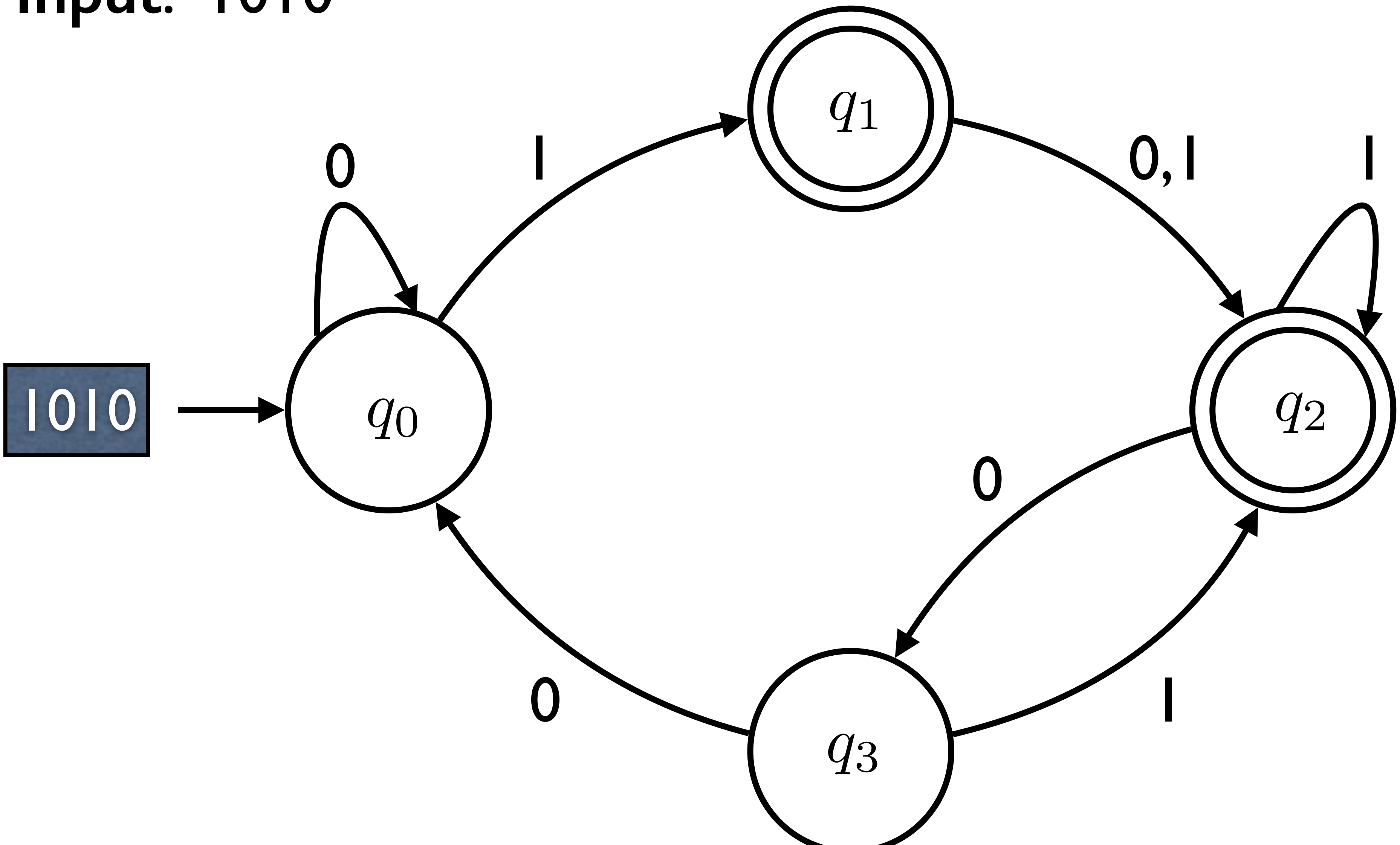
$\Sigma = \{0, 1\}$



Simulation of a DFA

$\Sigma = \{0, 1\}$

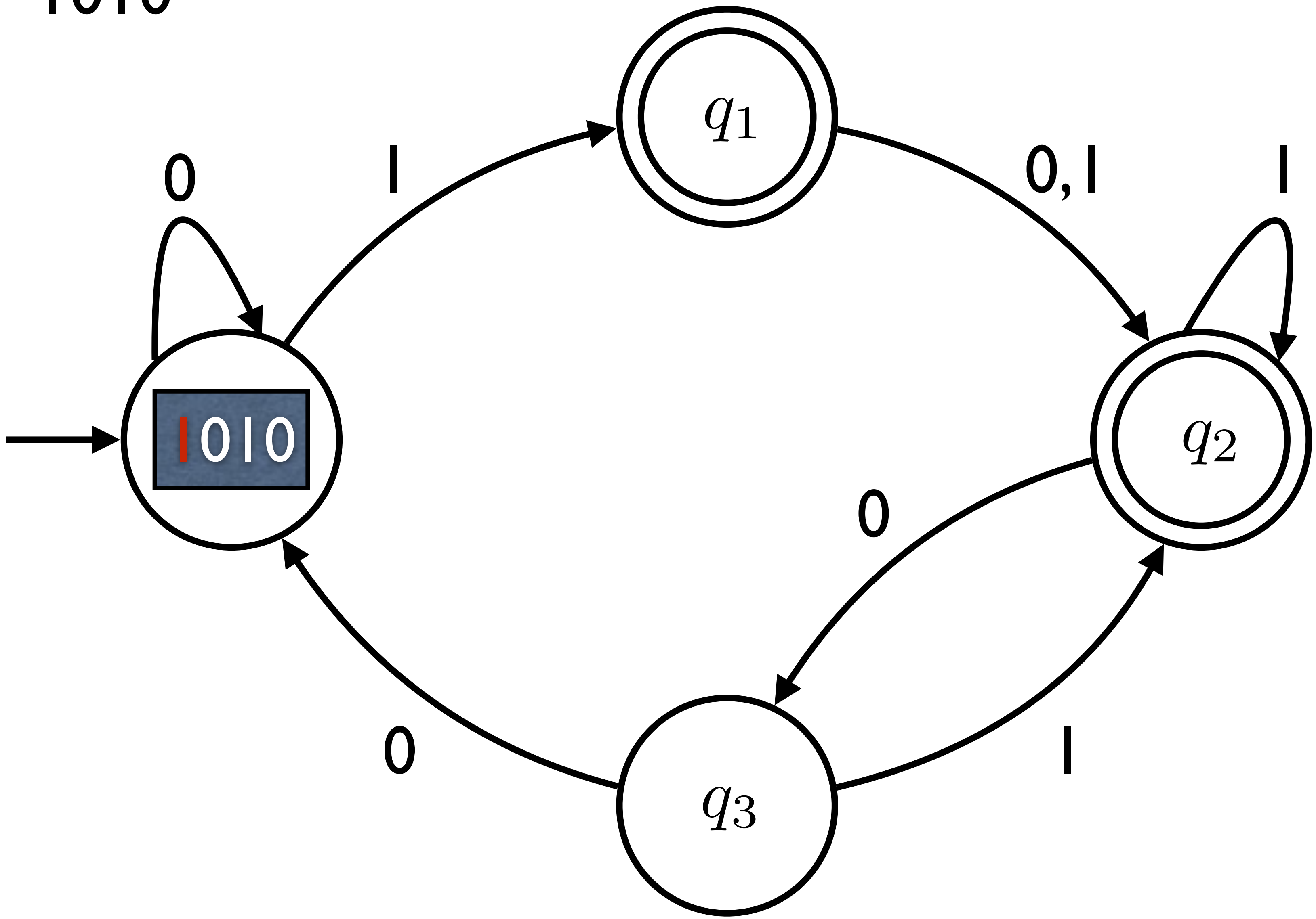
Input: 1010



Simulation of a DFA

$\Sigma = \{0, 1\}$

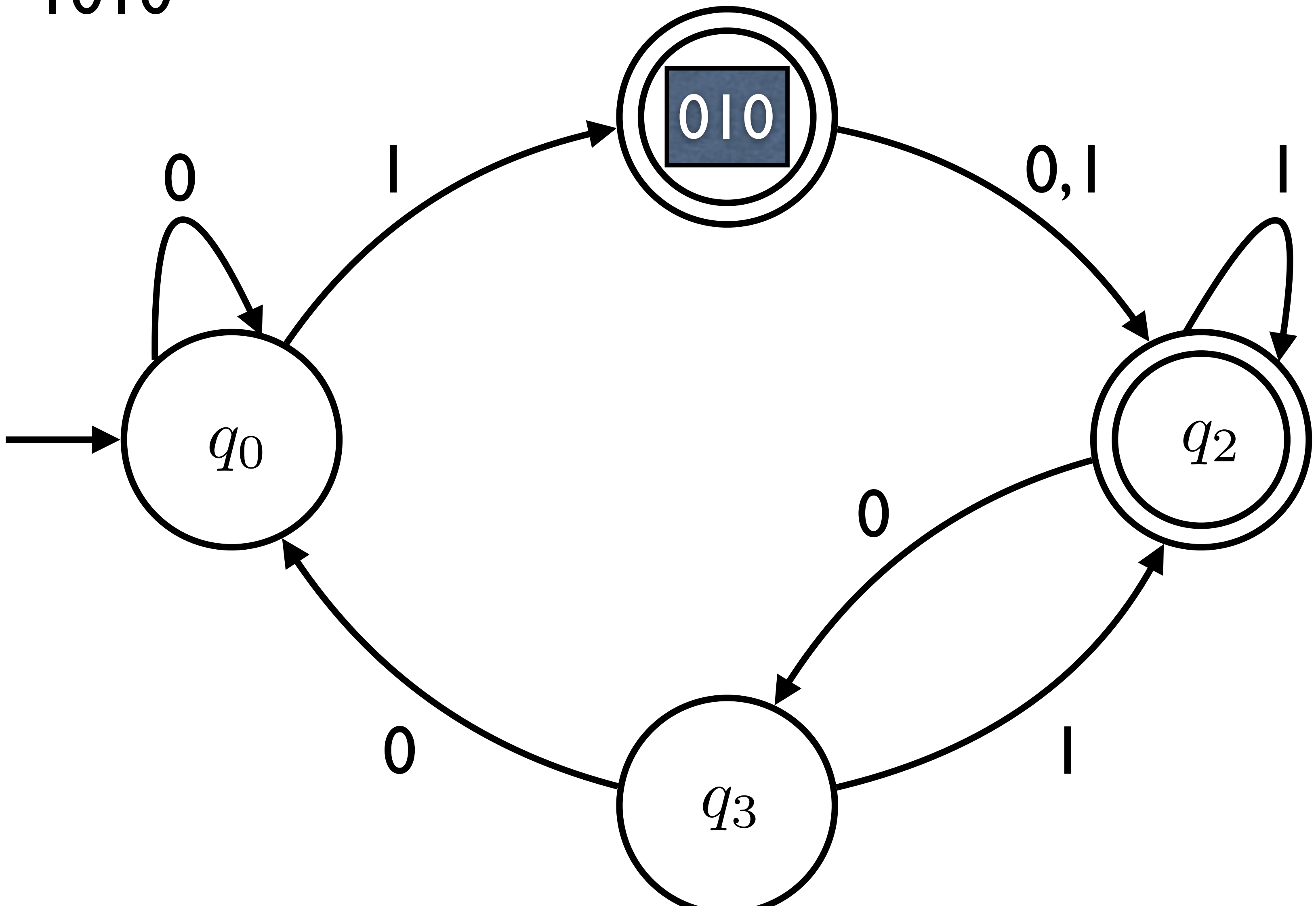
Input: 1010



Simulation of a DFA

$\Sigma = \{0, 1\}$

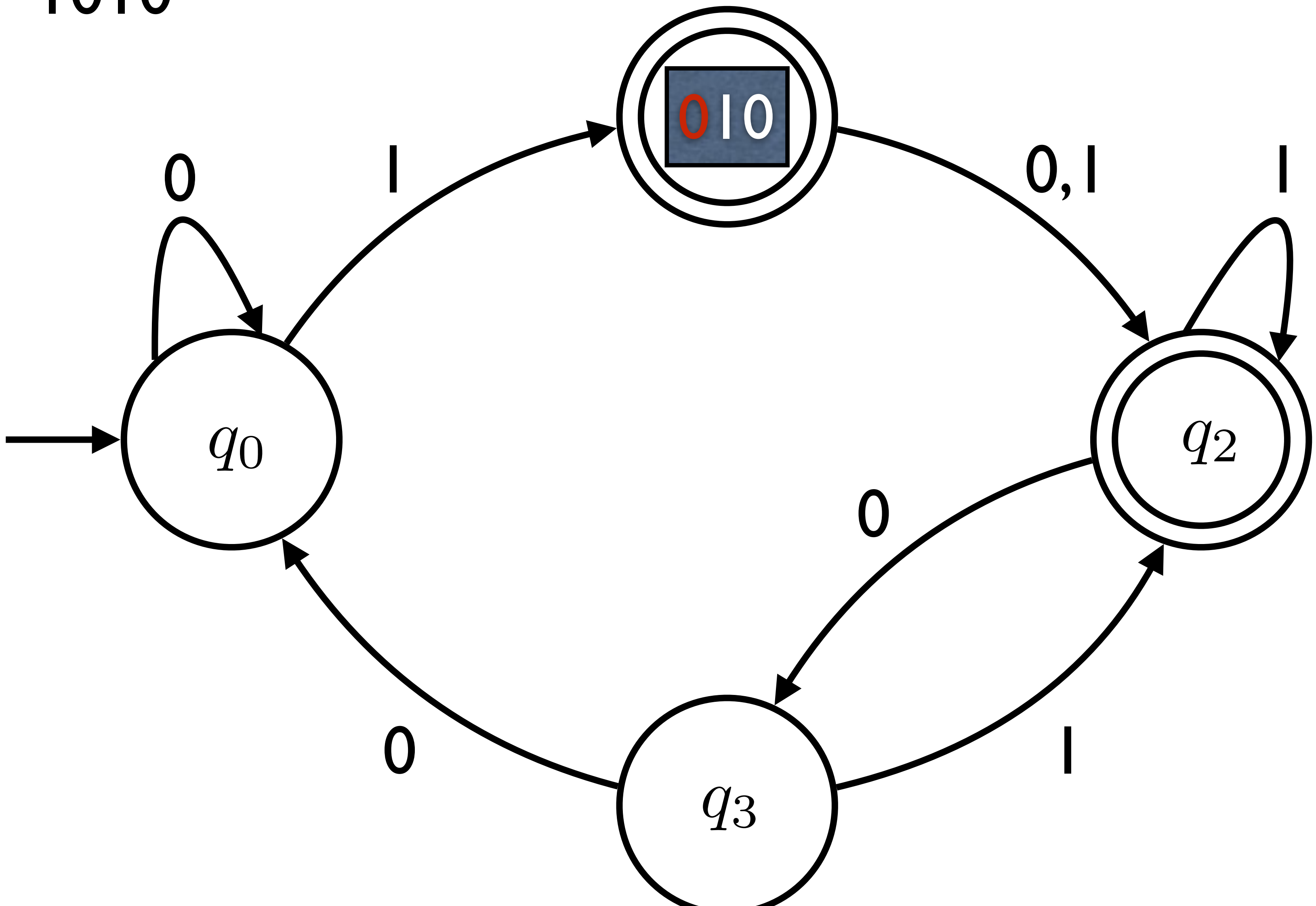
Input: 1010



Simulation of a DFA

$\Sigma = \{0, 1\}$

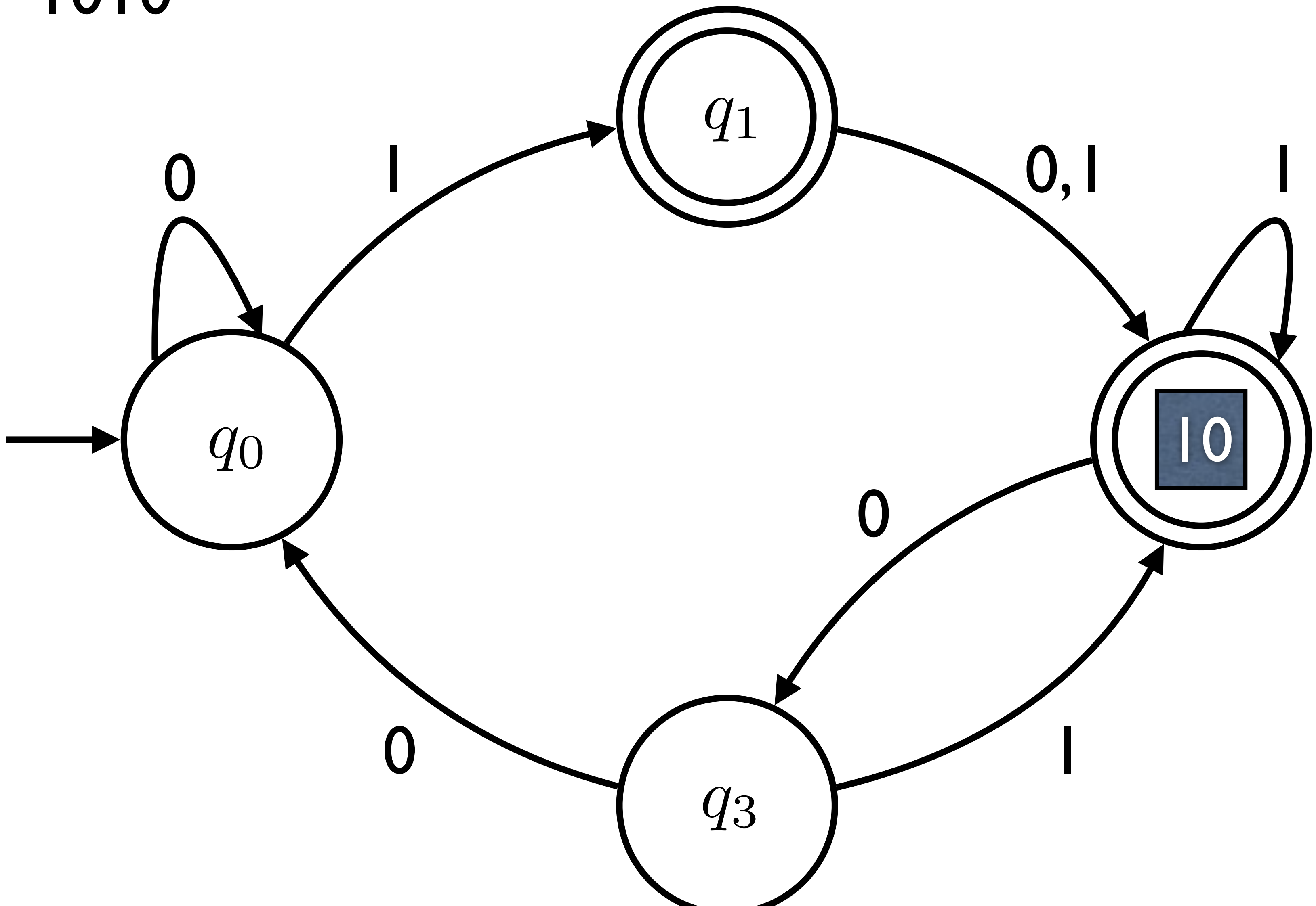
Input: 1010



Simulation of a DFA

$\Sigma = \{0, 1\}$

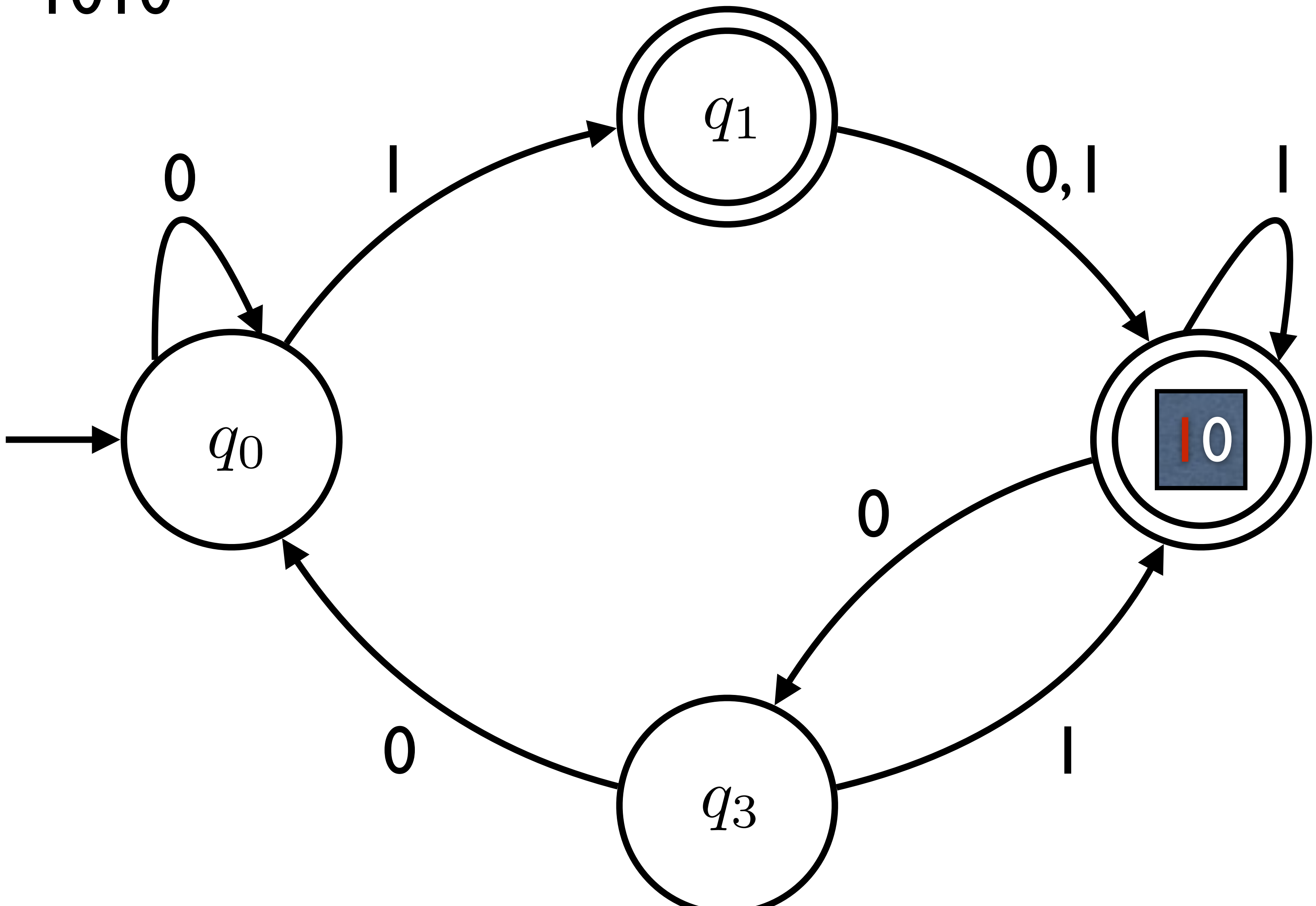
Input: 1010



Simulation of a DFA

$\Sigma = \{0, 1\}$

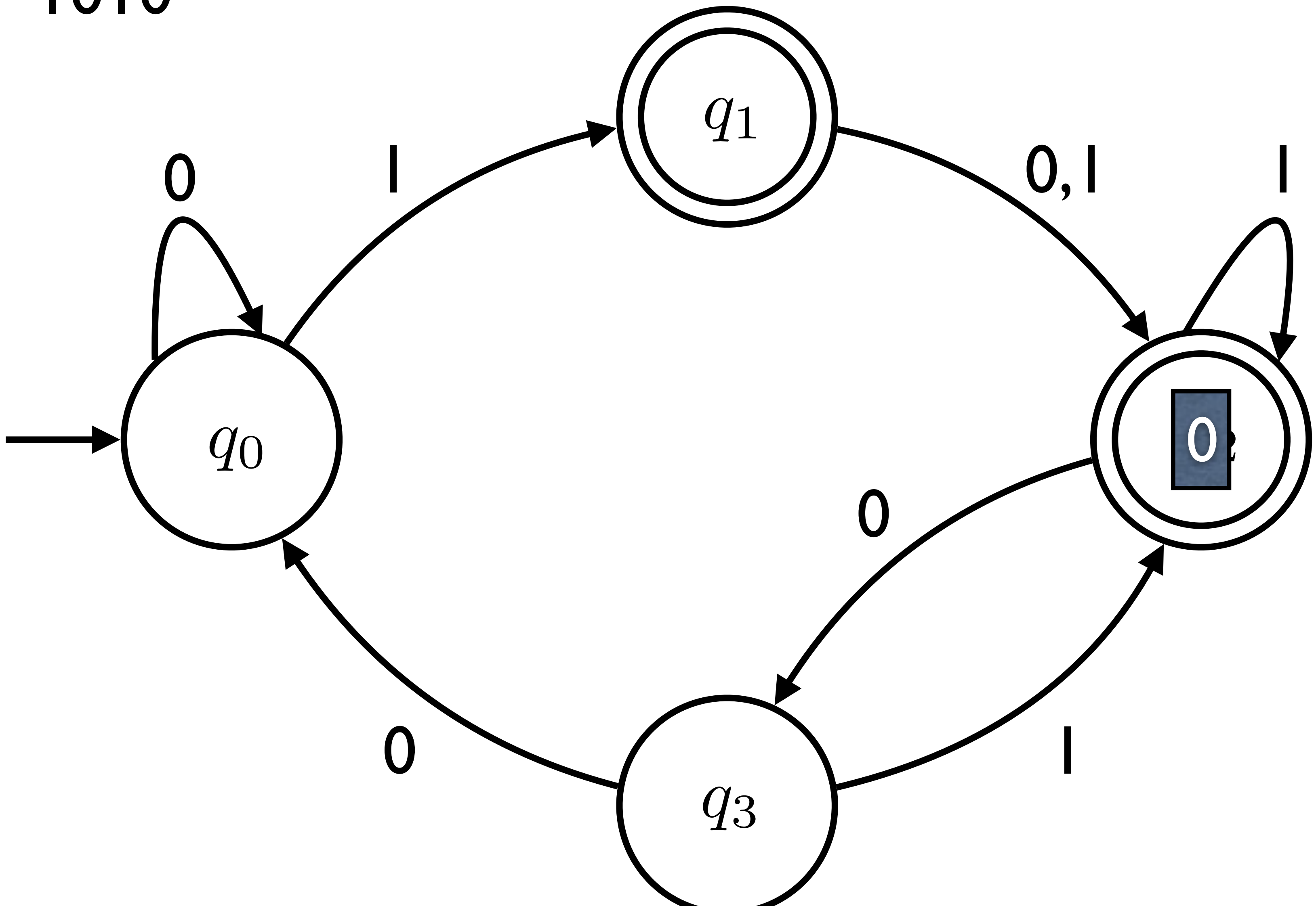
Input: 1010



Simulation of a DFA

$\Sigma = \{0, 1\}$

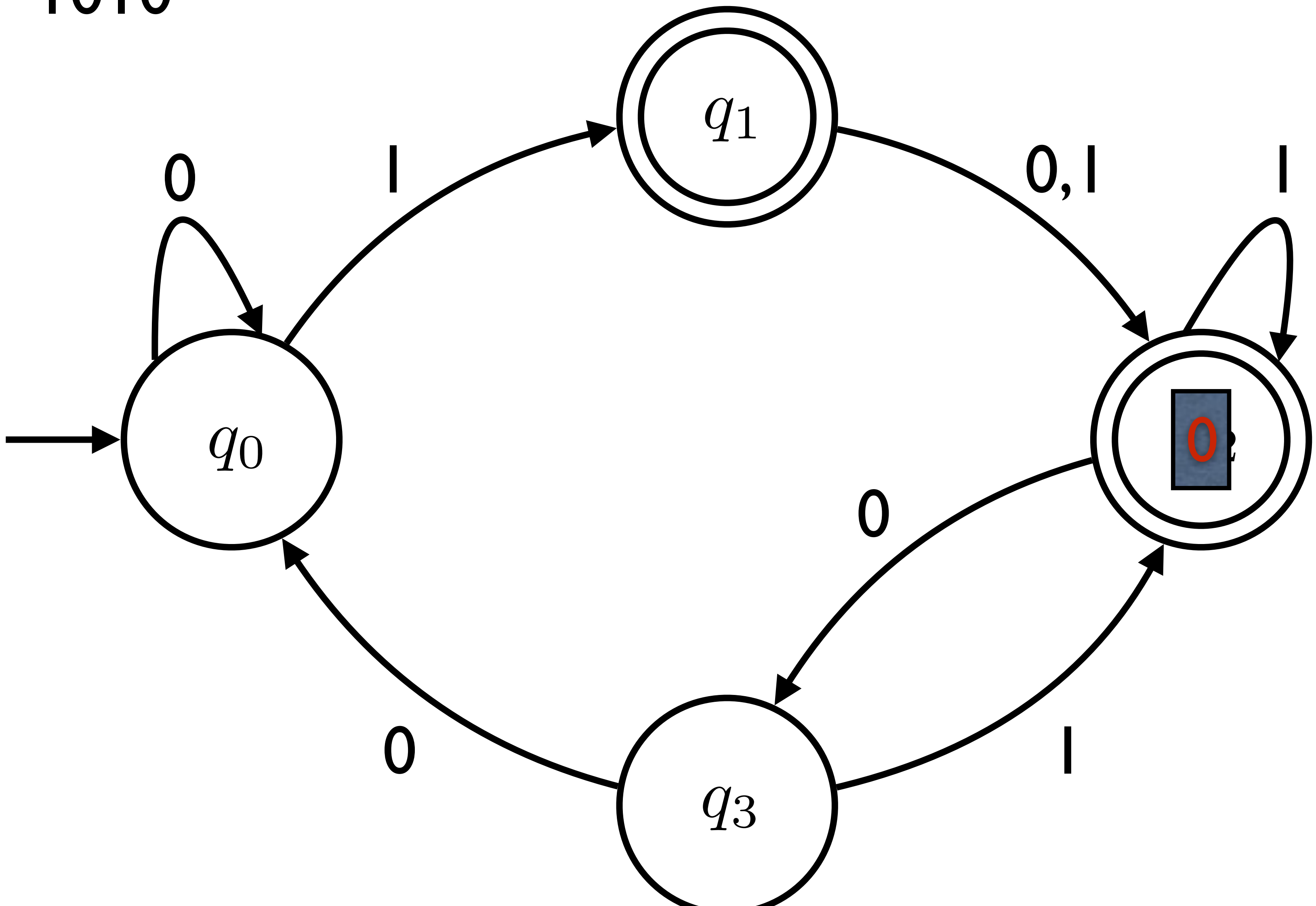
Input: 1010



Simulation of a DFA

$\Sigma = \{0, 1\}$

Input: 1010

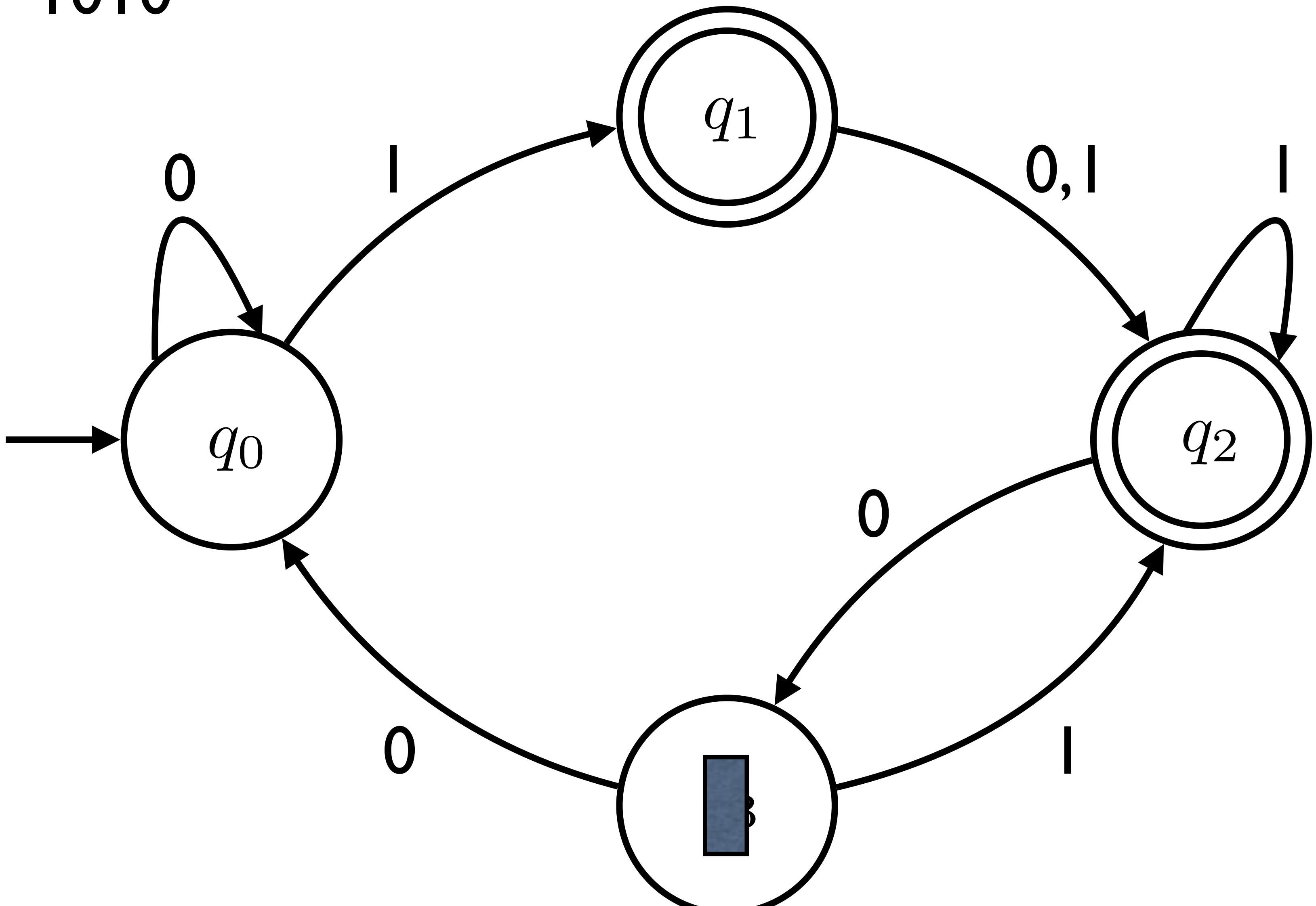


Simulation of a DFA

