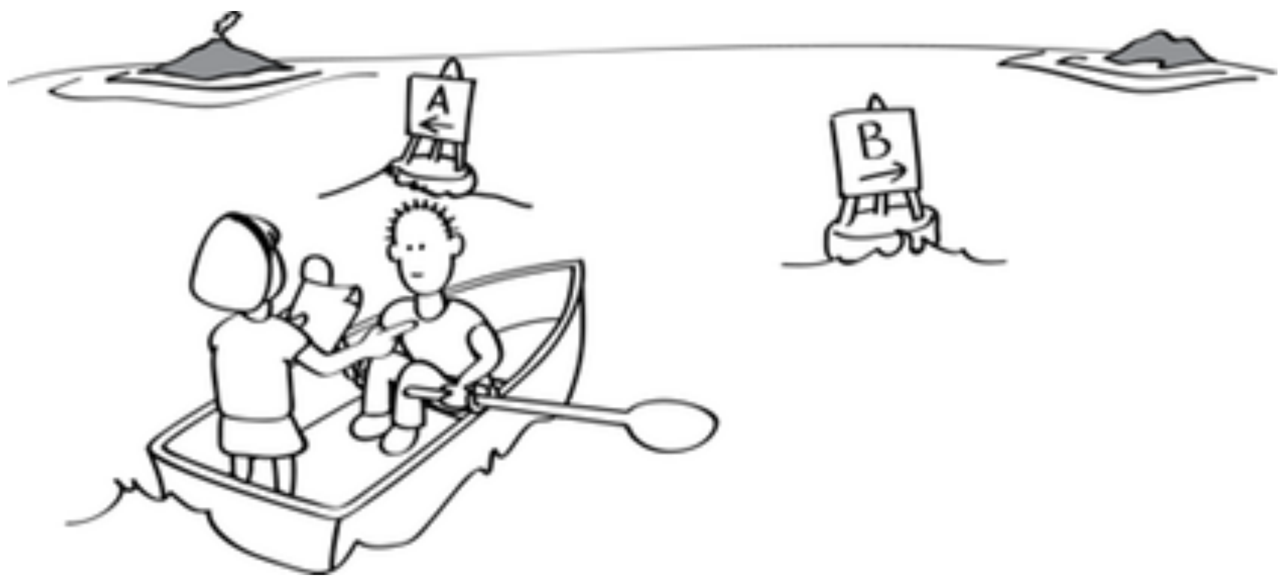
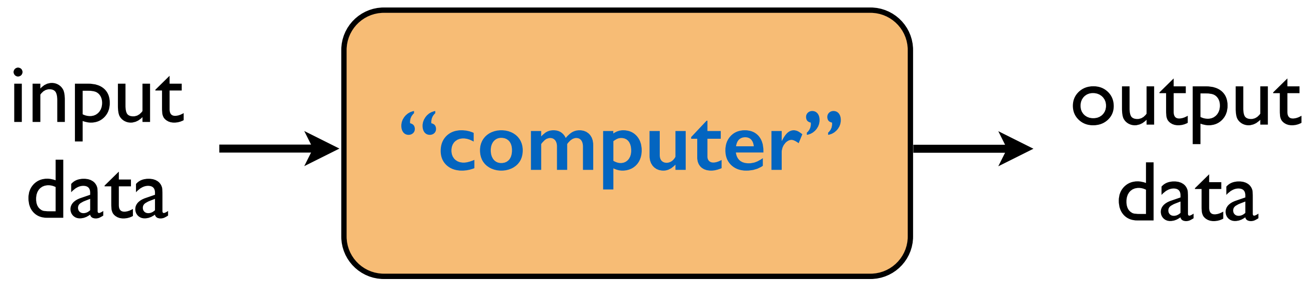
15-251
Great Theoretical Ideas in Computer Science

Lecture 3:
Deterministic Finite Automata (DFA)



September 6th, 2016

This Week



What is computation?

What is an algorithm?

How can we mathematically define them?

Let's assume two things about our world

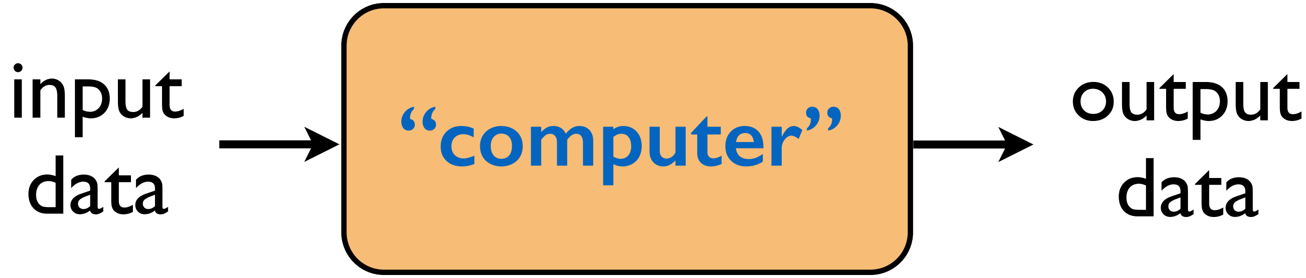
No universal machines exist.

+

isPrime

Sorting

We only have machines to solve decision problems.



What is computation?

What is an algorithm?

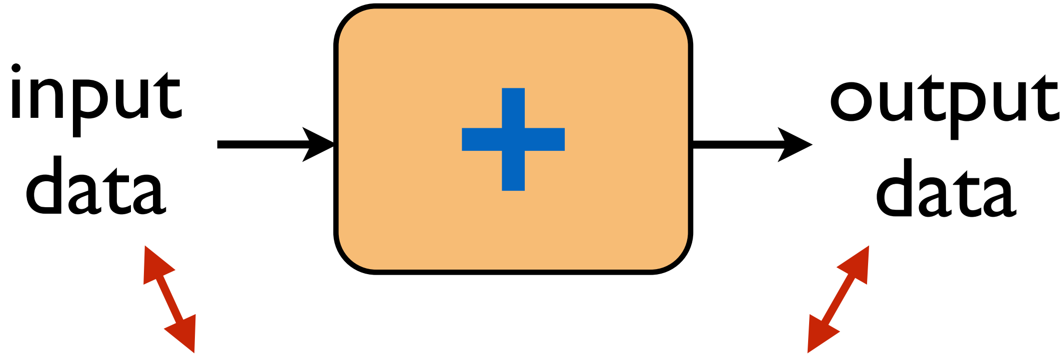
How can we mathematically define them?

Today:

How do we represent information/data?
What is a computational problem?

Introducing deterministic finite automata (DFA)

Examples of computational problems



Instance	Solution
0, 0	0
0, 1	1
1, 1	2
2, 2	4
2, 3	5
10, 1	11
100, 99	199
$\vdots$	$\vdots$

Examples of computational problems

Instance	Solution
0	No
1	No
2	Yes
3	Yes
4	No
$\vdots$	$\vdots$
251	Yes
$\vdots$	$\vdots$

Examples of computational problems

Instance	Solution
a	Yes
10101	Yes
selfless	No
dammitimmad	Yes
parahaziramarizaharap	Yes
⋮	⋮
⋮	⋮

Examples of computational problems

Instance

[vanilla, mind, Anil, yogurt, doesn't]

Solution

[Anil, doesn't, mind, vanilla, yogurt]

Representing information

Familiar idea:

Information in a computer is represented with 0s and 1s.

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

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

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

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

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

Can use whichever is convenient.

What is a computational problem?

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

The palindrome computational problem is:

Instance	Solution	
ϵ	1	Yes
0	1	Yes
1	1	Yes
00	1	Yes
01	0	No
10	0	No
11	1	Yes
000	1	Yes
001	0	No
$\vdots$	$\vdots$	

What is a computational problem?

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

The multiplication computational problem is:

Instance	Solution
$0\#0$	0
$0\#1$	0
$1\#0$	0
$1\#1$	1
$10\#2$	20
$11\#3$	33
$12345679\#9$	111111111
$\vdots$	$\vdots$

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?

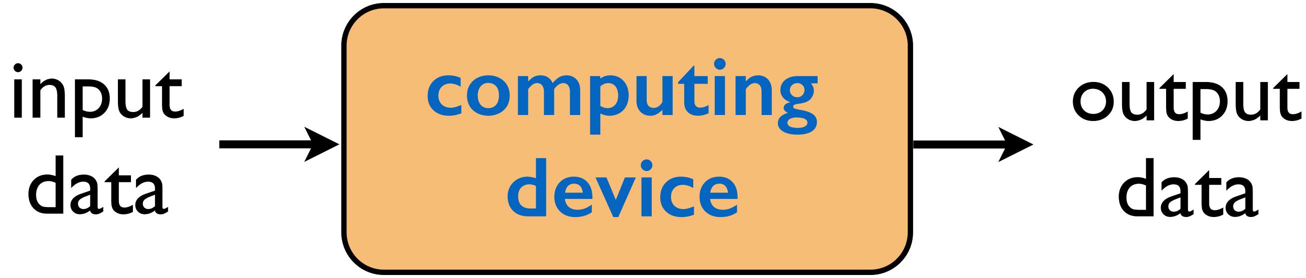
Important

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
$\vdots$	$\vdots$

$$L \subseteq \Sigma^*$$

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



What is computation?

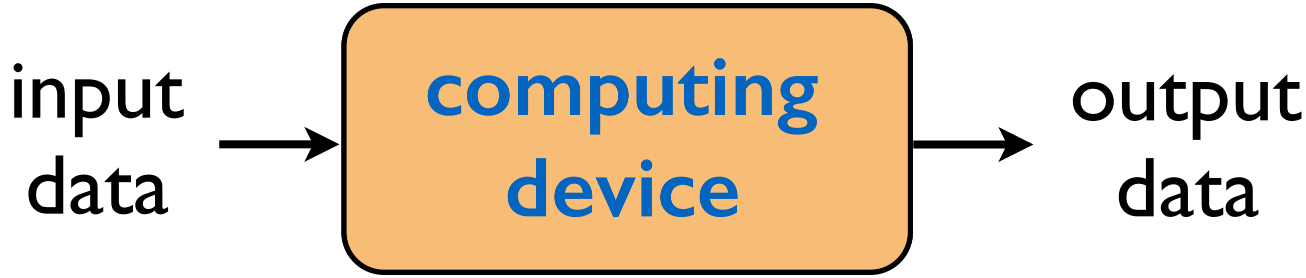
What is an algorithm?

How can we mathematically define them?

Today:

How do we represent information/data?
What is a computational problem?

Introducing deterministic finite automata (DFA)



What is computation?

What is an algorithm?

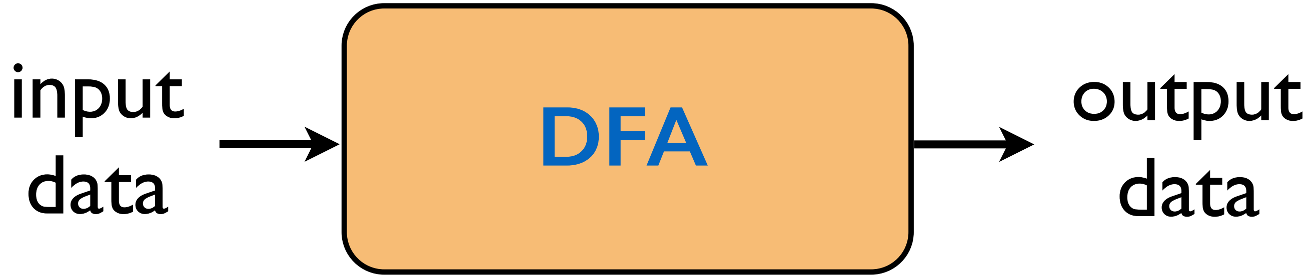
How can we mathematically define them?

Today:

How do we represent information/data?
What is a computational problem?

Introducing deterministic finite automata (DFA)

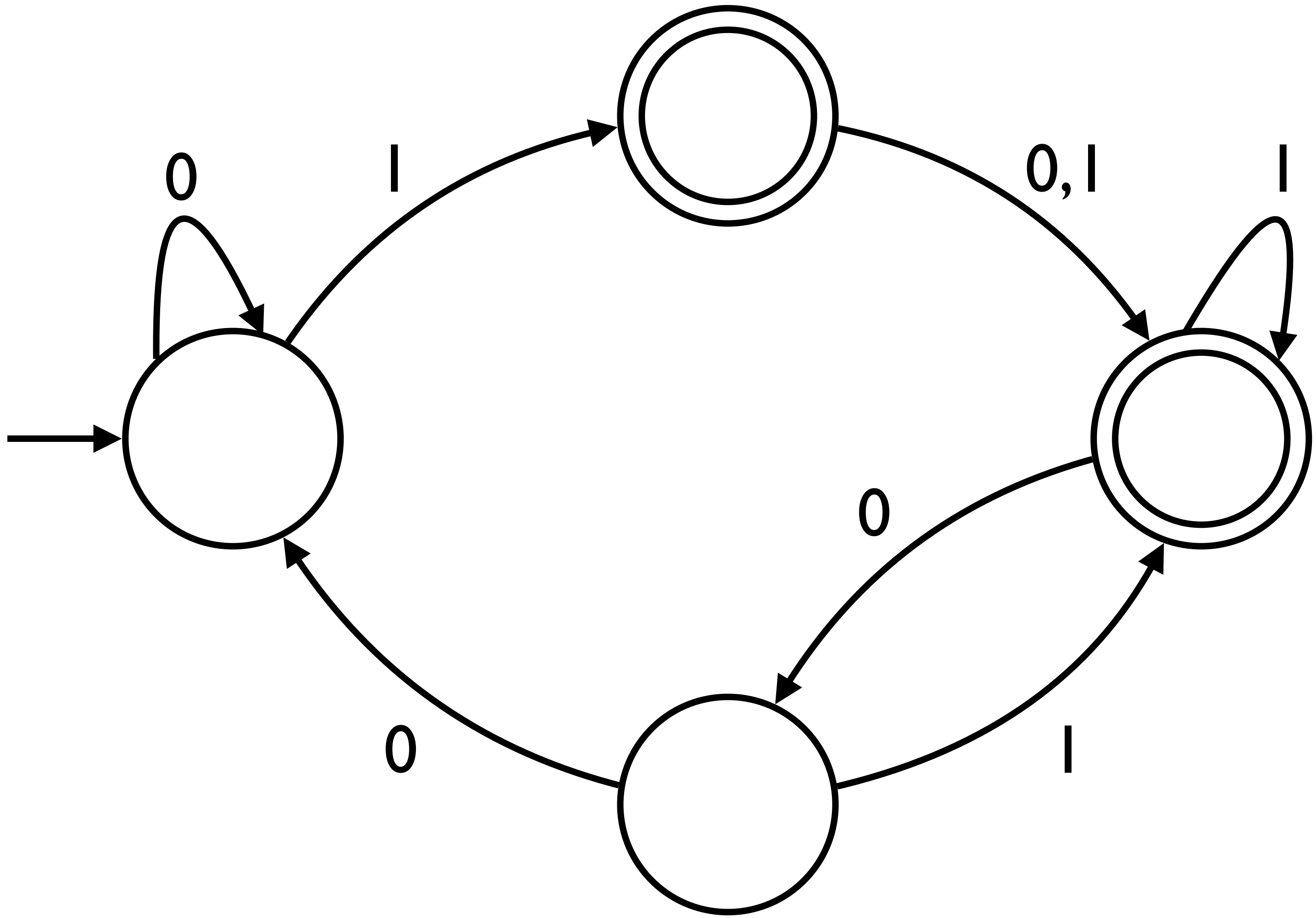
Introducing deterministic finite automata (DFA)