$\Sigma = \{0, 1\}$

Input: 1010

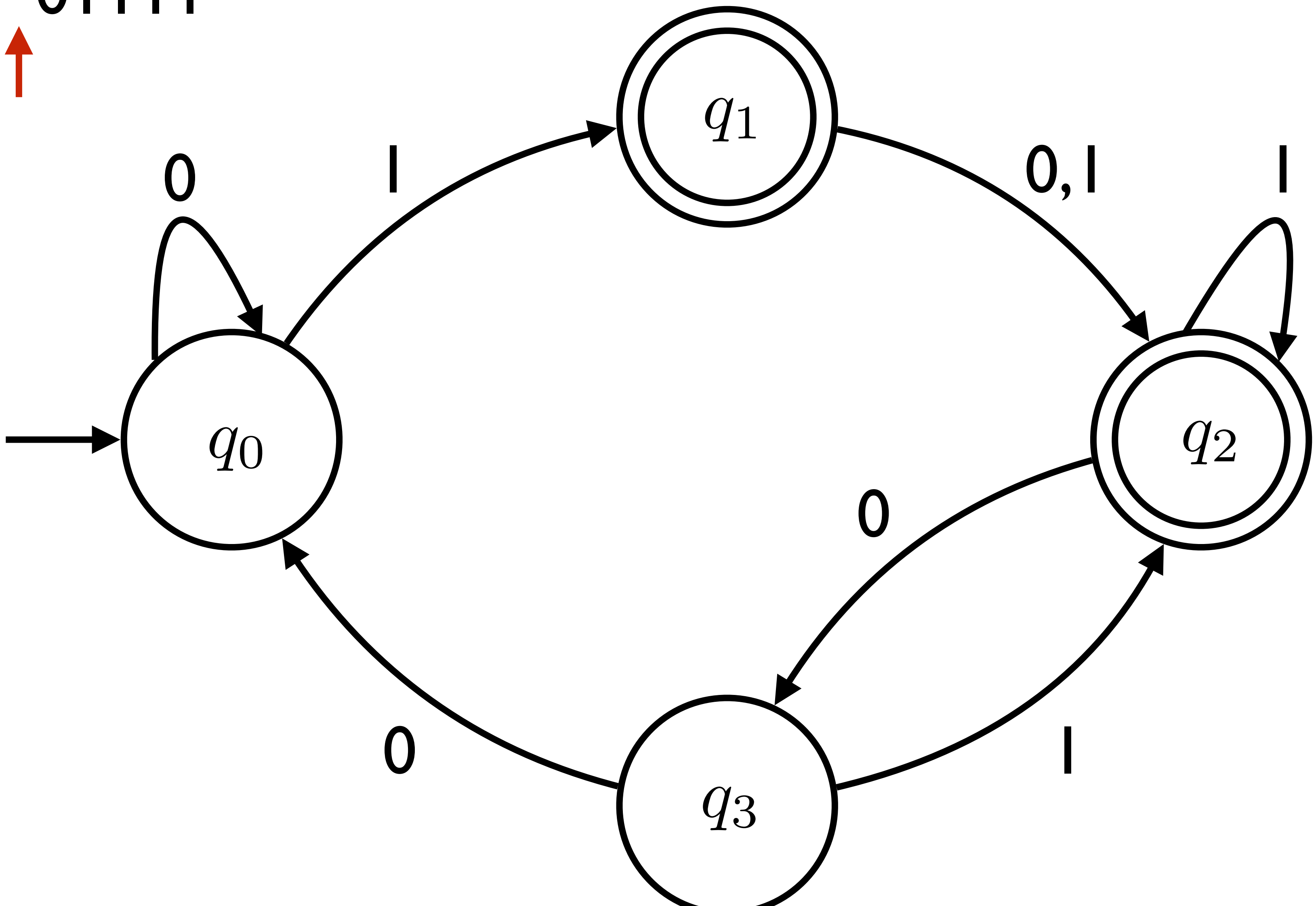
Decision: Reject



Simulation of a DFA

$\Sigma = \{0, 1\}$

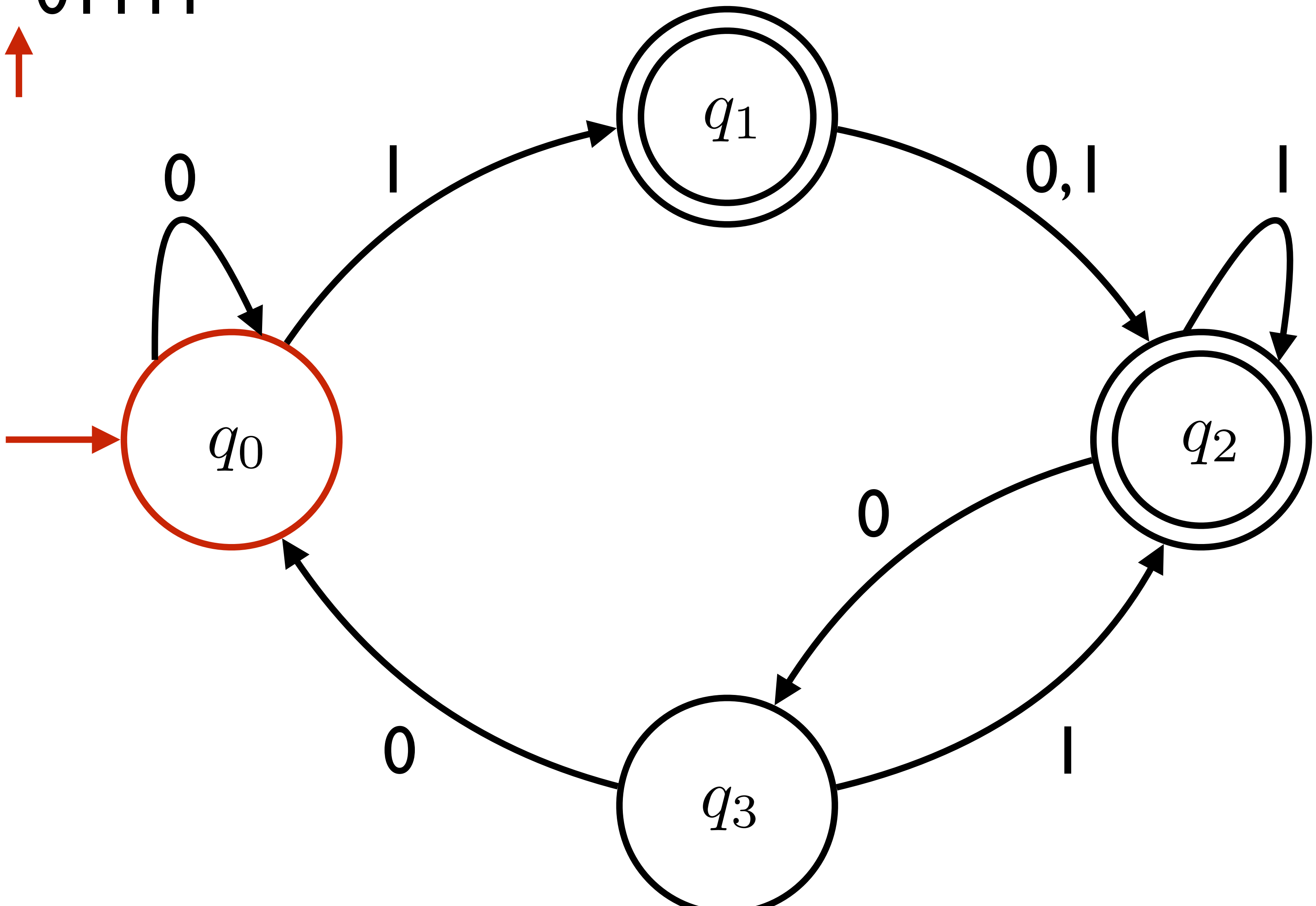
Input: 01111



Simulation of a DFA

$\Sigma = \{0, 1\}$

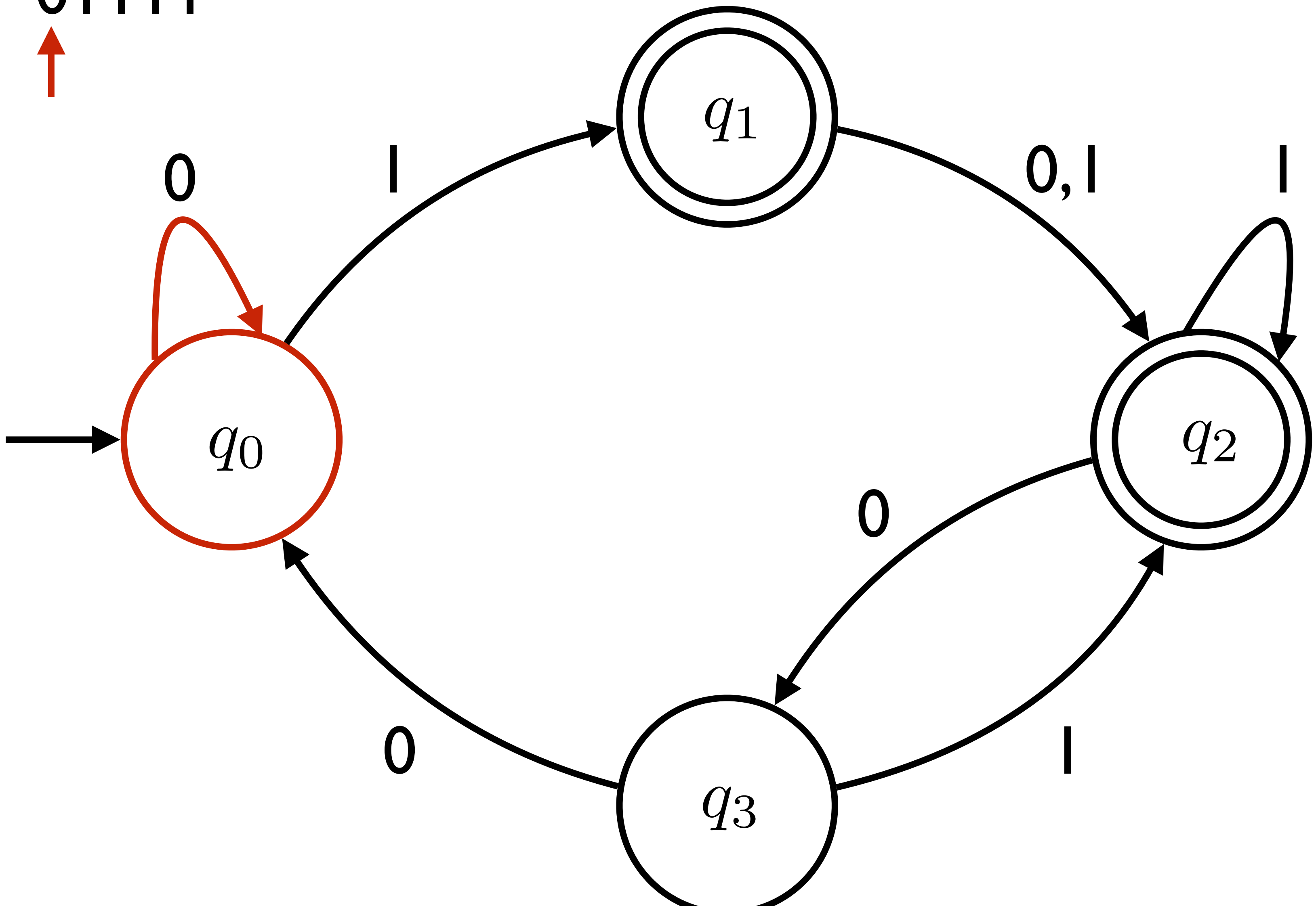
Input: 01111



Simulation of a DFA

$\Sigma = \{0, 1\}$

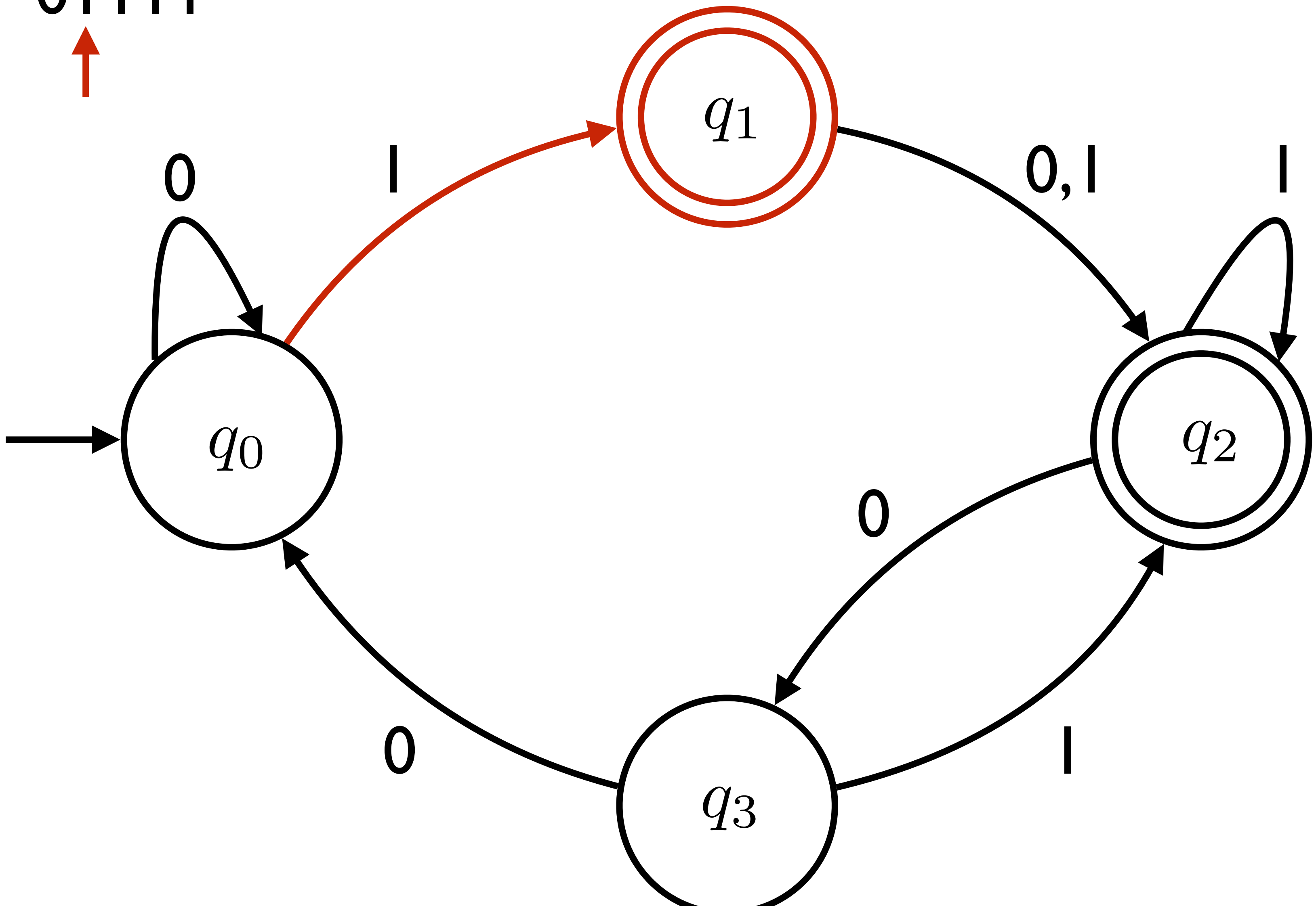
Input: 01111



Simulation of a DFA

$\Sigma = \{0, 1\}$

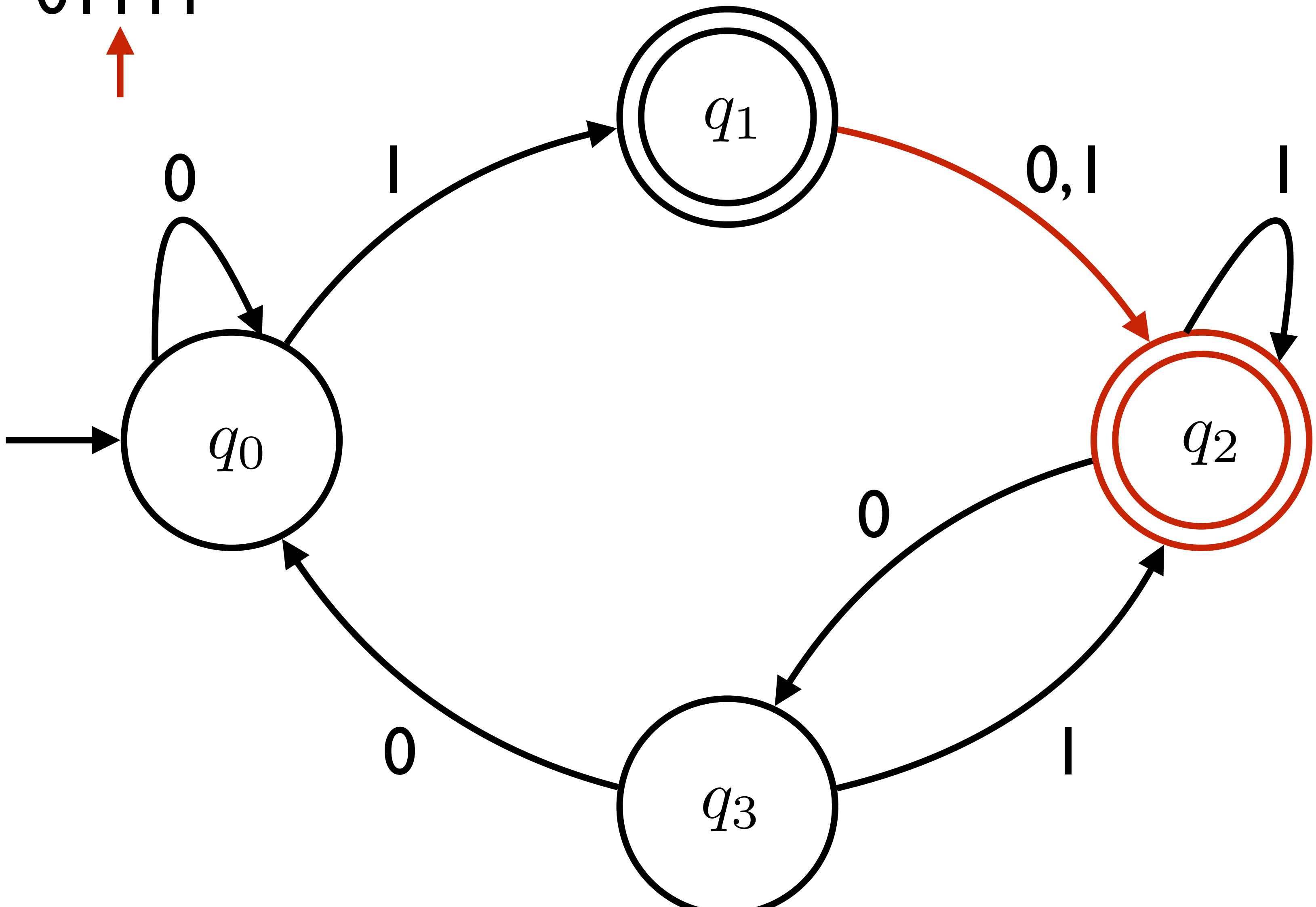
Input: 01111



Simulation of a DFA

$\Sigma = \{0, 1\}$

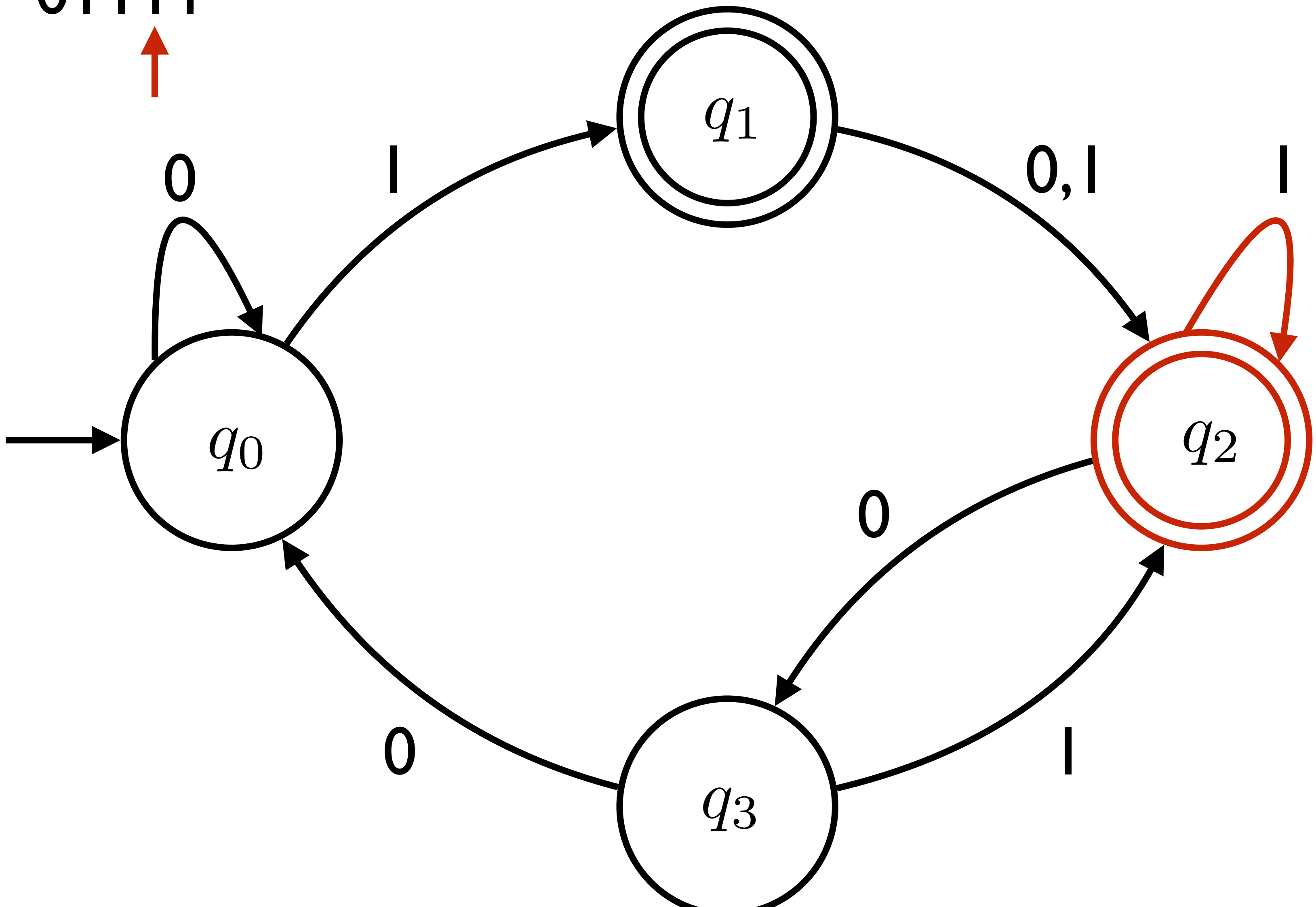
Input: 0 1 1 1 1



Simulation of a DFA

$\Sigma = \{0, 1\}$

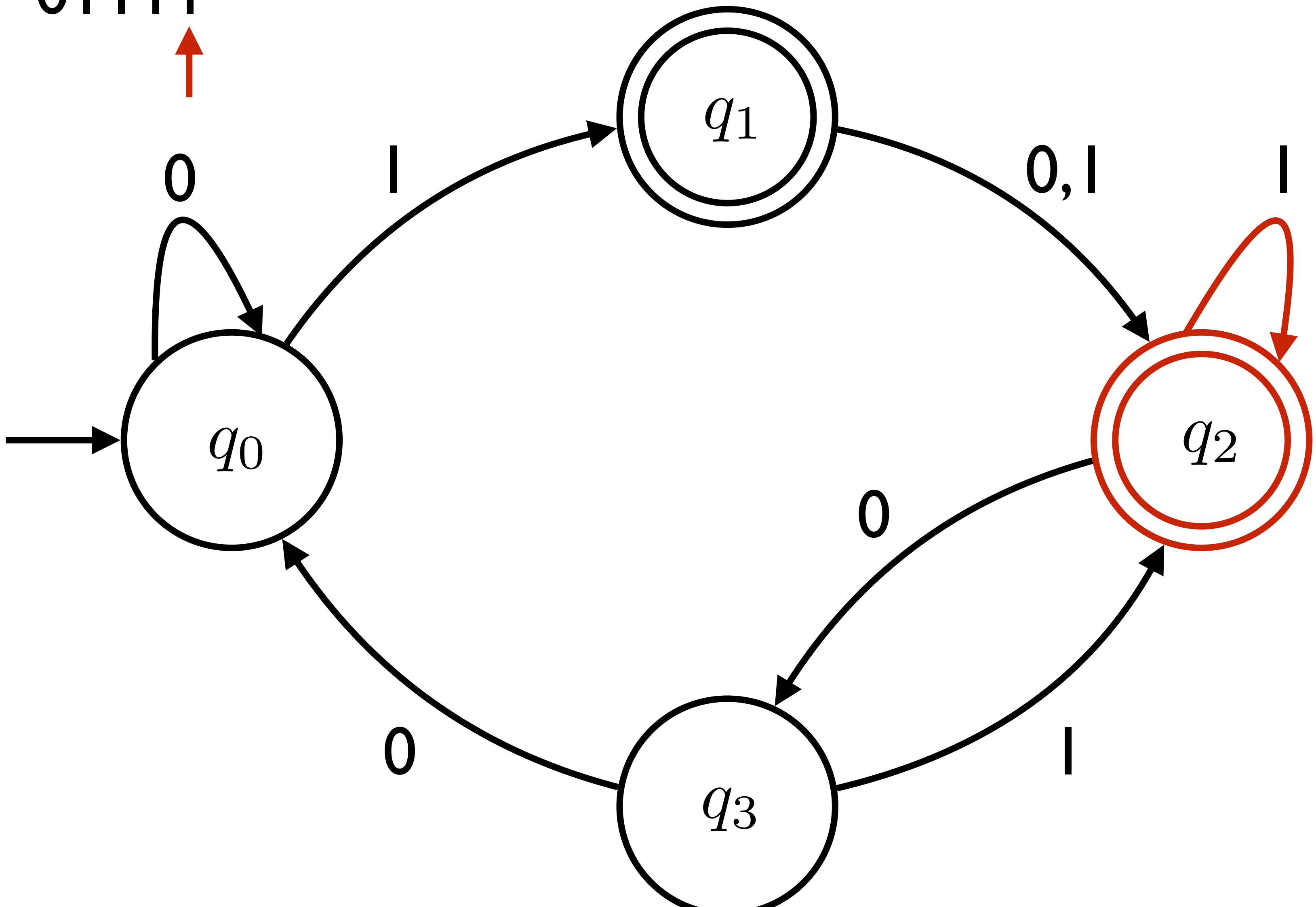
Input: 01111



Simulation of a DFA

$\Sigma = \{0, 1\}$

Input: 01111

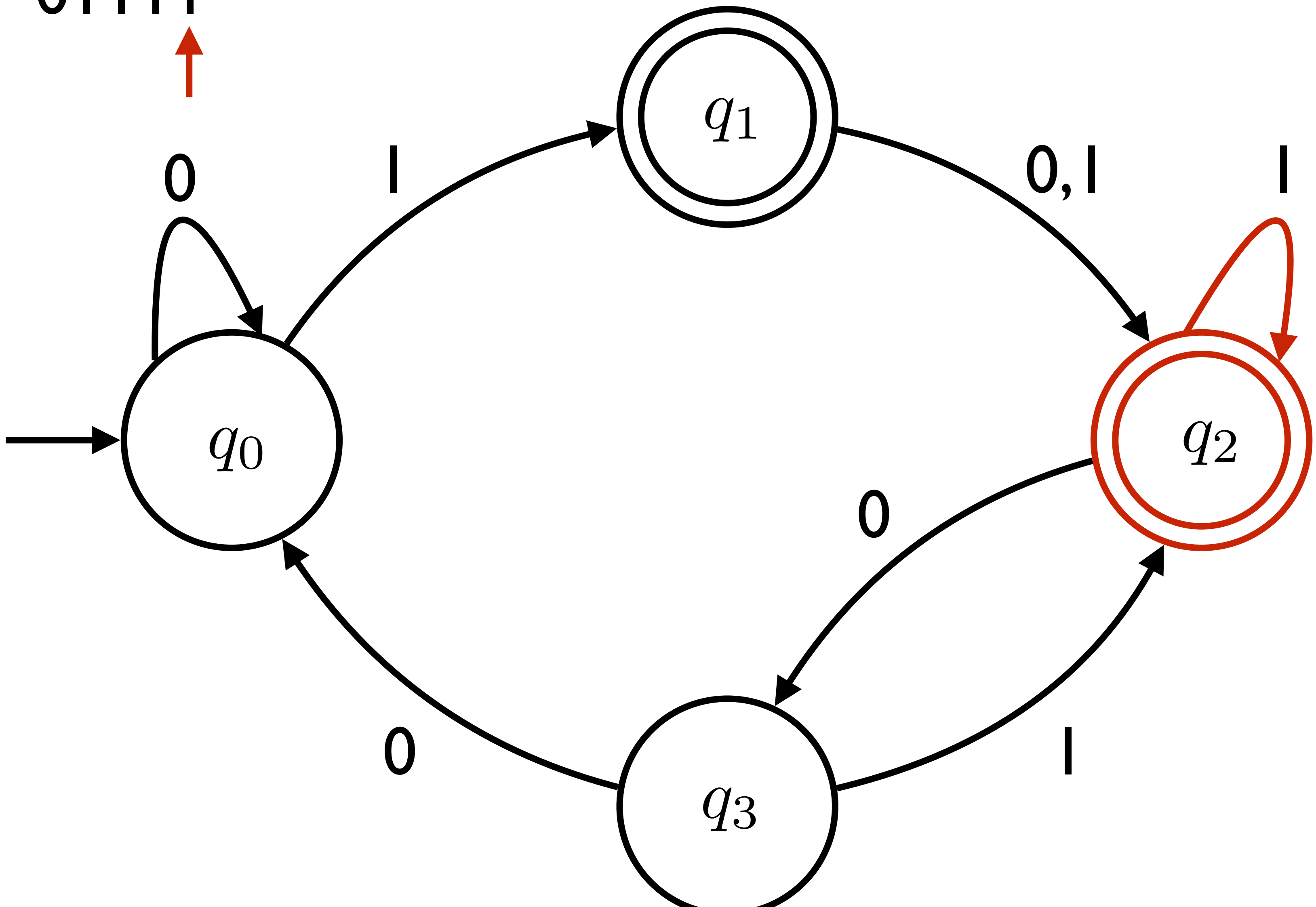


Simulation of a DFA

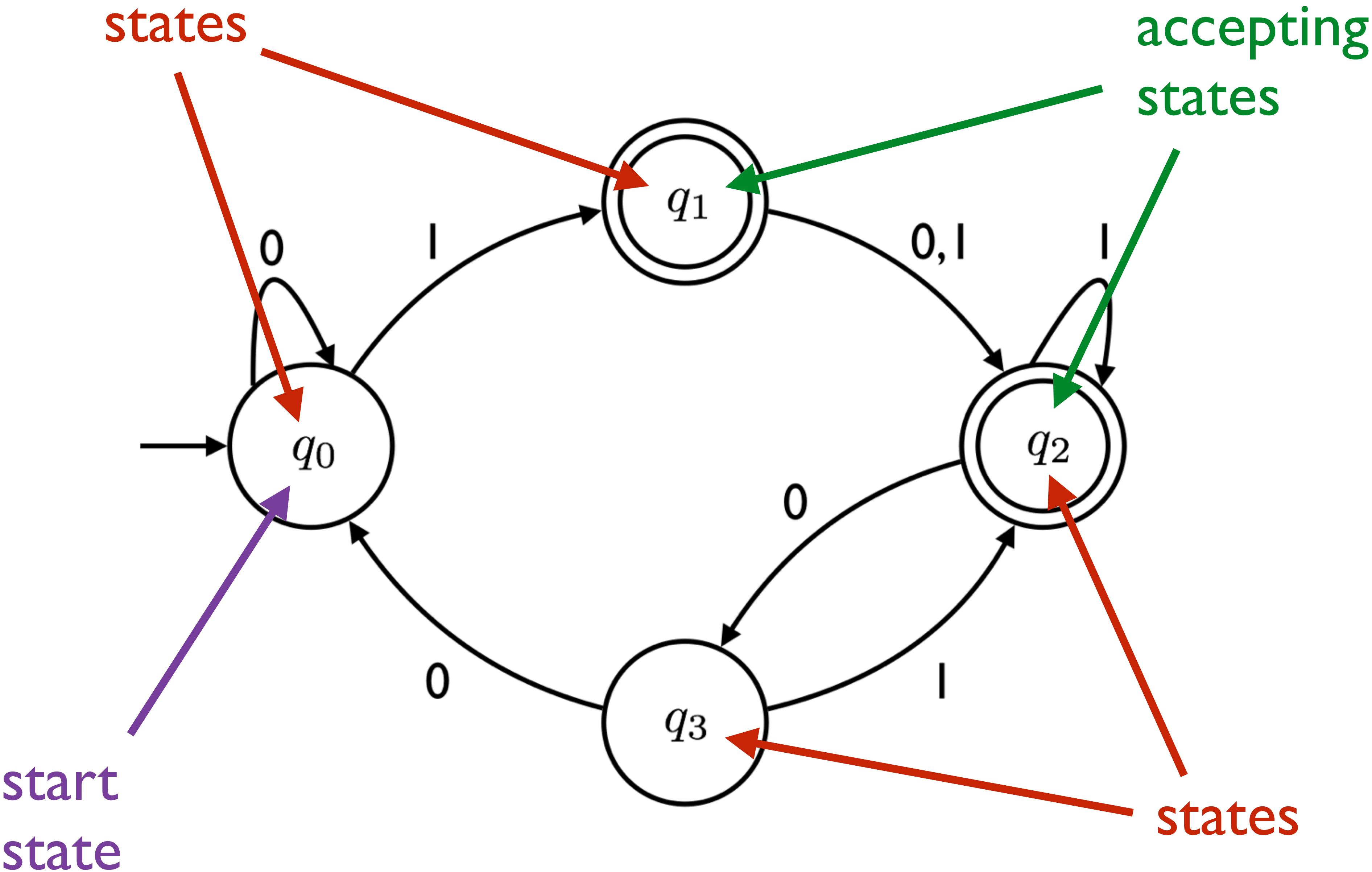