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

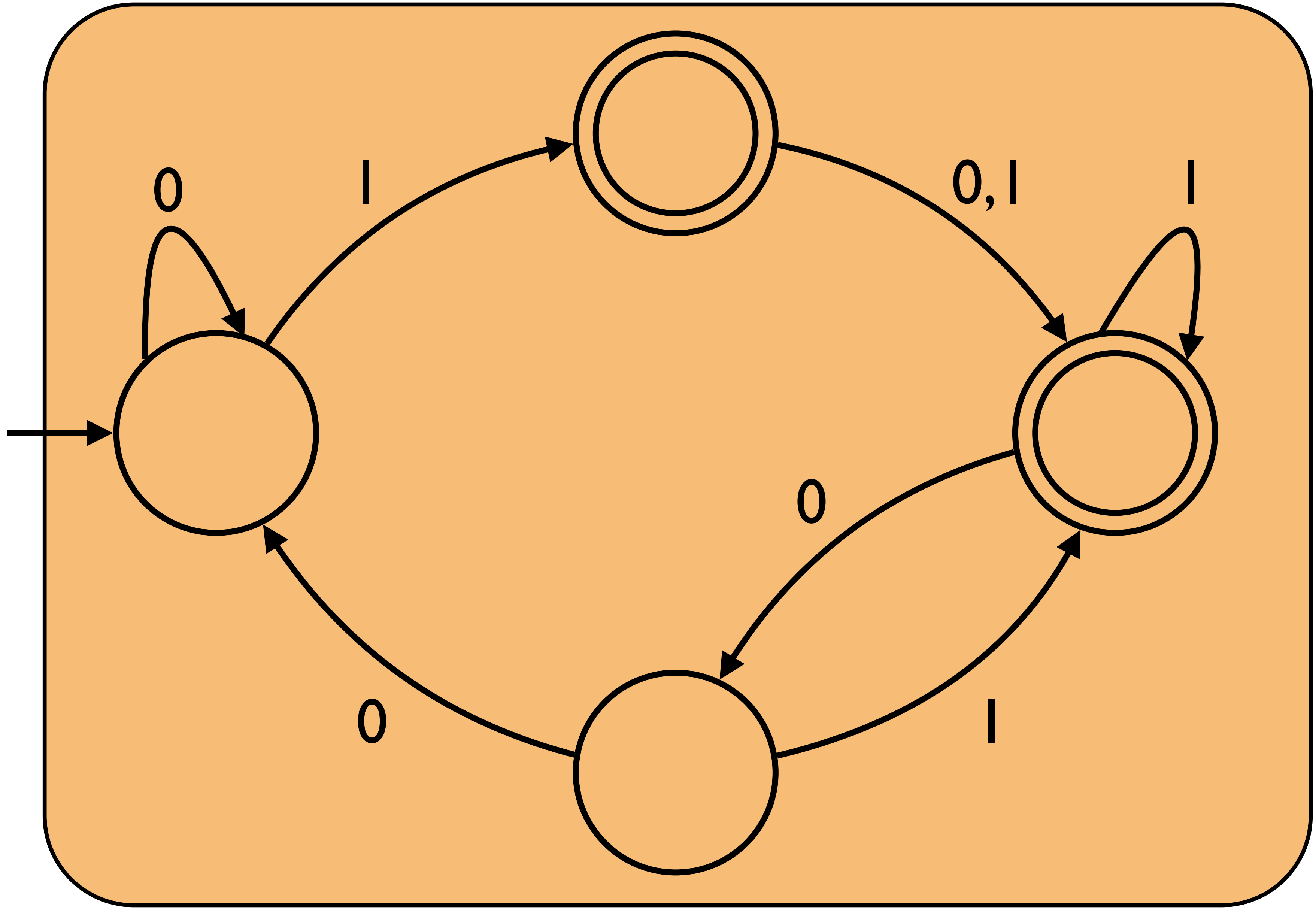
State diagram of a DFA

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



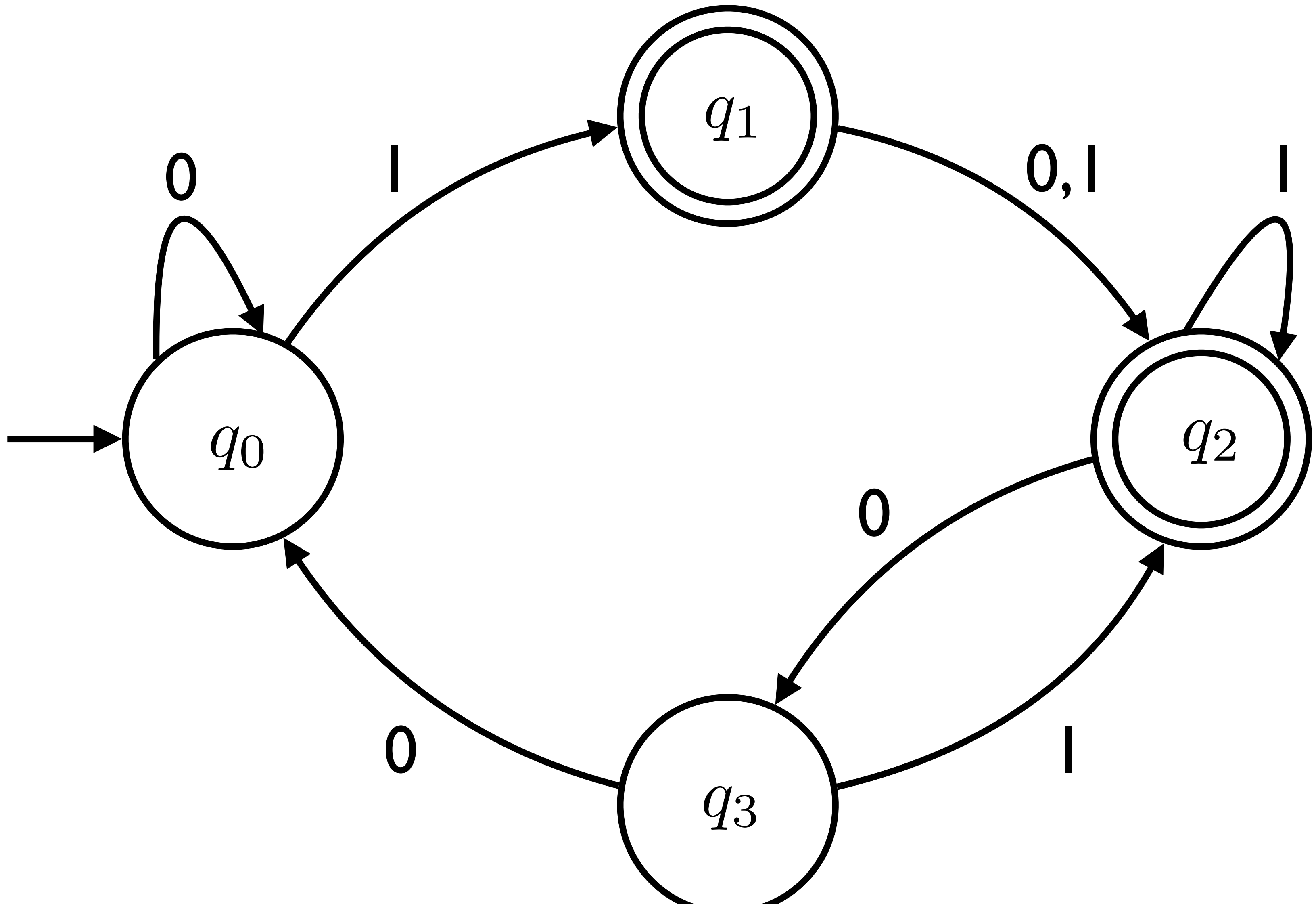
State diagram of a DFA

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



State diagram of a DFA

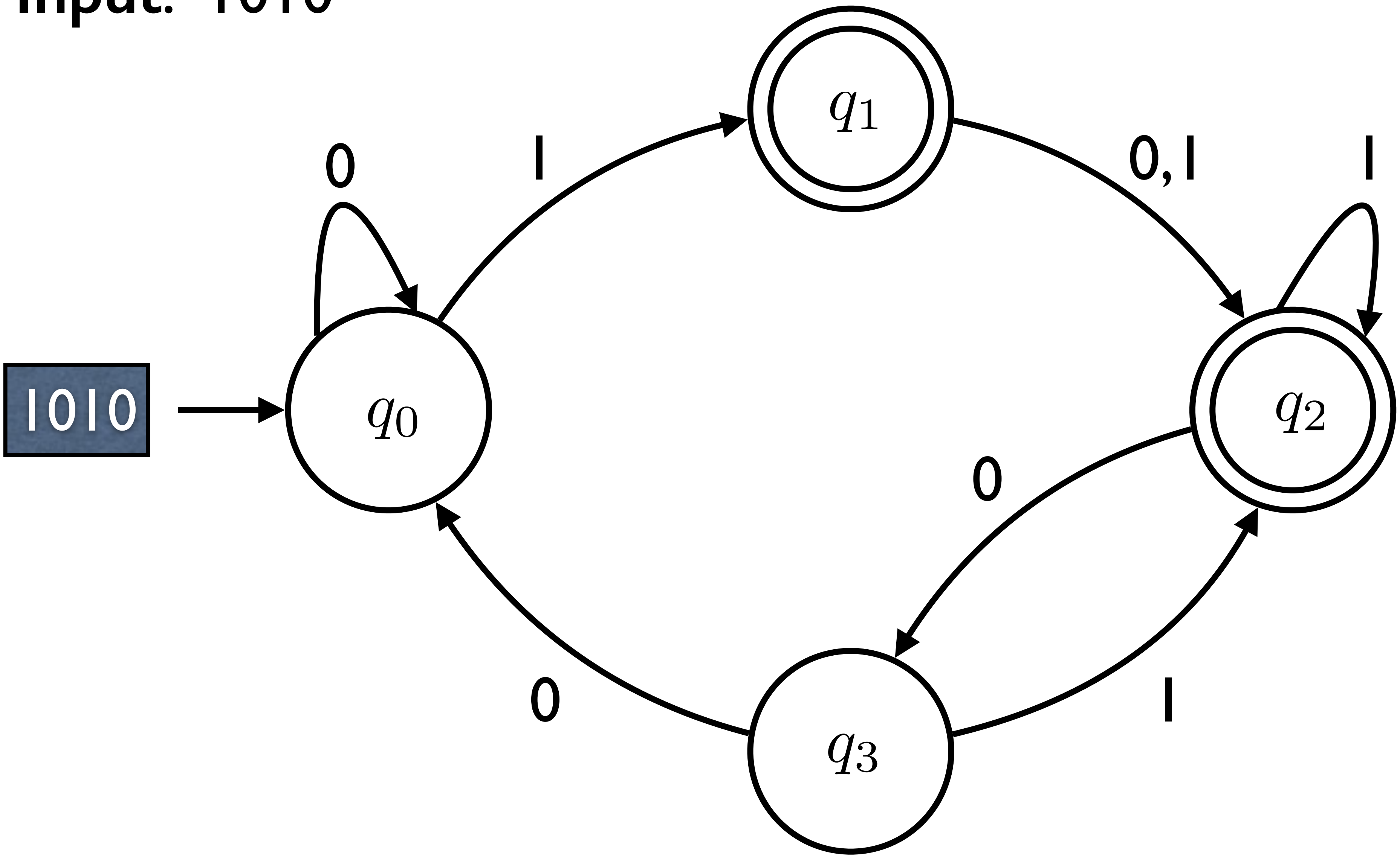
$\Sigma = \{0, 1\}$



Simulation of a DFA

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

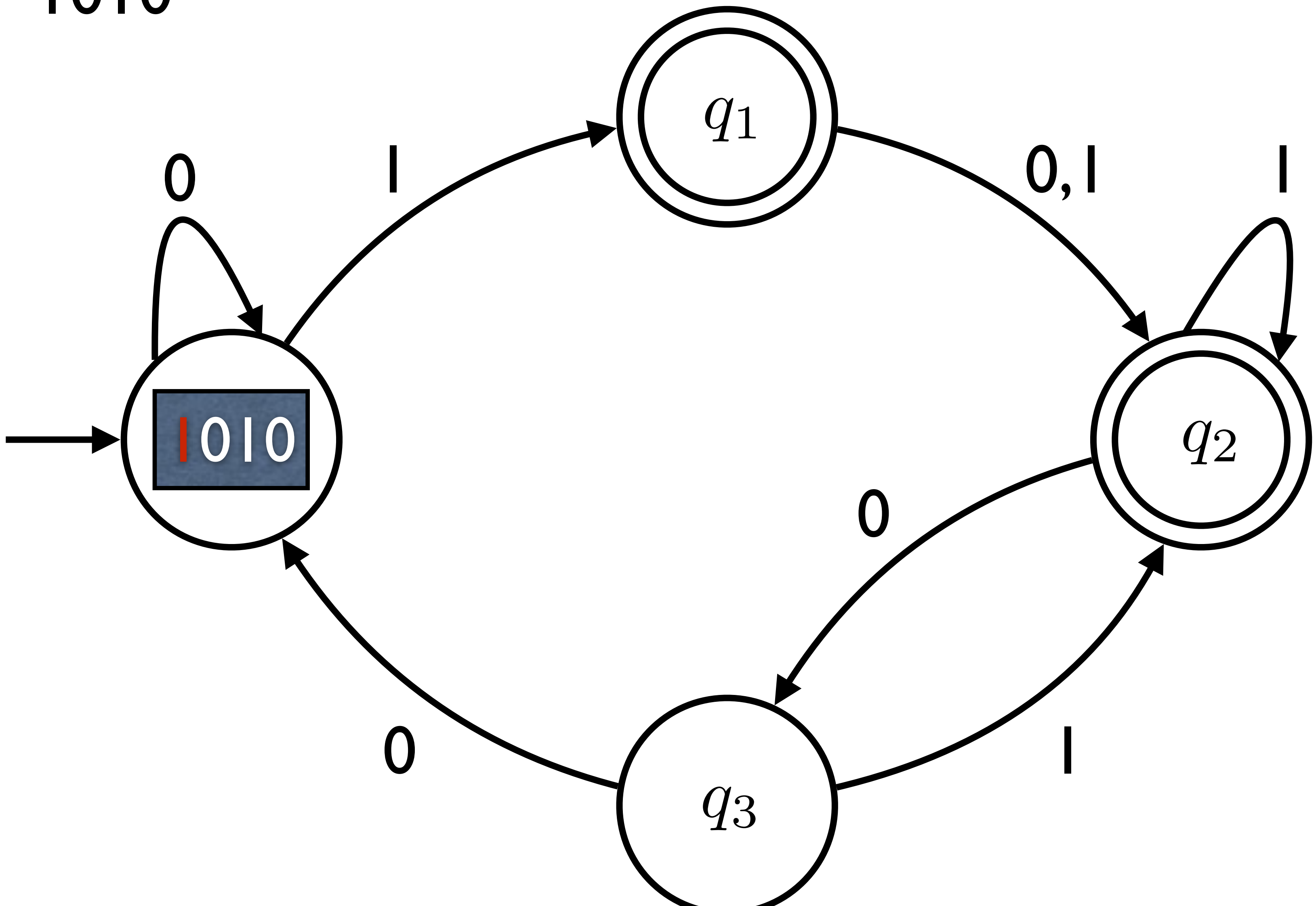
Input: 1010



Simulation of a DFA

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

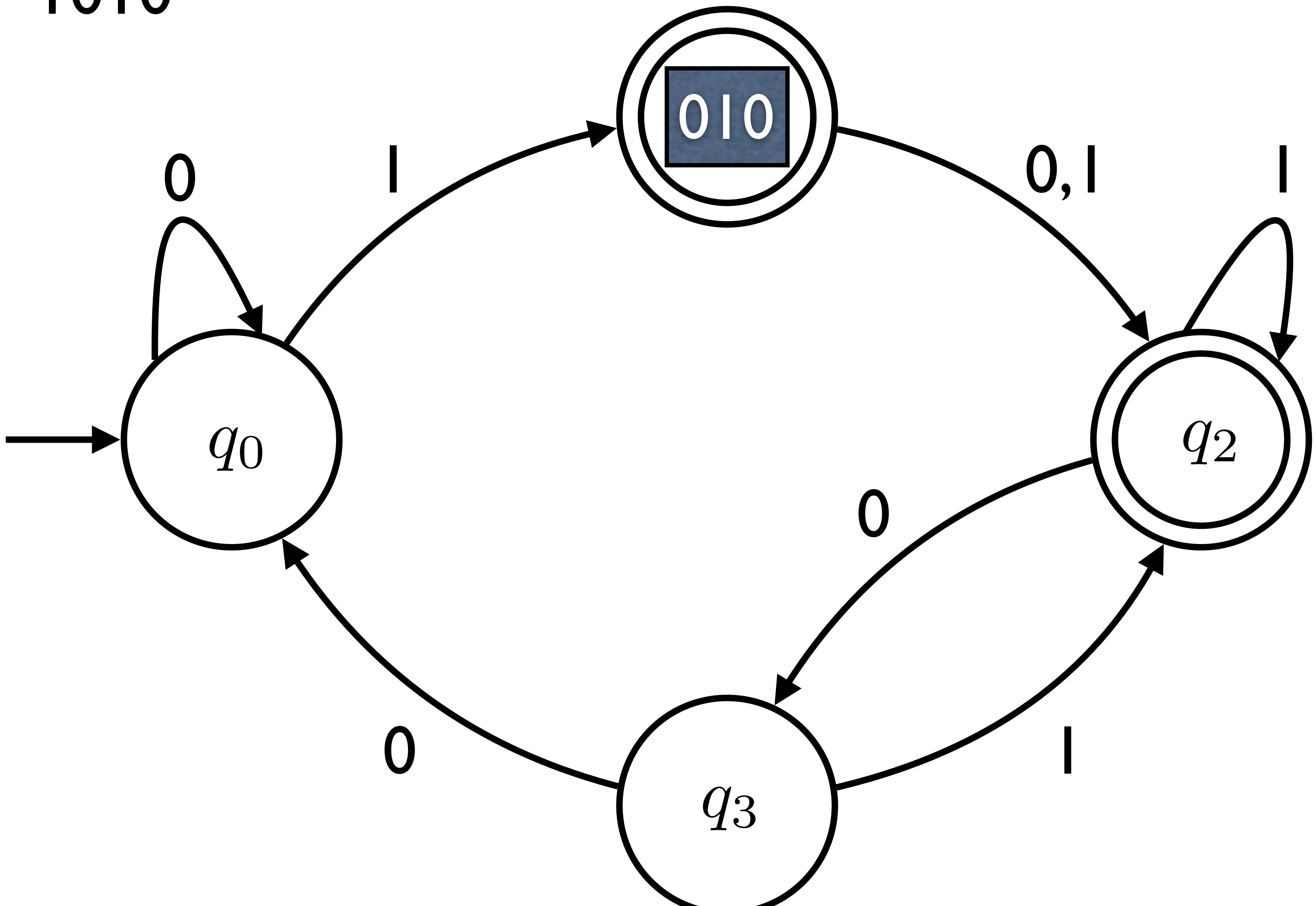
Input: 1010



Simulation of a DFA

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

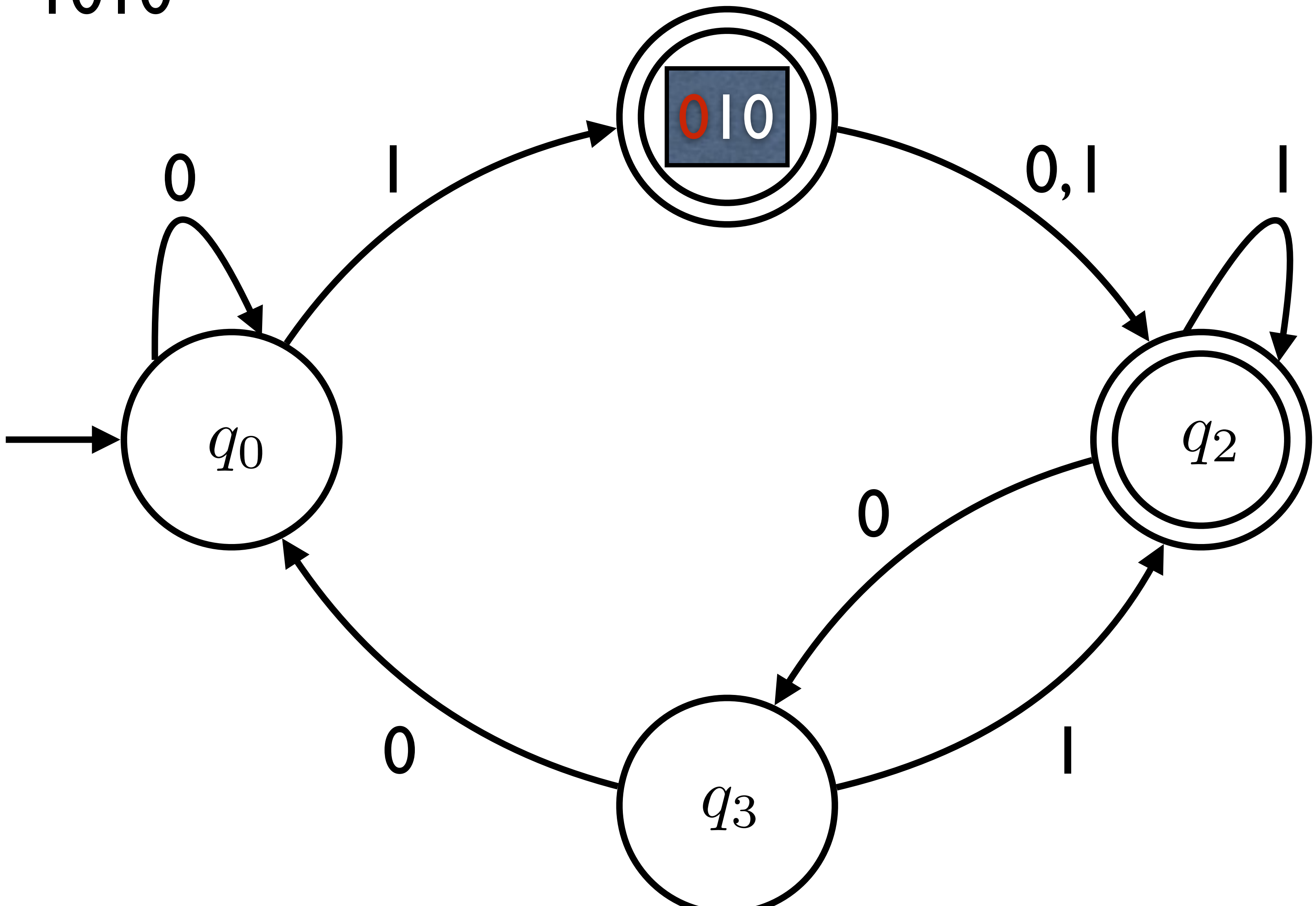
Input: 1010



Simulation of a DFA

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

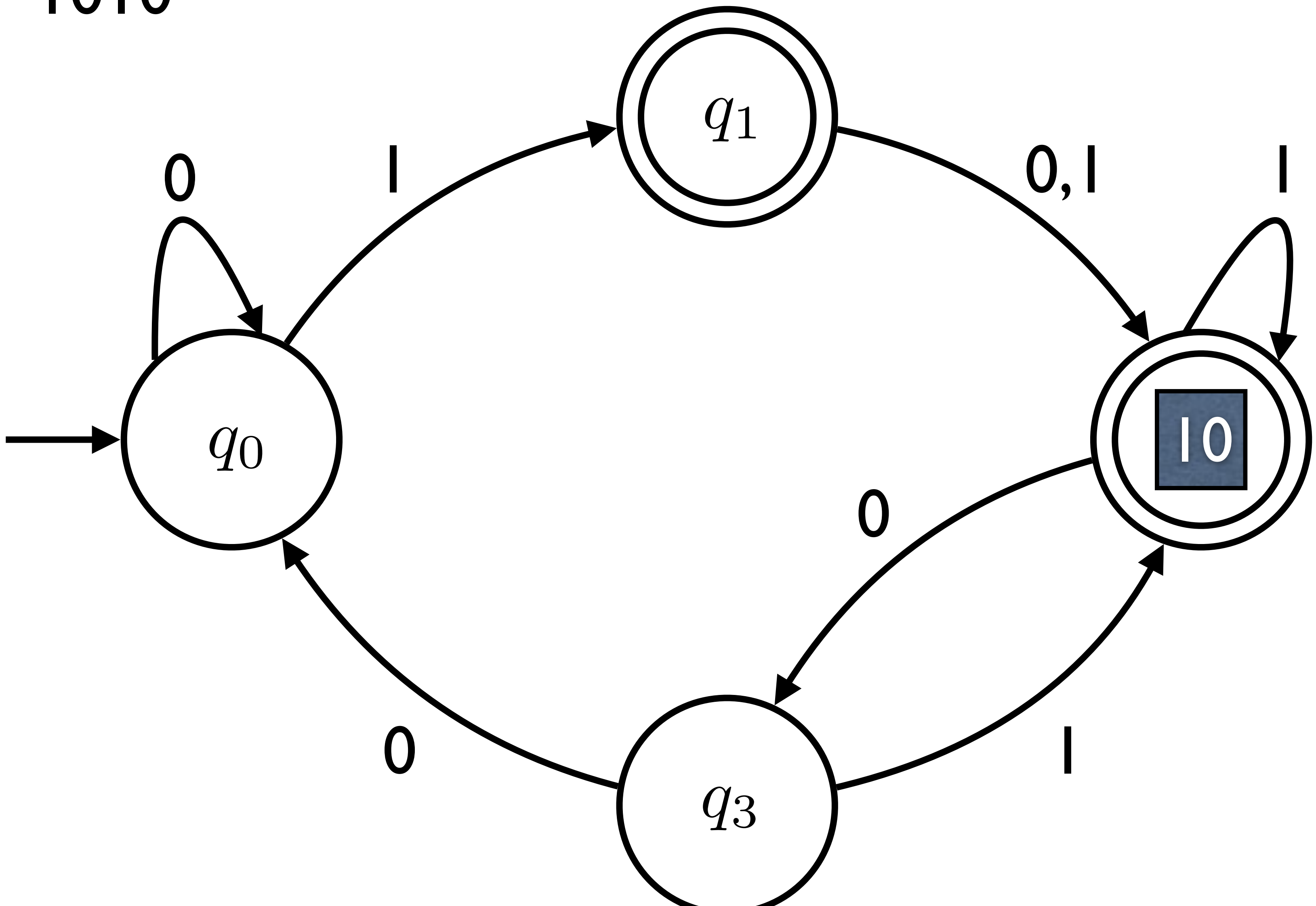
Input: 1010



Simulation of a DFA

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

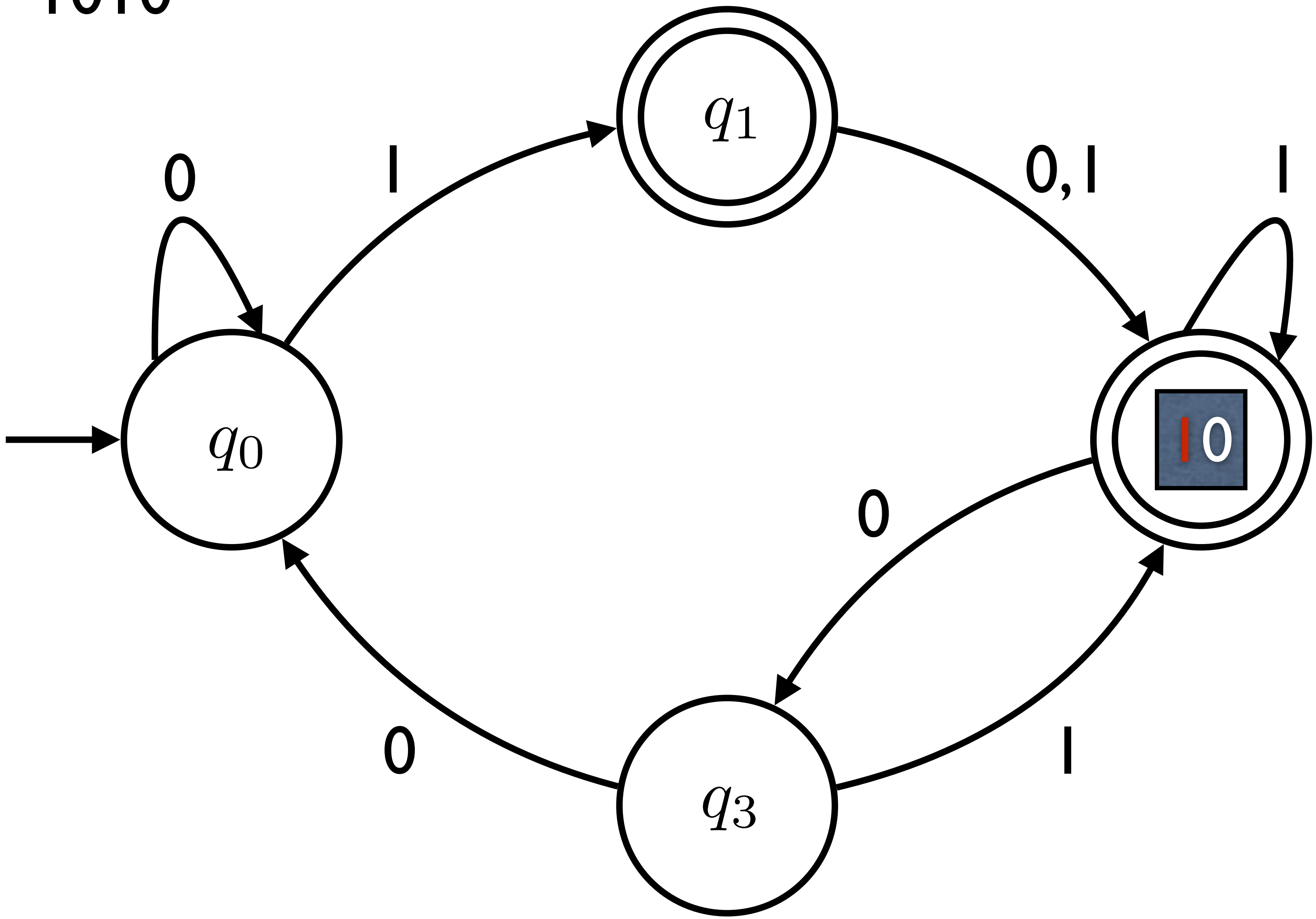
Input: 1010



Simulation of a DFA

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

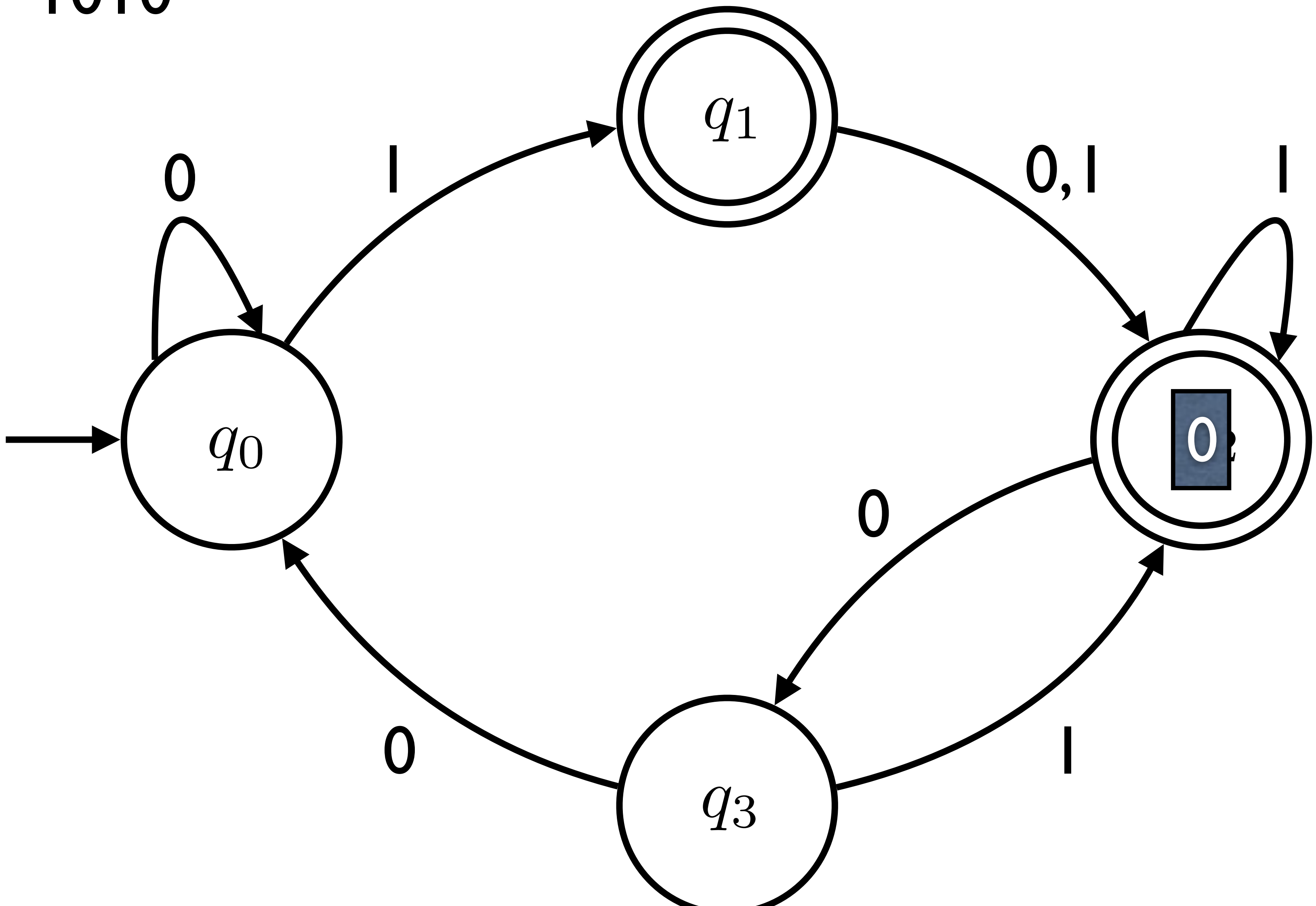
Input: 1010



Simulation of a DFA

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

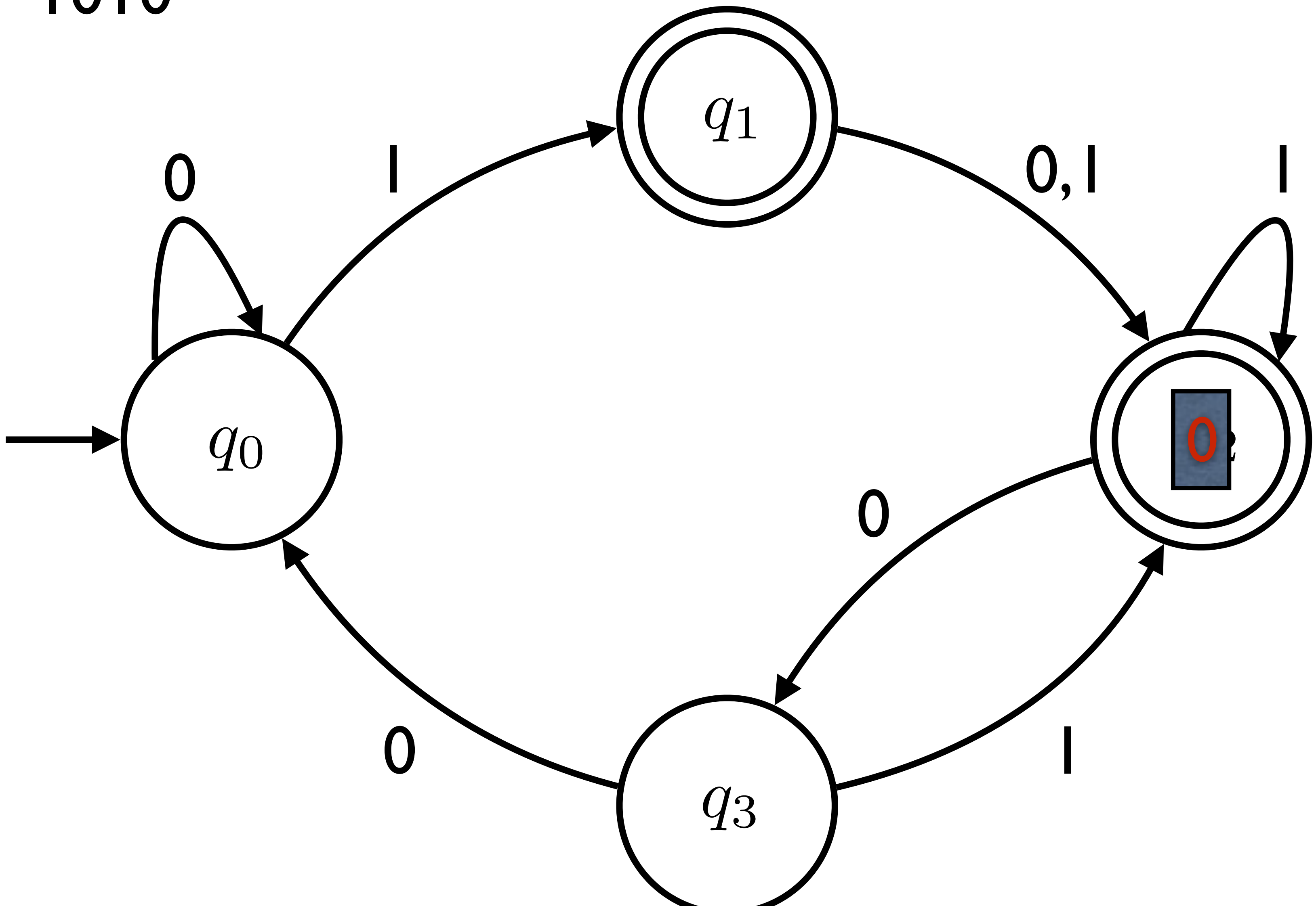
Input: 1010



Simulation of a DFA

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

Input: 1010

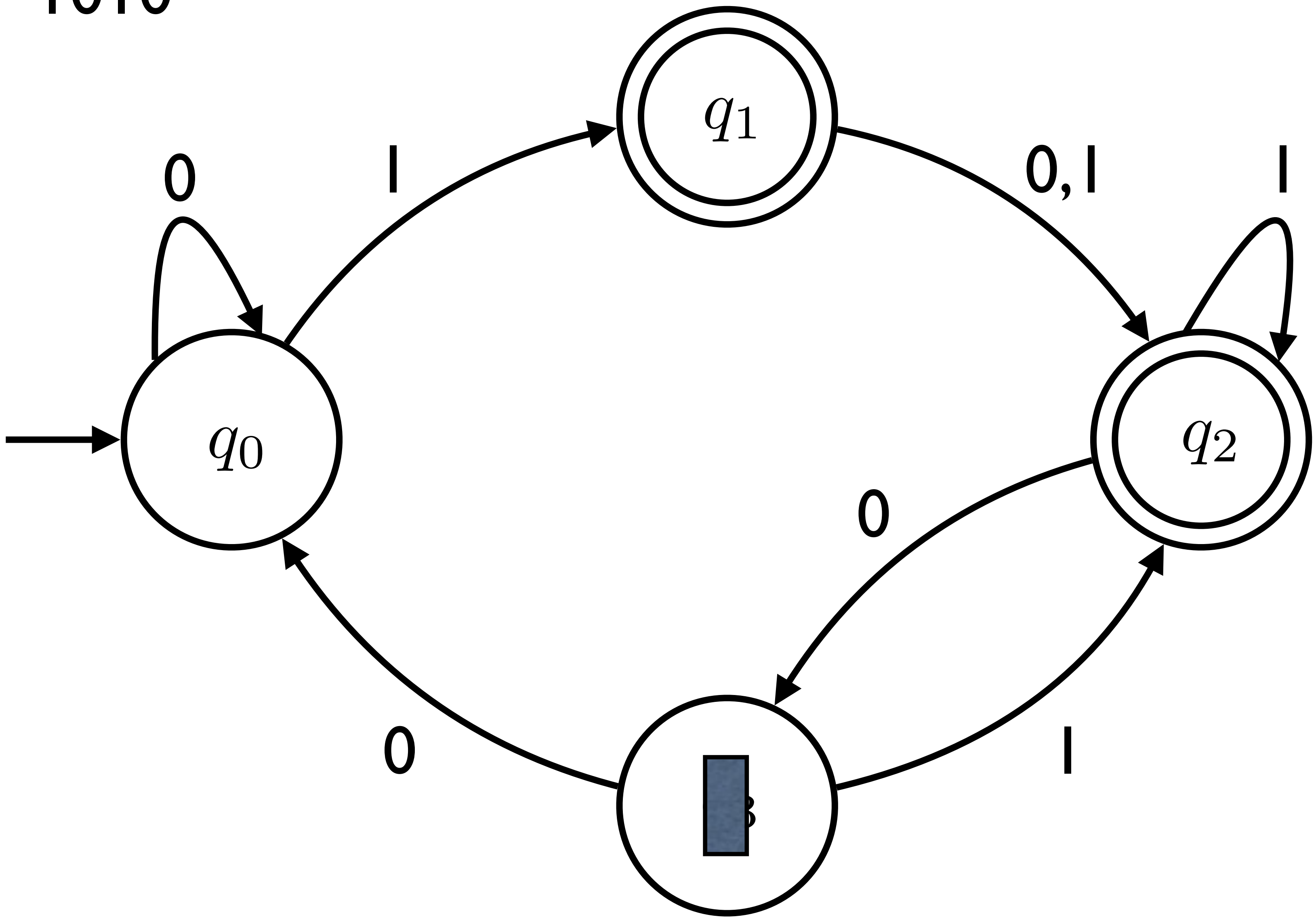


Simulation of a DFA

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

Input: 1010

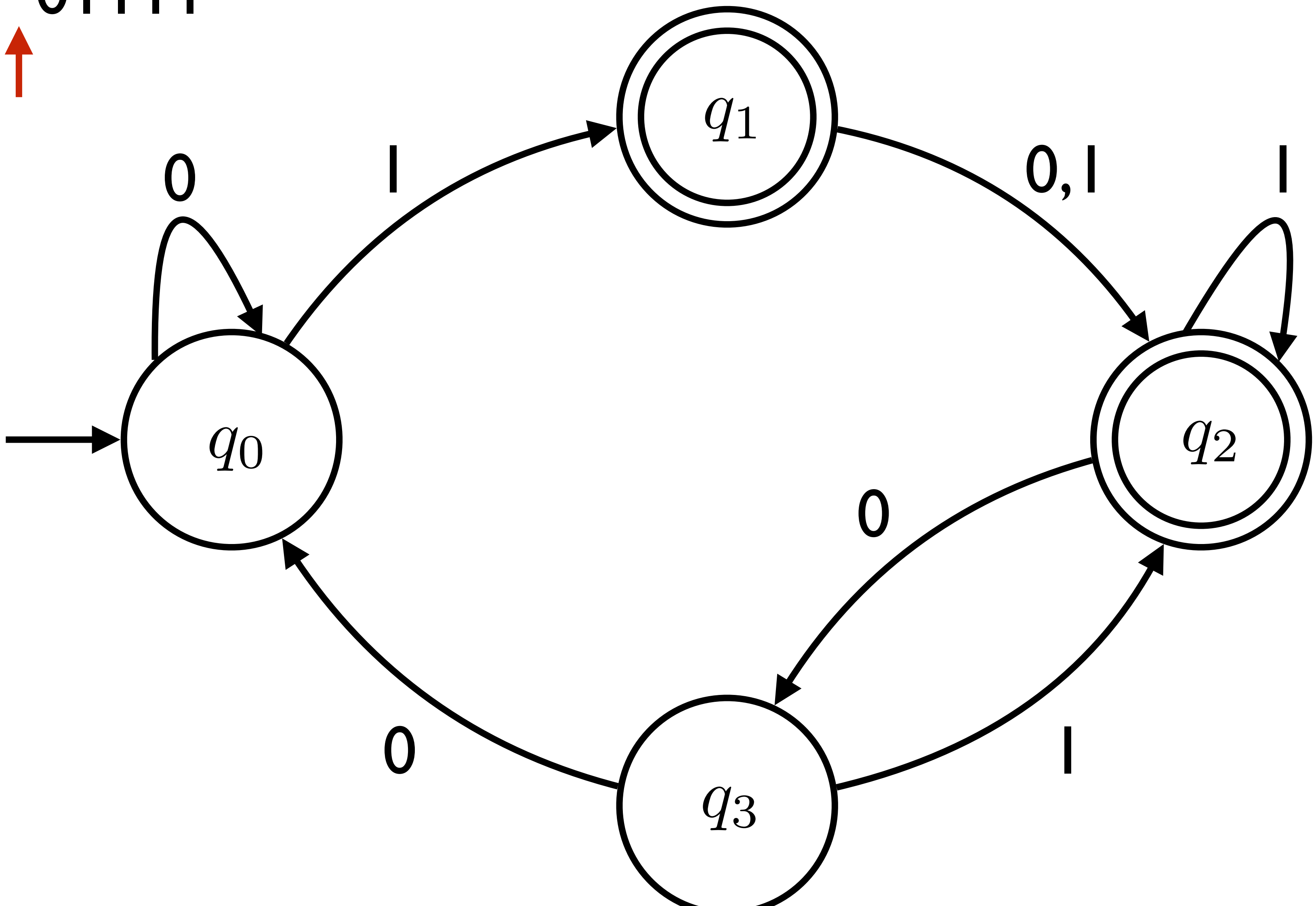
Decision: Reject



Simulation of a DFA