$\Sigma = \{0, 1\}$

Input: 01111

Decision: Accept



Anatomy of a DFA



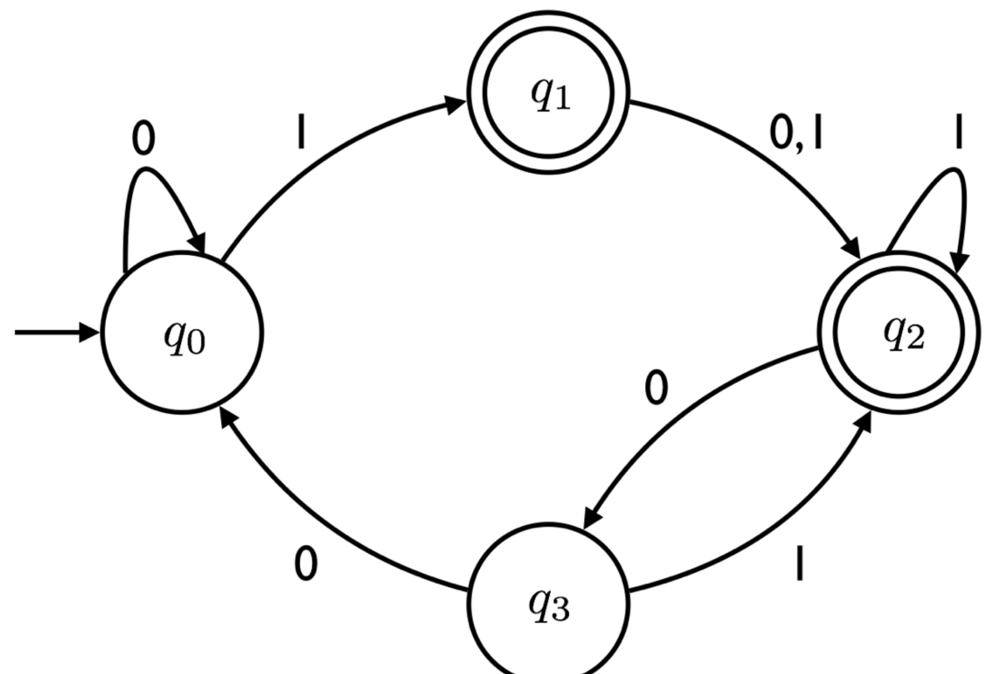
transition rule: labeled arrows

DFA as a programming language

input =

0	1	1	1	1

```
def foo(input):
    i = 0;
    STATE 0:
        if (i == input.length): return False;
        letter = input[i];
        i++;
        switch(letter):
            case ‘0’:  go to STATE 0;
            case ‘1’:  go to STATE 1;

    STATE 1:
        if (i == input.length): return True;
        letter = input[i];
        i++;
        switch(letter):
            case ‘0’:  go to STATE 2;
            case ‘1’:  go to STATE 2;
    ...
```



DFA as a programming language

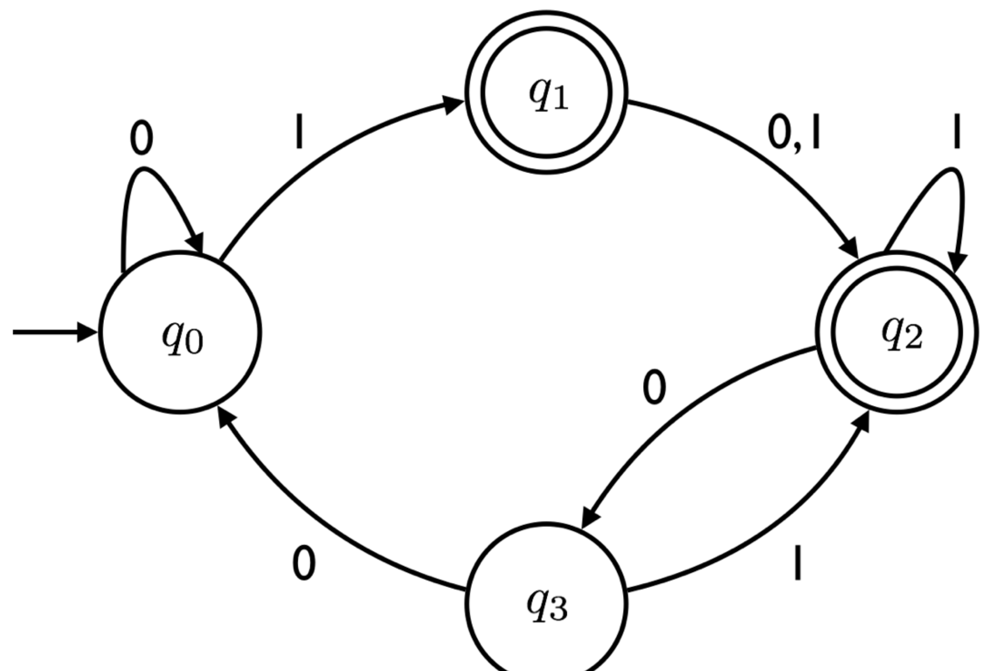
```
def foo(input):
    i = 0;
    STATE 0:
        if (i == input.length): return False;
        letter = input[i];
        i++;
        switch(letter):