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

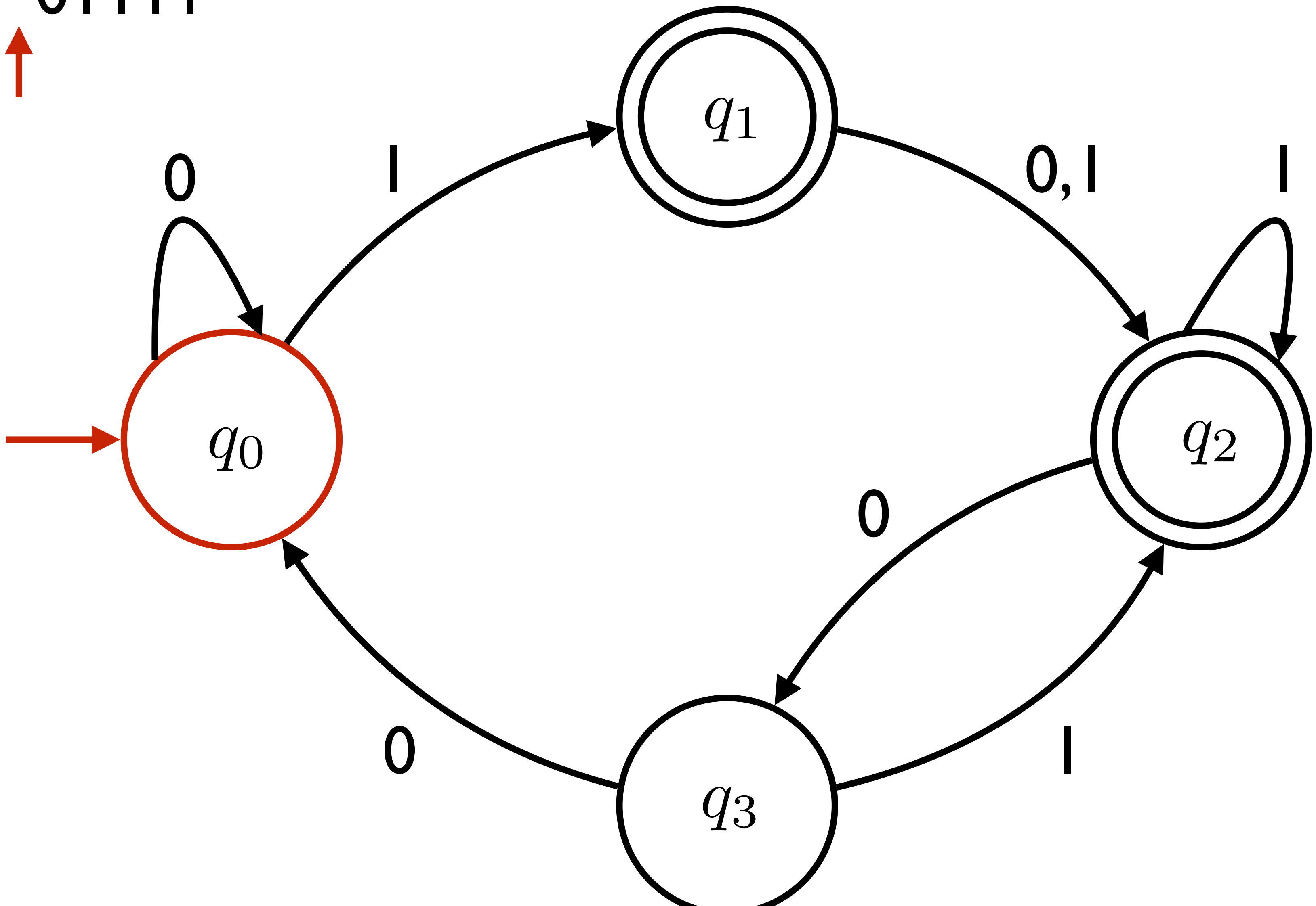
Input: 01111



Simulation of a DFA

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

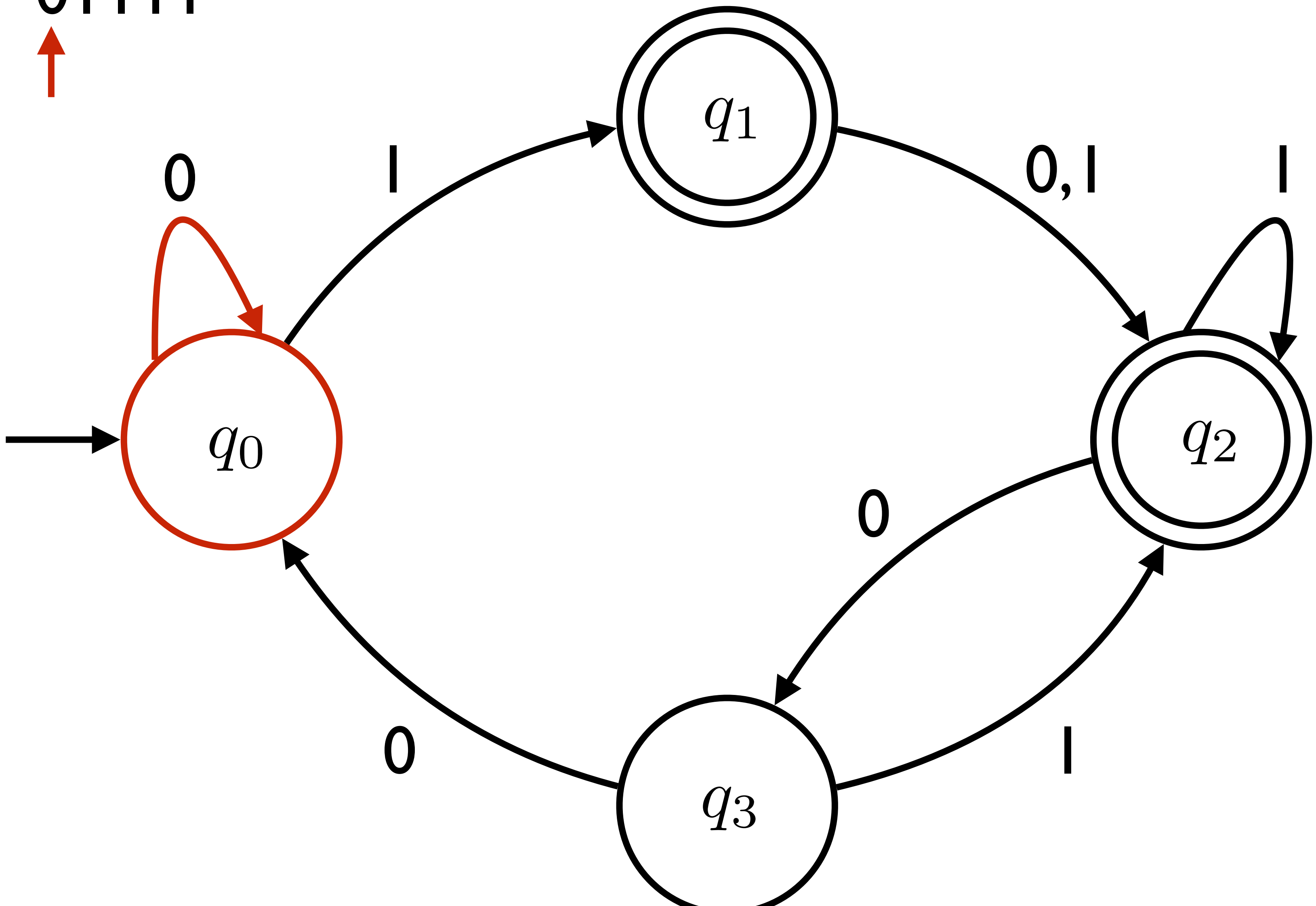
Input: 01111



Simulation of a DFA

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

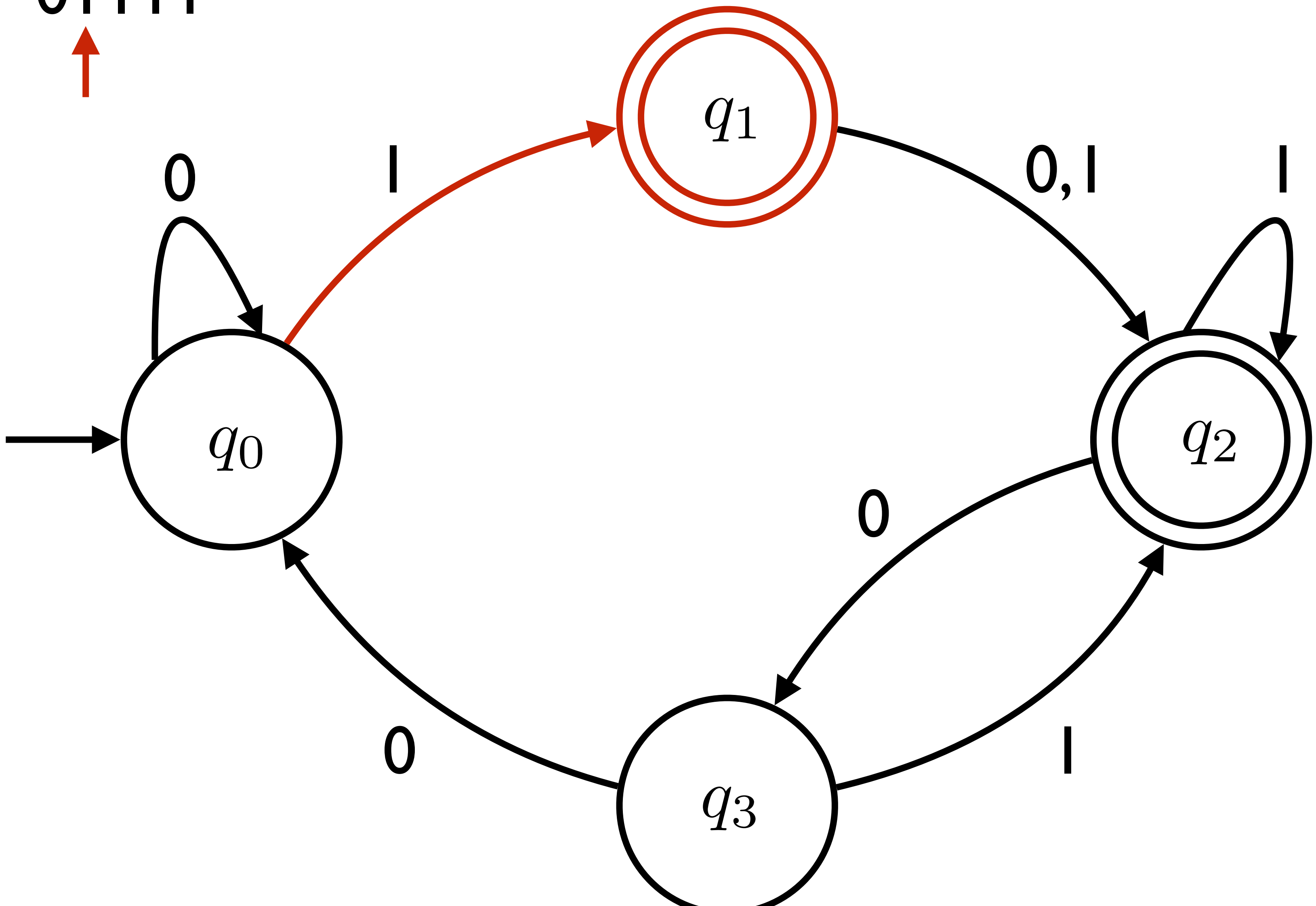
Input: 01111



Simulation of a DFA

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

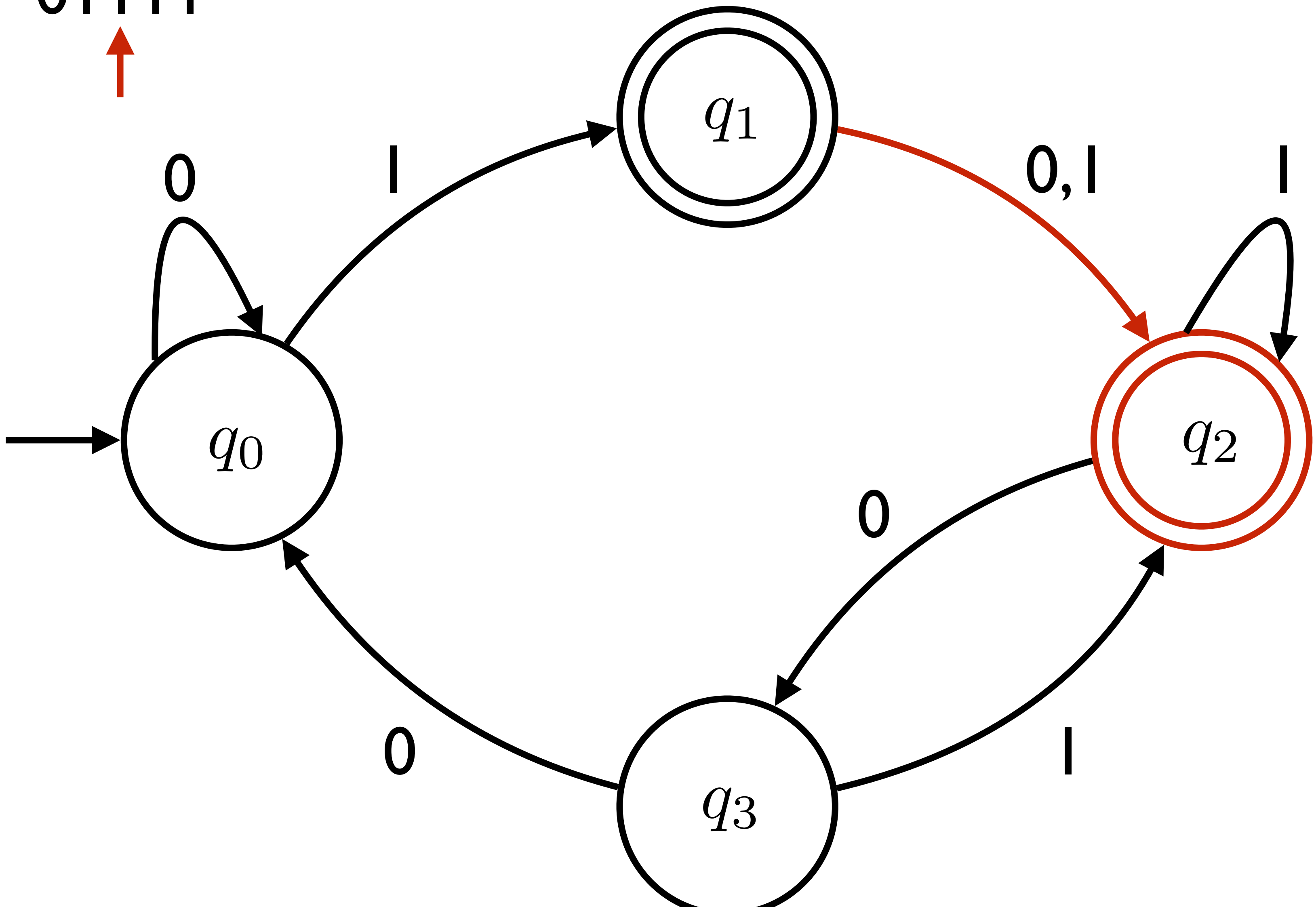
Input: 01111



Simulation of a DFA

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

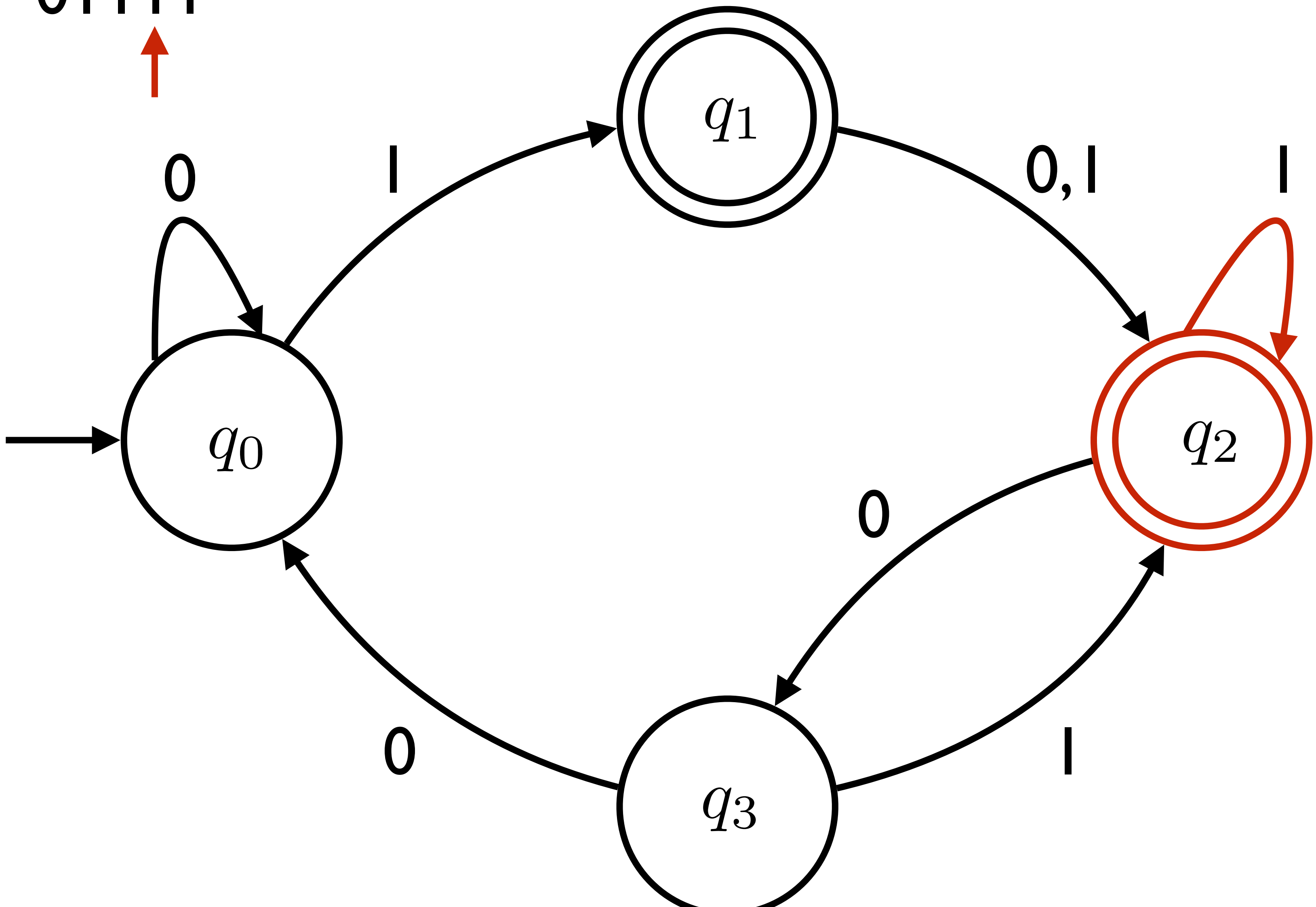
Input: 0 1 1 1 1



Simulation of a DFA

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

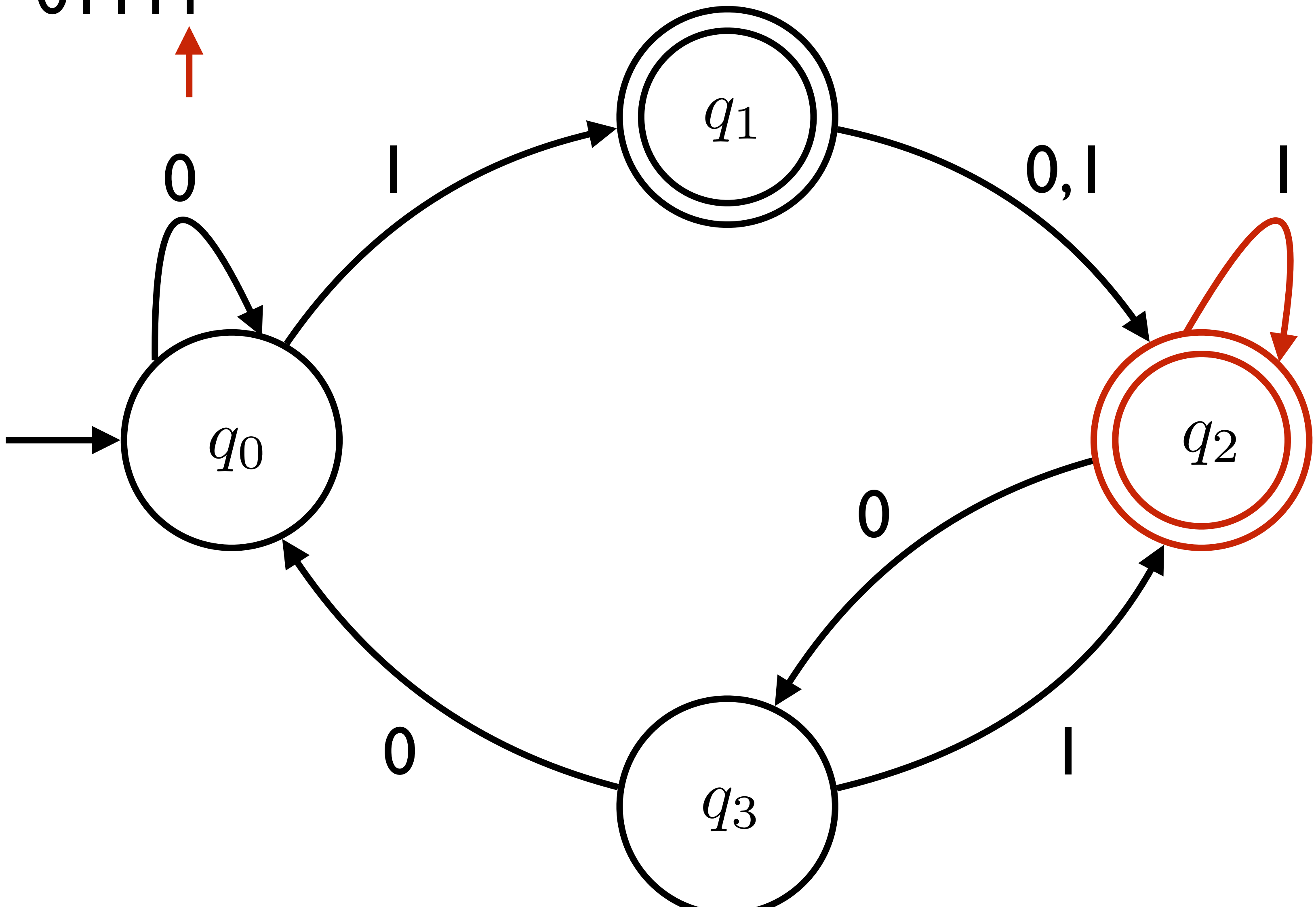
Input: 01111



Simulation of a DFA

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

Input: 01111

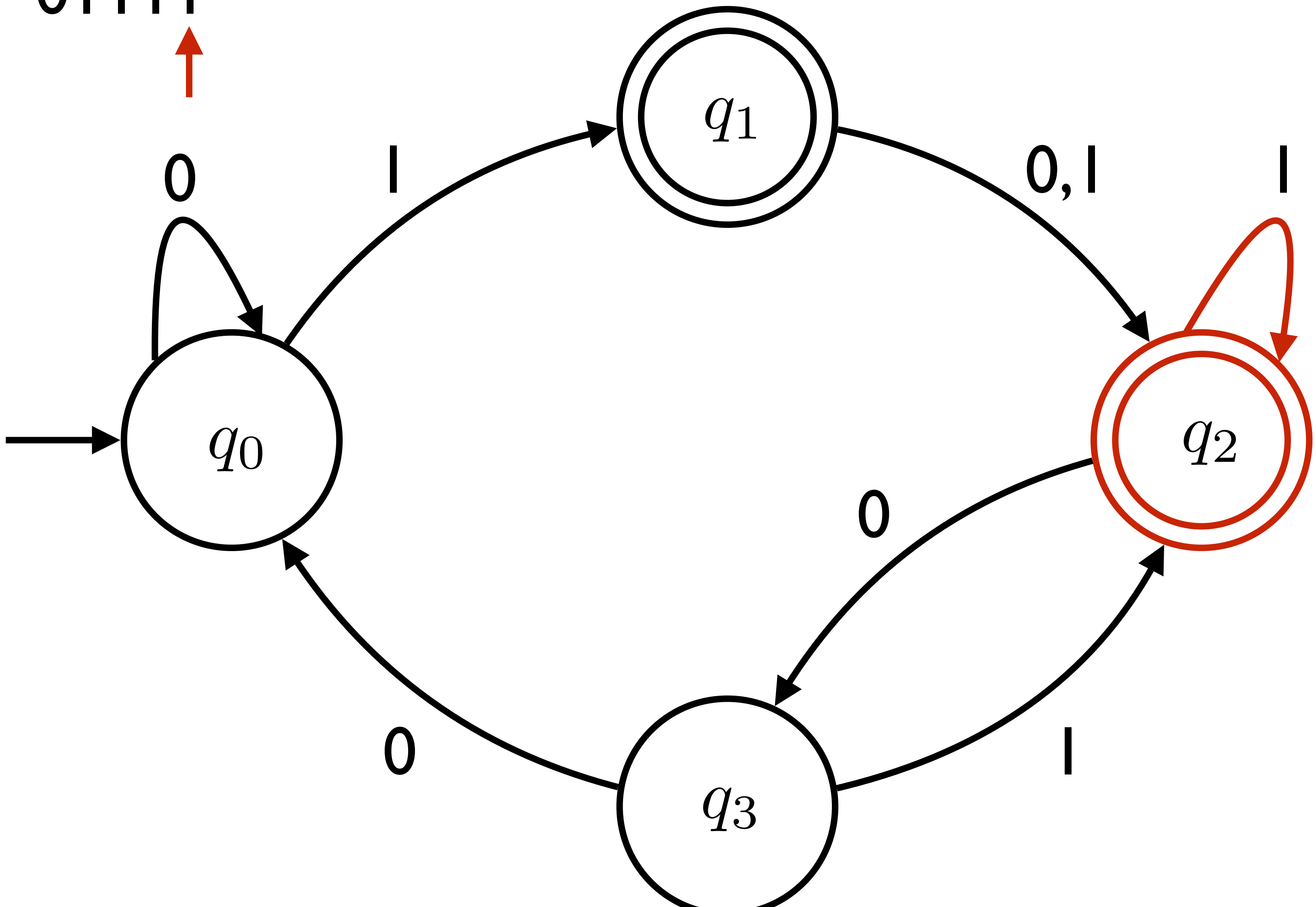


Simulation of a DFA

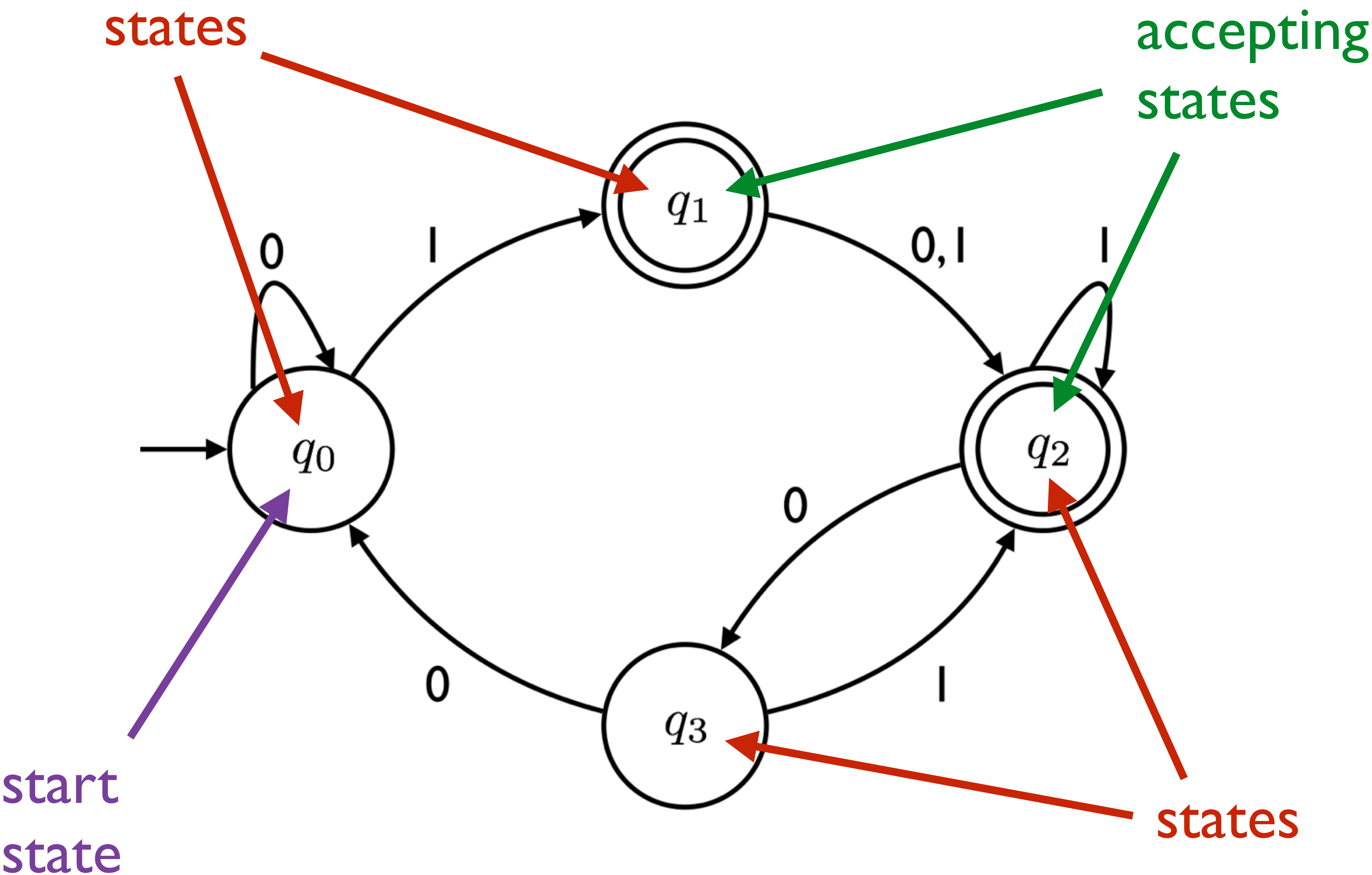
$\Sigma = \{0, 1\}$

Input: 01111

Decision: Accept



Anatomy of a DFA

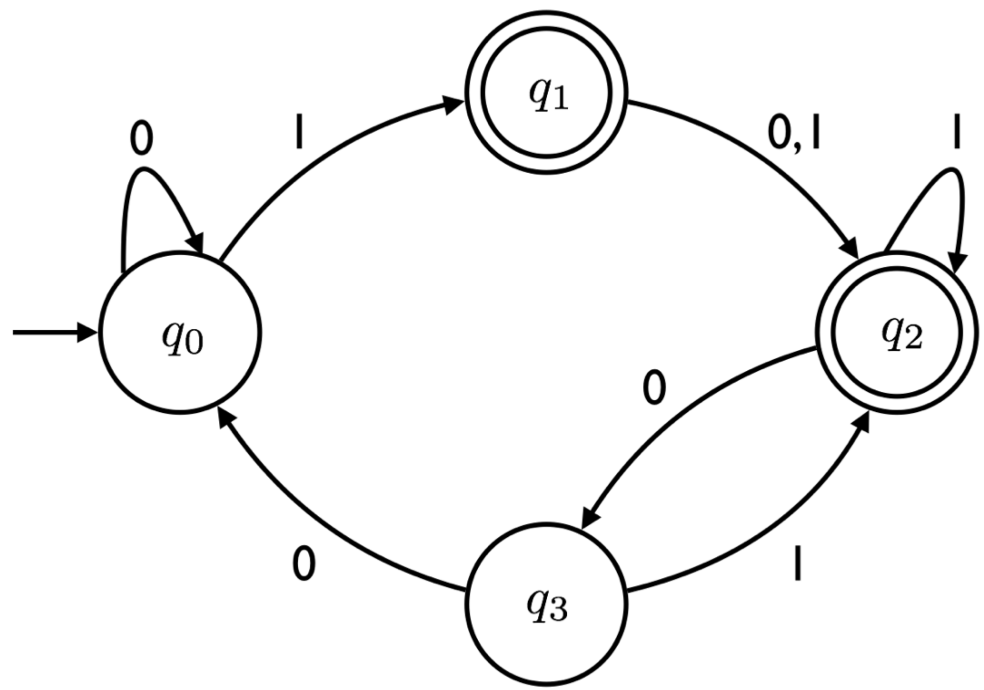


transition rule: labeled arrows

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;
  ...
```



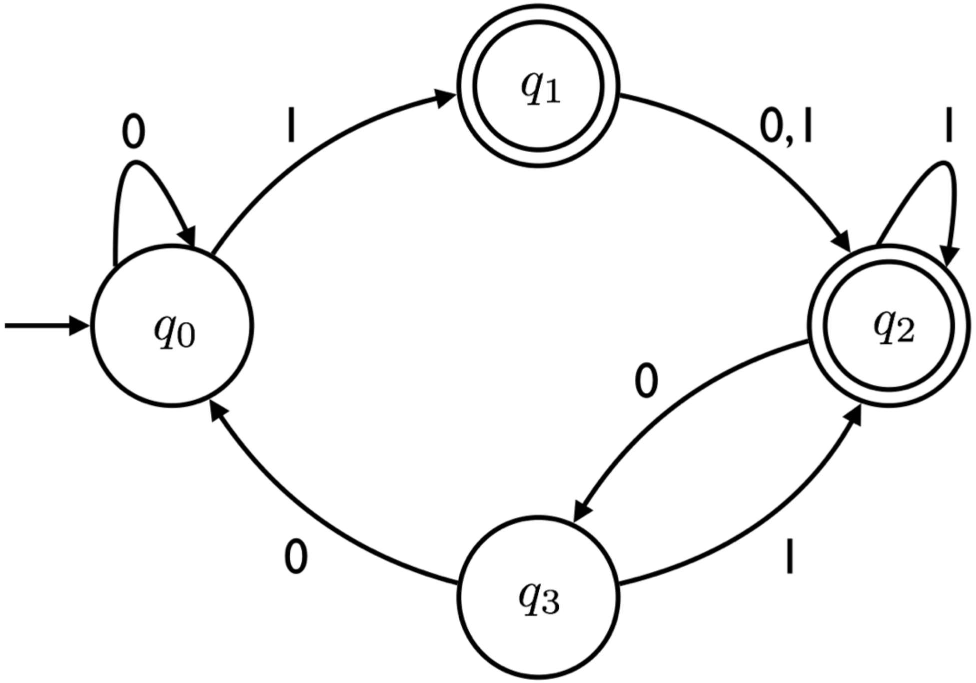
DFA as a programming language

```
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

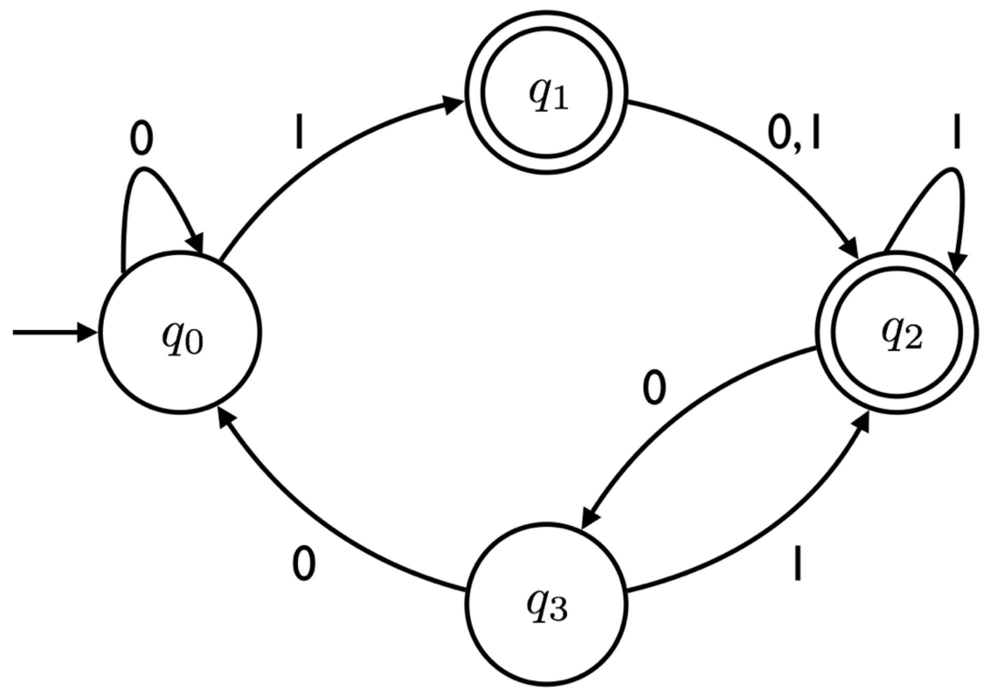
def foo(input):
    i = 0;
    STATE 0:
        if (i == input.length): return **False**;
        letter = input[i];
        i++;
        switch(letter):
            case ‘0’: go to STATE **0**;
            case ‘1’: go to STATE **1**;

    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.



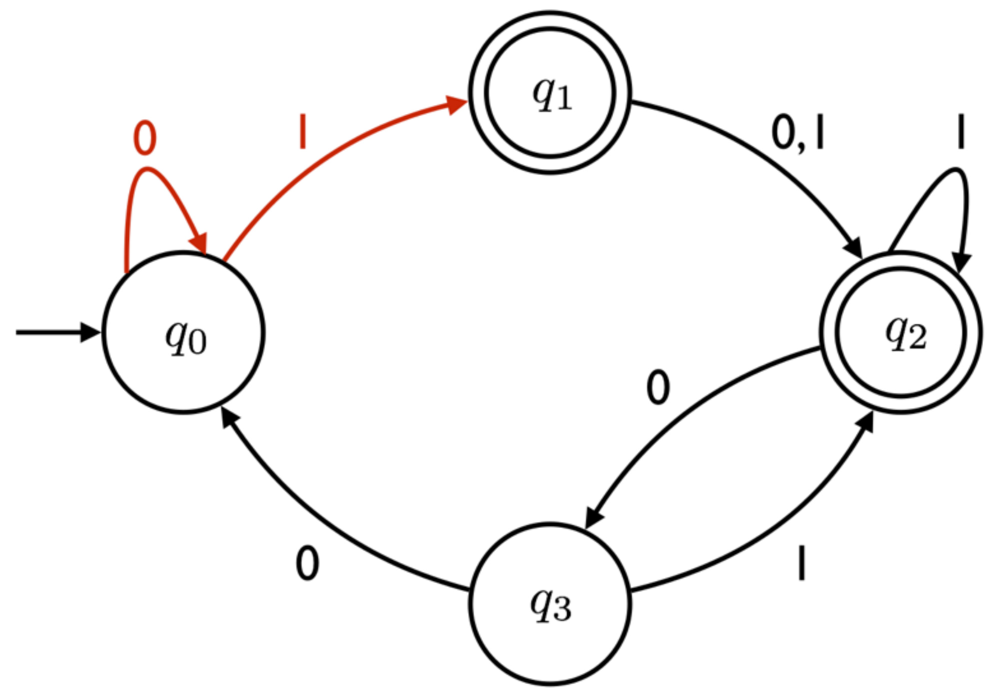
DFA as a programming language

```
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.

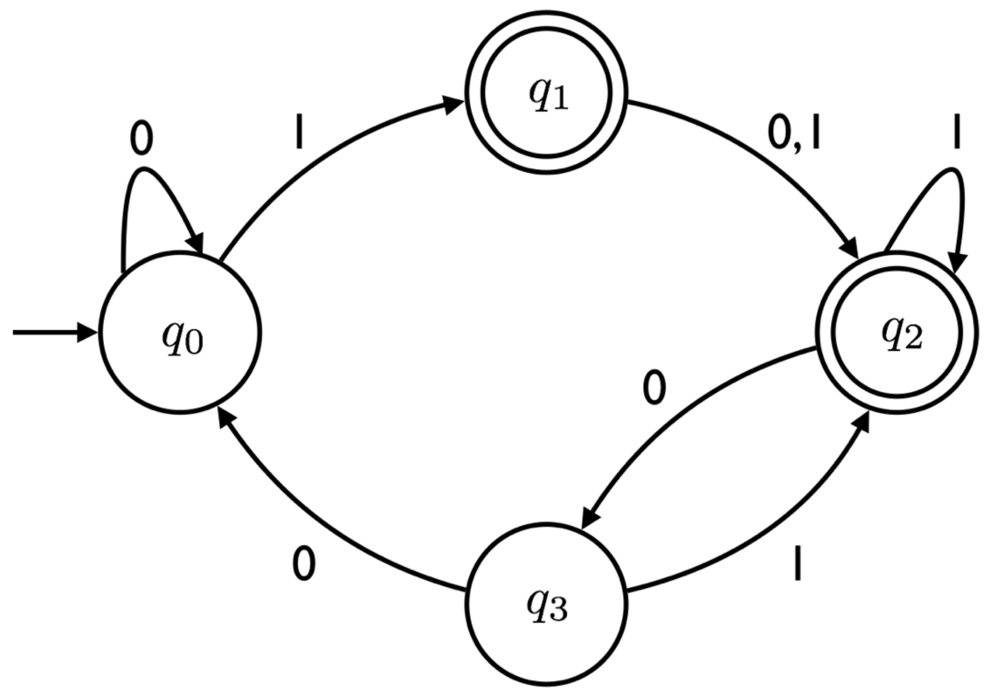


DFA as a programming language

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

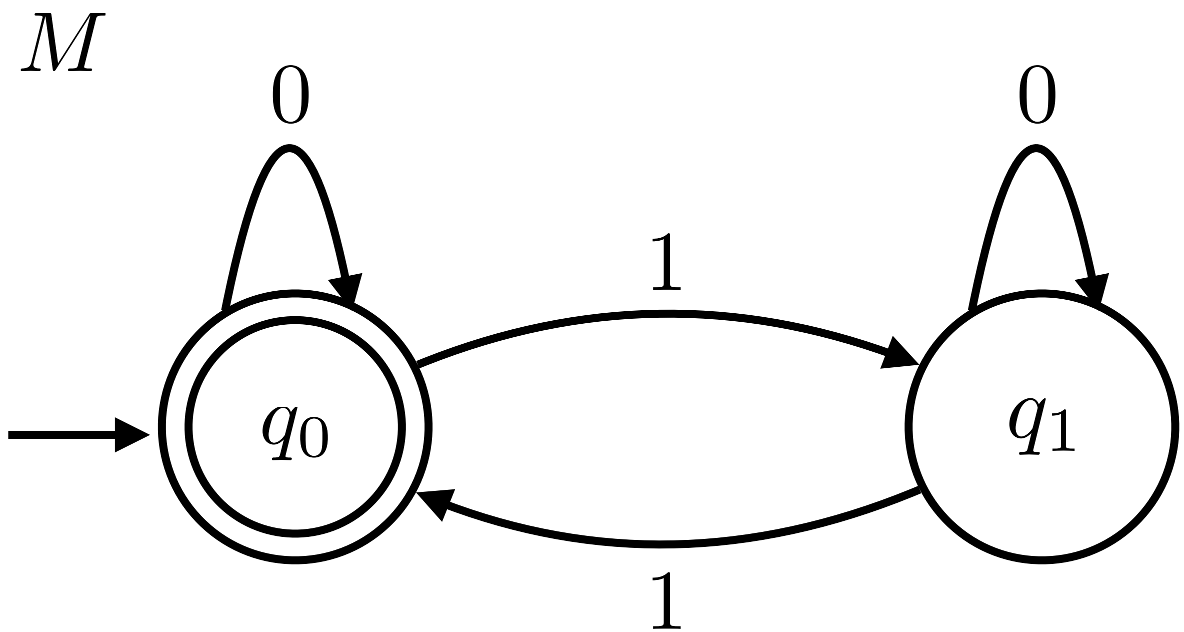
If $L = L(M)$, we say that M *decides* L.

computes

recognizes

accepts

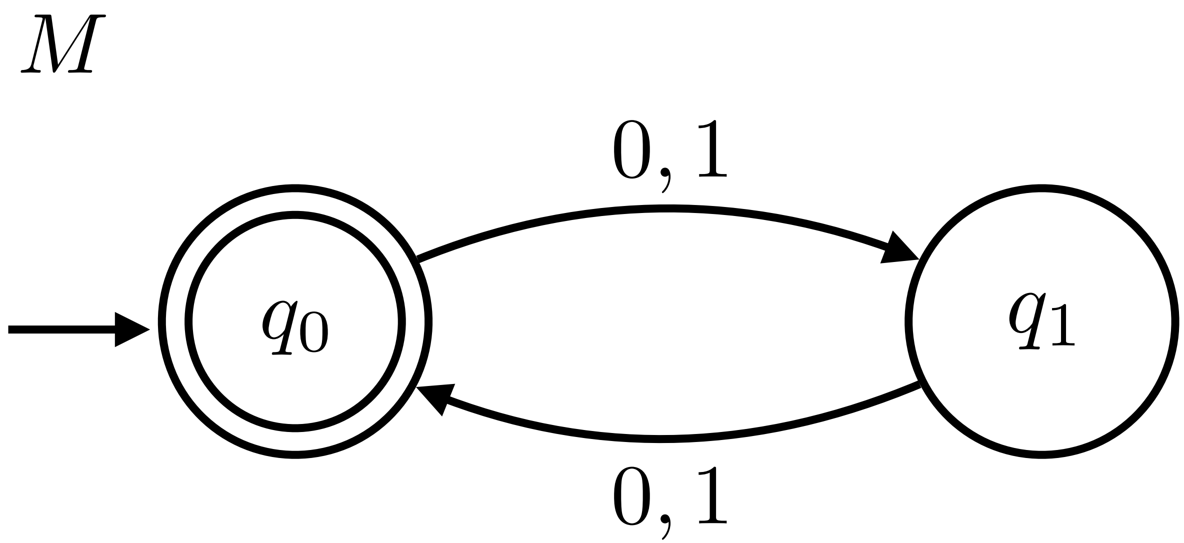
DFA Examples



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

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

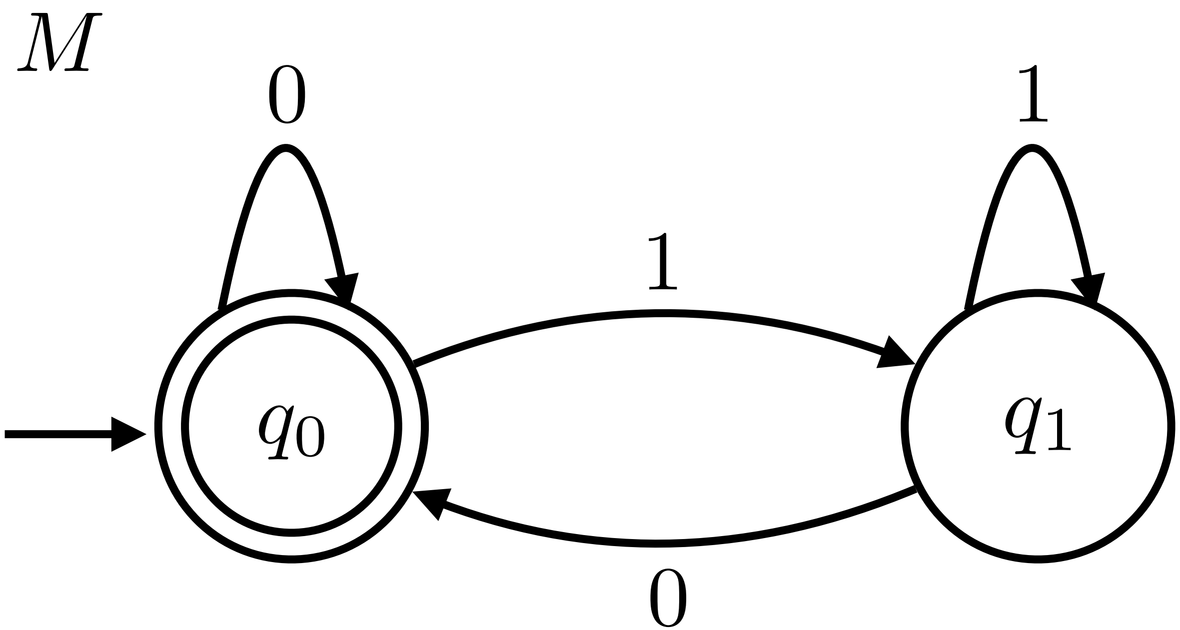
DFA Examples



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

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

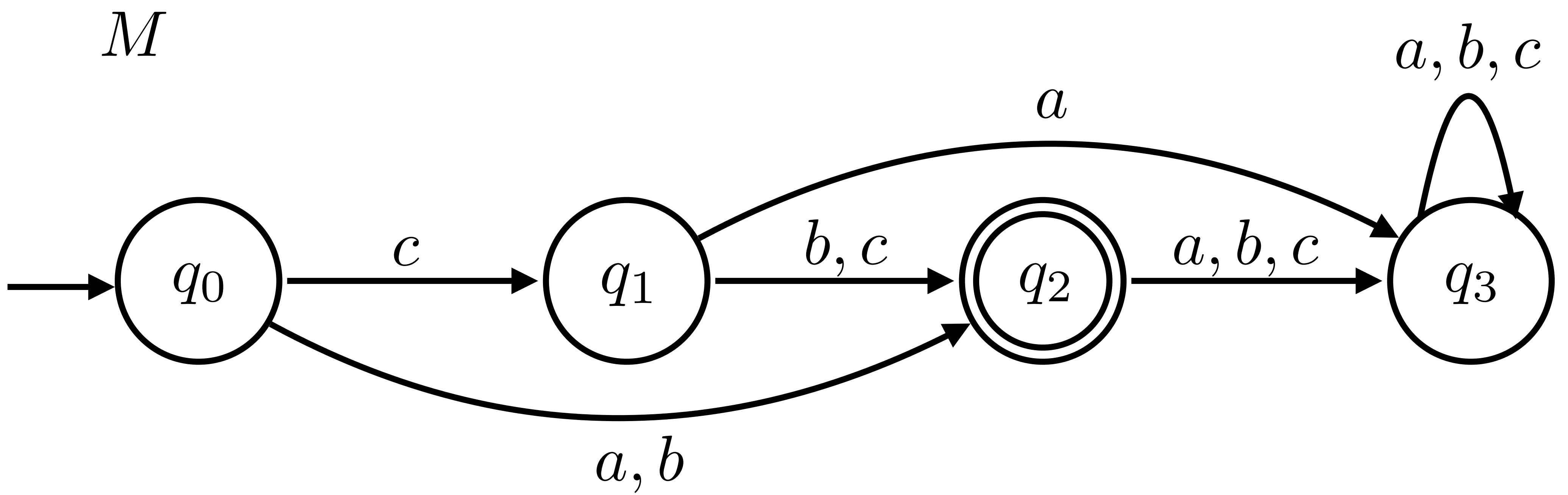
DFA Examples



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

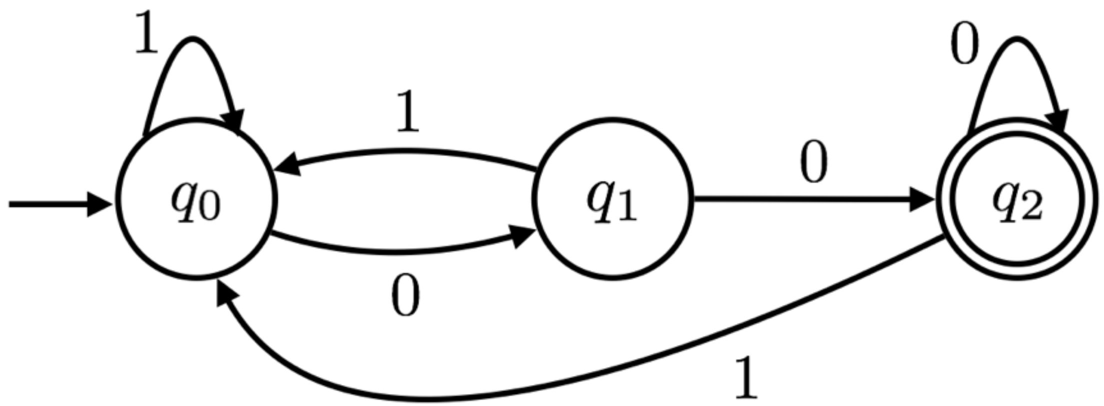
DFA Examples

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



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

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 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

None of the above

Beats me

DFA Examples

Draw a DFA that decides

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

Hint: How do you decide all strings that end with a 0 ?

How do you decide all strings that end with a 1 ?

DFA construction practice

$$L = \{110, 101\}$$

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

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

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

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

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

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

Formal definition: DFA

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

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

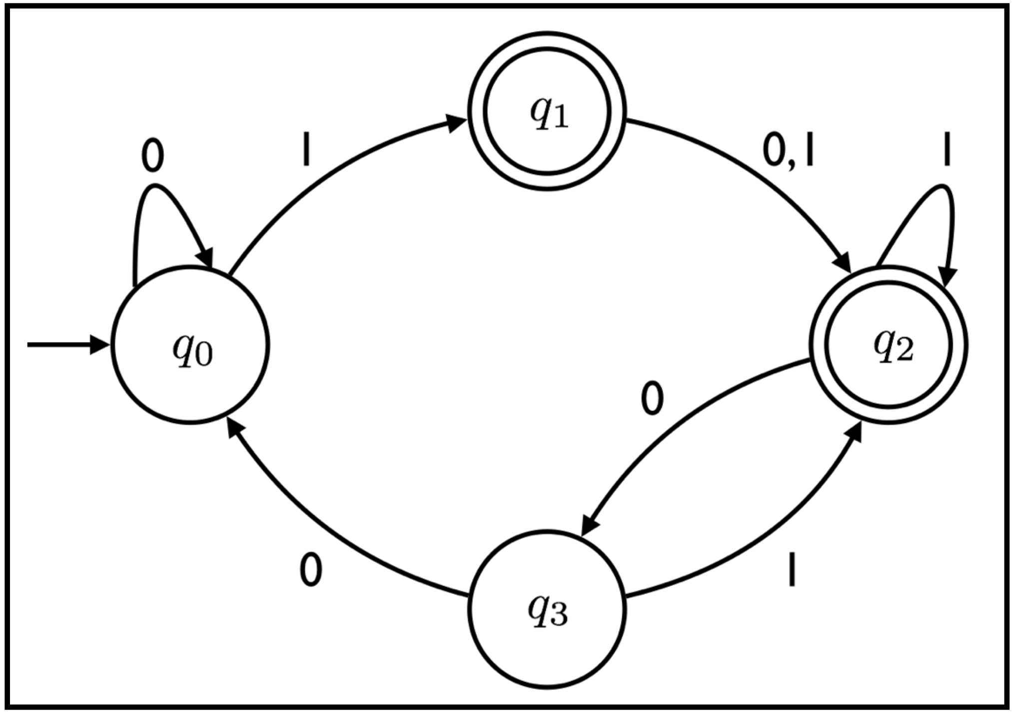
where

- Q is a finite set (which we call the set of states);
- Σ is a finite 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.

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:

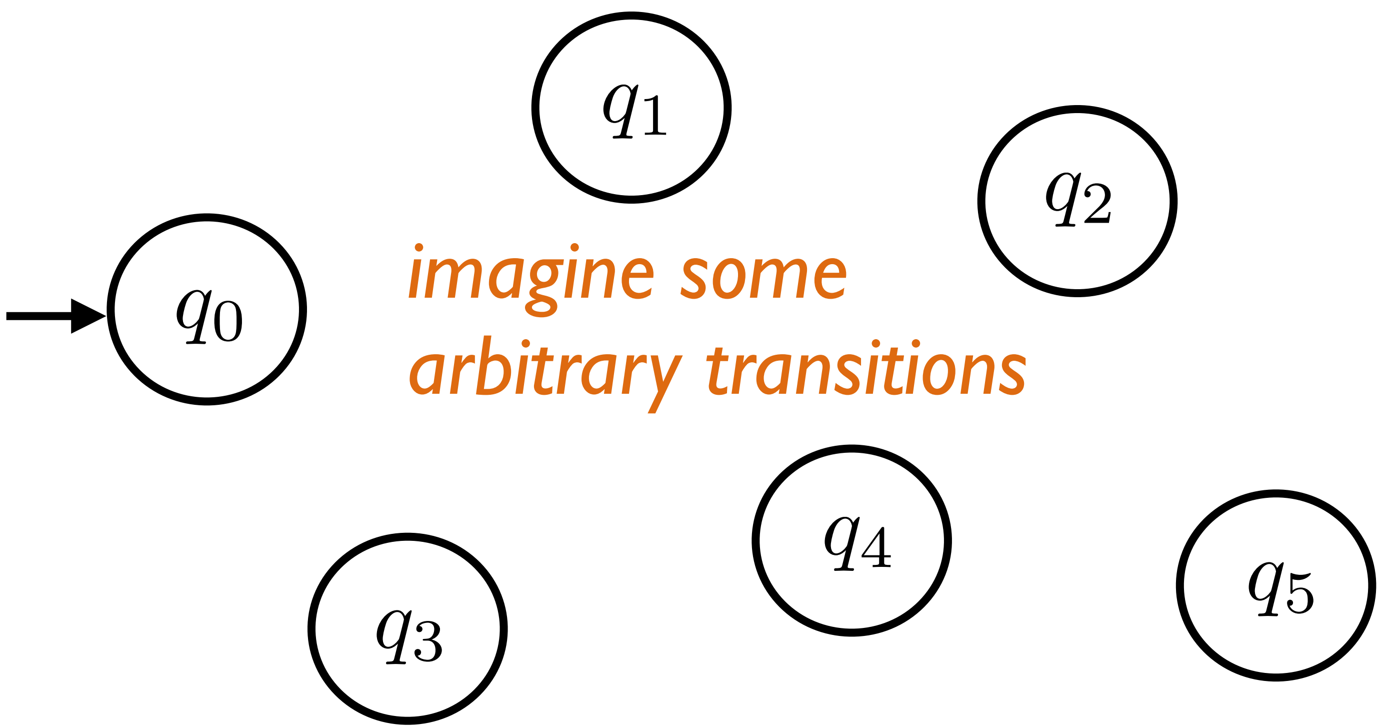