            case '0':  go to STATE 0;
            case '1':  go to STATE 1;

    STATE 1:
        if (i == input.length): return True;
        letter = input[i];
        i++;
        switch(letter):
            case '0':  go to STATE 2;
            case '1':  go to STATE 2;
    ...
```

input =

0	1	1	1	1

Have we reached end of input?

Is it an accepting state?



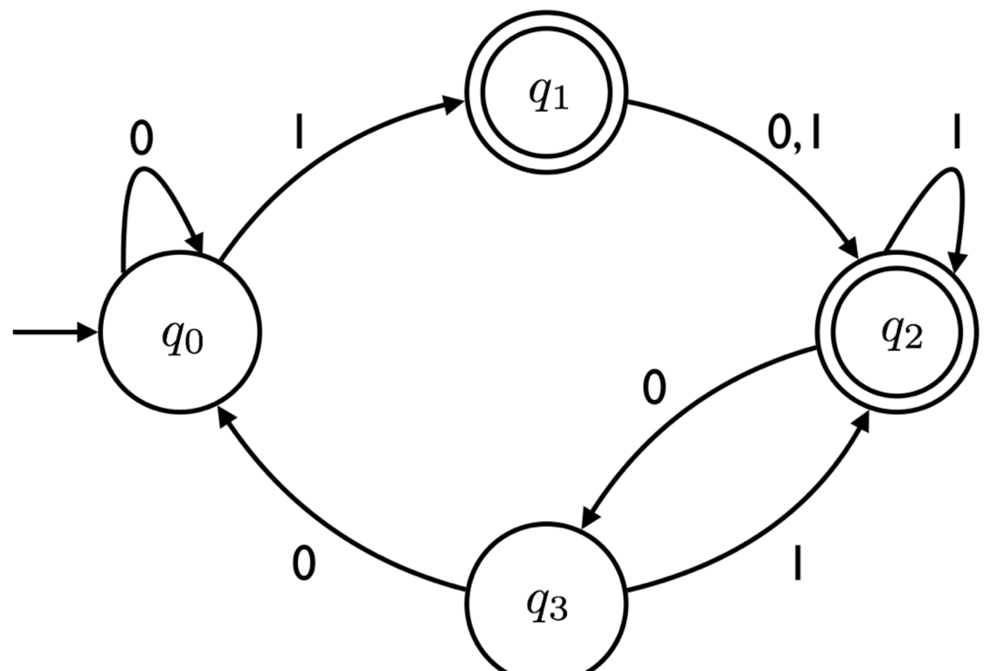
DFA as a programming language

```
def foo(input):
  i = 0;
  STATE 0:
    if (i == input.length): return False;
    letter = input[i];
    i++;
    switch(letter):
      case '0':  go to STATE 0;
      case '1':  go to STATE 1;

  STATE 1:
    if (i == input.length): return True;
    letter = input[i];
    i++;
    switch(letter):
      case '0':  go to STATE 2;
      case '1':  go to STATE 2;
  ...
```

input =

0	1	1	1	1

Read current letter.