$L = \{x \in \{0,1\}^* : \text{10 and 01 occur equally often in } x.\}$ is regular!

A non-regular language

Warm-up:

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

Input: 0000000011111111

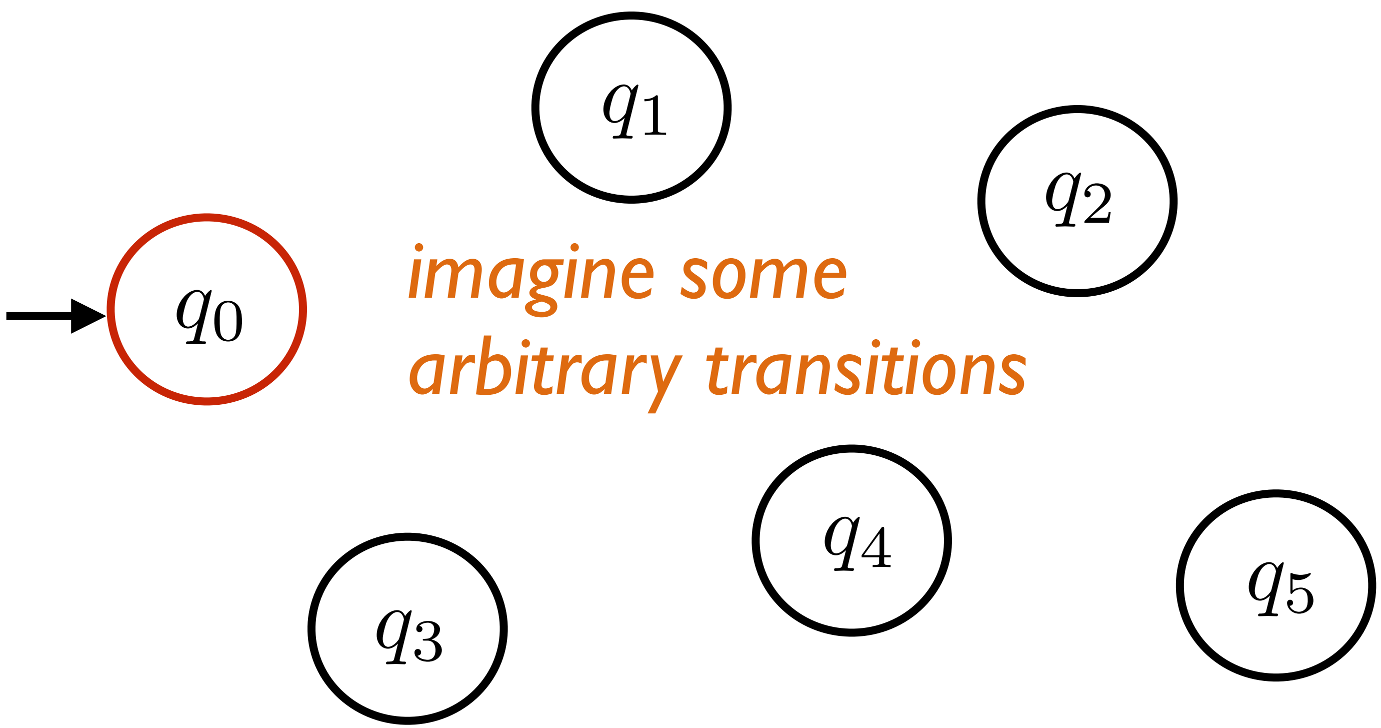


A non-regular language

Warm-up:

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

Input: 0000000011111111

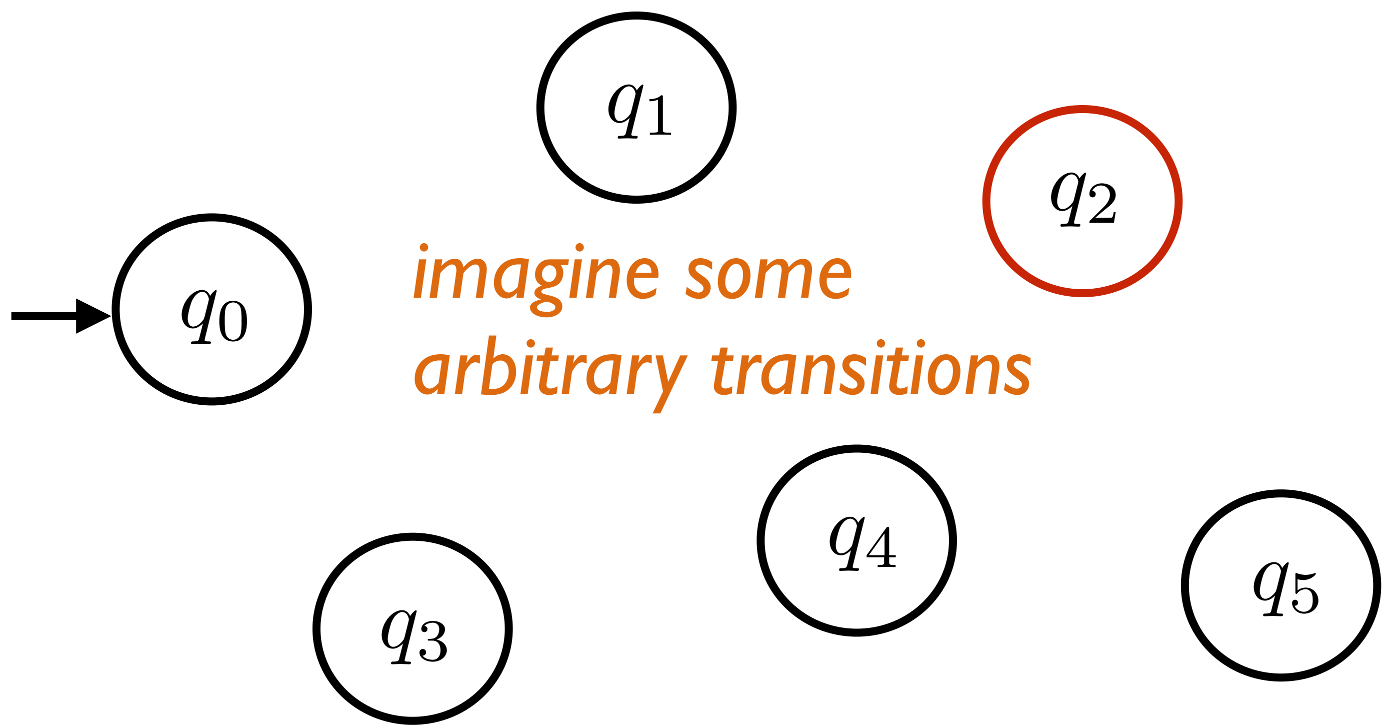


A non-regular language

Warm-up:

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

Input: 0000000011111111

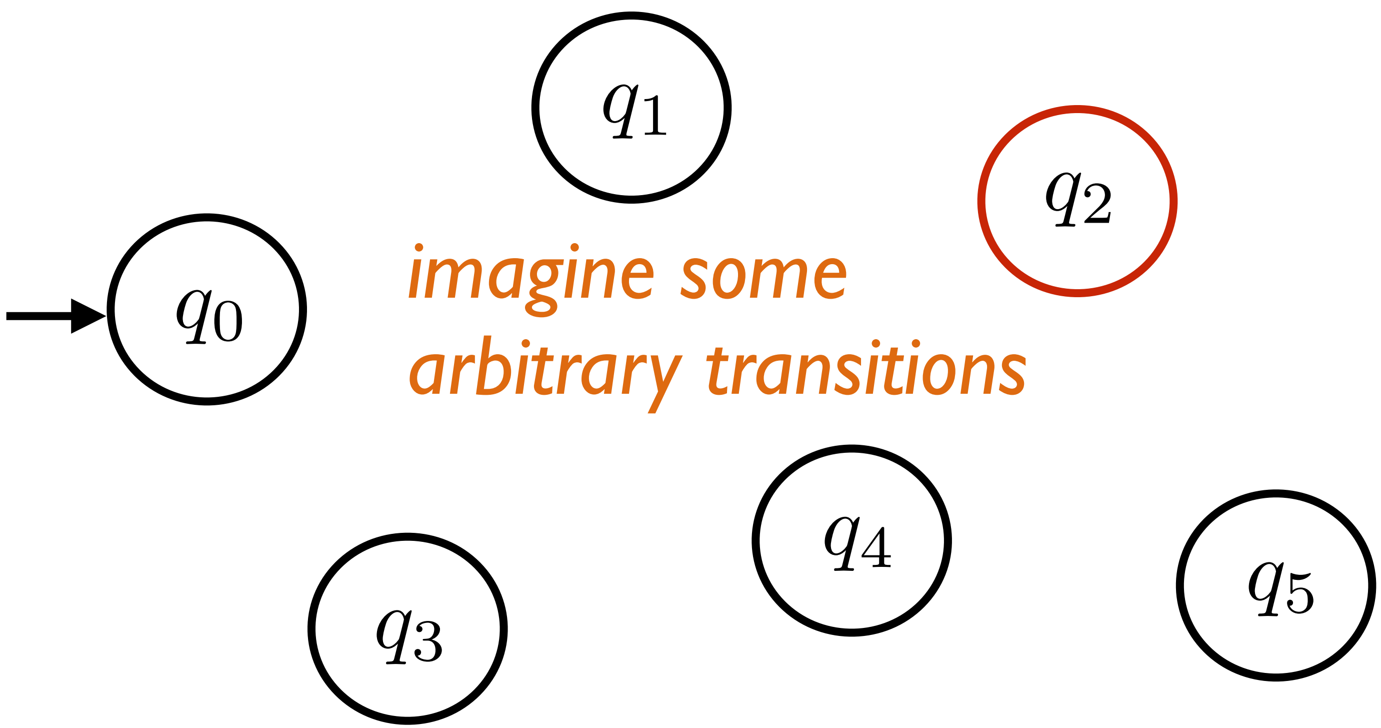


A non-regular language

Warm-up:

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

Input: 0000000011111111

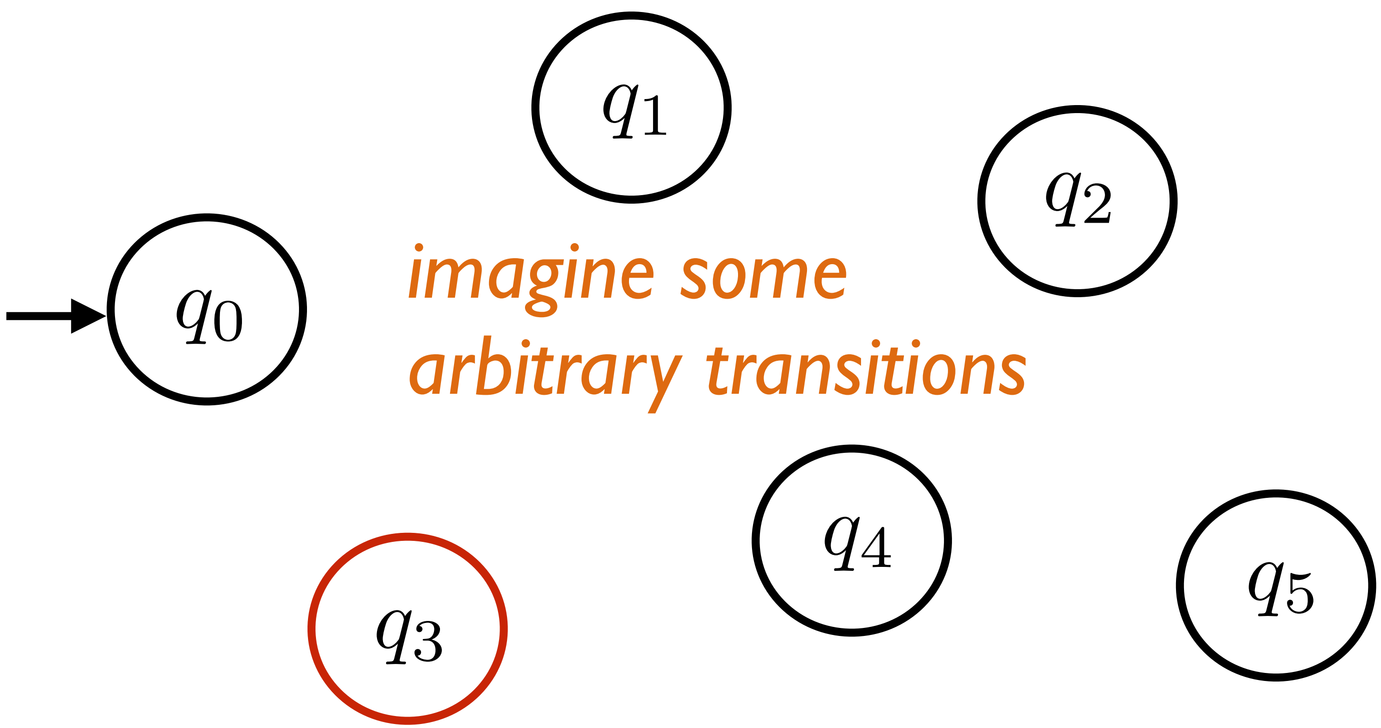


A non-regular language

Warm-up:

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

Input: 0000000011111111

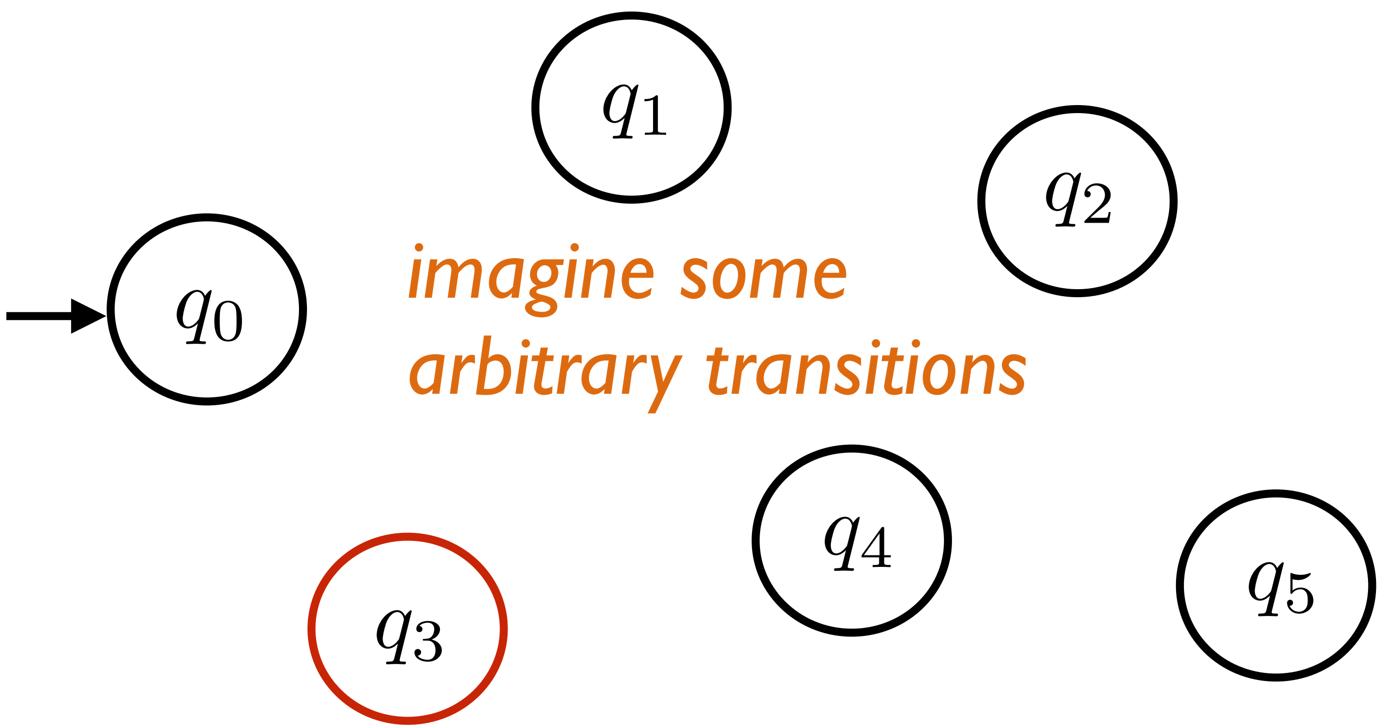


A non-regular language

Warm-up:

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

Input: 0000000011111111

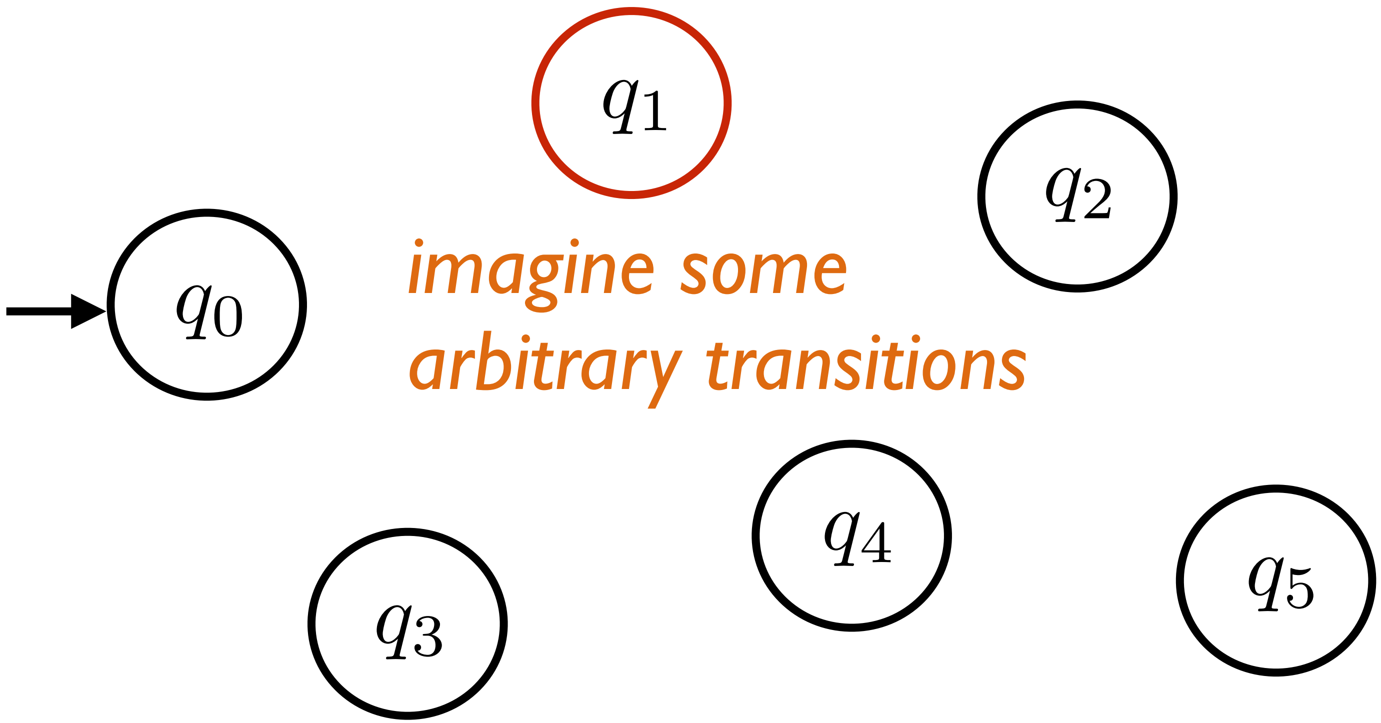


A non-regular language

Warm-up:

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

Input: 0000000011111111

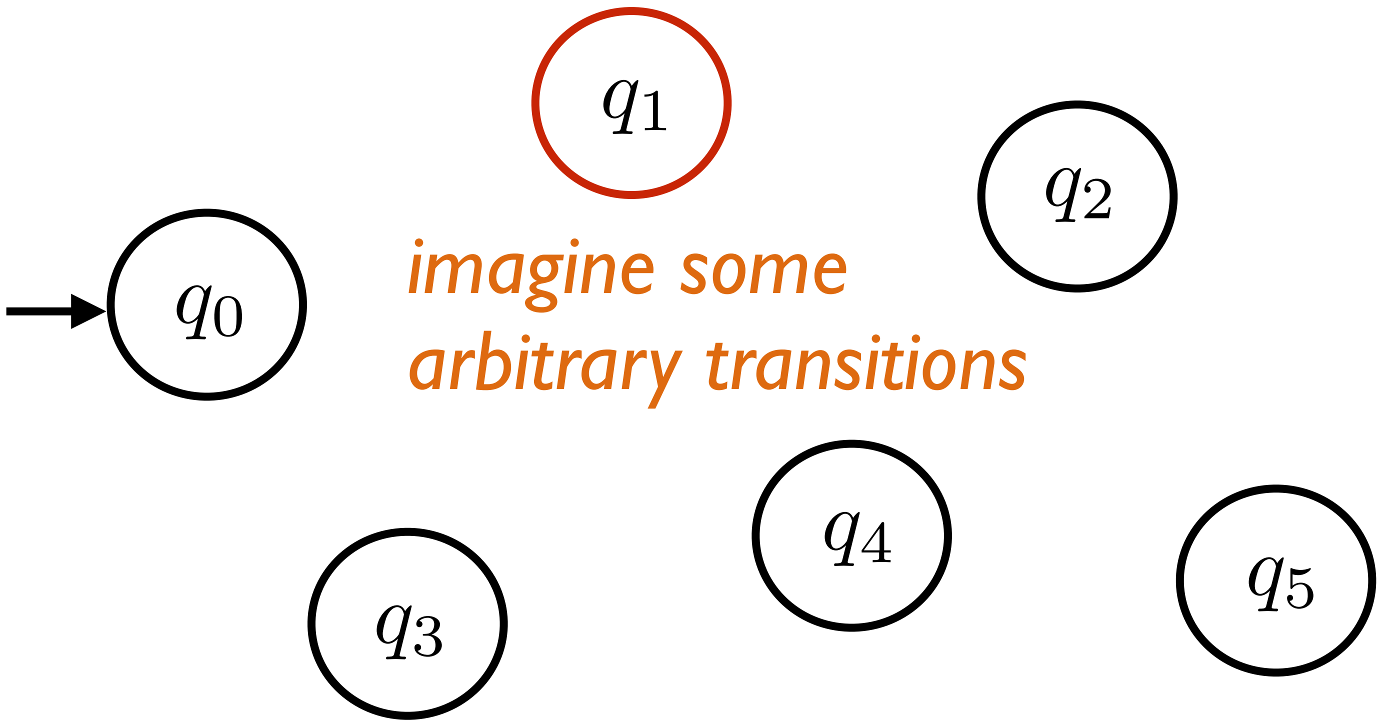


A non-regular language

Warm-up:

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

Input: 0000000011111111

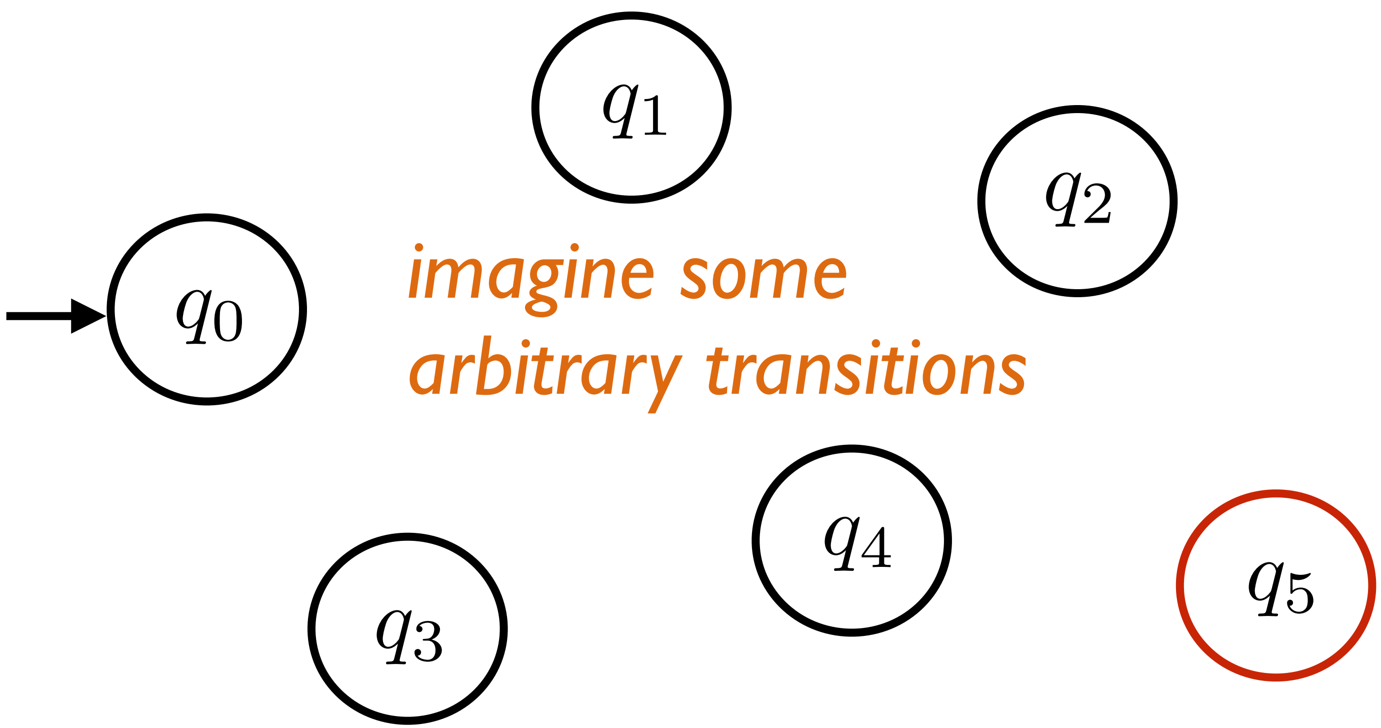


A non-regular language

Warm-up:

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

Input: 0000000011111111

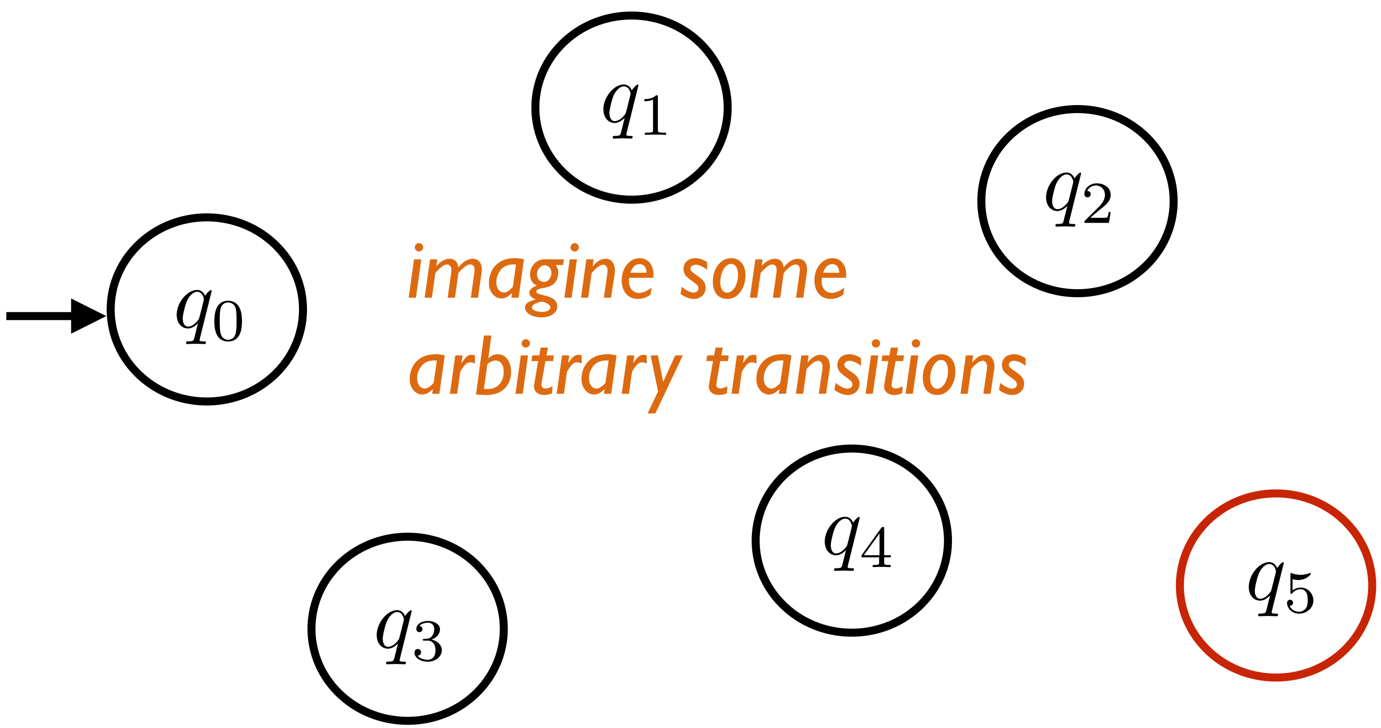


A non-regular language

Warm-up:

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

Input: 0000000011111111

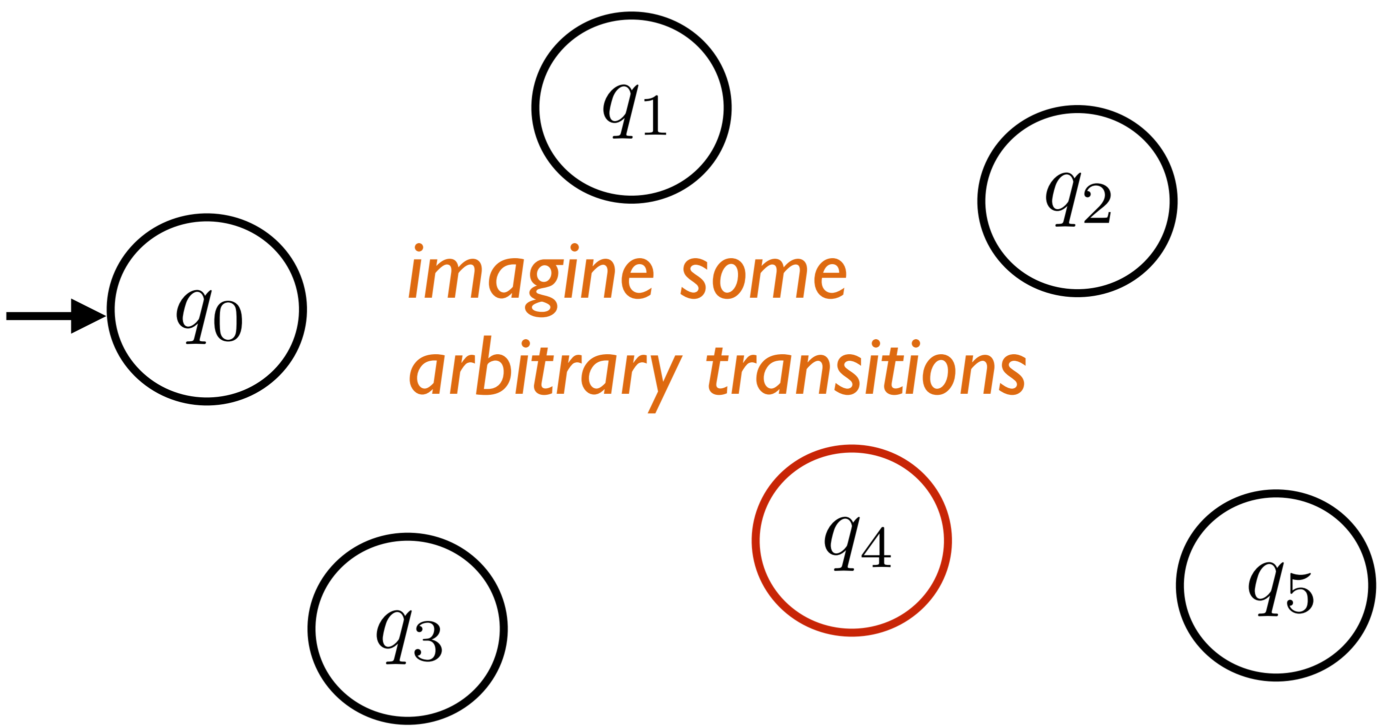


A non-regular language

Warm-up:

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

Input: 0000000011111111

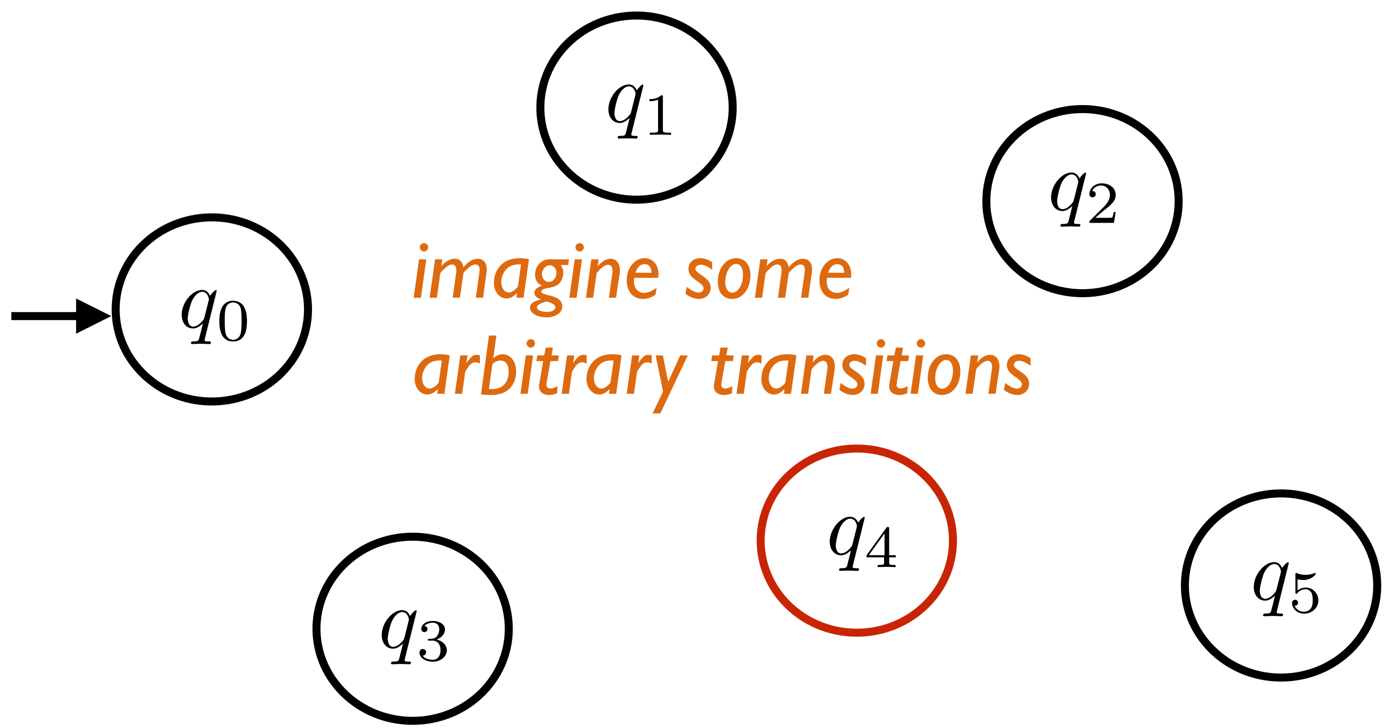


A non-regular language

Warm-up:

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

Input: 0000000011111111

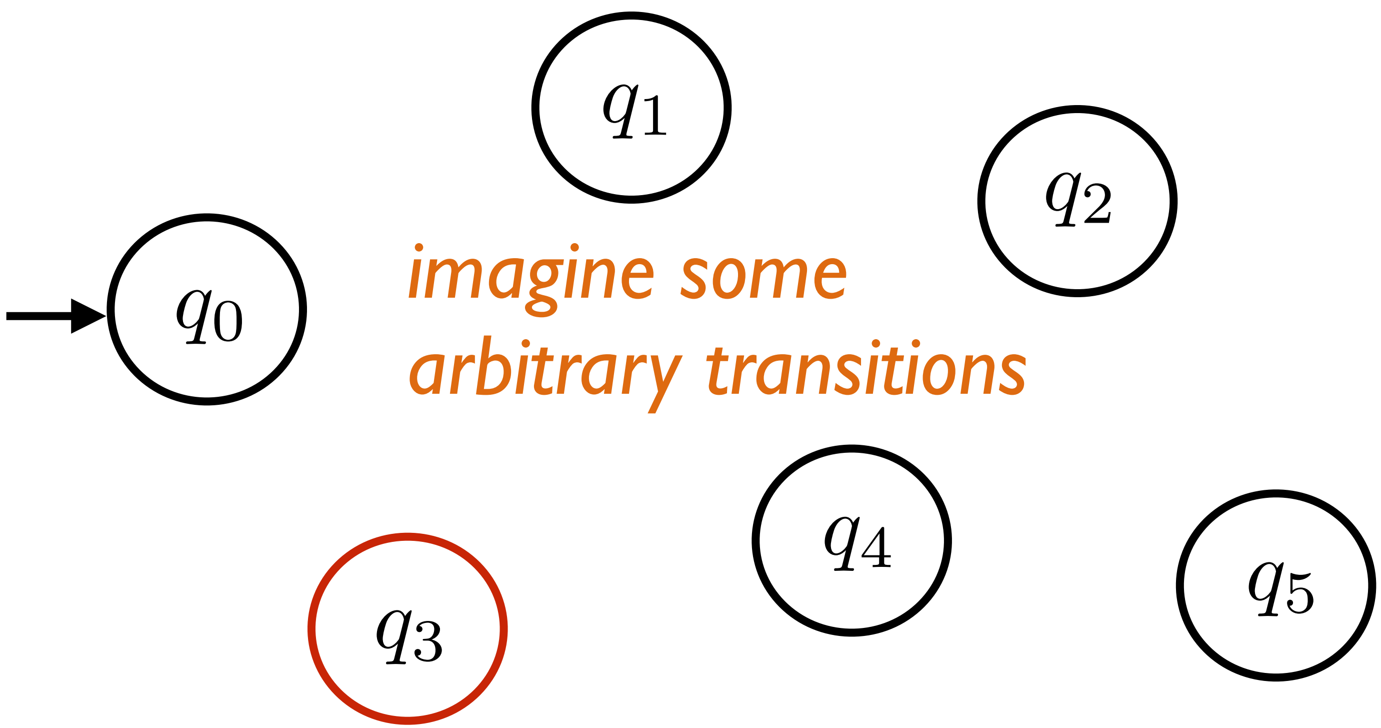


A non-regular language

Warm-up:

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

Input: 0000000011111111

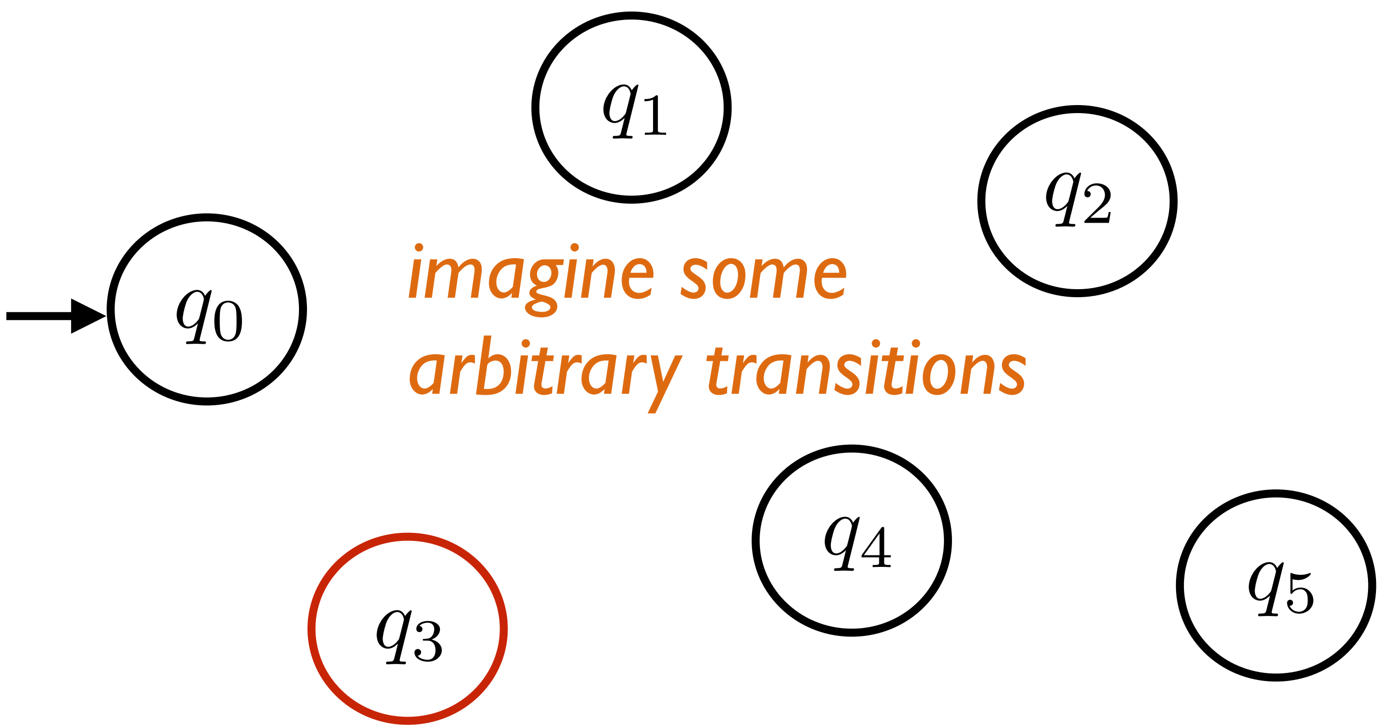


A non-regular language

Warm-up:

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

Input: 0000000011111111

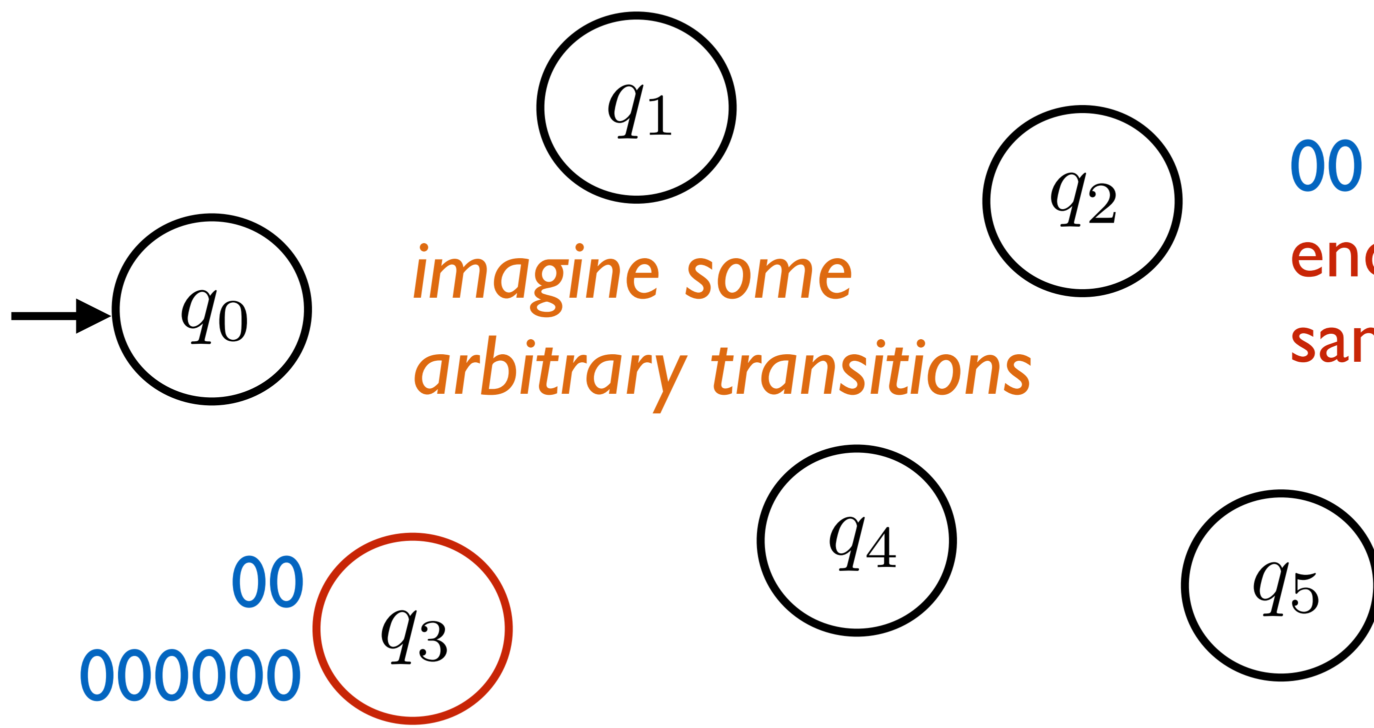


A non-regular language

Warm-up:

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

Input: 0000000011111111



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

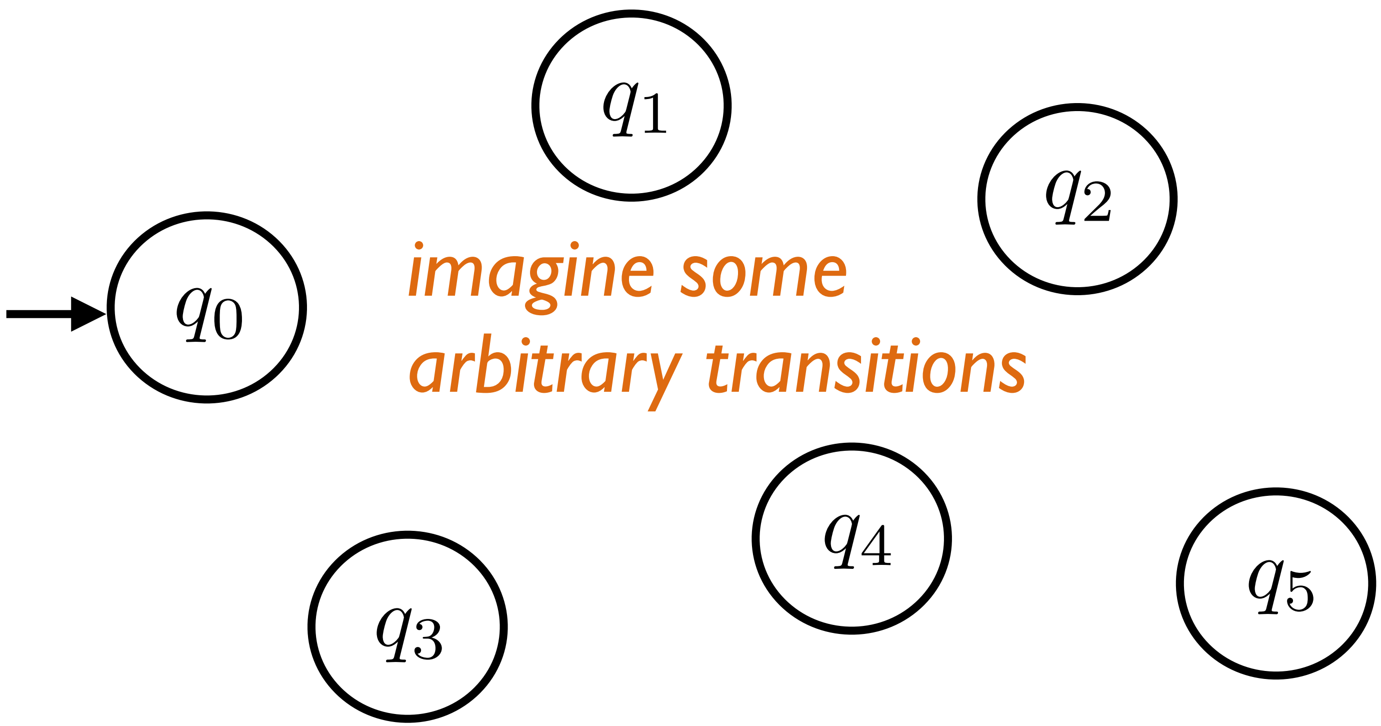
Warm-up:

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.

So 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:

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.

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

Regular languages

All languages

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

Regular languages

$\{110, 101\}$

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

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

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

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

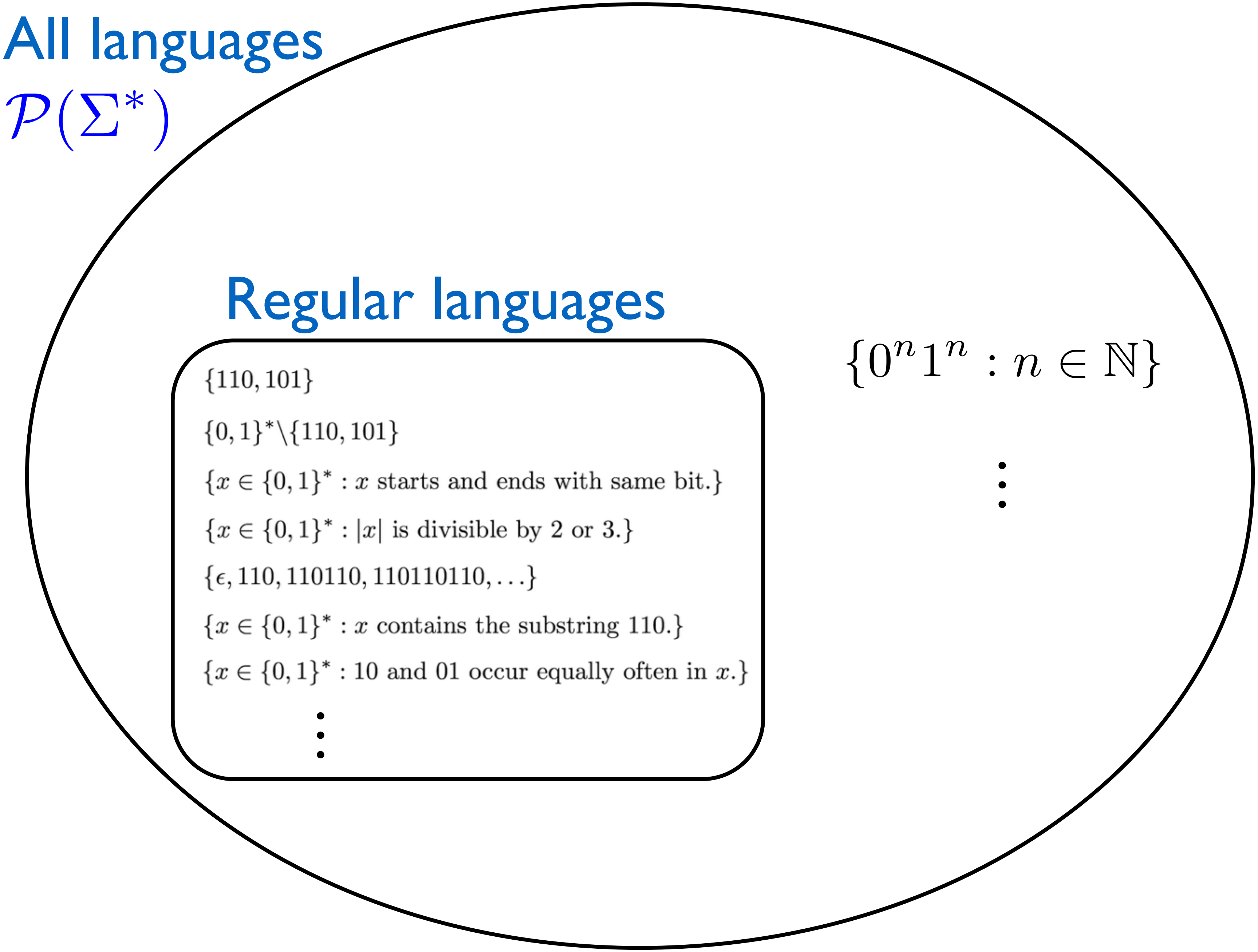
$\{x \in \{0,1\}^* : x \text{ contains the substring 110.}\}$

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

$\vdots$

?

Regular languages



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?

Closure properties or regular languages

Closed under union:

L_1, L_2 regular $\implies$ $L_1 \cup L_2$ regular.

$$L_1 \cup L_2 = \{x \in \Sigma^* : x \in L_1 \text{ or } x \in L_2\}$$

Closed under concatenation:

L_1, L_2 regular $\implies$ $L_1 \cdot L_2$ regular.

$$L_1 \cdot L_2 = \{xy : x \in L_1, y \in L_2\}$$

Closed under star:

L regular $\implies$ L^* regular.

$$L^* = \{x_1 x_2 \cdots x_k : k \geq 0, \forall i \; x_i \in L\}$$

Closure properties or regular languages

Fact:

Can define regular languages inductively as follows:

- $\emptyset$ is regular.
- For every $a \in \Sigma$, $\{a\}$ is regular.
- L_1, L_2 regular $\implies$ $L_1 \cup L_2$ regular.
- L_1, L_2 regular $\implies$ $L_1 \cdot L_2$ regular.
- L regular $\implies$ L^* regular.

Regular expression:

$$a(a \cup b)^* a \cup b(a \cup b)^* b \cup a \cup b$$

Regular languages are closed under union

Theorem:

Let Σ be some finite alphabet.
If $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$ are regular, then so is $L_1 \cup L_2$.