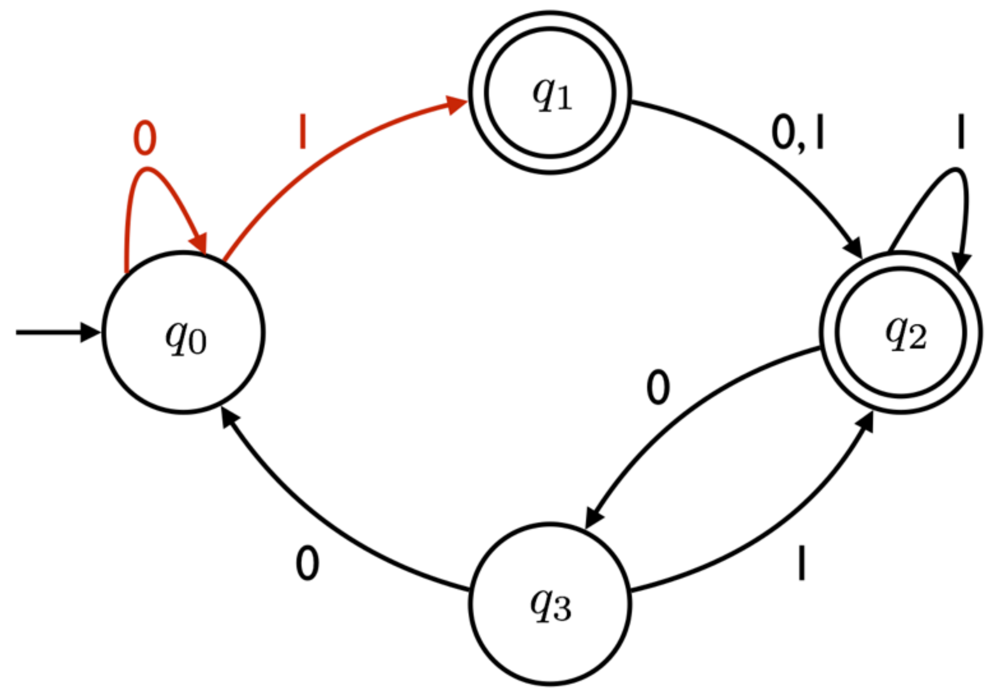
DFA as a programming language

```
def foo(input):
    i = 0;
    STATE 0:
        if (i == input.length): return False;
        letter = input[i];
        i++;
        switch(letter):
            case '0':  go to STATE 0;
            case '1':  go to STATE 1;

    STATE 1:
        if (i == input.length): return True;
        letter = input[i];
        i++;
        switch(letter):
            case '0':  go to STATE 2;
            case '1':  go to STATE 2;

    ...
```

input =

0	1	1	1	1

Depending on the letter change the state.

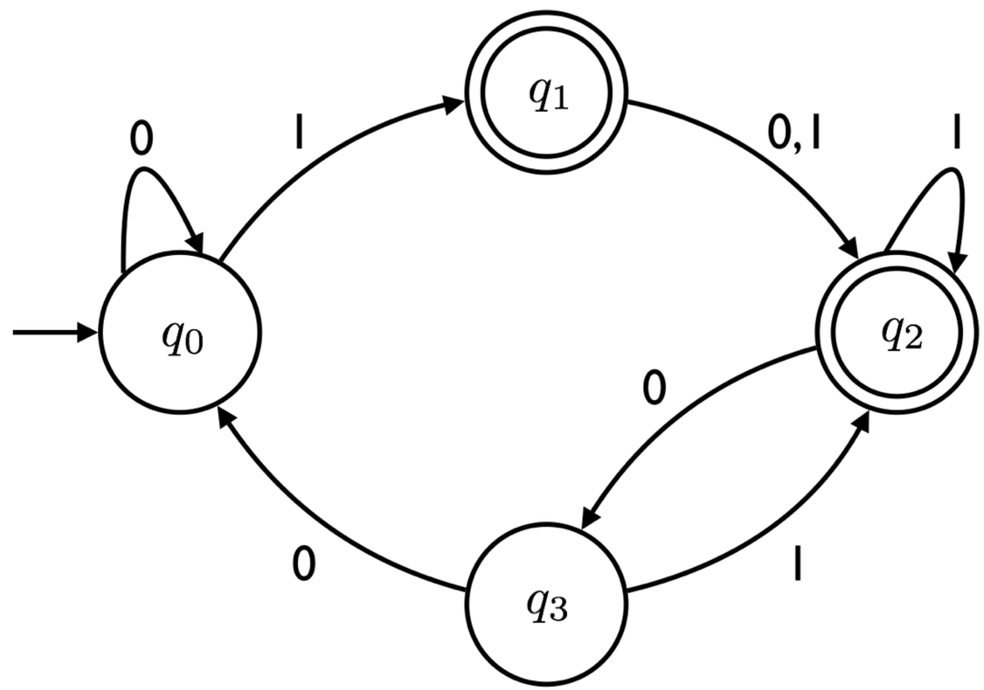


DFA as a programming language

```
def foo(input):
    i = 0;
    STATE 0:
        if (i == input.length): return False;
        letter = input[i];
        i++;
        switch(letter):
           case ‘0’:  go to STATE 0;
           case ‘1’:  go to STATE 1;

    STATE 1:
        if (i == input.length): return True;
        letter = input[i];
        i++;
        switch(letter):
           case ‘0’:  go to STATE 2;
           case ‘1’:  go to STATE 2;
     …
```

input =

0	1	1	1	1



Definition: Language decided by a DFA

Let M be a DFA.

We let $L(M)$ denote the set of strings that M **accepts**.

So, $L(M) = \{x \in \Sigma^* : M(x) \text{ accepts.}\} \subseteq \Sigma^*$

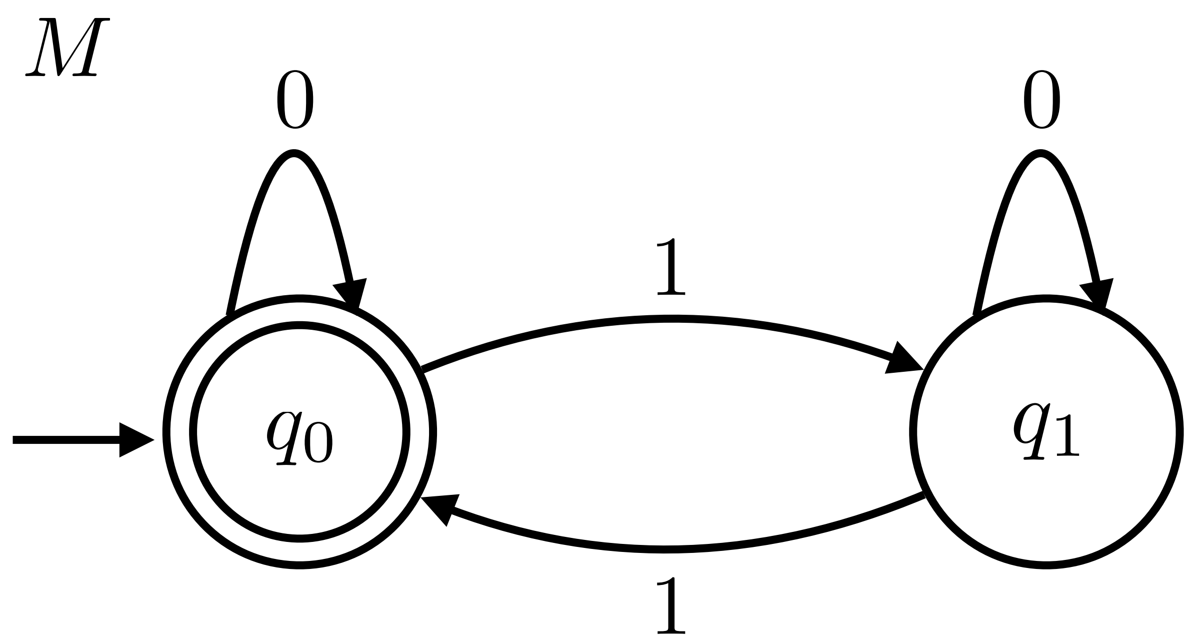
If $L = L(M)$, we say that M *recognizes* L.

accepts

decides

computes

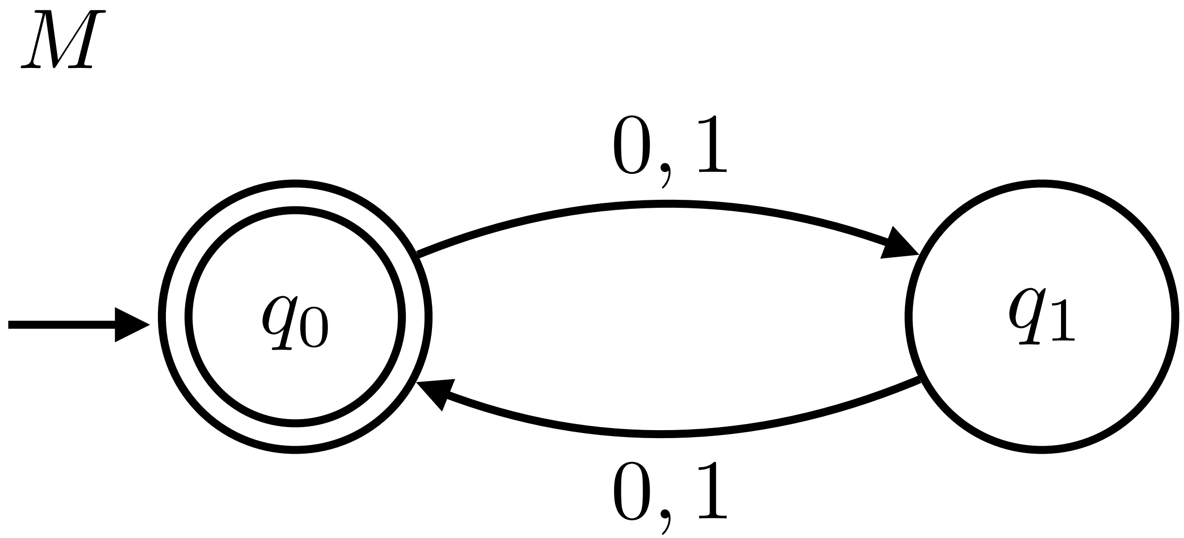
DFA Examples



$L(M) =$ **all binary strings with an even number of 1's**

$= \{x \in \{0,1\}^* : x \text{ has an even number of 1's}\}$

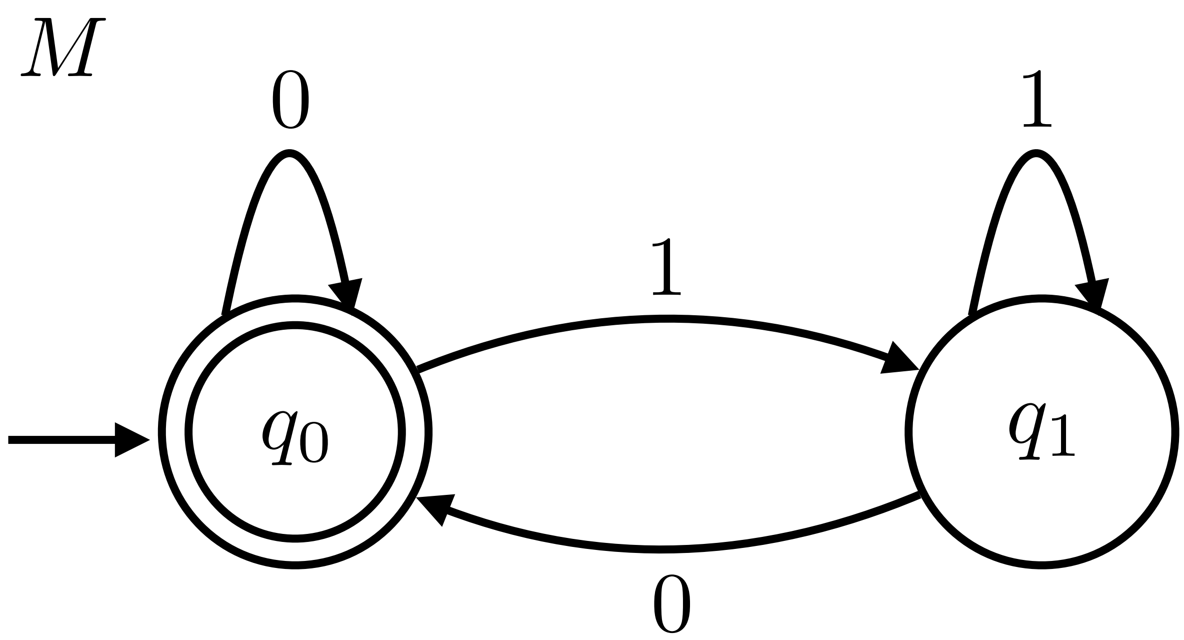
DFA Examples



$L(M) =$ all binary strings with even length

$= \{x \in \{0,1\}^* : |x| \text{ is even}\}$

DFA Examples

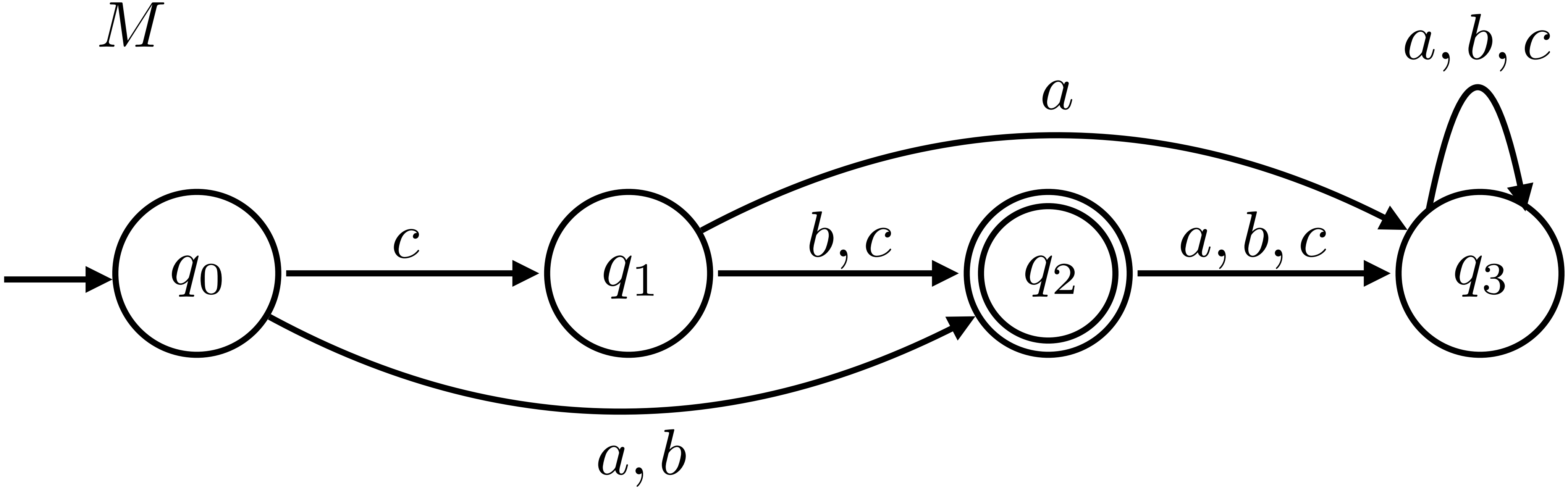


$$L(M) = \{x \in \{0,1\}^* : x \text{ ends with a } 0\} \cup \{\epsilon\}$$

DFA Examples

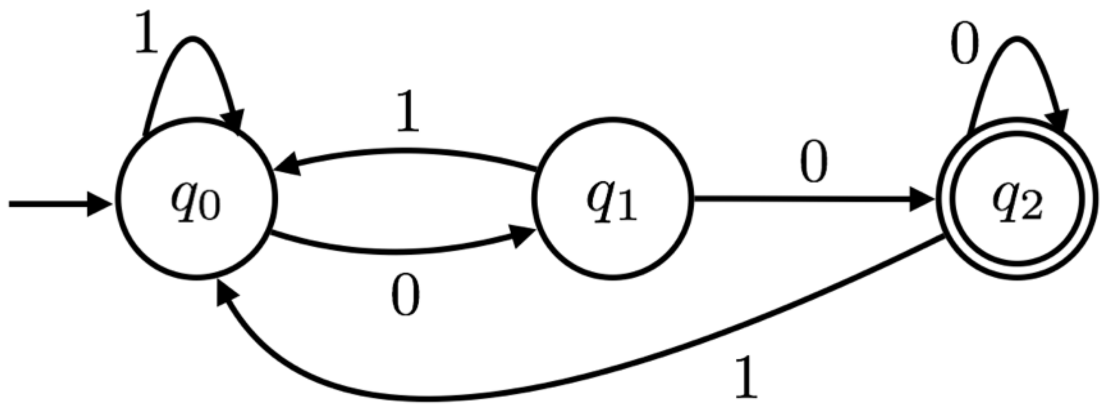
$$\Sigma = \{a, b, c\}$$

M



$$L(M) = \{a, b, cb, cc\}$$

Poll



The set of all words that contain at least three 0's

The set of all words that contain at least two 0's

The set of all words that contain 000 as a substring

The set of all words that contain 00 as a substring

The set of all words ending in 000

The set of all words ending in 00

The set of all words ending in 0

None of the above

Beats me

DFA construction practice

$L = \{110, 101\}$

$L = \{0,1\}^* \setminus \{110, 101\}$

$L = \{x \in \{0,1\}^* : x \text{ starts and ends with same bit.}\}$

$L = \{x \in \{0,1\}^* : |x| \text{ is divisible by 2 or 3.}\}$

$L = \{\epsilon, 110, 110110, 110110110, \ldots\}$

$L = \{x \in \{0,1\}^* : x \text{ contains the substring 110.}\}$

$L = \{x \in \{0,1\}^* : \text{10 and 01 occur equally often in } x.\}$

Formal definition: DFA

A deterministic finite automaton (DFA) M is a 5-tuple

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

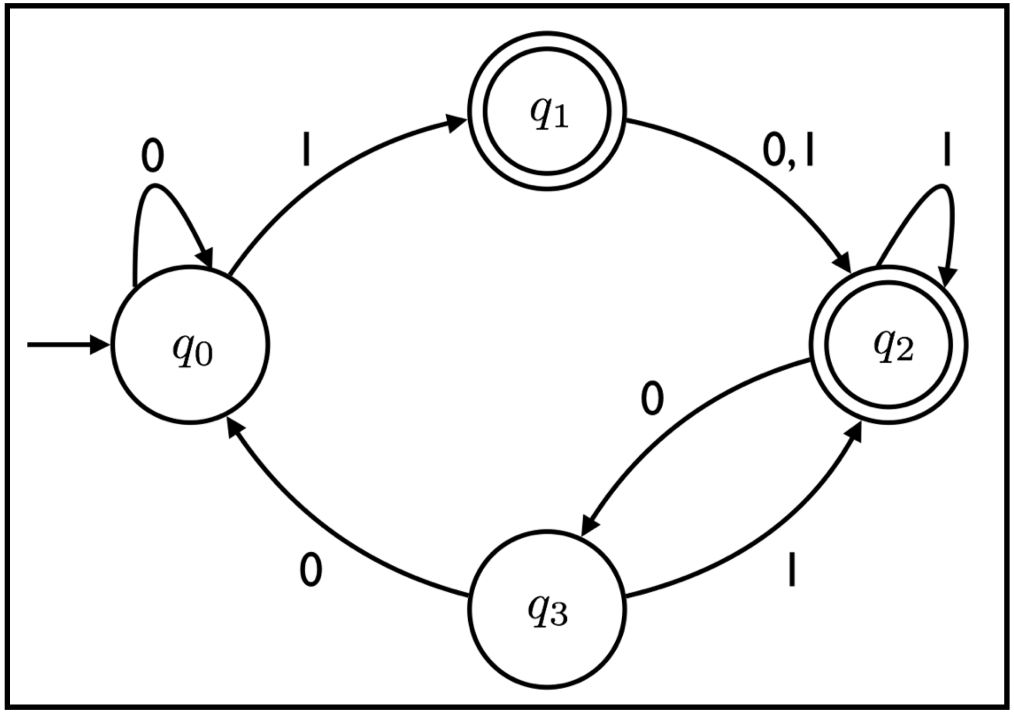
where

- Q is a finite, non-empty set (which we call the set of states);
- Σ is a finite, non-empty set (which we call the alphabet);
- δ is a function of the form $\delta : Q \times \Sigma \to Q$ (which we call the transition function);
- $q_0 \in Q$ is an element of Q (which we call the start state);
- $F \subseteq Q$ is a subset of Q (which we call the set of accepting states).