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be the decider for L_1 and $M' = (Q', \Sigma, \delta', q_0', F')$ be the decider for L_2.
We construct a DFA $M'' = (Q'', \Sigma, \delta'', q_0'', F'')$ that decides $L_1 \cup L_2$, as follows:

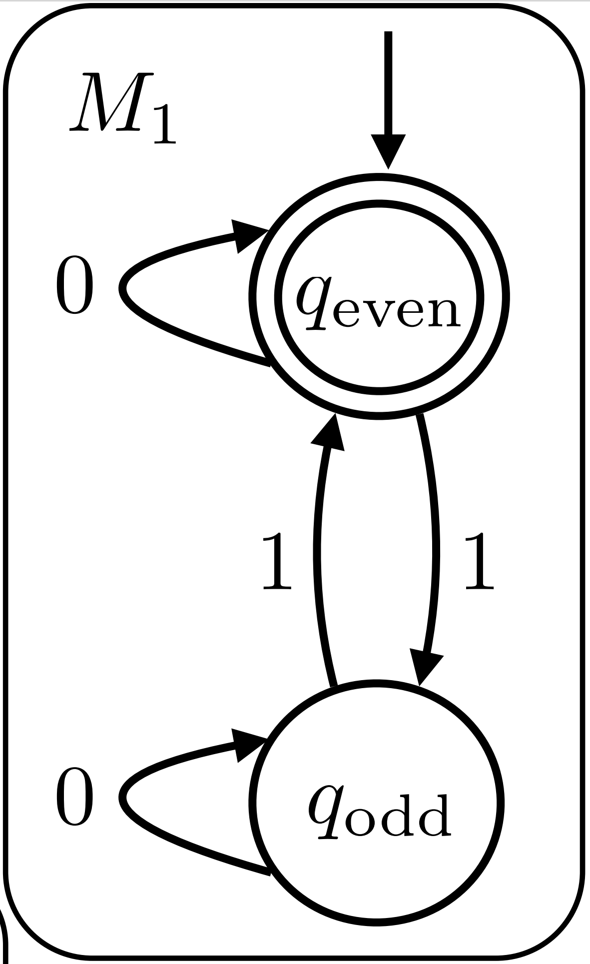
$\vdots$

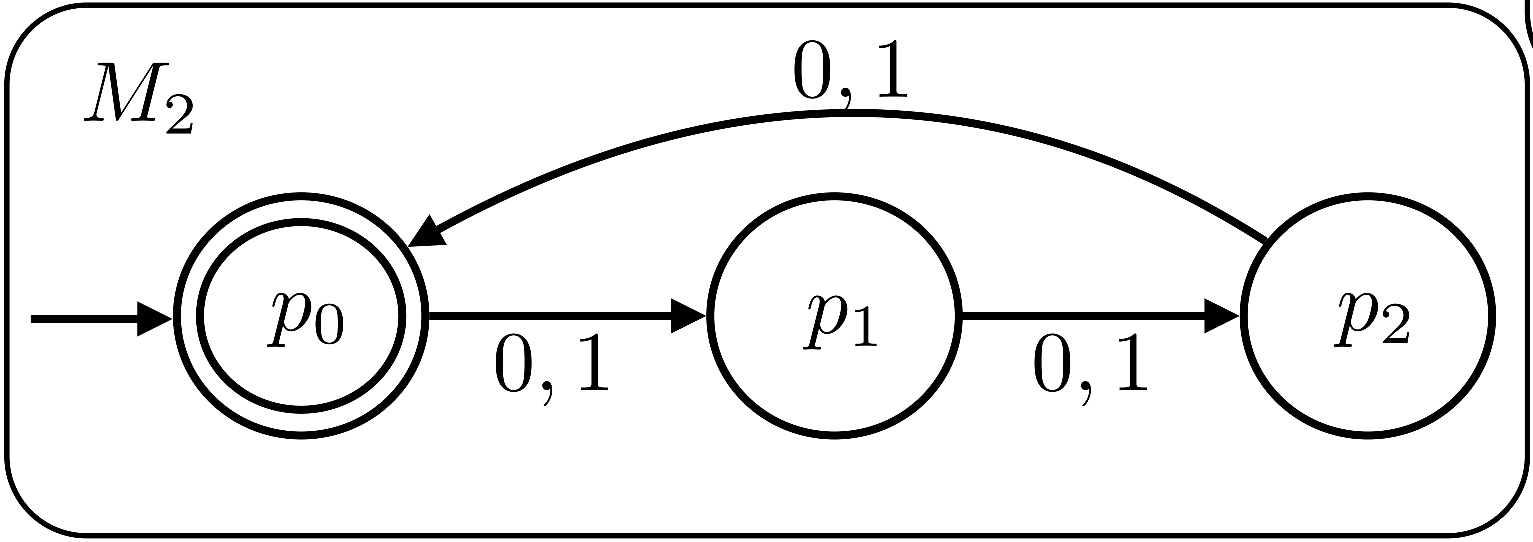
Regular languages are closed under union

Example

$L_1 =$ strings with even number of 1's

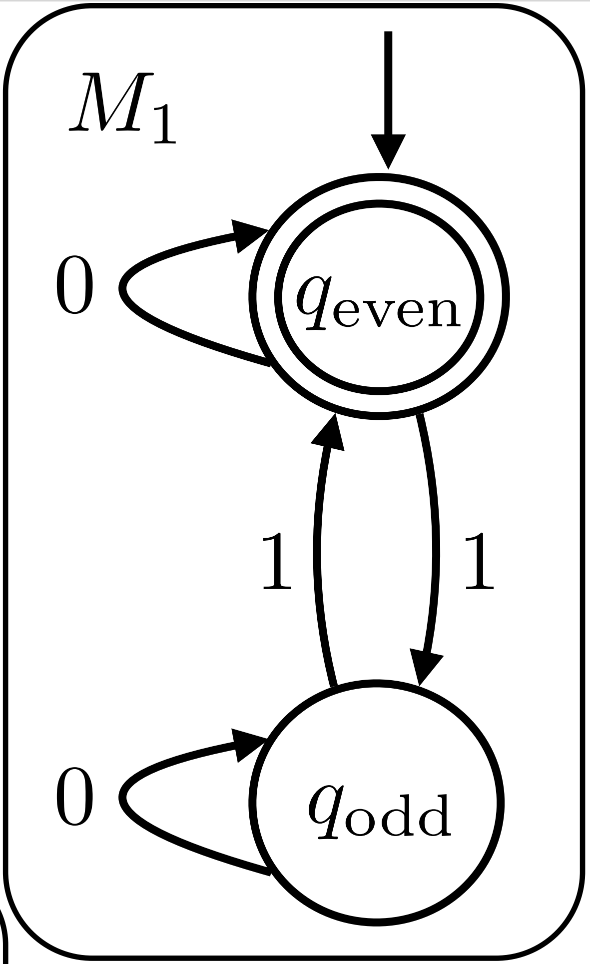
$L_2 =$ strings with length divisible by 3.

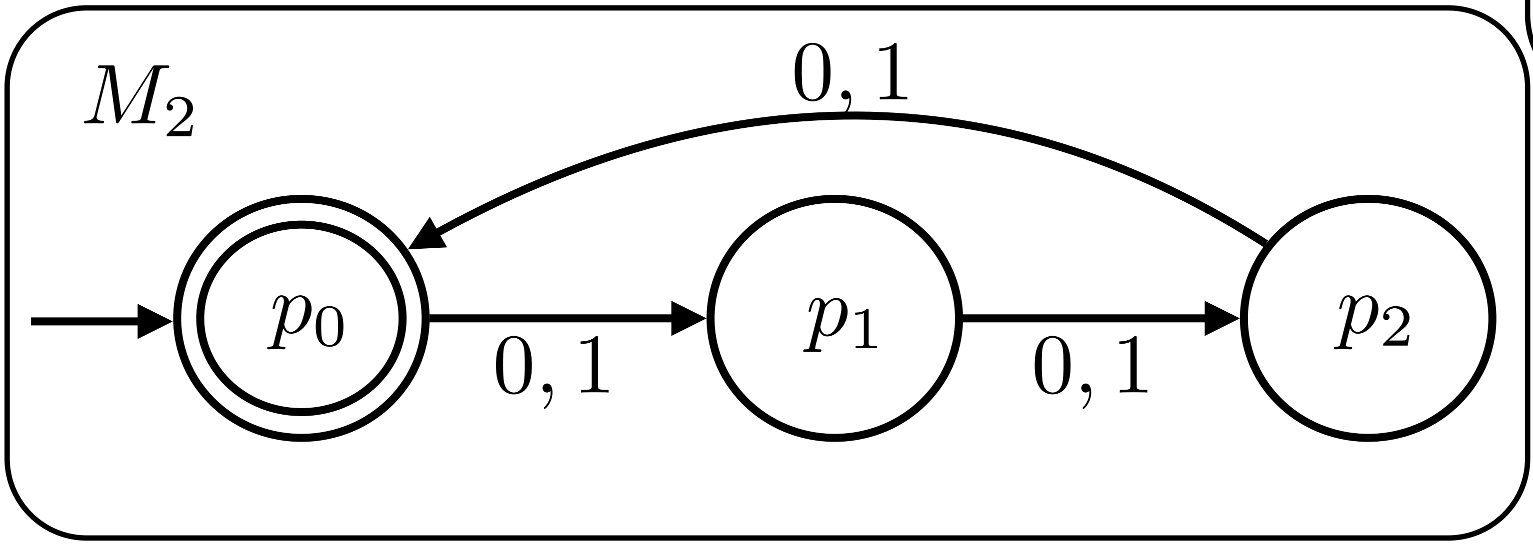




Regular languages are closed under union

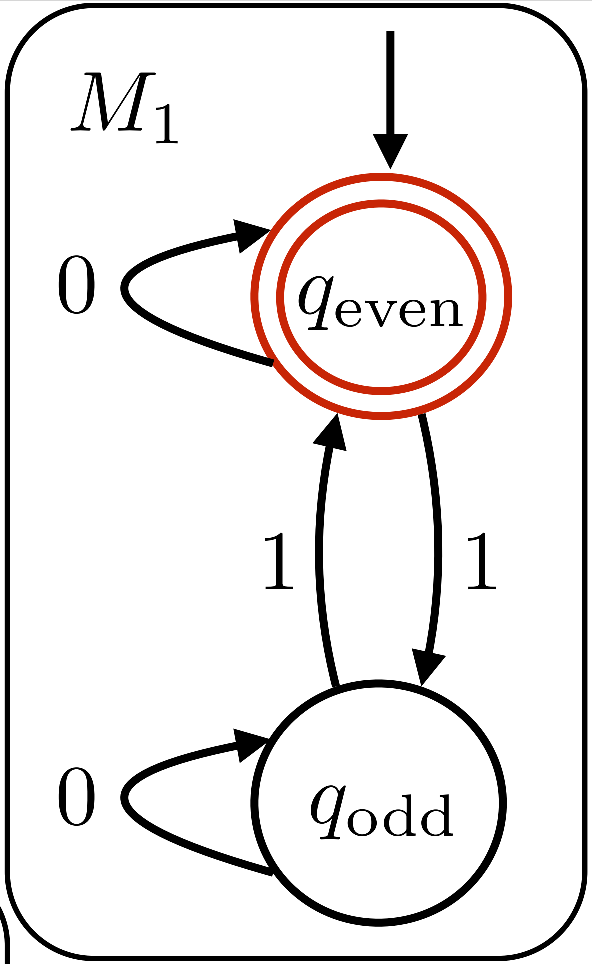
Input: 101001

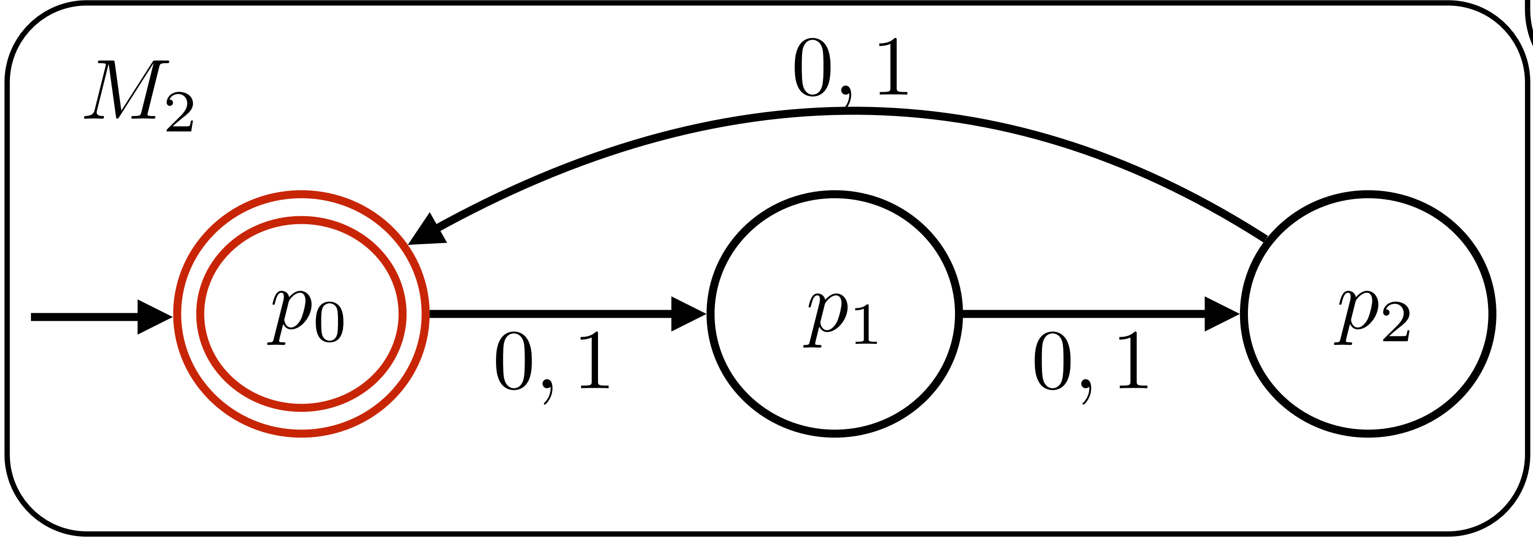




Regular languages are closed under union

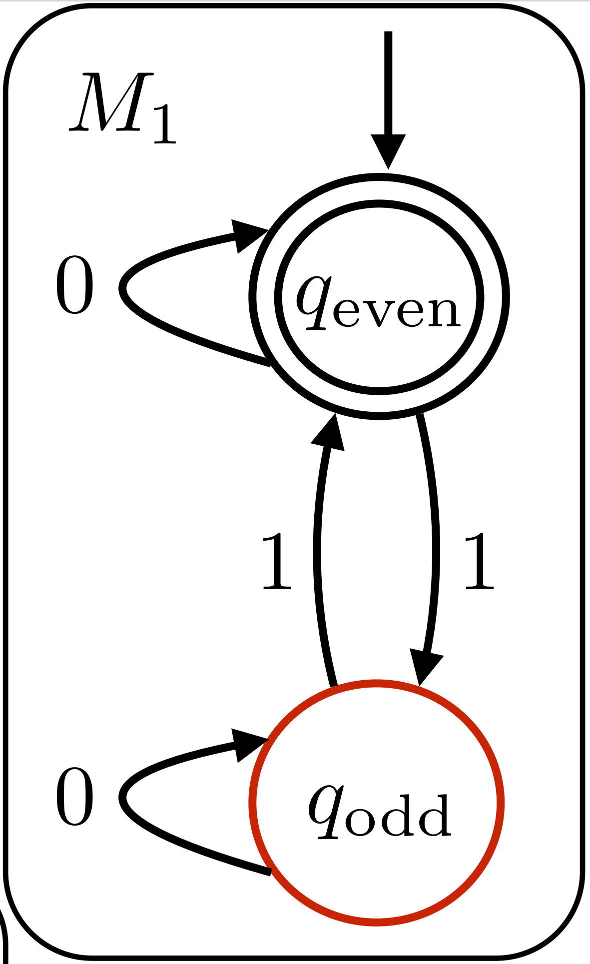
Input: 101001

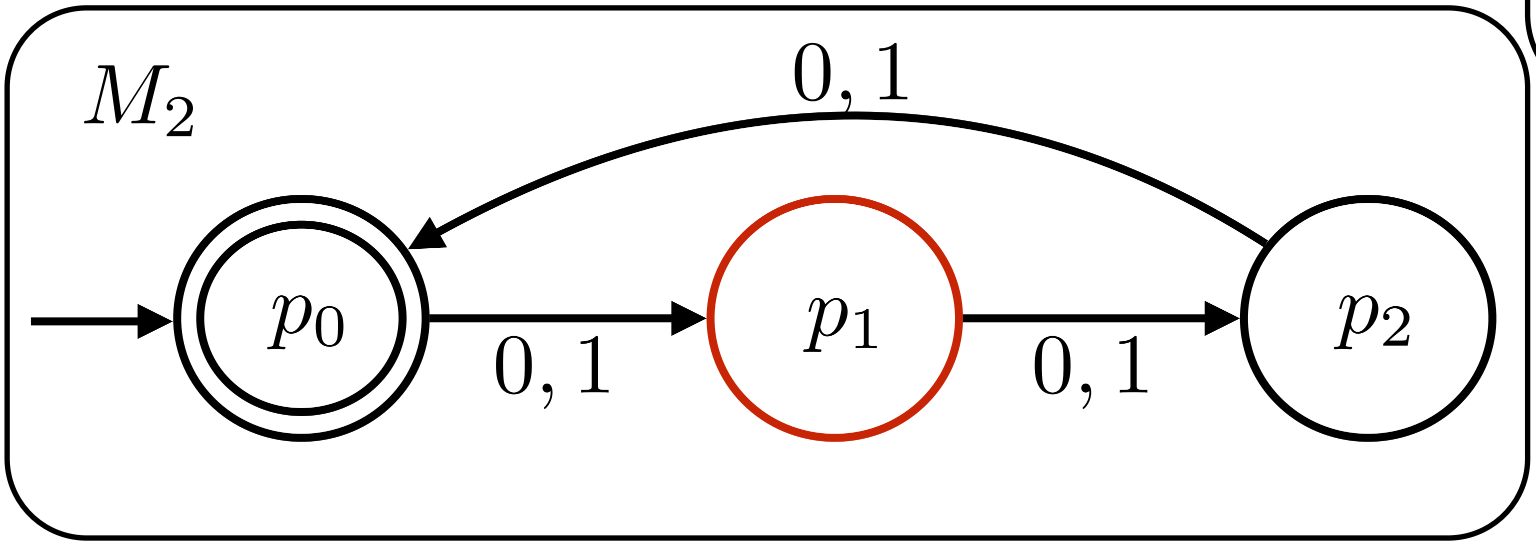




Regular languages are closed under union

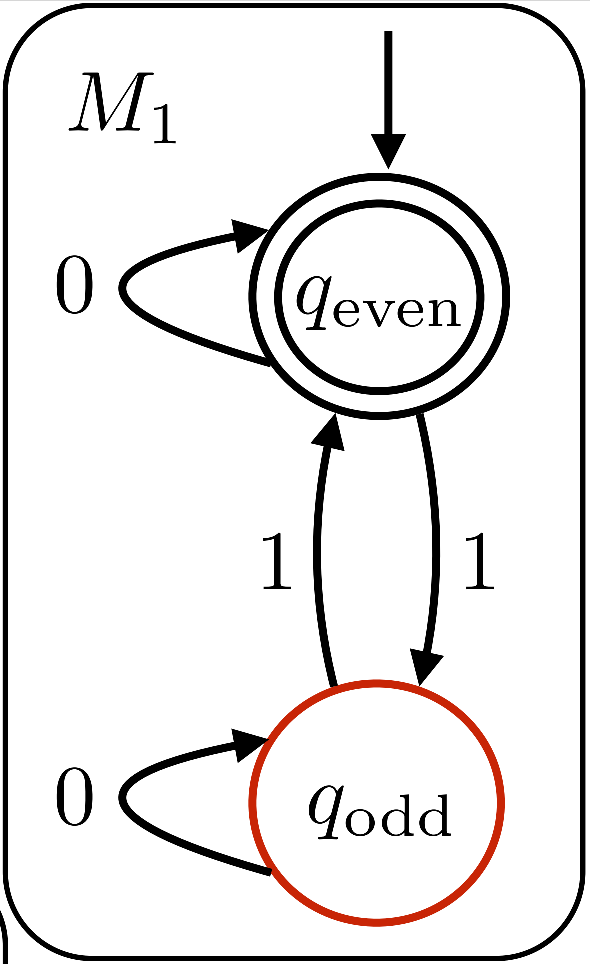
Input: 101001

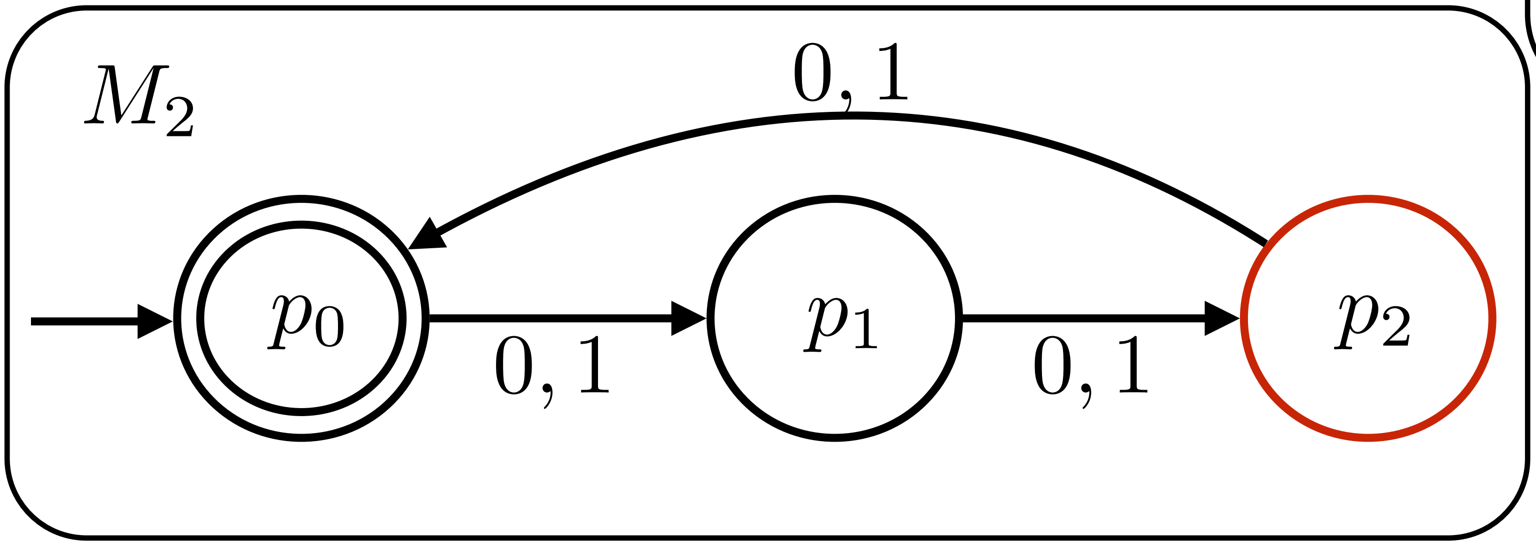




Regular languages are closed under union

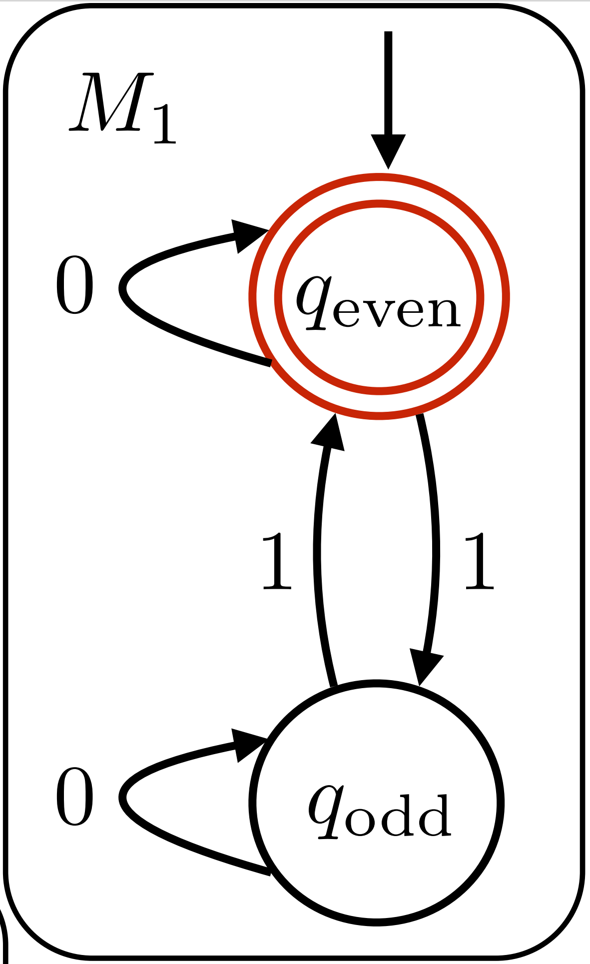
Input: 101001

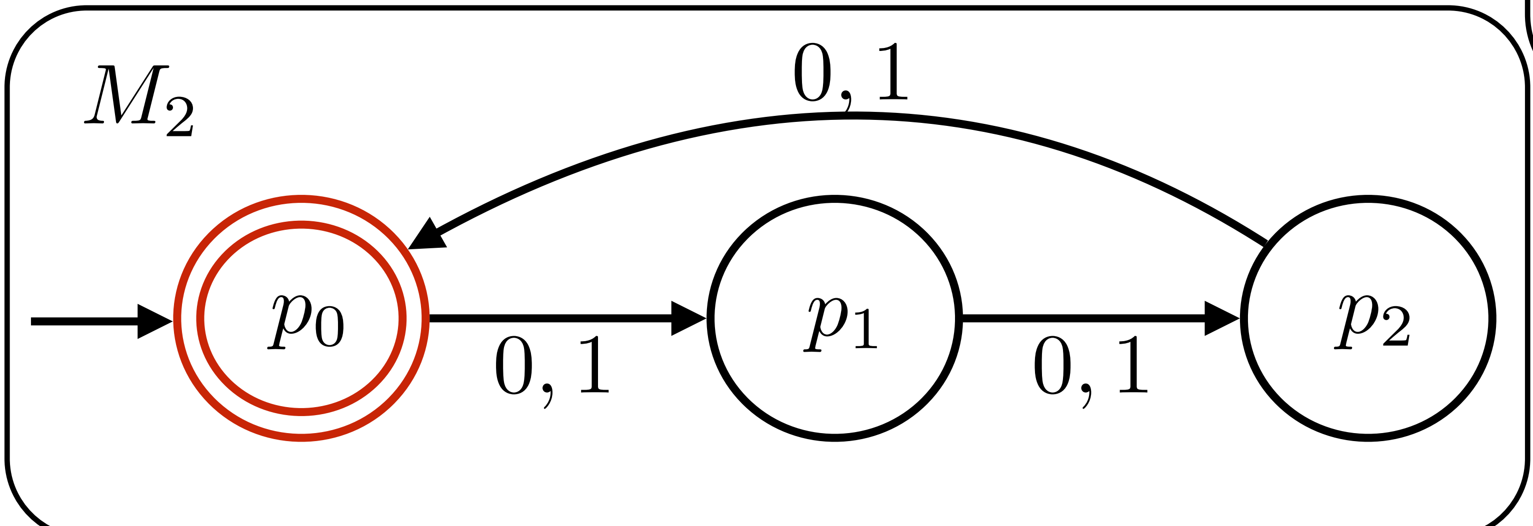




Regular languages are closed under union

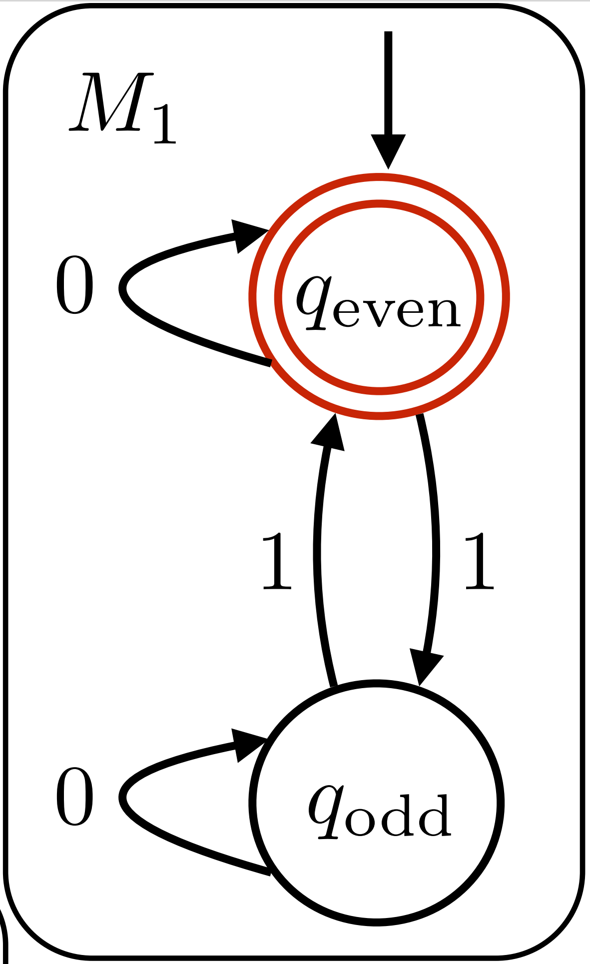
Input: 101001

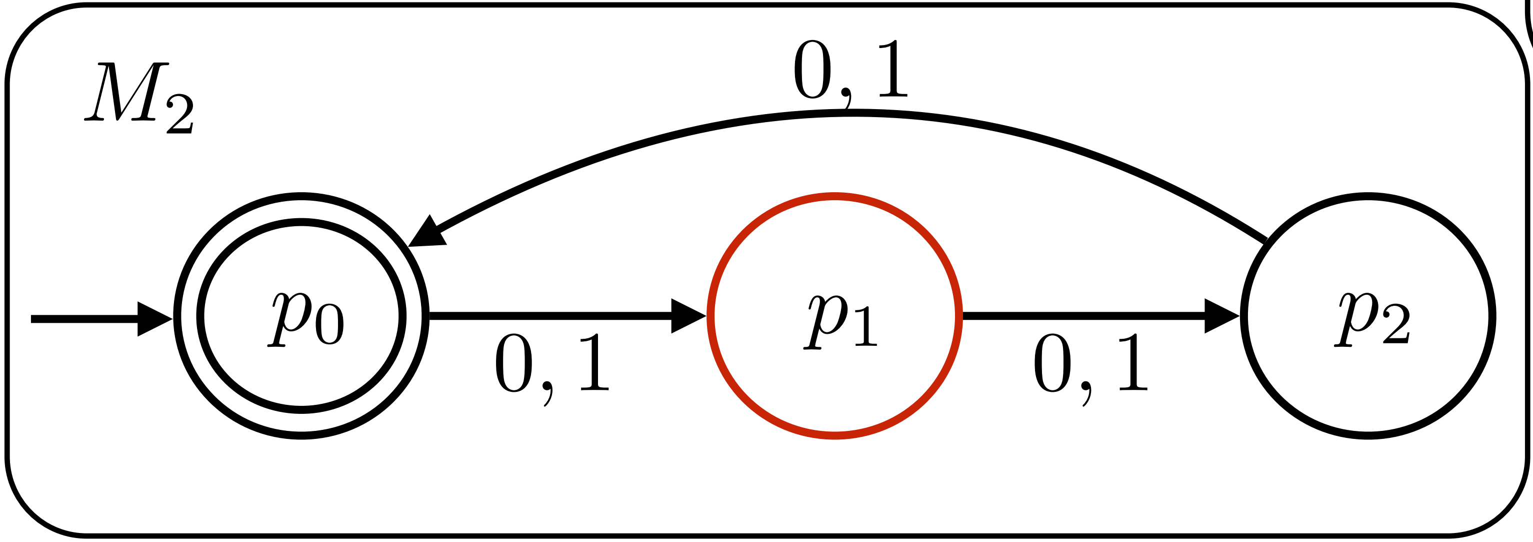




Regular languages are closed under union

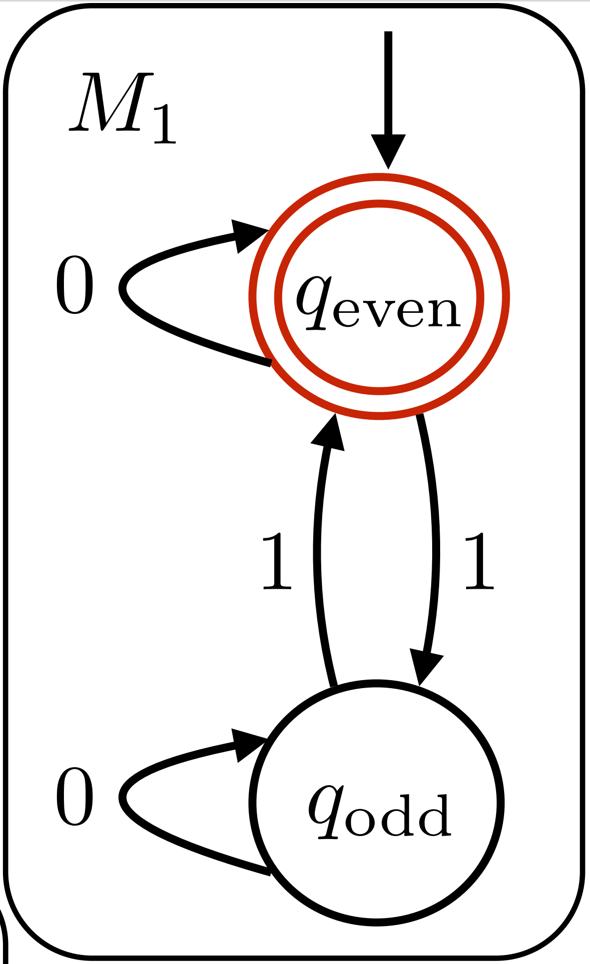
Input: 101001

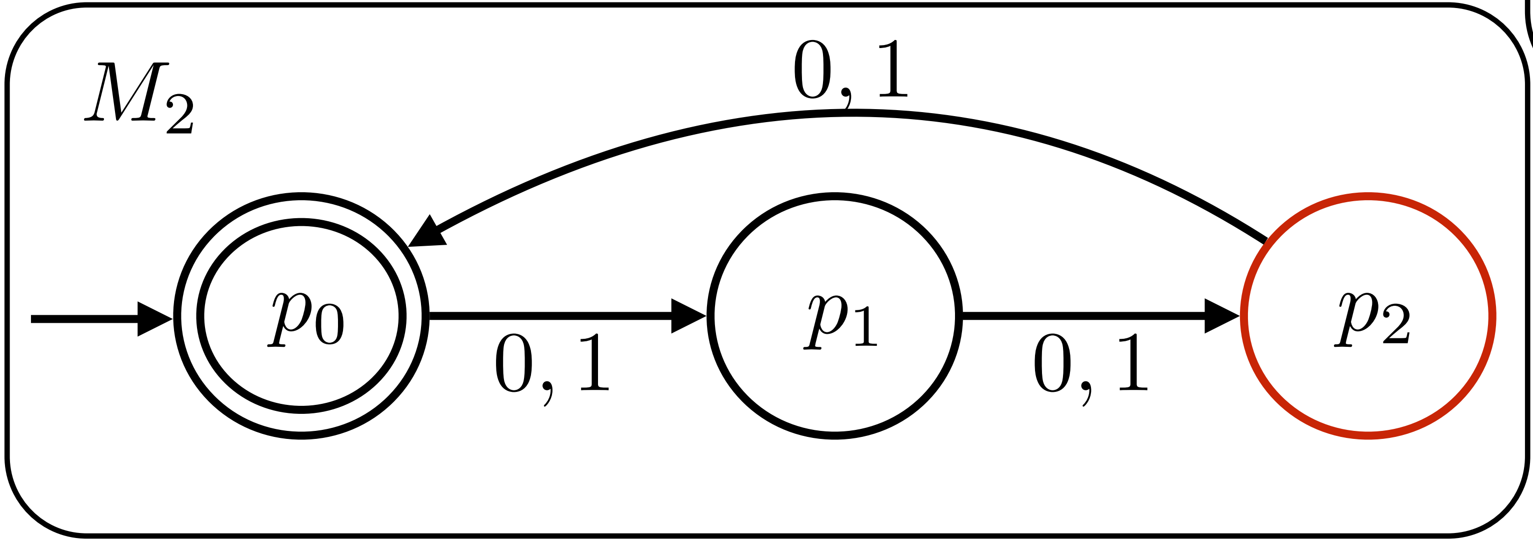




Regular languages are closed under union

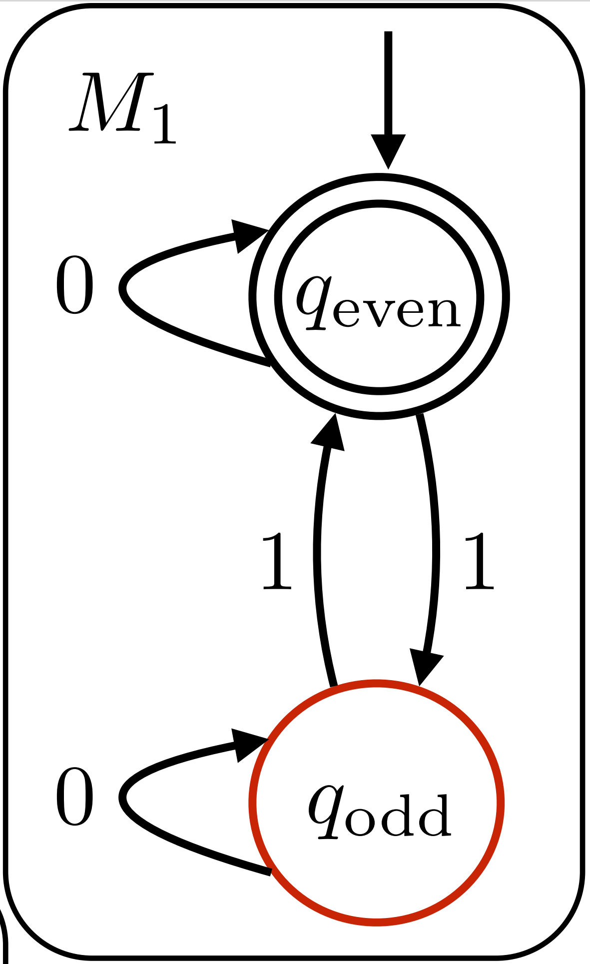
Input: 101001

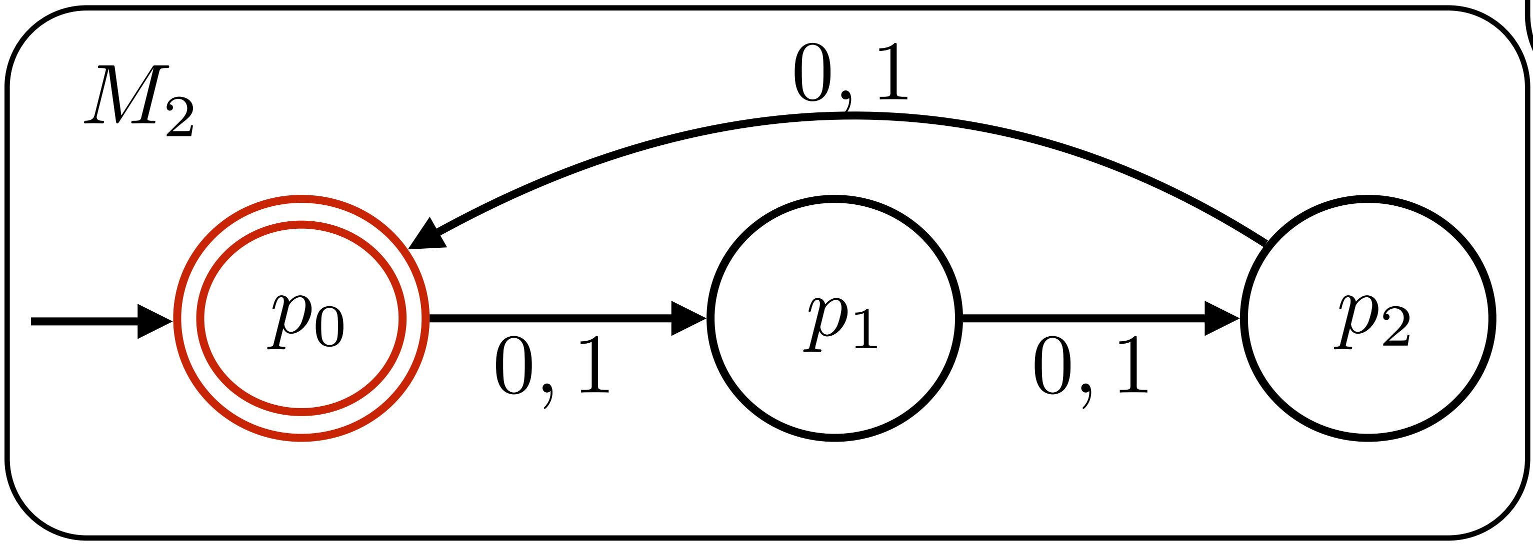




Regular languages are closed under union

Input: 101001

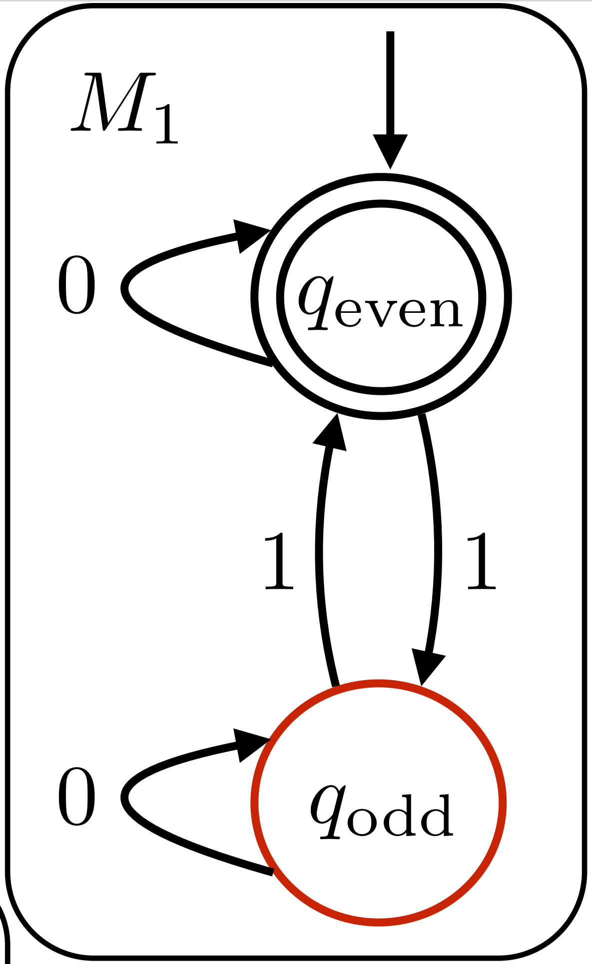


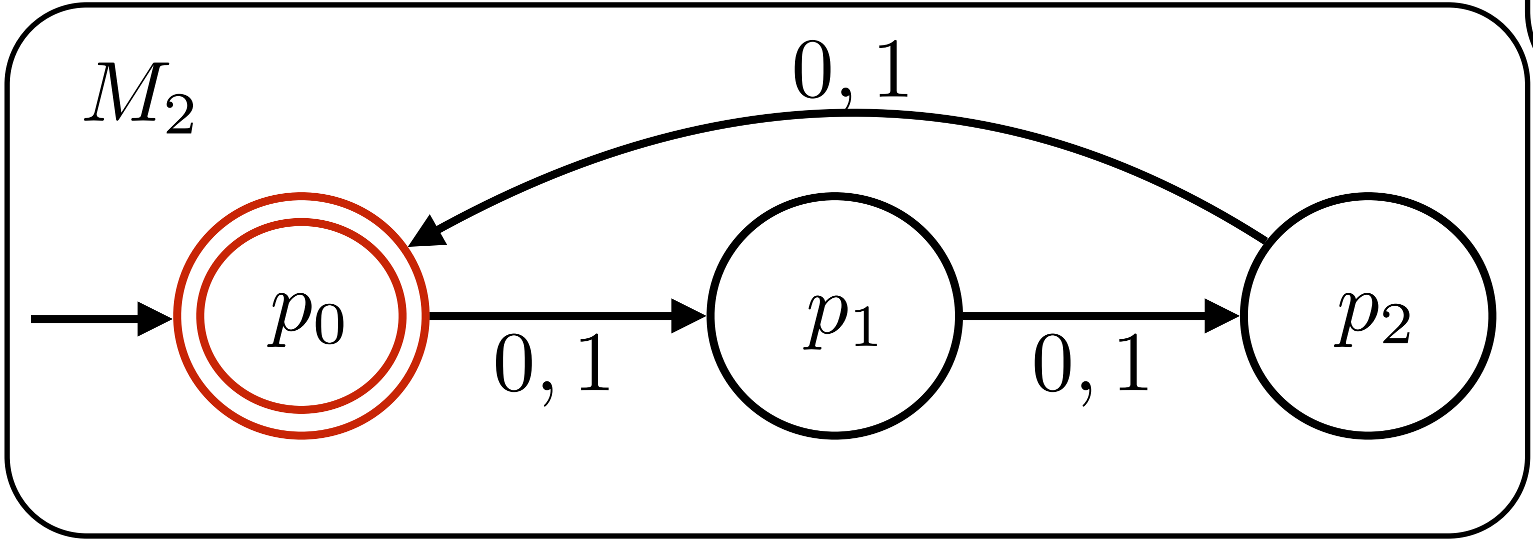


Regular languages are closed under union

Input: 101001

Accept

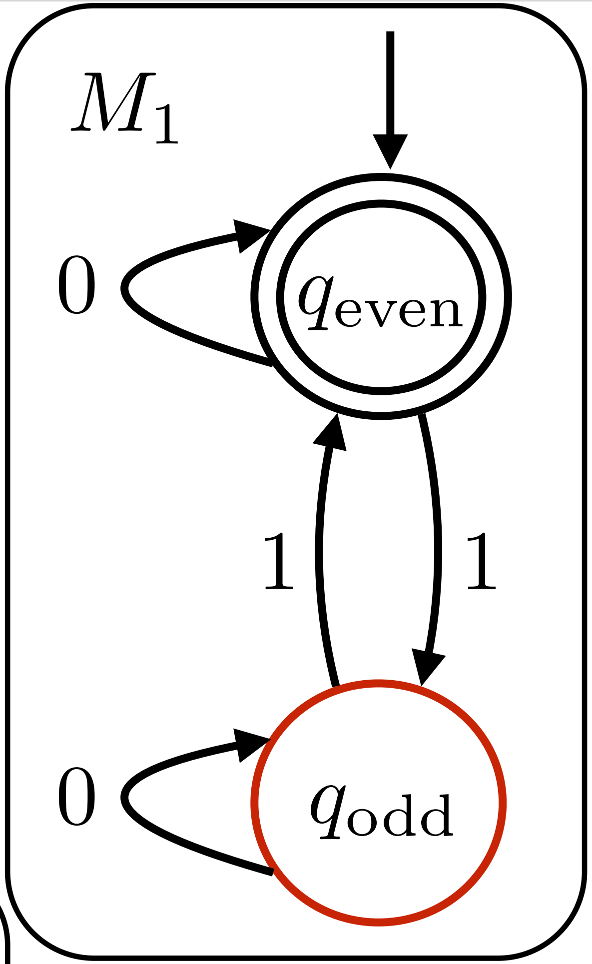


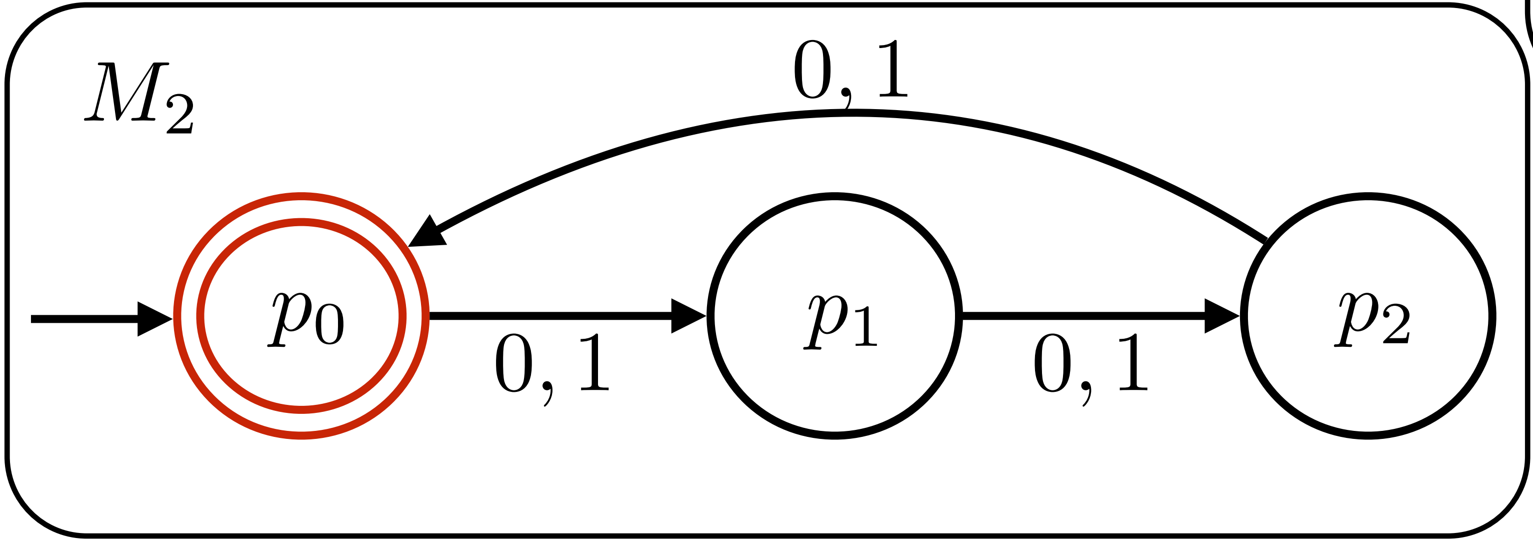


Regular languages are closed under union

Main idea:

Construct a DFA that keeps track of both at once.

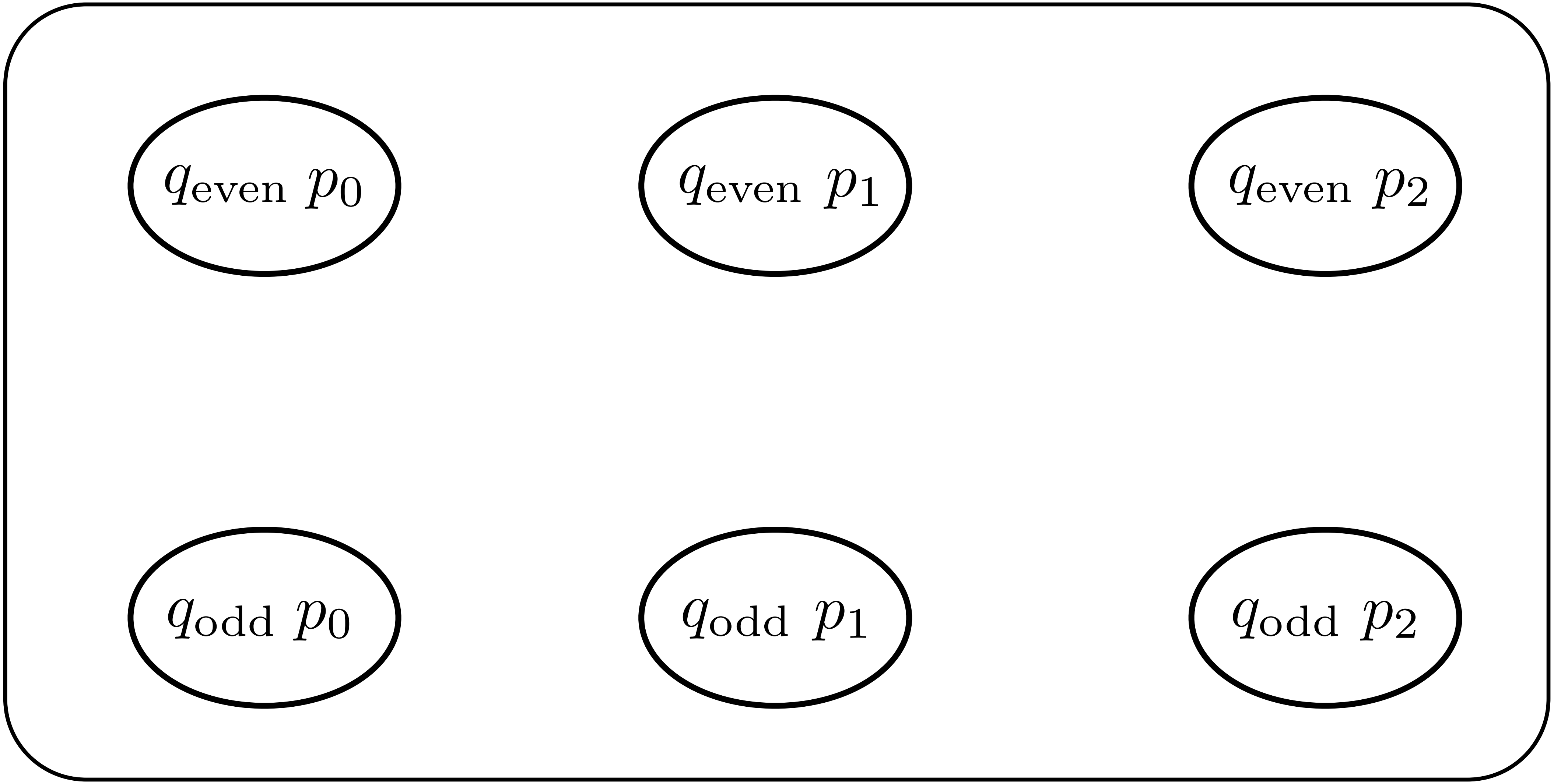




Regular languages are closed under union

Main idea:

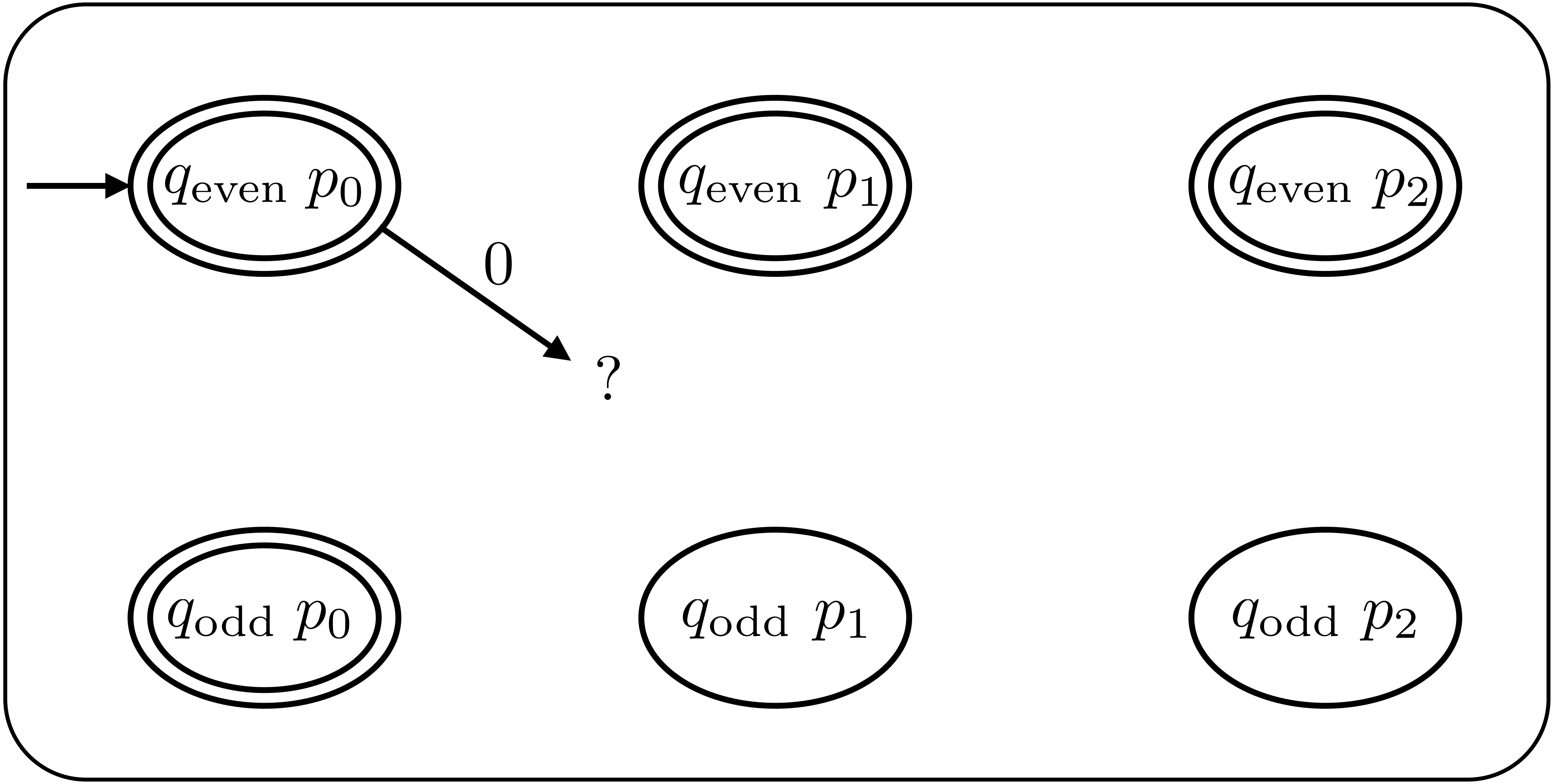
Construct a DFA that keeps track of both at once.