Formal definition: DFA

A deterministic finite automaton (DFA) M is a 5-tuple

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



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

$$\Sigma = \{0, 1\}$$

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

δ	0	1
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_3	q_2
q_3	q_0	q_2

q_0 is the start state

$$F = \{q_1, q_2\}$$

Formal definition: DFA accepting a string

Let $w = w_1 w_2 \cdots w_n$ be a string over an alphabet Σ.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA.

We say that M ***accepts*** the string w
if there exists a sequence of states $r_0, r_1, \ldots, r_n \in Q$
such that

- $r_0 = q_0$;
- $\delta(r_{i-1}, w_i) = r_i$ for each $i \in \{1, 2, \ldots, n\}$;
- $r_n \in F$.

Otherwise we say M ***rejects*** the string w .

Formal definition: DFA accepting a string

Simplifying notation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA.

$\delta : Q \times \Sigma \to Q$ can be extended to $\delta^* : Q \times \Sigma^* \to Q$

as follows:

for $q \in Q, w \in \Sigma^*$,

$\delta^*(q, w) =$ state we end up in when we start at q and read w

In fact, even OK to drop $*$ from the notation.

M ***accepts*** w if $\delta(q_0, w) \in F$.

Otherwise M ***rejects*** w.

Definition: Regular languages

Definition: A language L is called ***regular*** if $L = L(M)$ for some DFA M.

Regular languages

All languages
$\mathcal{P}(\Sigma^*)$

Regular languages

$\{110, 101\}$

$\{0,1\}^* \backslash \{110, 101\}$

$\{x \in \{0,1\}^* : x \text{ starts and ends with same bit.}\}$

$\{x \in \{0,1\}^* : |x| \text{ is divisible by 2 or 3.}\}$

$\{\epsilon, 110, 110110, 110110110, \ldots\}$

$\{x \in \{0,1\}^* : x \text{ contains the substring 110.}\}$

$\{x \in \{0,1\}^* : \text{10 and 01 occur equally often in } x.\}$

$\vdots$

?

Regular languages

Questions:

1. Are all languages regular?

   (Are all decision problems computable by a DFA?)

2. Are there other ways to tell if a language is regular?

A non-regular language

Theorem:

The language $L = \{0^n 1^n : n \in \mathbb{N}\}$ is **not** regular.

Note on notation:

For $a \in \Sigma$, a^n denotes the string $\underbrace{aa \cdots a}_{\text{n times}}$.

$$a^0 = \epsilon$$

For $u, v \in \Sigma^*$, uv denotes u concatenated with v.

So $L = \{\epsilon, 01, 0011, 000111, 00001111, \ldots\}$.

A non-regular language

Theorem:

The language $L = \{0^n 1^n : n \in \mathbb{N}\}$ is **not** regular.

Intuition:

Seems like the DFA would need to remember how many 0's it sees.

But it has a constant number of states.
And no other way of remembering things.

Careful though:

$L = \{x \in \{0,1\}^* : 10 \text{ and } 01 \text{ occur equally often in } x.\}$ is regular!

A non-regular language

Theorem:

The language $L = \{0^n 1^n : n \in \mathbb{N}\}$ is **not** regular.

A key component of the proof:

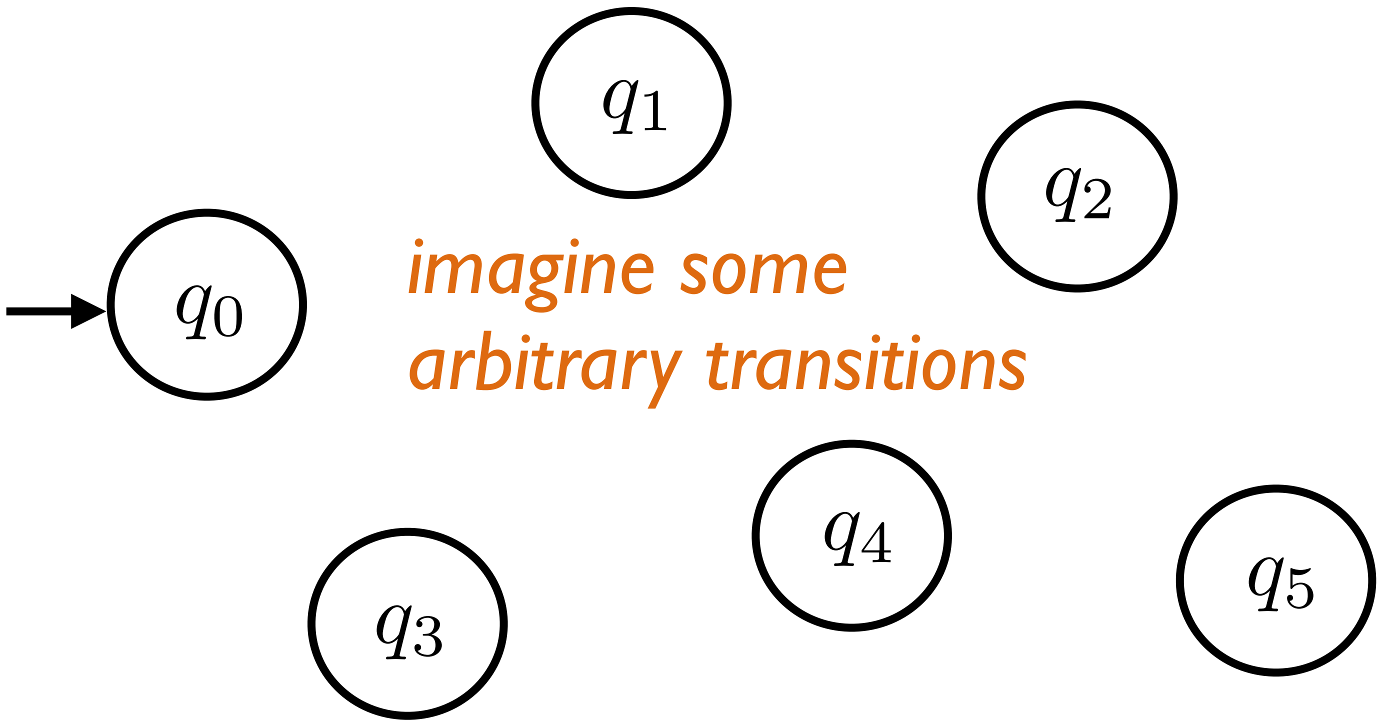
Pigeonhole principle (PHP)

A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

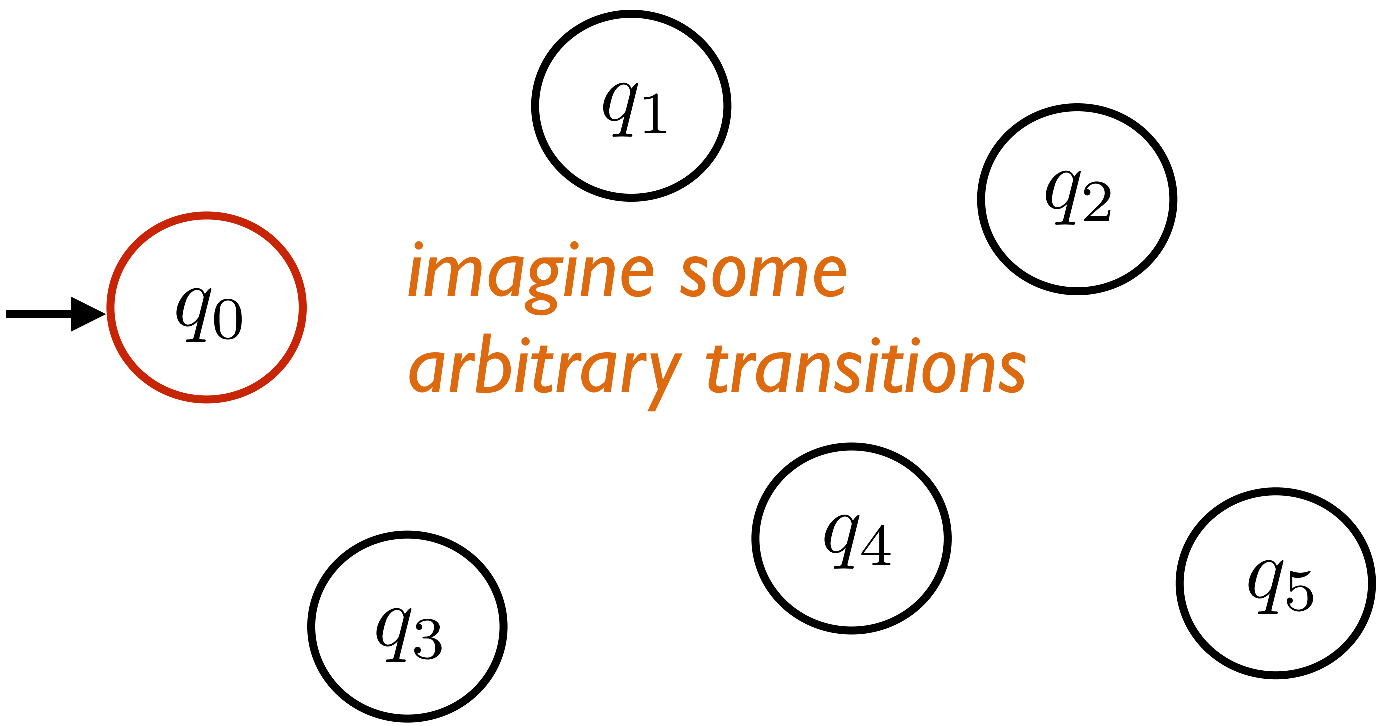


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

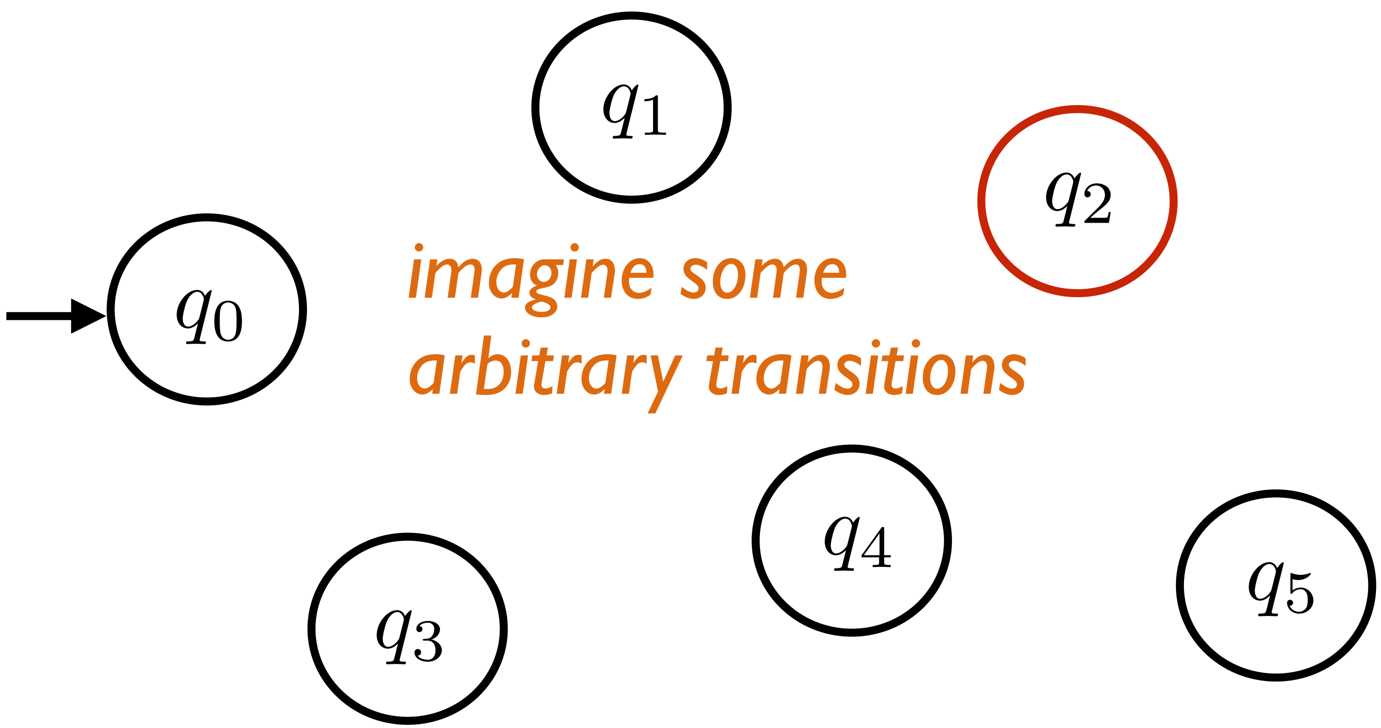


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

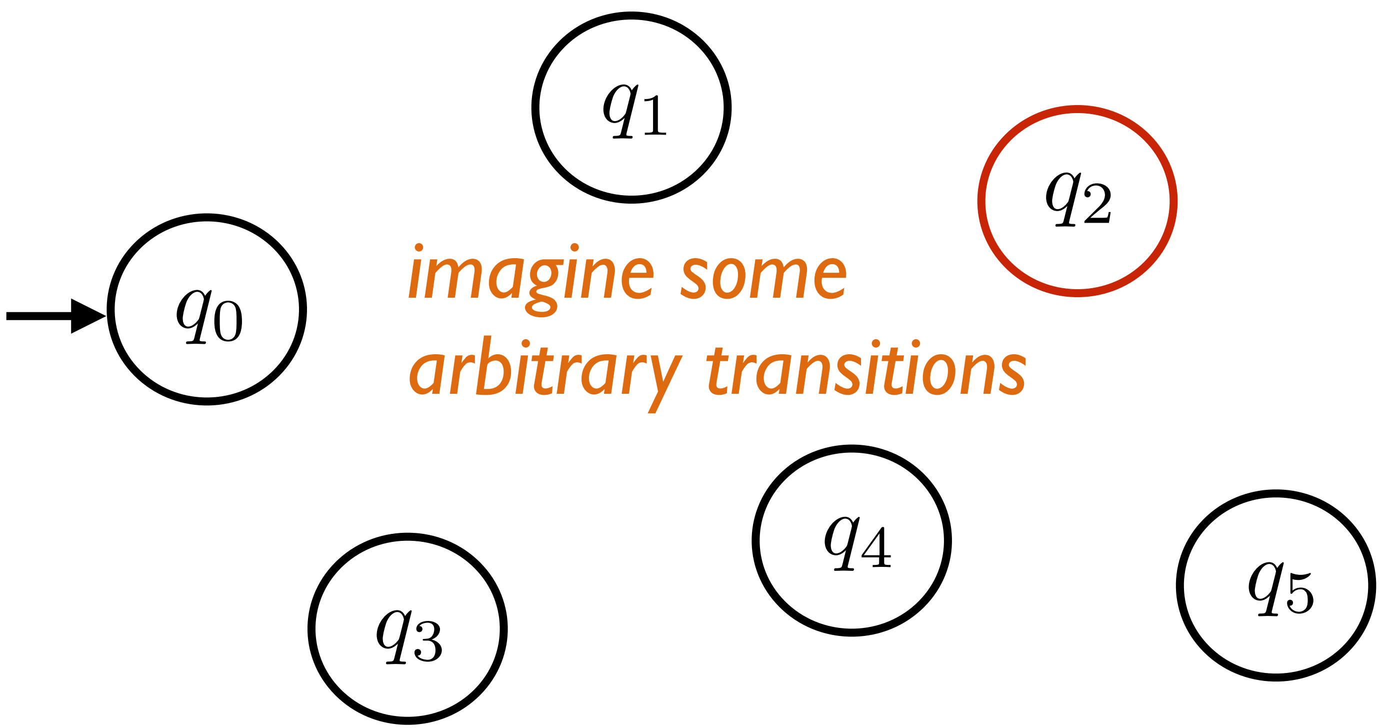


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

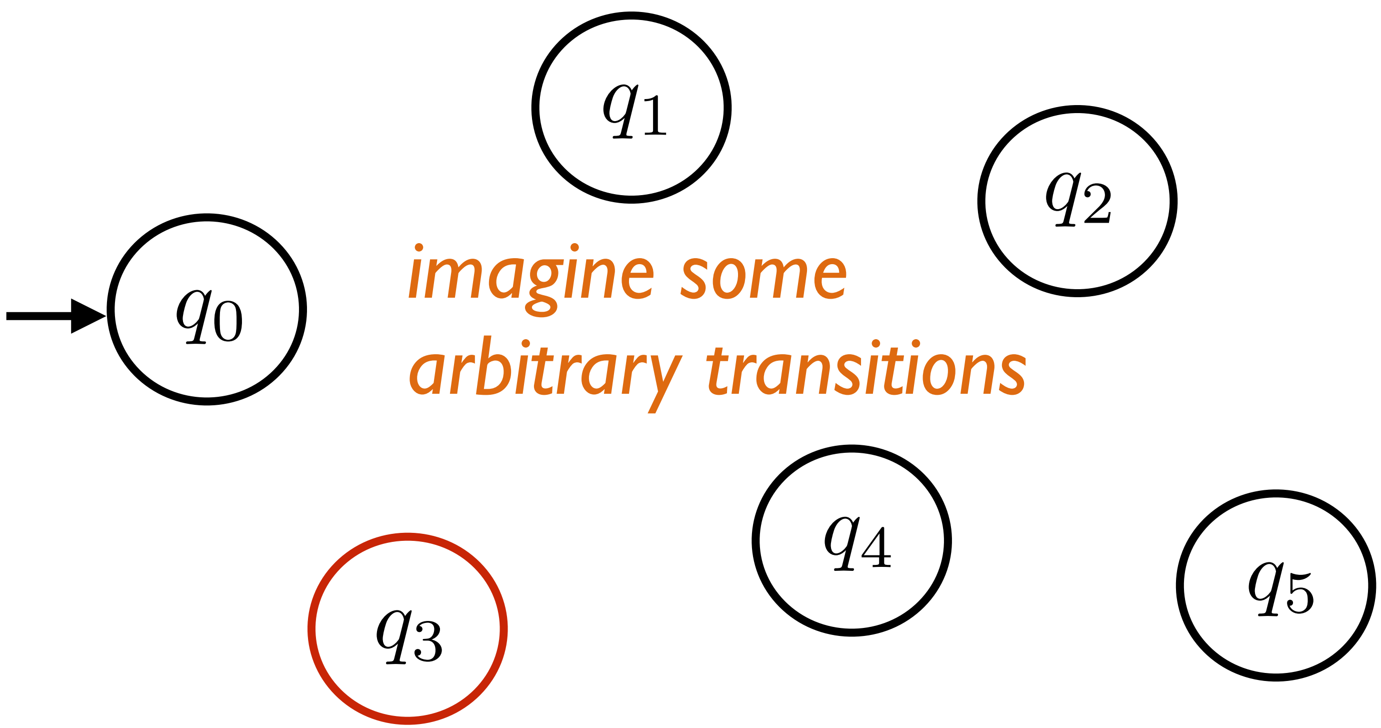


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

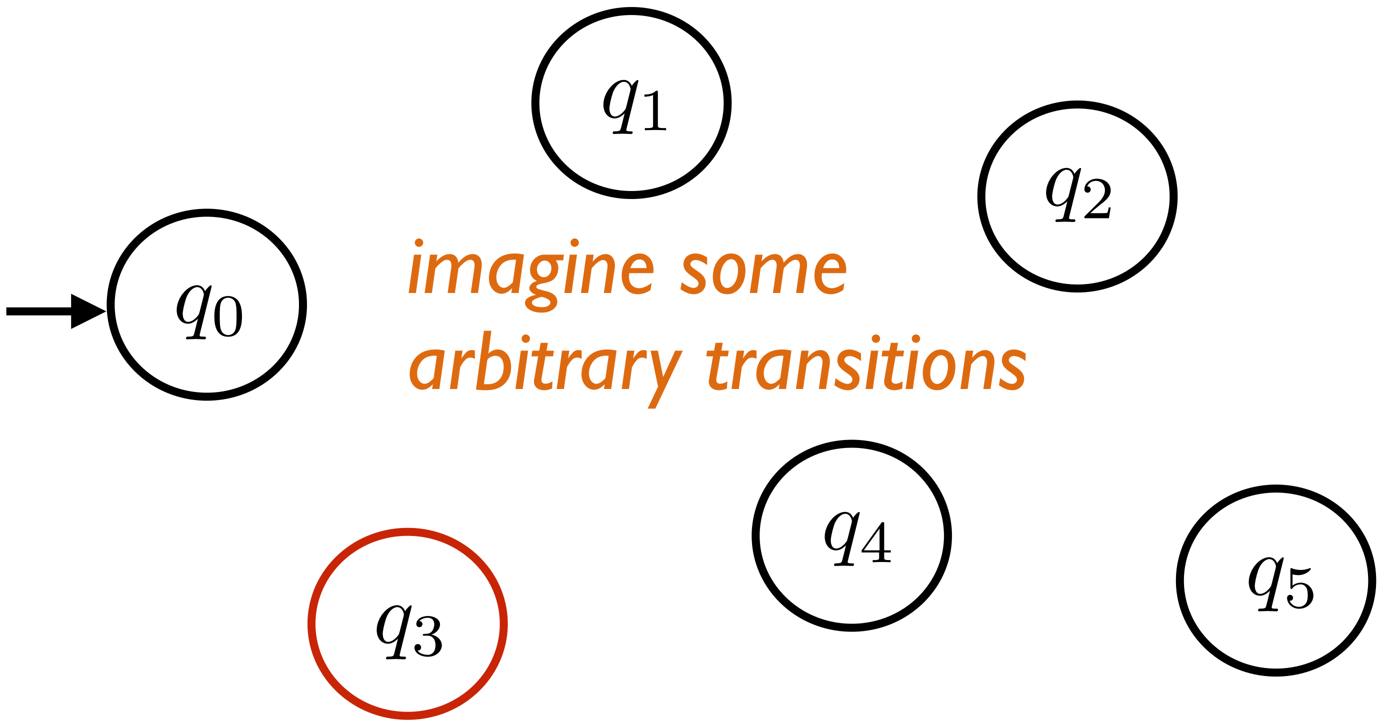


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

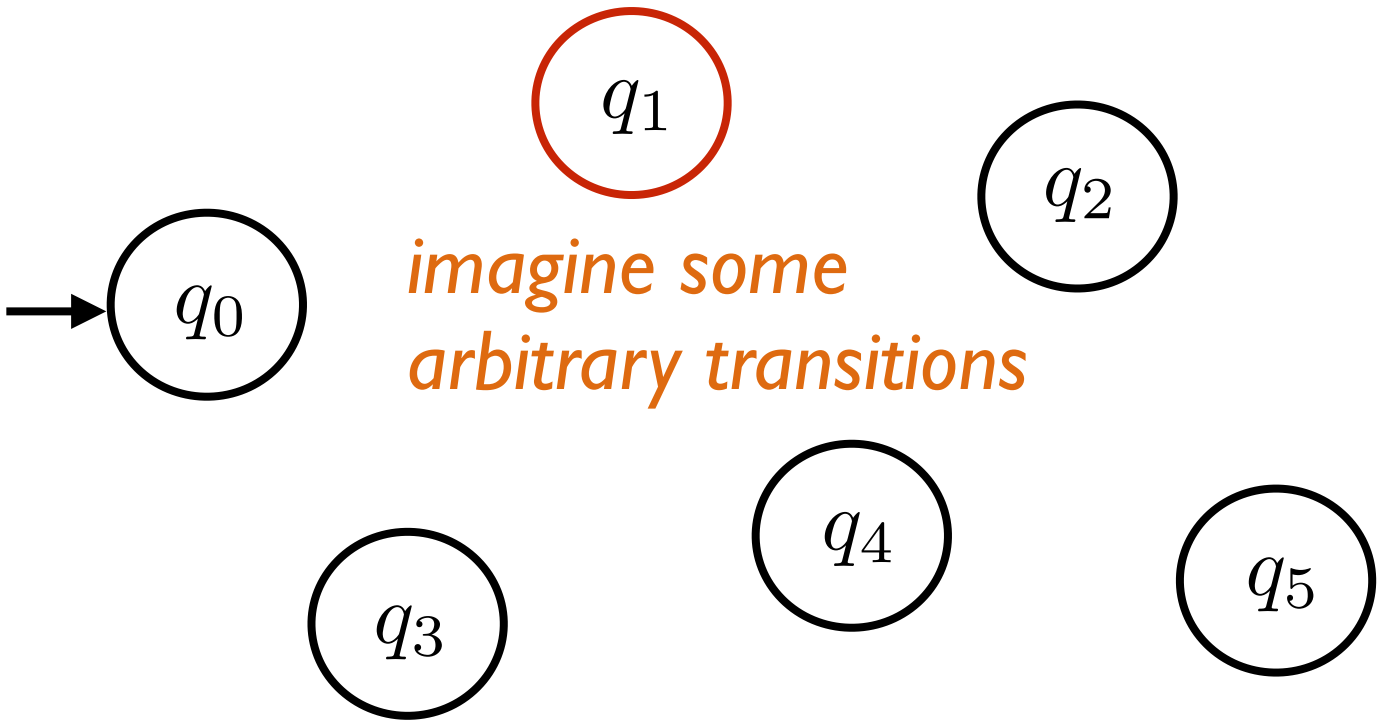


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

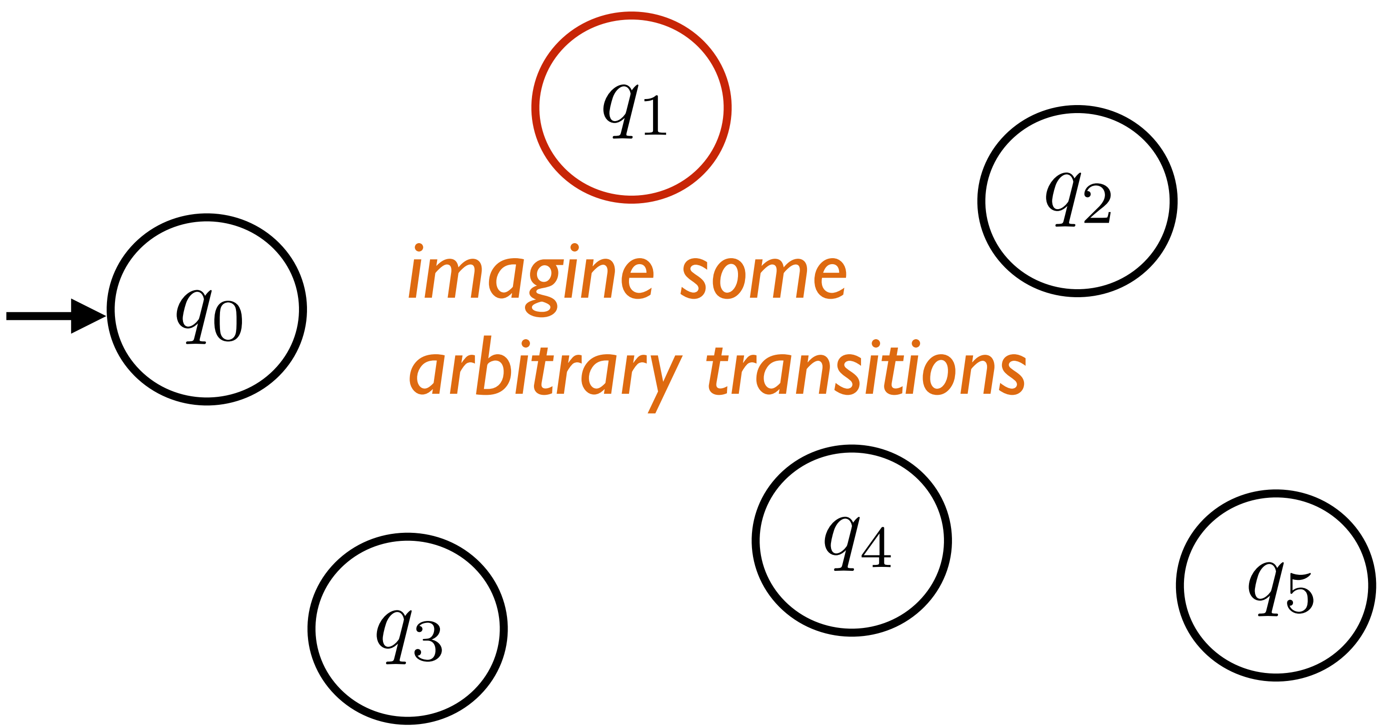


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

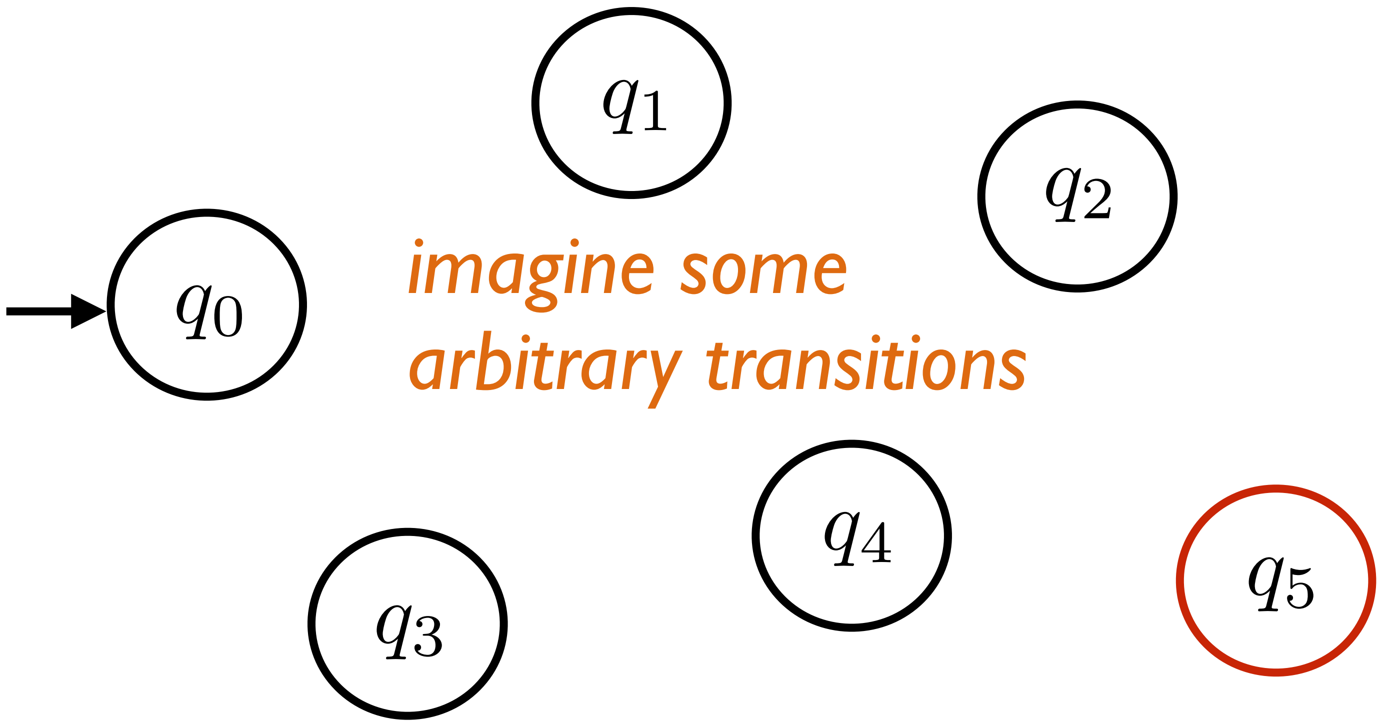


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 00000000011111111

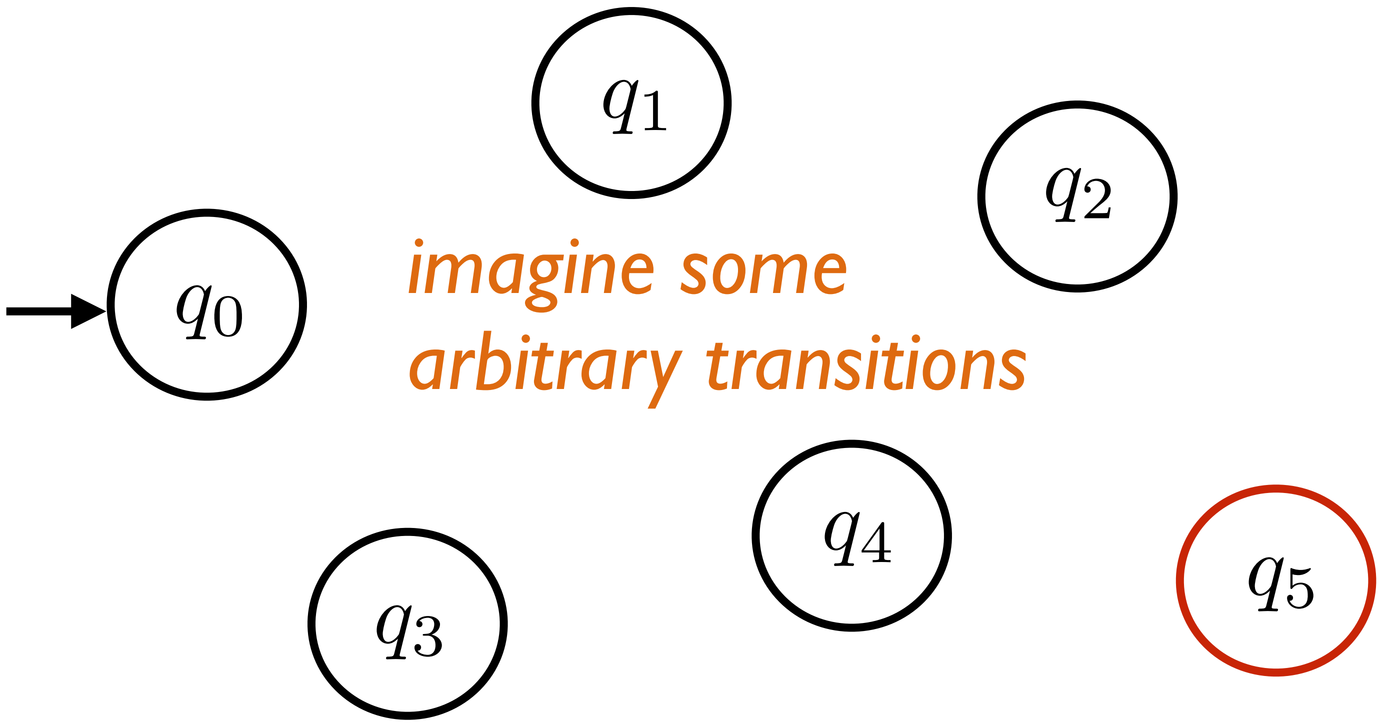


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

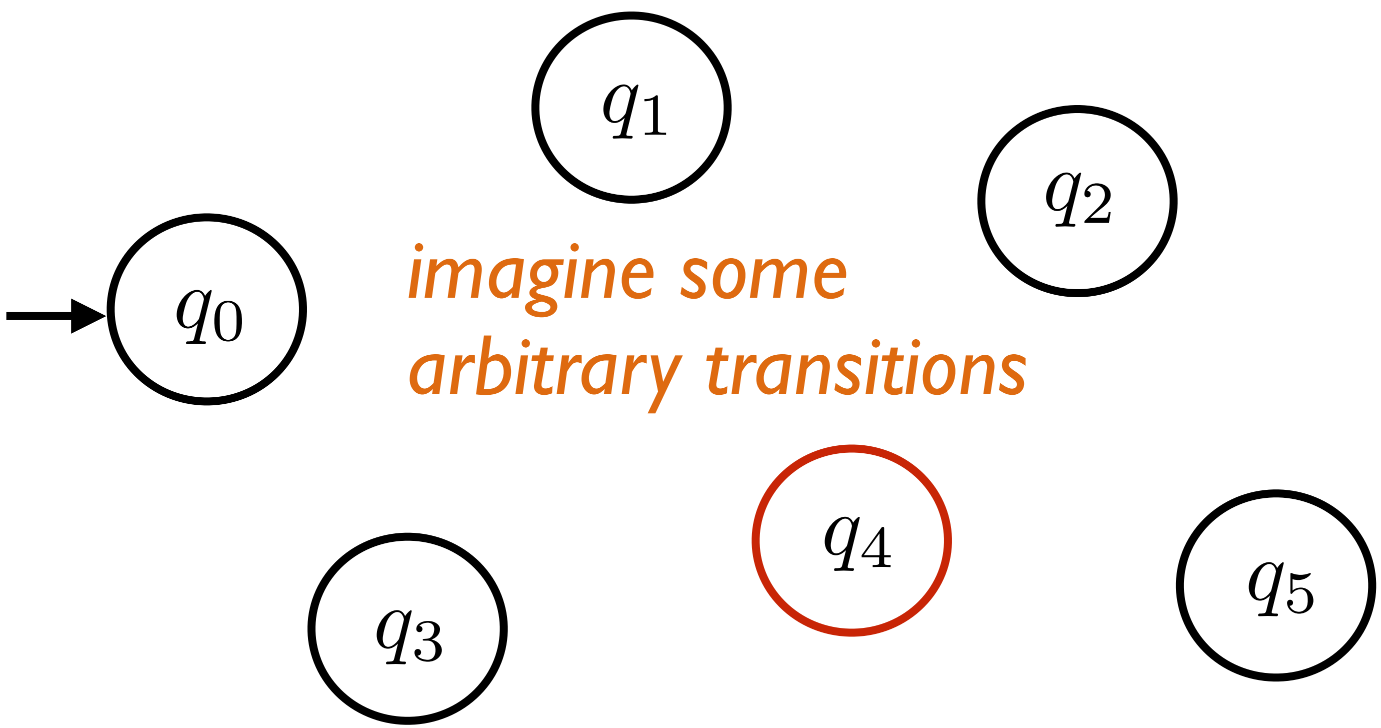


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

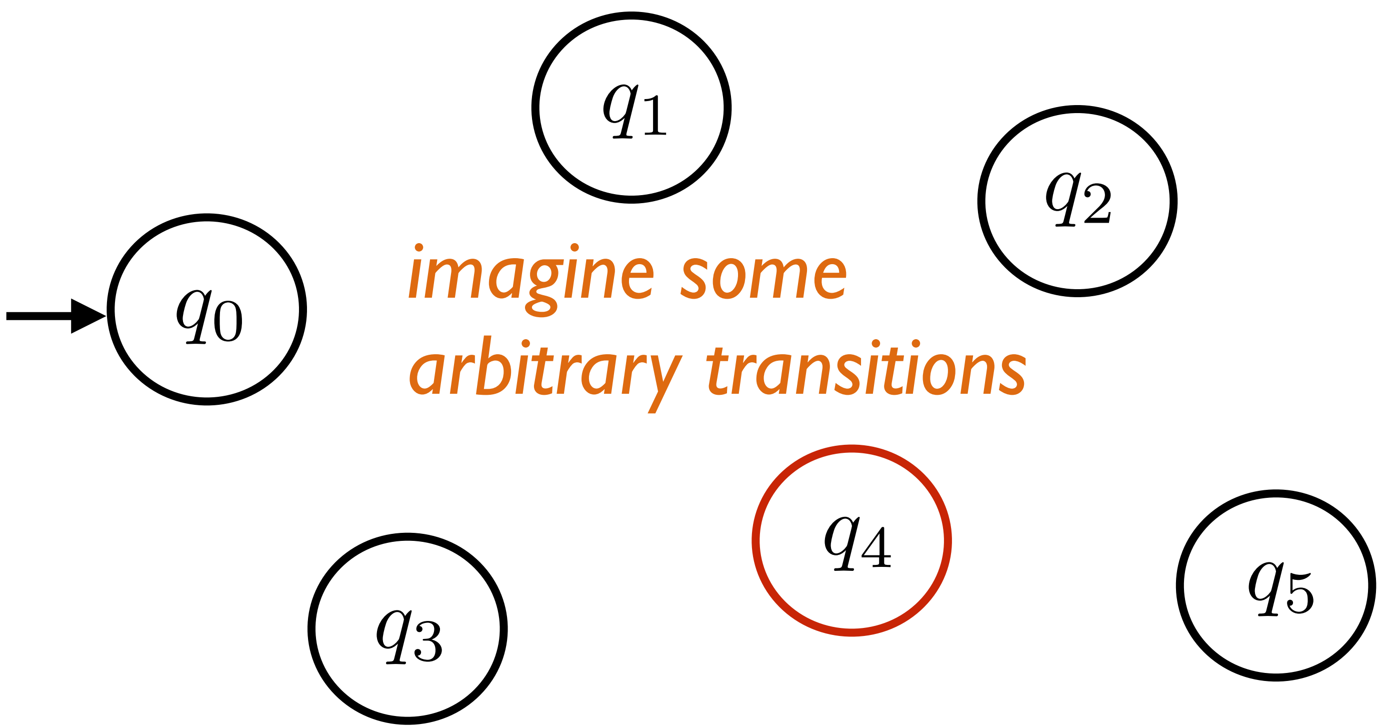


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

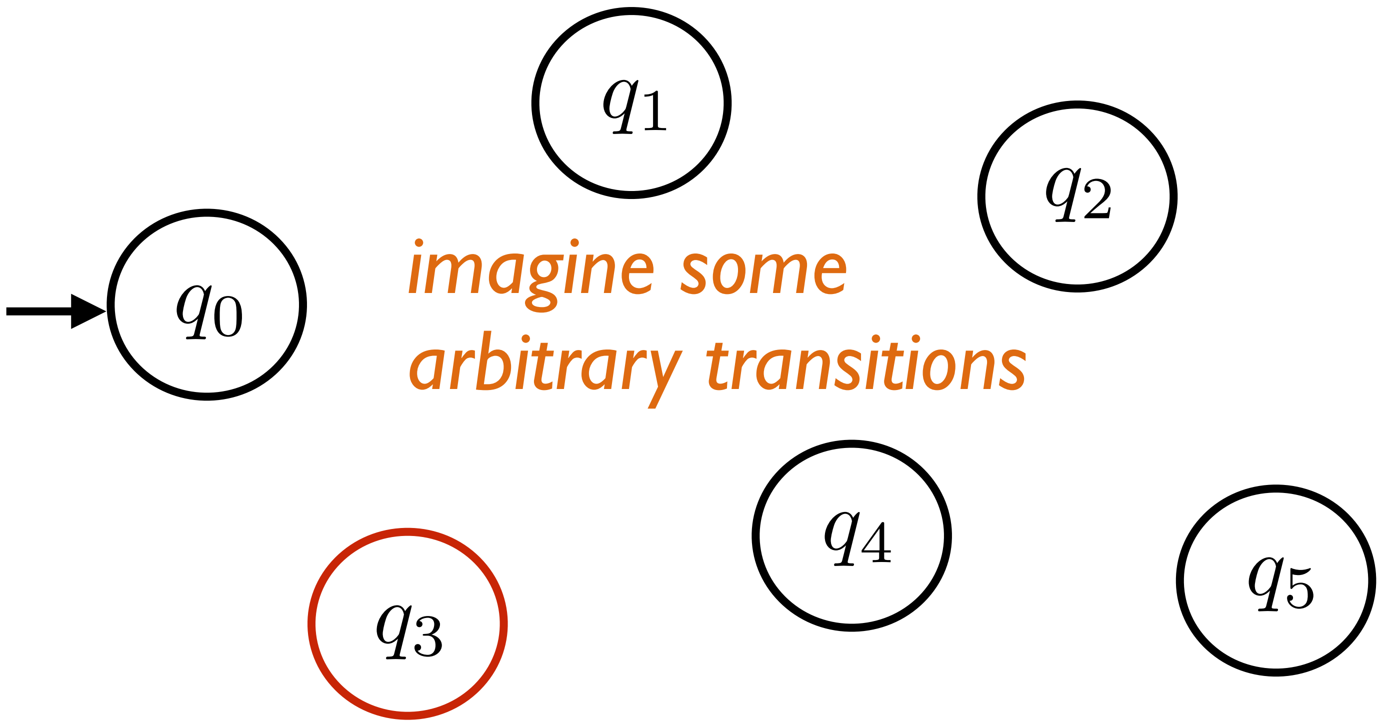


A non-regular language

Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

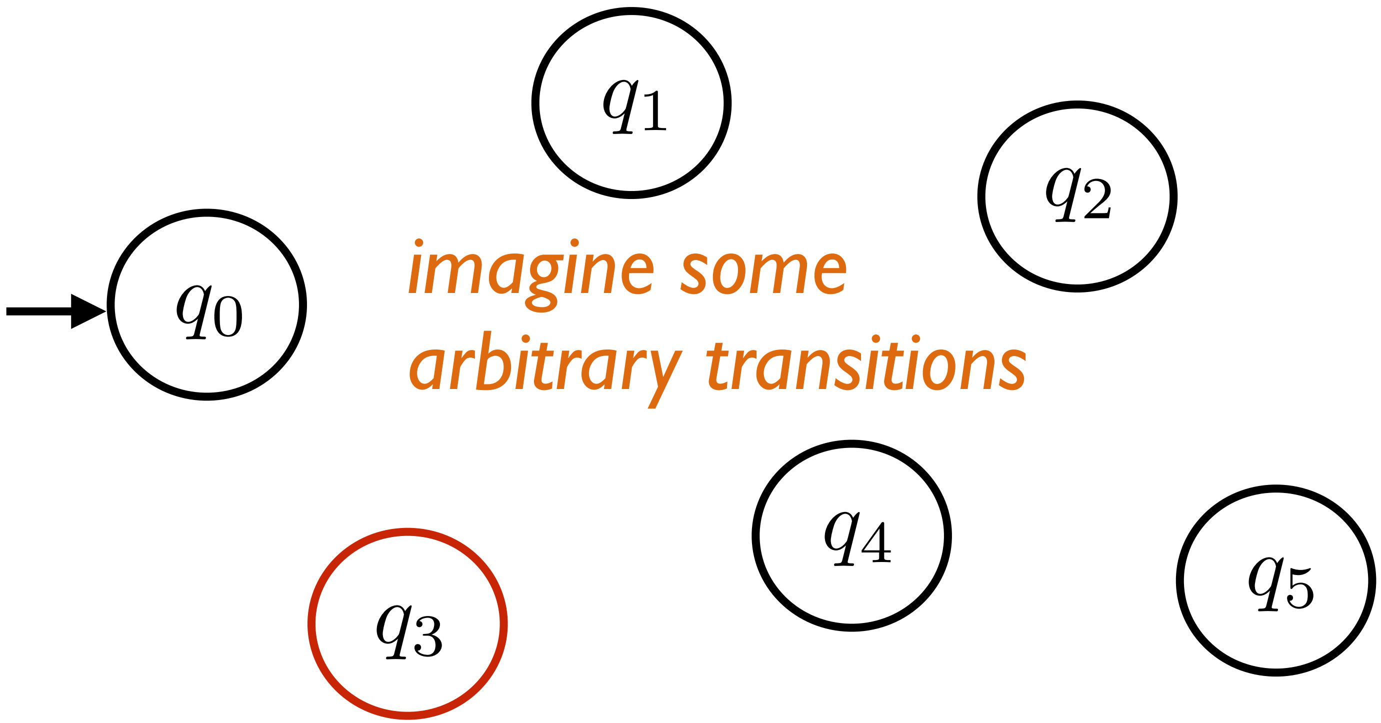
Input: 0000000011111111



A non-regular language

Warm-up:
Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111



A non-regular language

Warm-up:

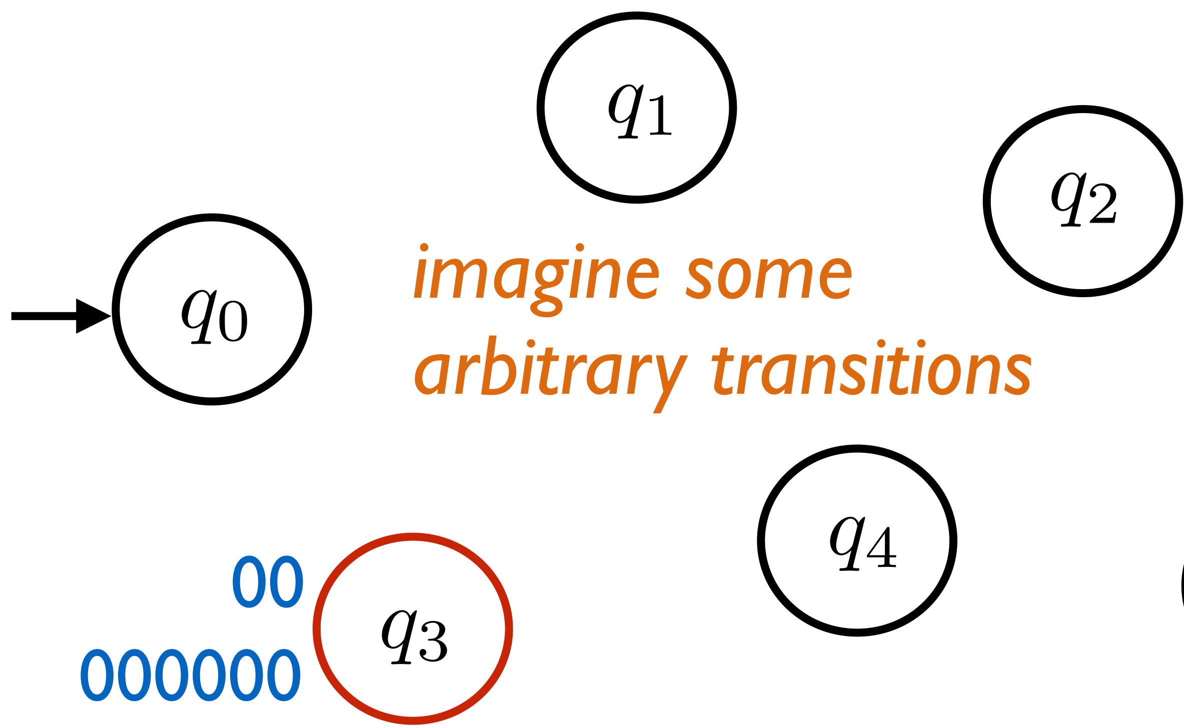
Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

After 00 and 000000 we ended up in the same state q_3.

$\Longrightarrow$

0011 and 00000011 end up in the same state.



But

0011 $\longrightarrow$ accept

00000011 $\longrightarrow$ reject

A non-regular language

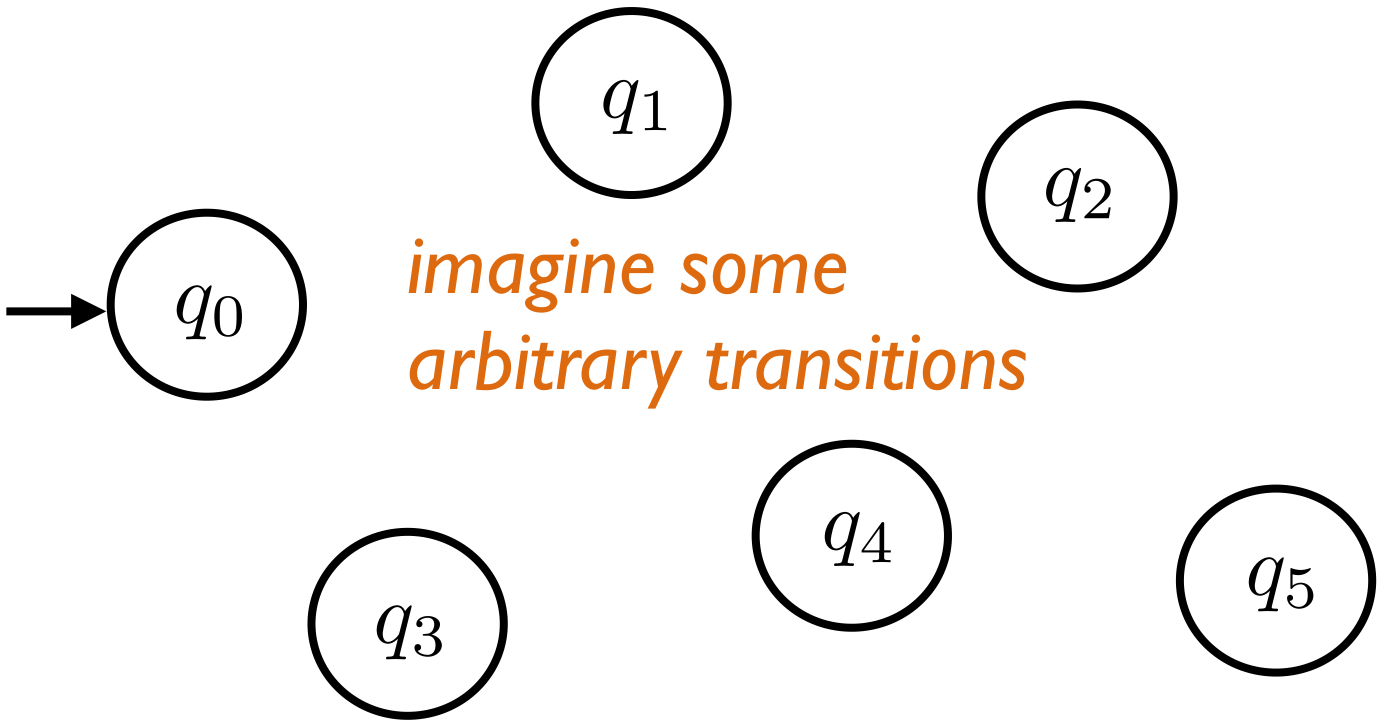
Warm-up:

Suppose a DFA with 6 states decides $L = \{0^n 1^n : n \in \mathbb{N}\}$.

Input: 0000000011111111

Pigeonhole Principle

Where will 0000000 go?



0
00
000
0000
00000
000000

A non-regular language

Theorem:

The language $L = \{0^n 1^n : n \in \mathbb{N}\}$ is **not** regular.

Proof: Proof is by contradiction. So suppose L is regular.
This means there is a DFA M that decides L.
Let k denote the number of states of M.
Let r_n denote the state M is in after reading 0^n.
By PHP, there exists $i, j \in \{0, 1, \ldots, k\}$, $i \neq j$, such that $r_i = r_j$. So 0^i and 0^j end up in the same state.
For any string w, $0^i w$ and $0^j w$ end up in the same state.
But for $w = 1^i$, $0^i w$ should end up in an accepting state,
and $0^j w$ should end up in a rejecting state.
This is the desired contradiction. $\square$

Proving a language is not regular

Usually the proof goes like this:

1. Assume (to reach a contradiction) that the language is regular. So a DFA decides it.

2. Argue by PHP that there are two strings x and y that lead to the same state in the DFA.

   (For any string z, xz and yz lead to the same state.)

3. Find a string z such that $xz \in L$ but $yz \notin L$.

Proving a language is not regular

Exercise (test your understanding):

Show that the following language is not regular:

$$L = \{c^{251}a^n b^{2n} : n \in \mathbb{N}\}.$$

$(\Sigma = \{a, b, c\})$

Regular languages

All languages
$\mathcal{P}(\Sigma^*)$

Regular languages

$\{110, 101\}$

$\{0,1\}^* \backslash \{110, 101\}$

$\{x \in \{0,1\}^* : x \text{ starts and ends with same bit.}\}$

$\{x \in \{0,1\}^* : |x| \text{ is divisible by 2 or 3.}\}$

$\{\epsilon, 110, 110110, 110110110, \ldots\}$

$\{x \in \{0,1\}^* : x \text{ contains the substring 110.}\}$

$\{x \in \{0,1\}^* : \text{10 and 01 occur equally often in } x.\}$

$\vdots$

?

Regular languages

All languages

$\mathcal{P}(\Sigma^*)$

Regular languages

$\{110, 101\}$

$\{0,1\}^* \backslash \{110, 101\}$

$\{x \in \{0,1\}^* : x \text{ starts and ends with same bit.}\}$

$\{x \in \{0,1\}^* : |x| \text{ is divisible by 2 or 3.}\}$

$\{\epsilon, 110, 110110, 110110110, \ldots\}$

$\{x \in \{0,1\}^* : x \text{ contains the substring 110.}\}$

$\{x \in \{0,1\}^* : \text{10 and 01 occur equally often in } x.\}$

$\vdots$

$\{0^n 1^n : n \in \mathbb{N}\}$

$\vdots$

Another non-regular language?

Question: Are all unary languages regular?

(a language L is unary if $L \subseteq \Sigma^*$, where $|\Sigma| = 1$.)

Theorem:

The language $\{a^{2^n} : n \in \mathbb{N}\}$ is **not** regular.

Regular languages

Questions:

1. Are all languages regular?
(Are all decision problems computable by a DFA?)

2. Are there other ways to tell if a language is regular?

Next Time

Closure properties of regular languages