Regular languages are closed under union

Main idea:

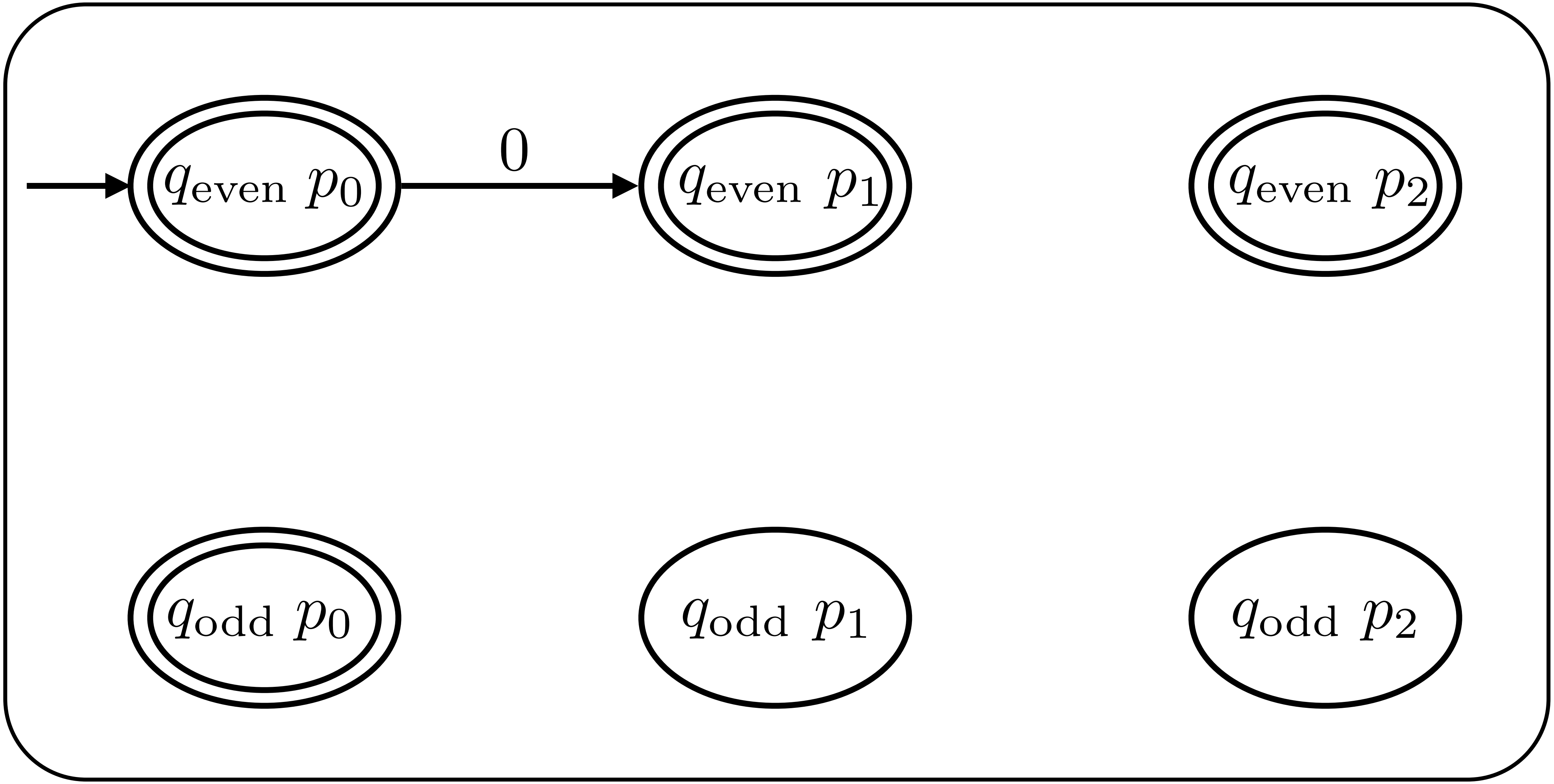
Construct a DFA that keeps track of both at once.



Regular languages are closed under union

Main idea:

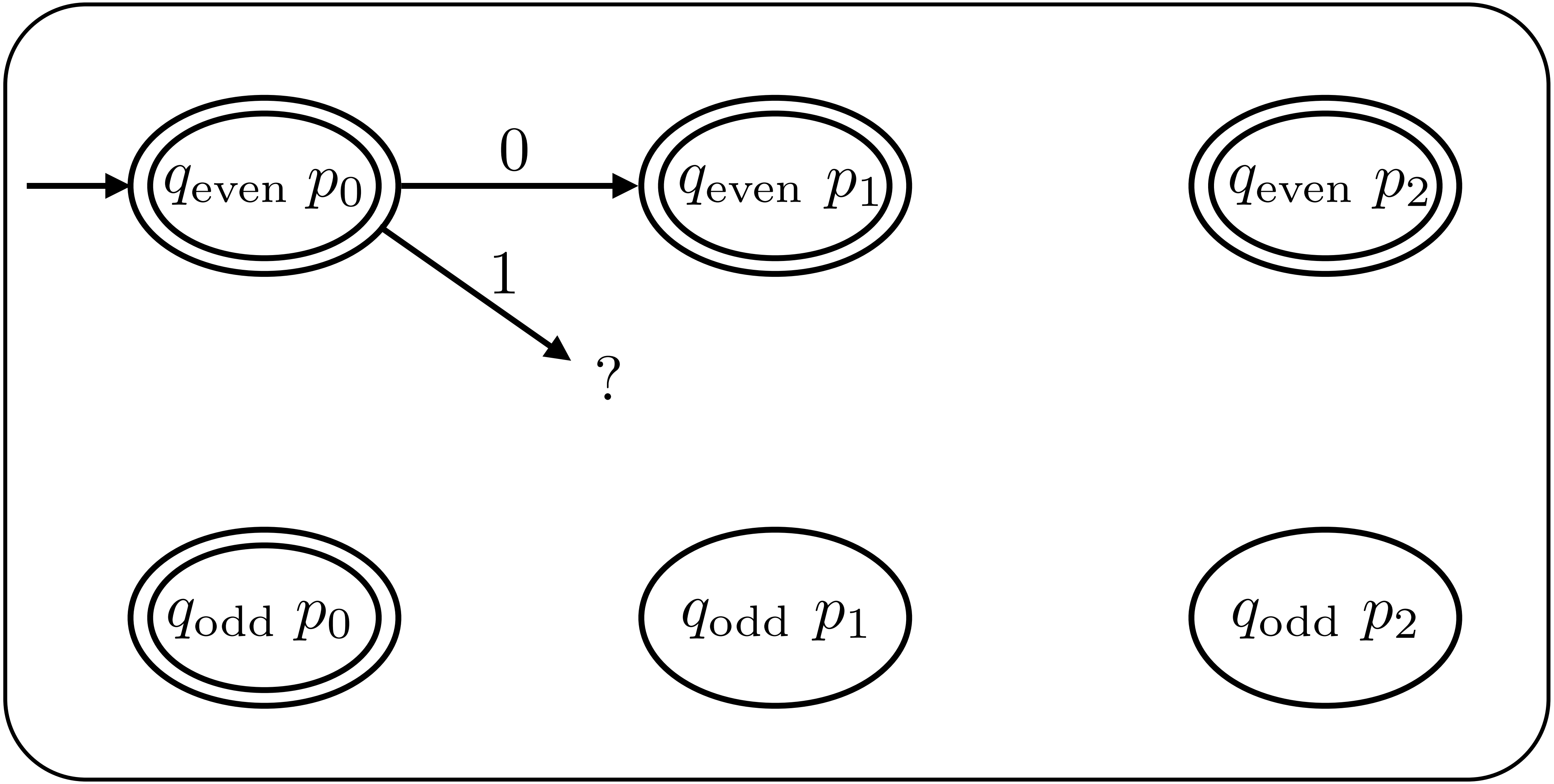
Construct a DFA that keeps track of both at once.



Regular languages are closed under union

Main idea:

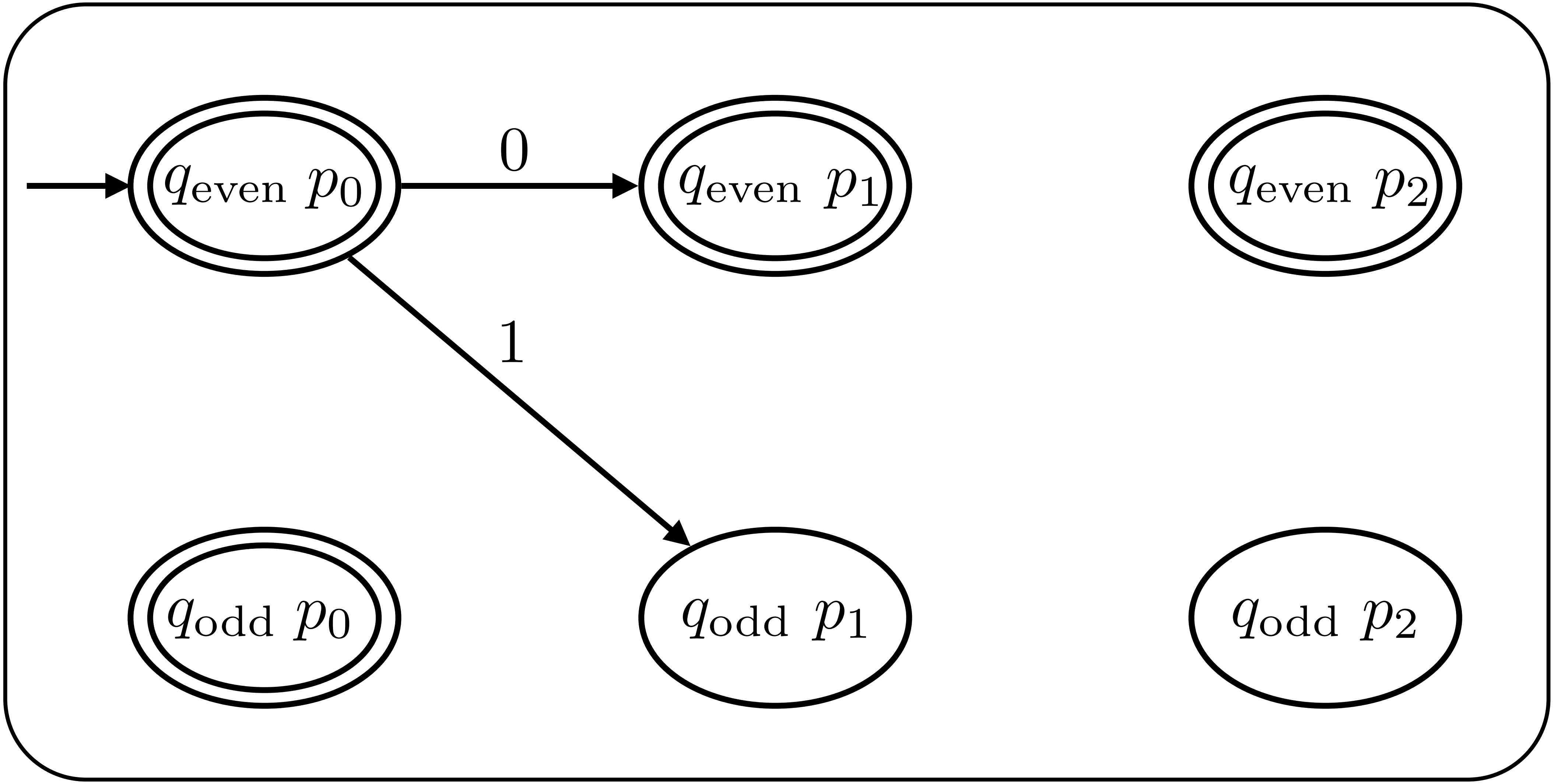
Construct a DFA that keeps track of both at once.



Regular languages are closed under union

Main idea:

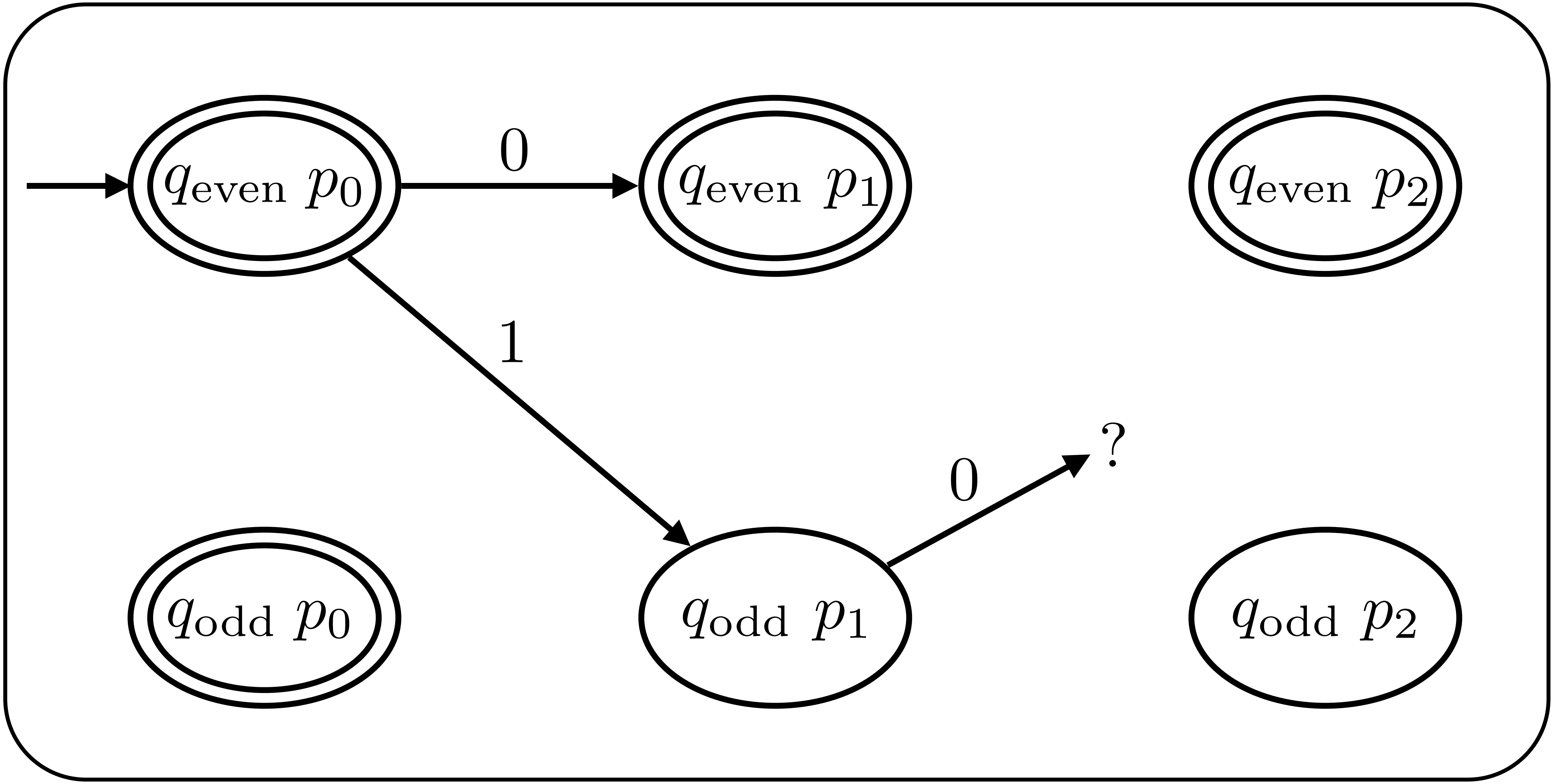
Construct a DFA that keeps track of both at once.



Regular languages are closed under union

Main idea:

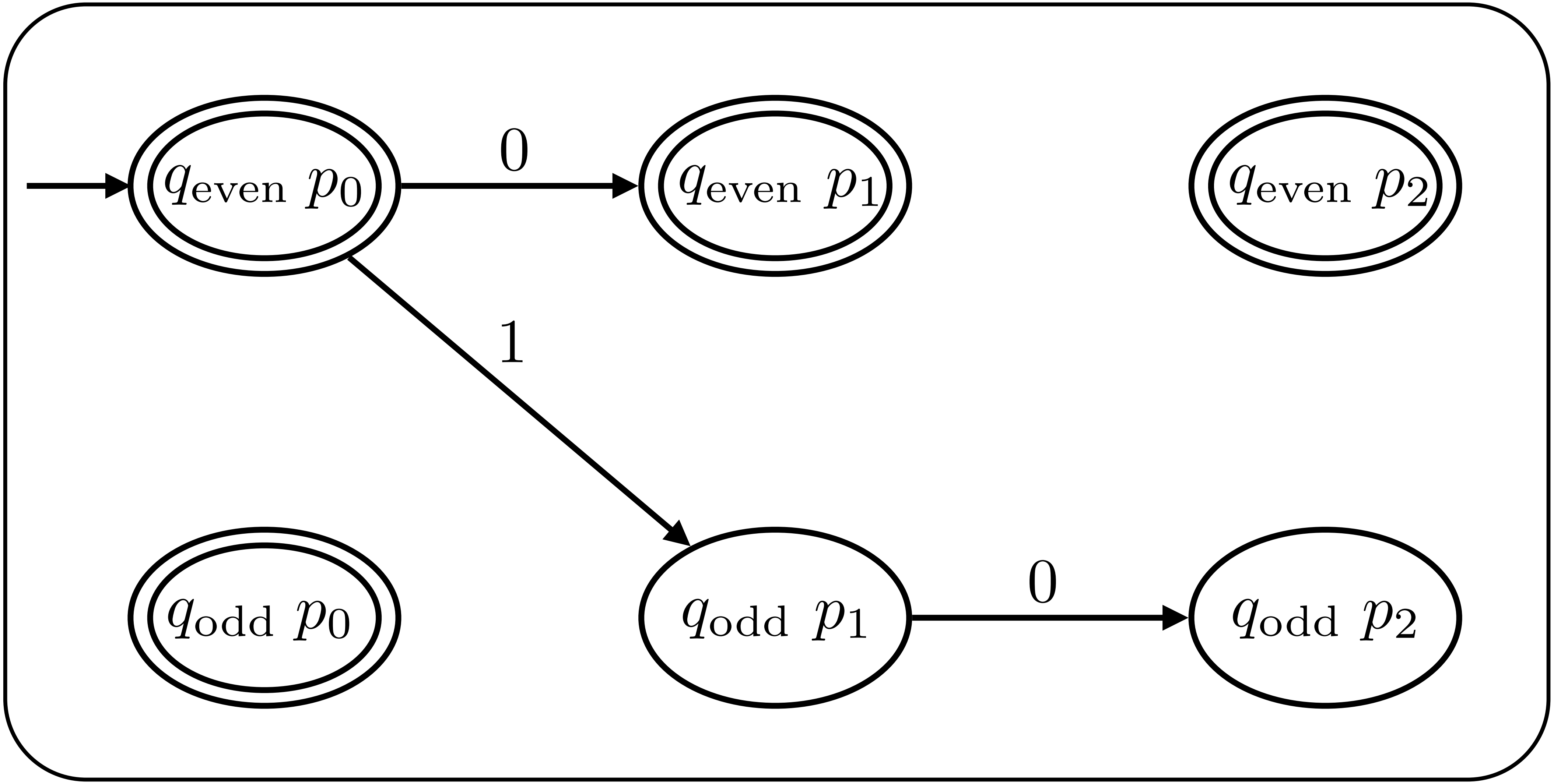
Construct a DFA that keeps track of both at once.



Regular languages are closed under union

Main idea:

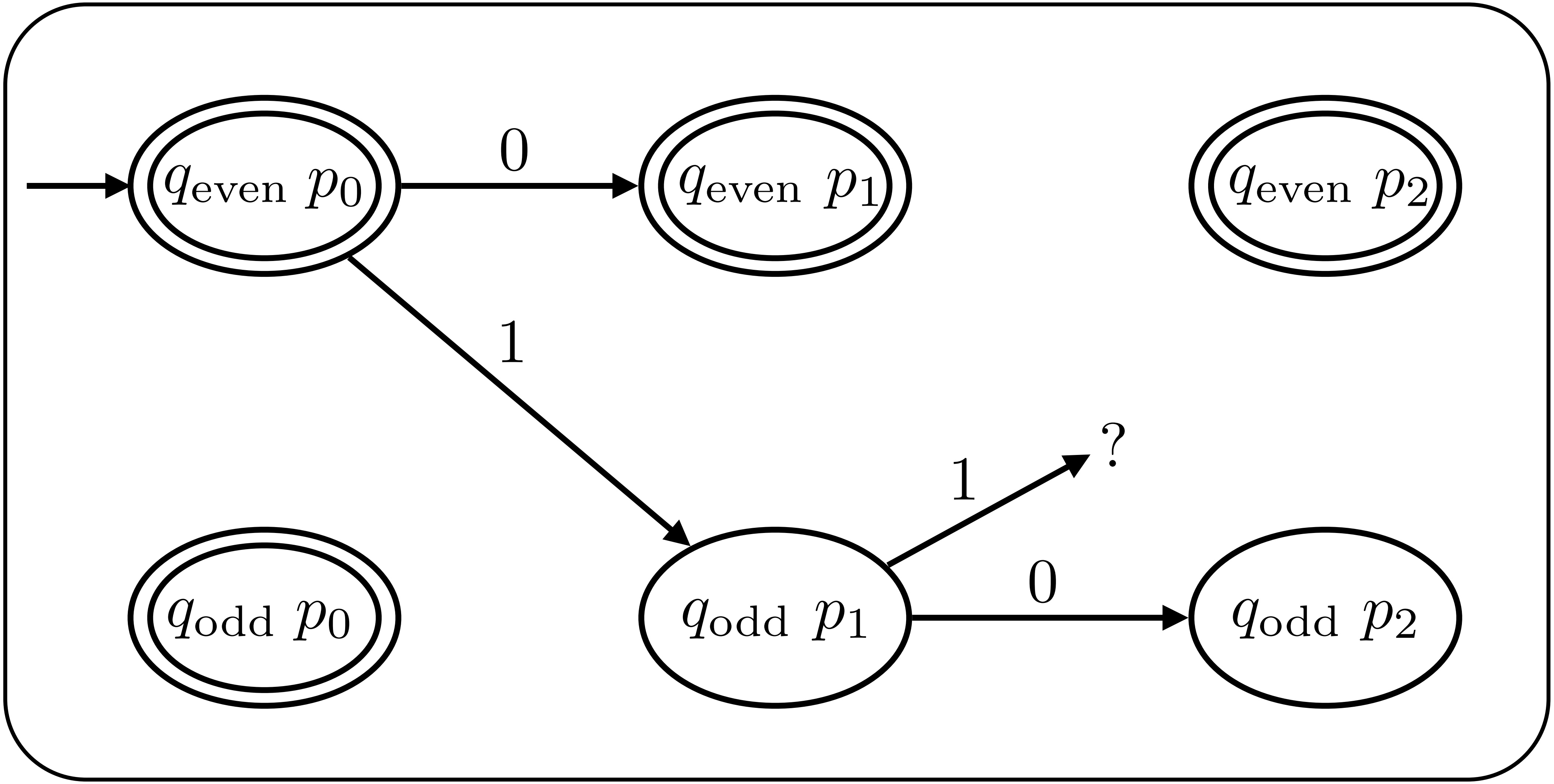
Construct a DFA that keeps track of both at once.



Regular languages are closed under union

Main idea:

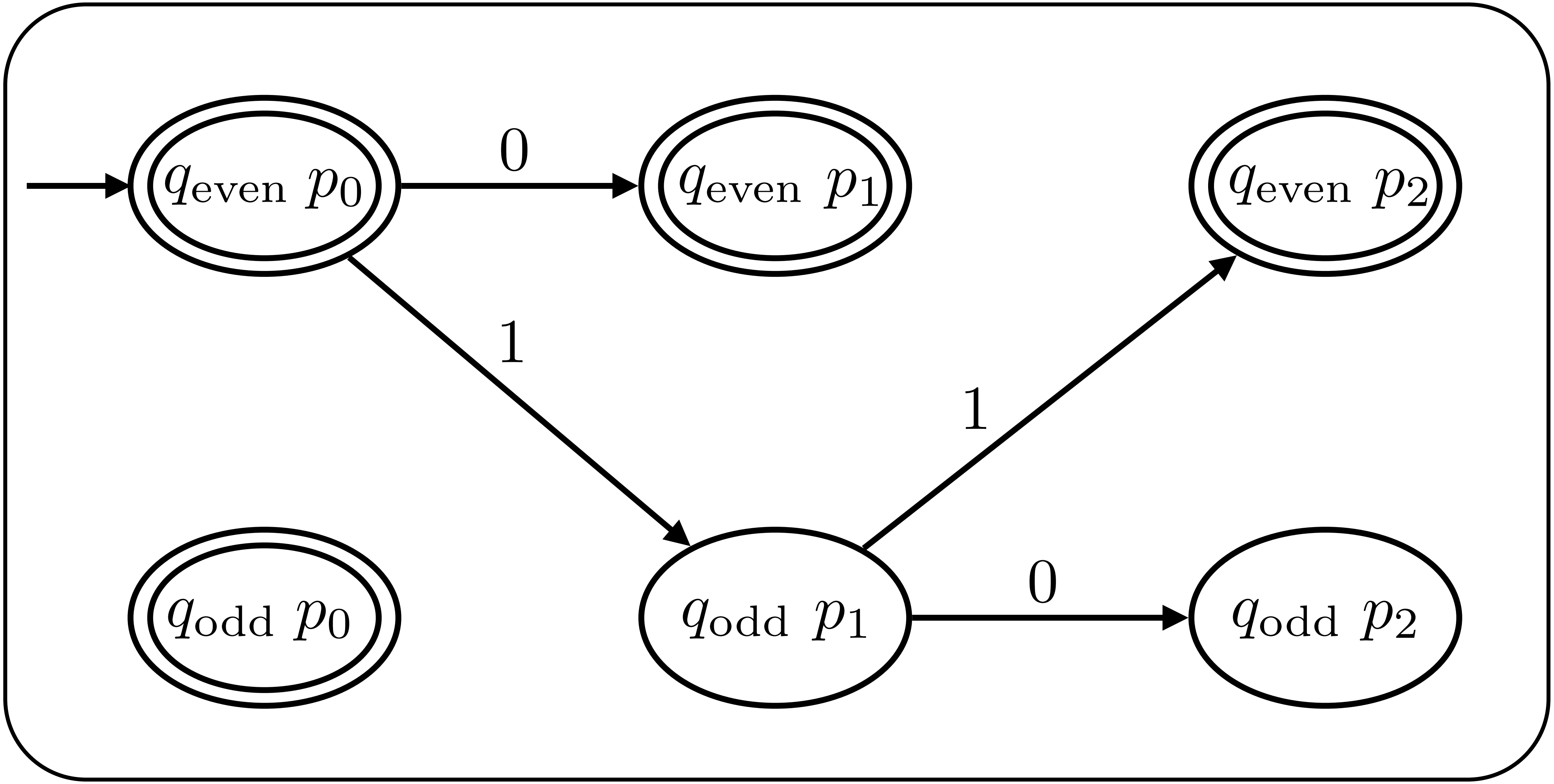
Construct a DFA that keeps track of both at once.



Regular languages are closed under union

Main idea:

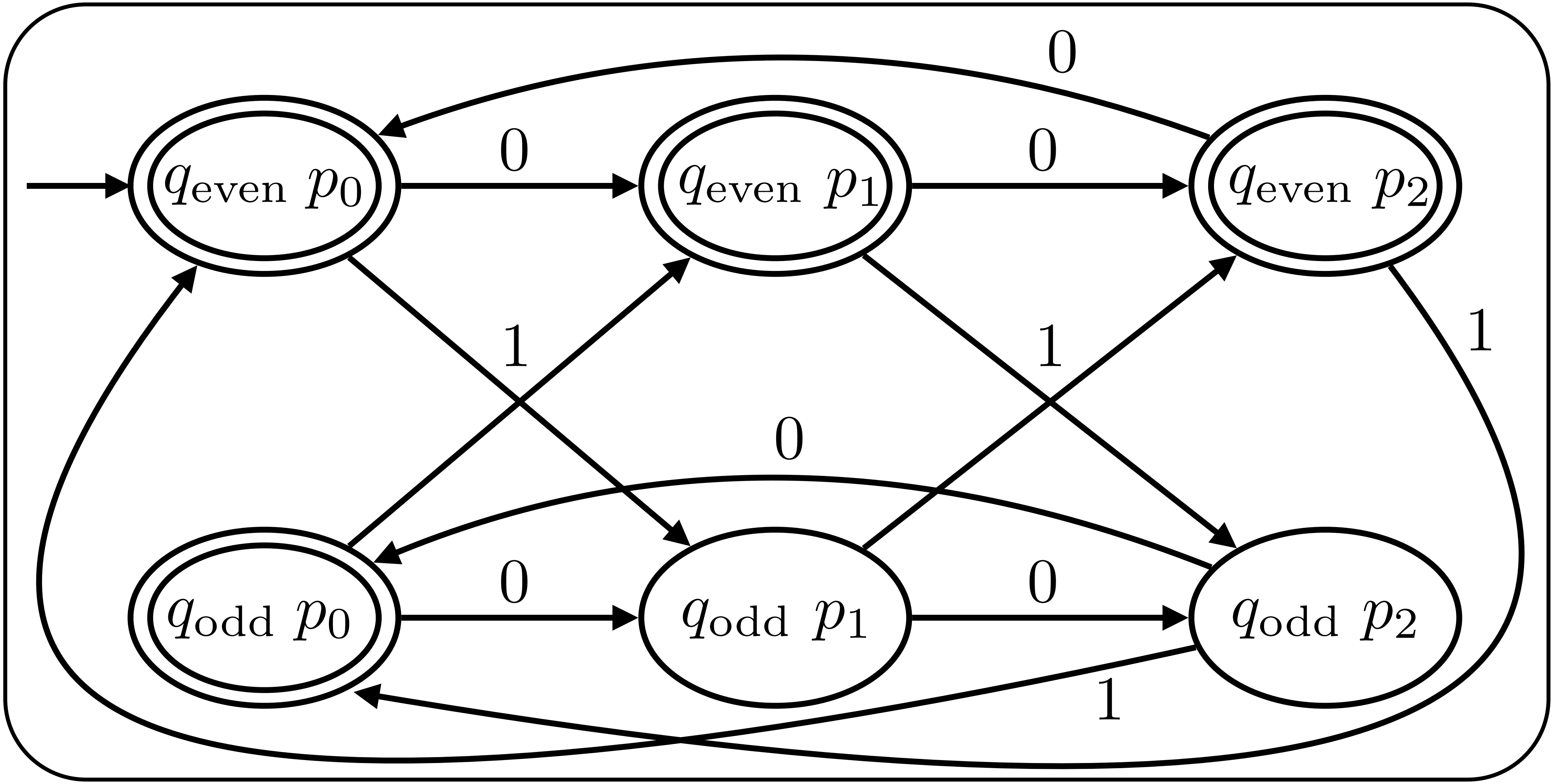
Construct a DFA that keeps track of both at once.



Regular languages are closed under union

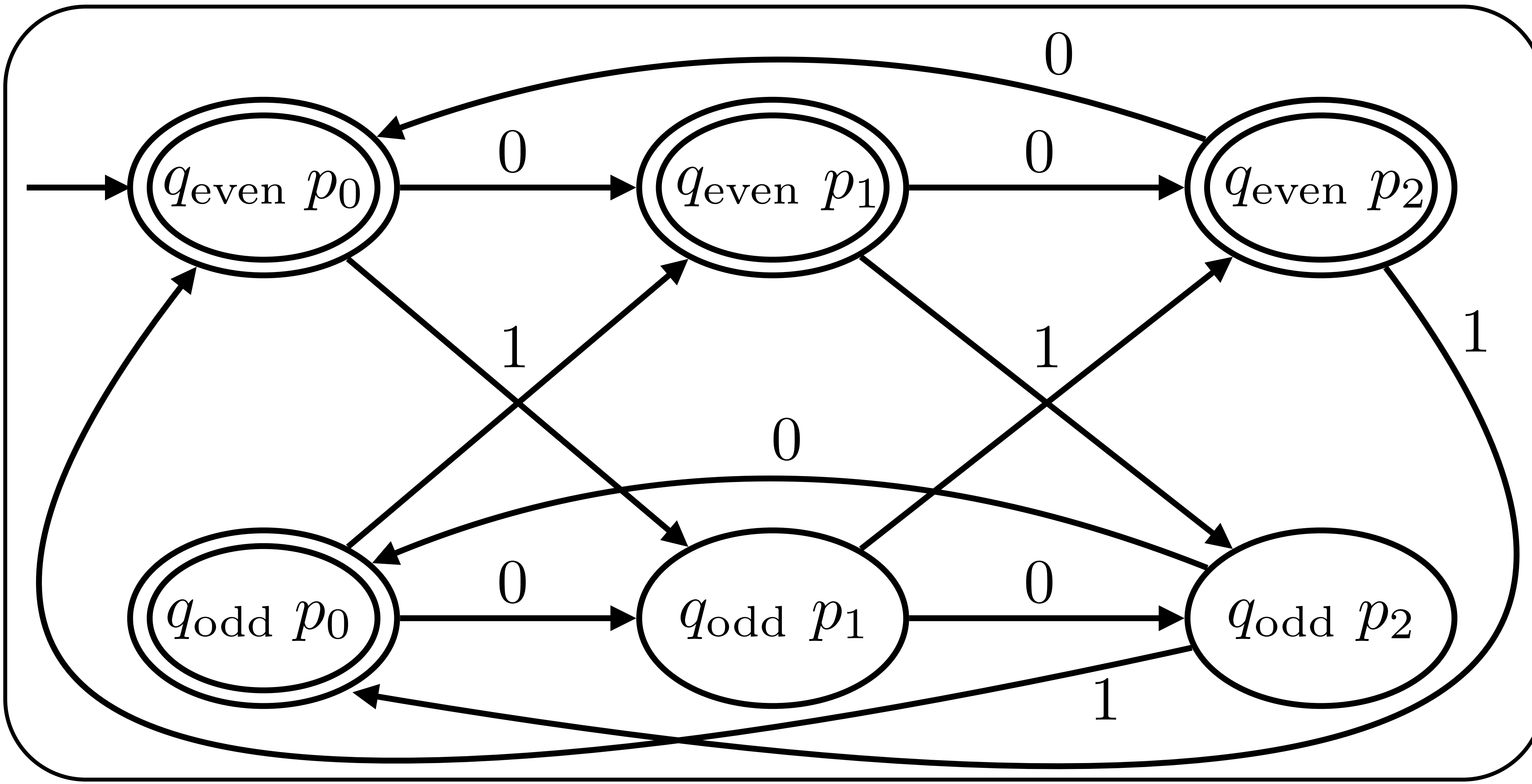
Main idea:

Construct a DFA that keeps track of both at once.



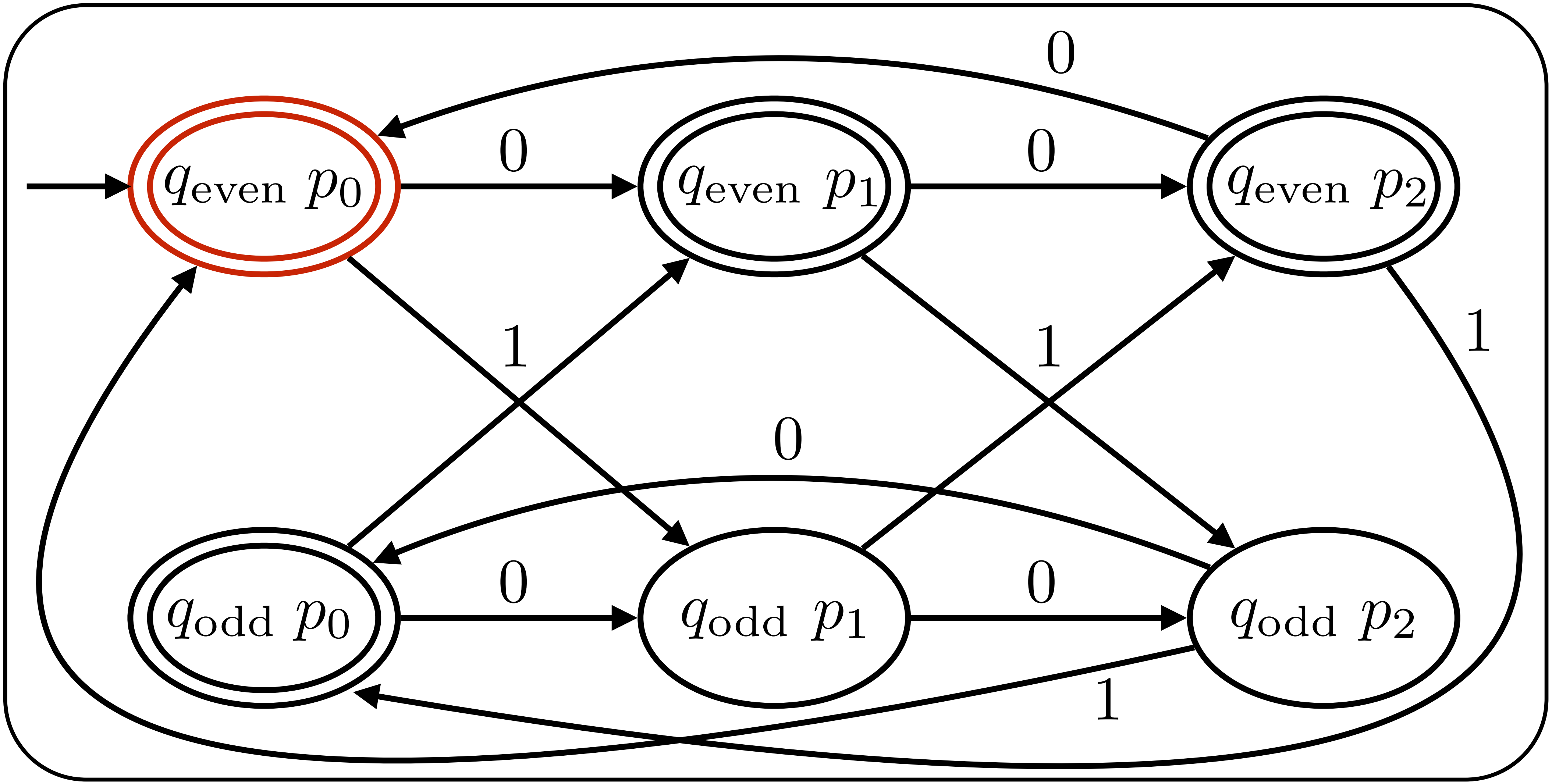
Regular languages are closed under union

Input: 101001



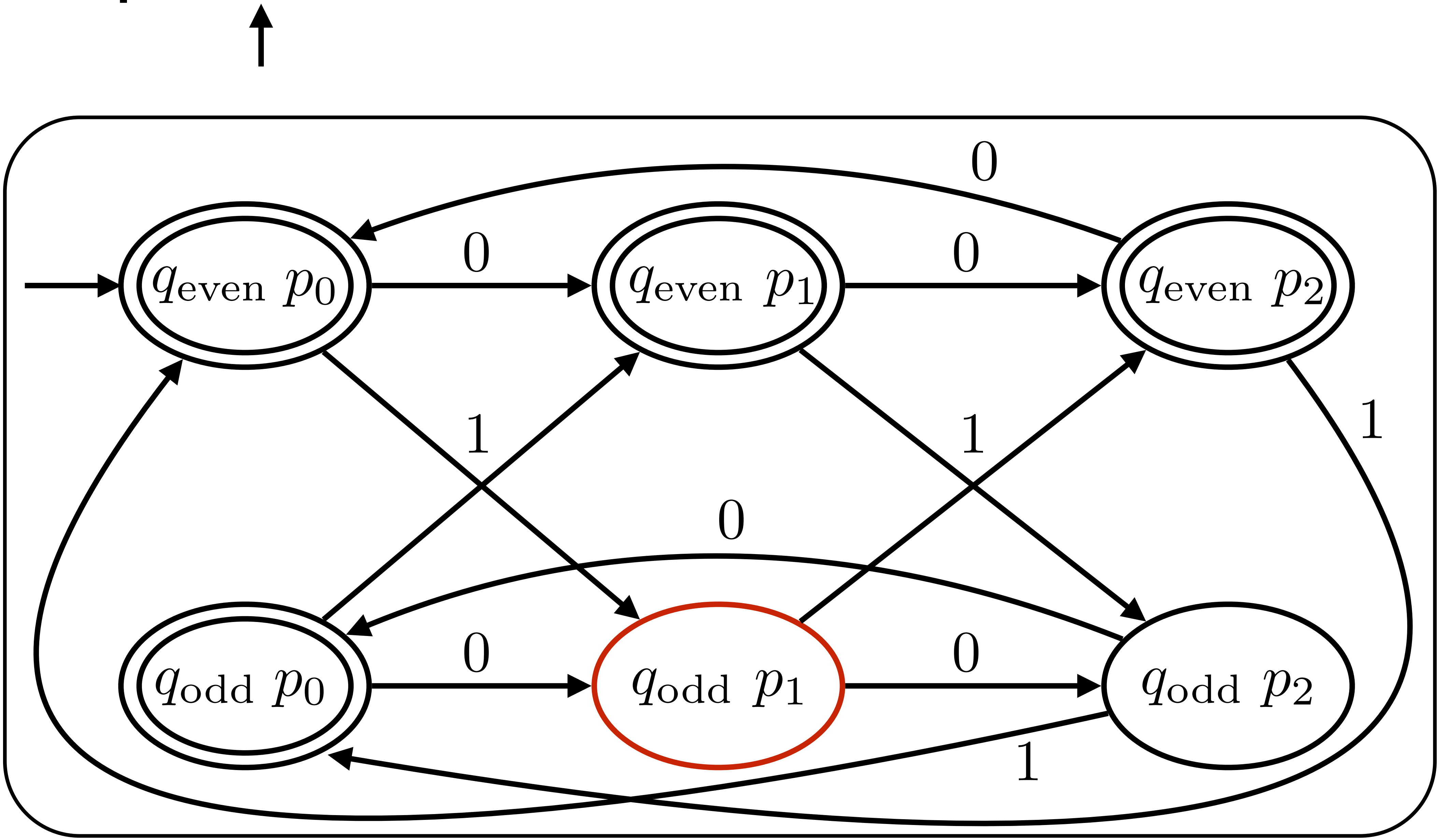
Regular languages are closed under union

Input: 101001



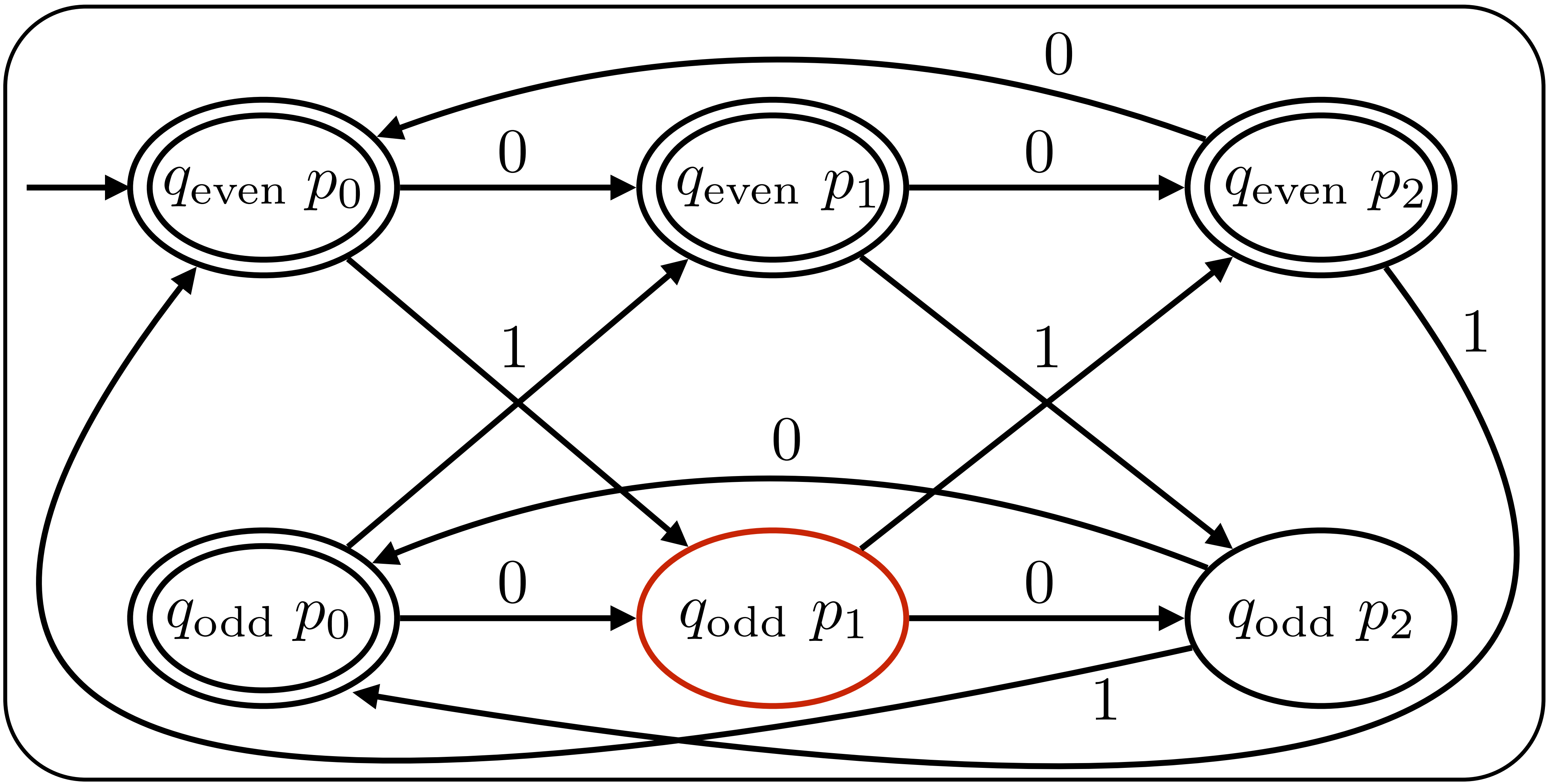
Regular languages are closed under union

Input: 101001



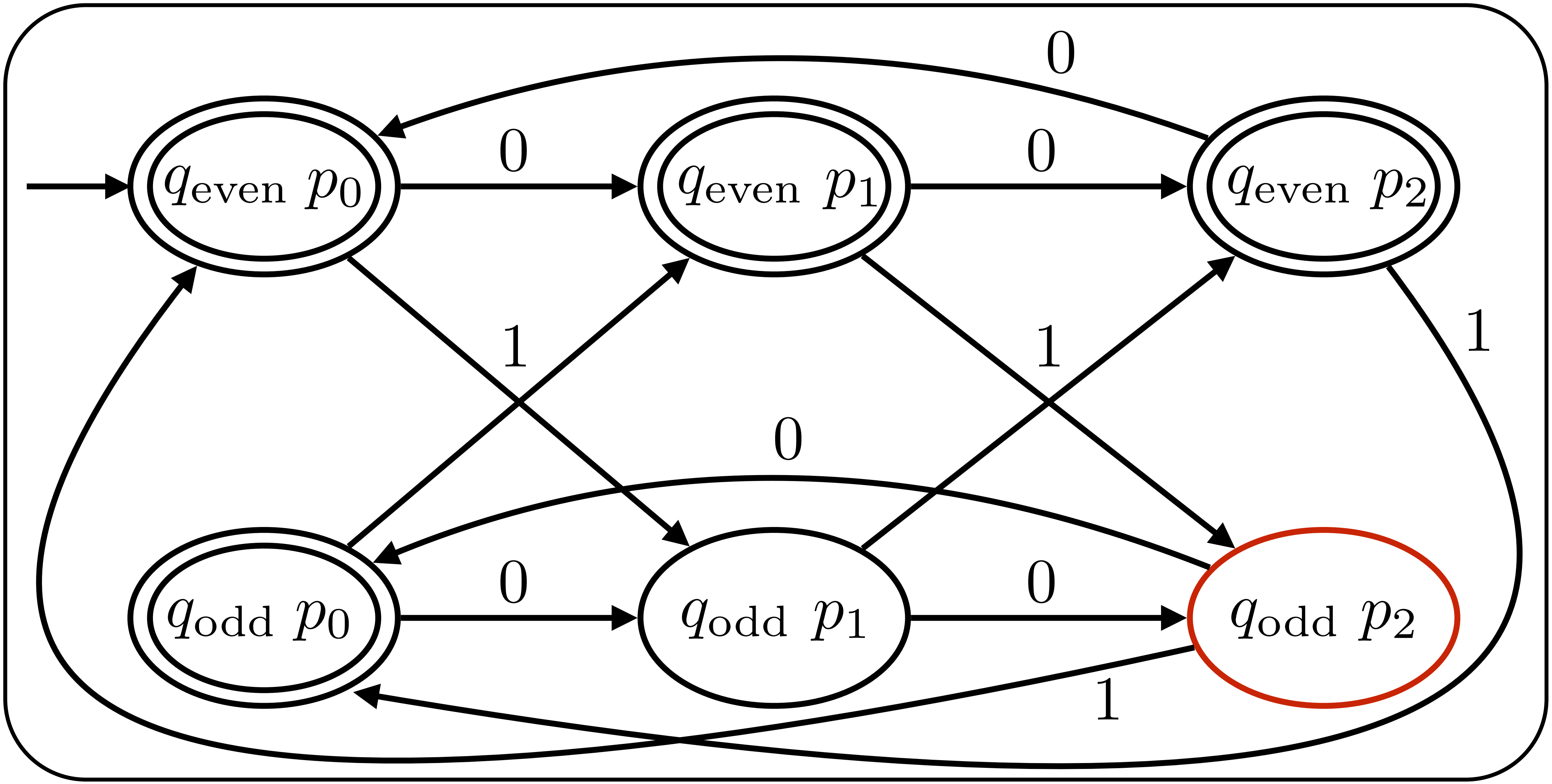
Regular languages are closed under union

Input: 101001



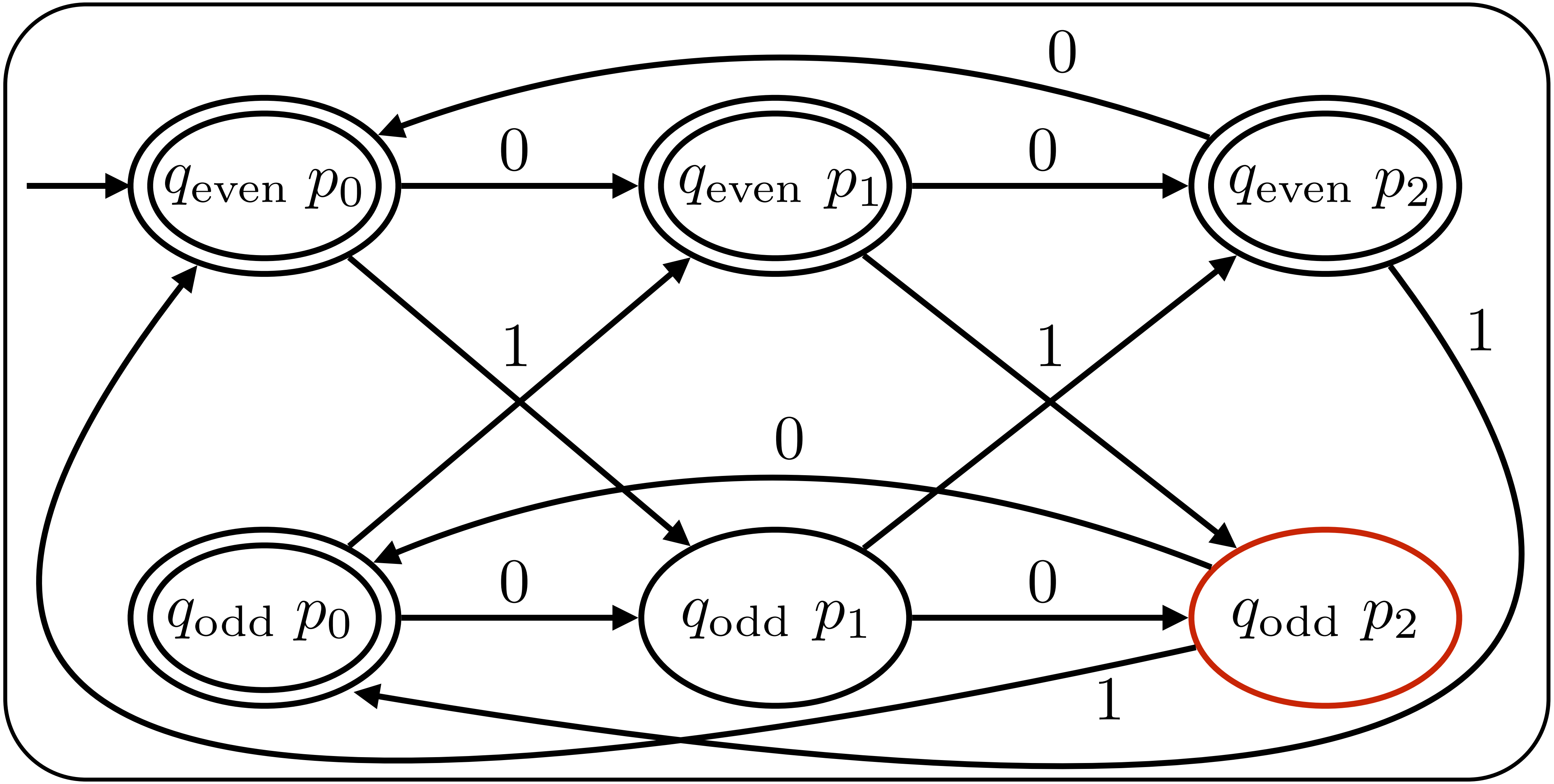
Regular languages are closed under union

Input: 101001



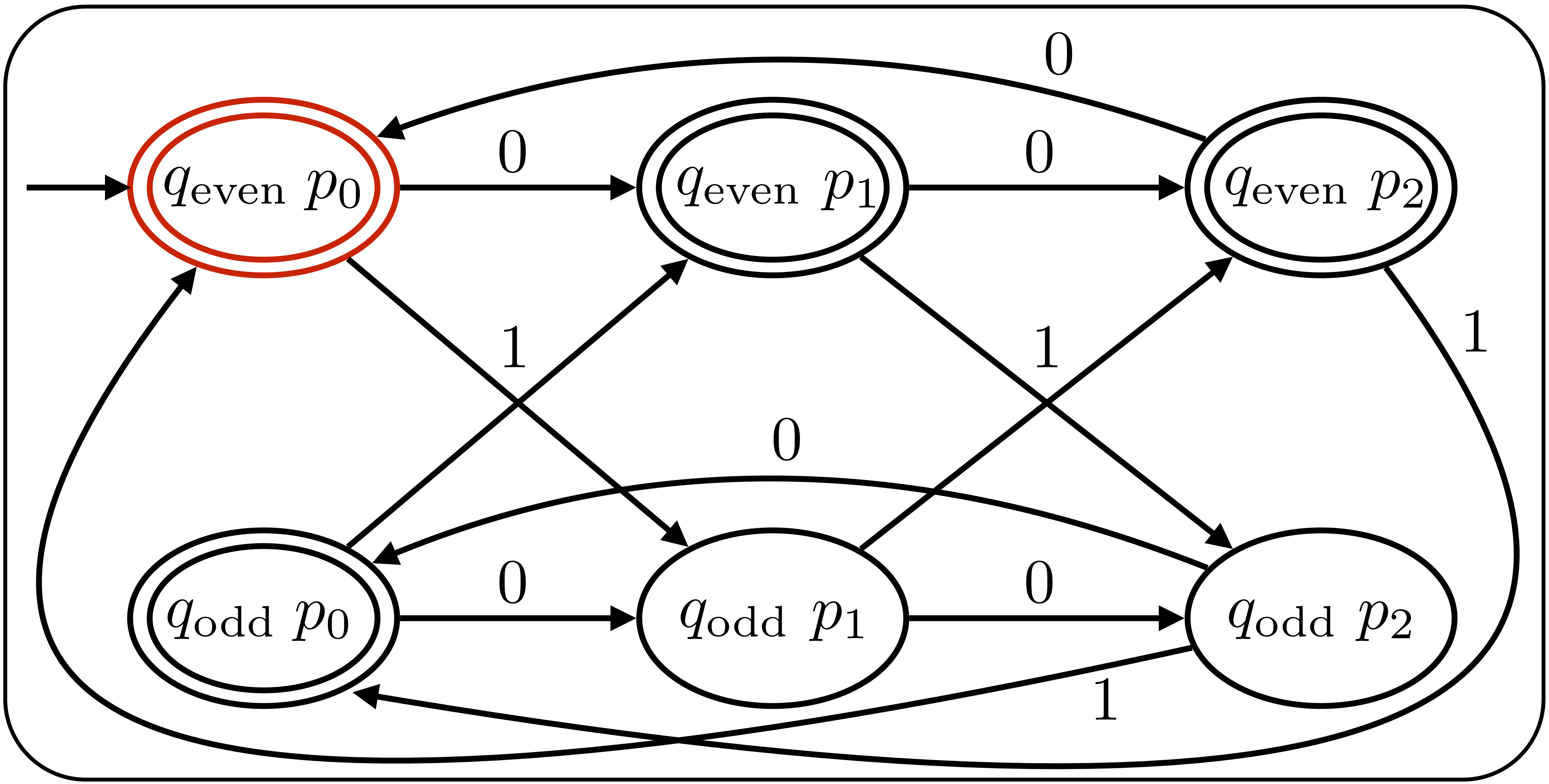
Regular languages are closed under union

Input: 101001



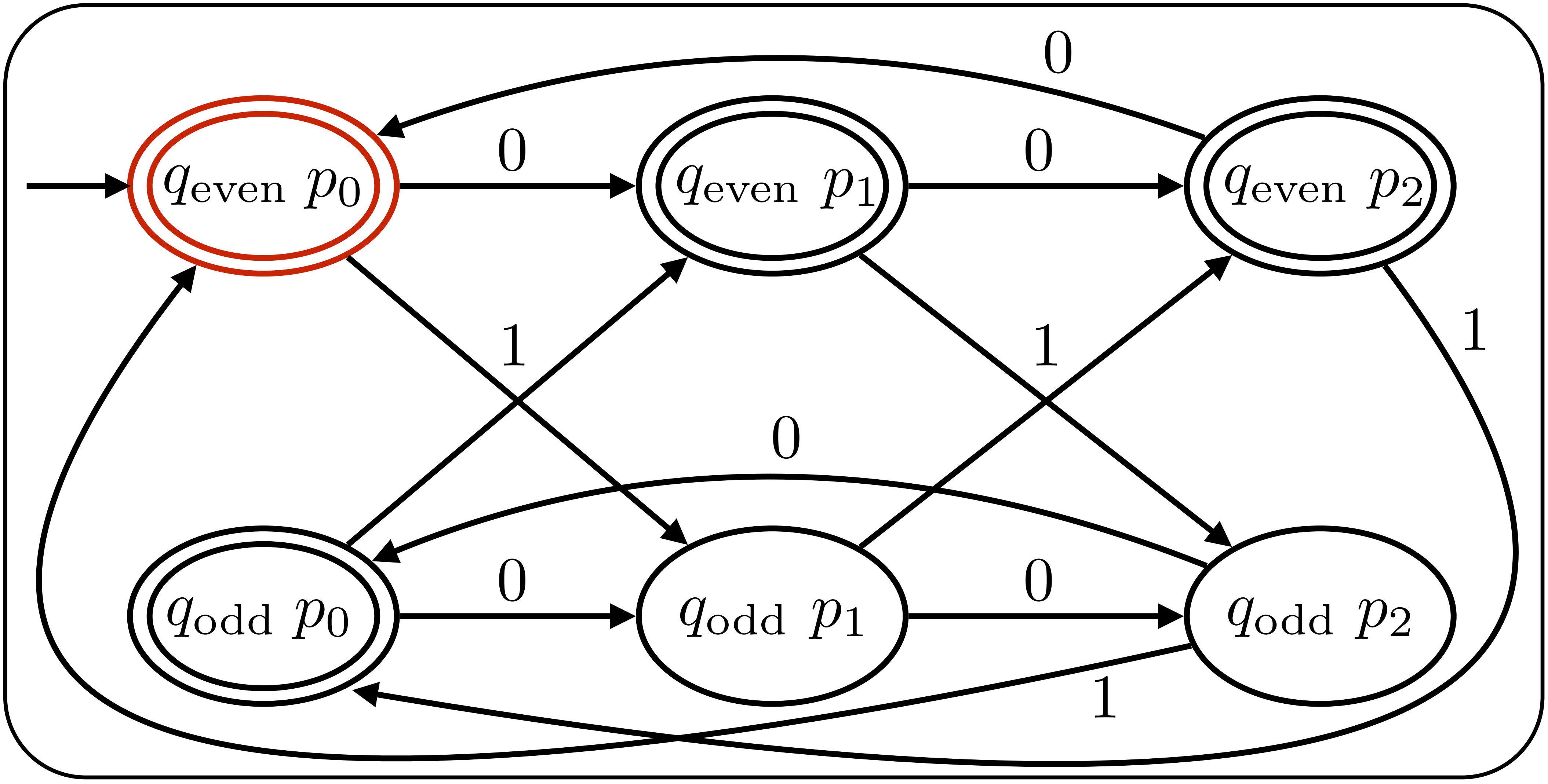
Regular languages are closed under union

Input: 101001



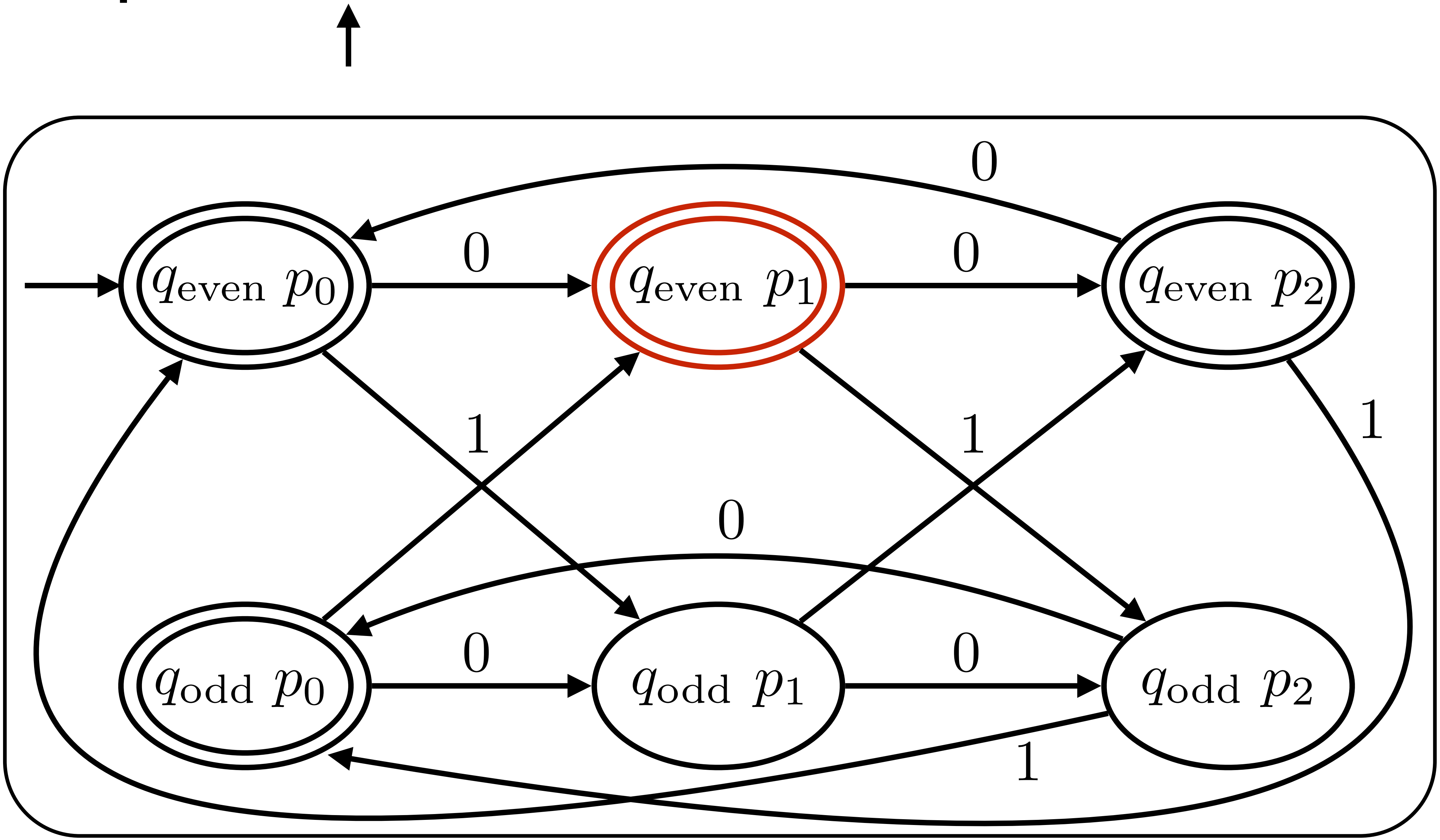
Regular languages are closed under union

Input: 101001



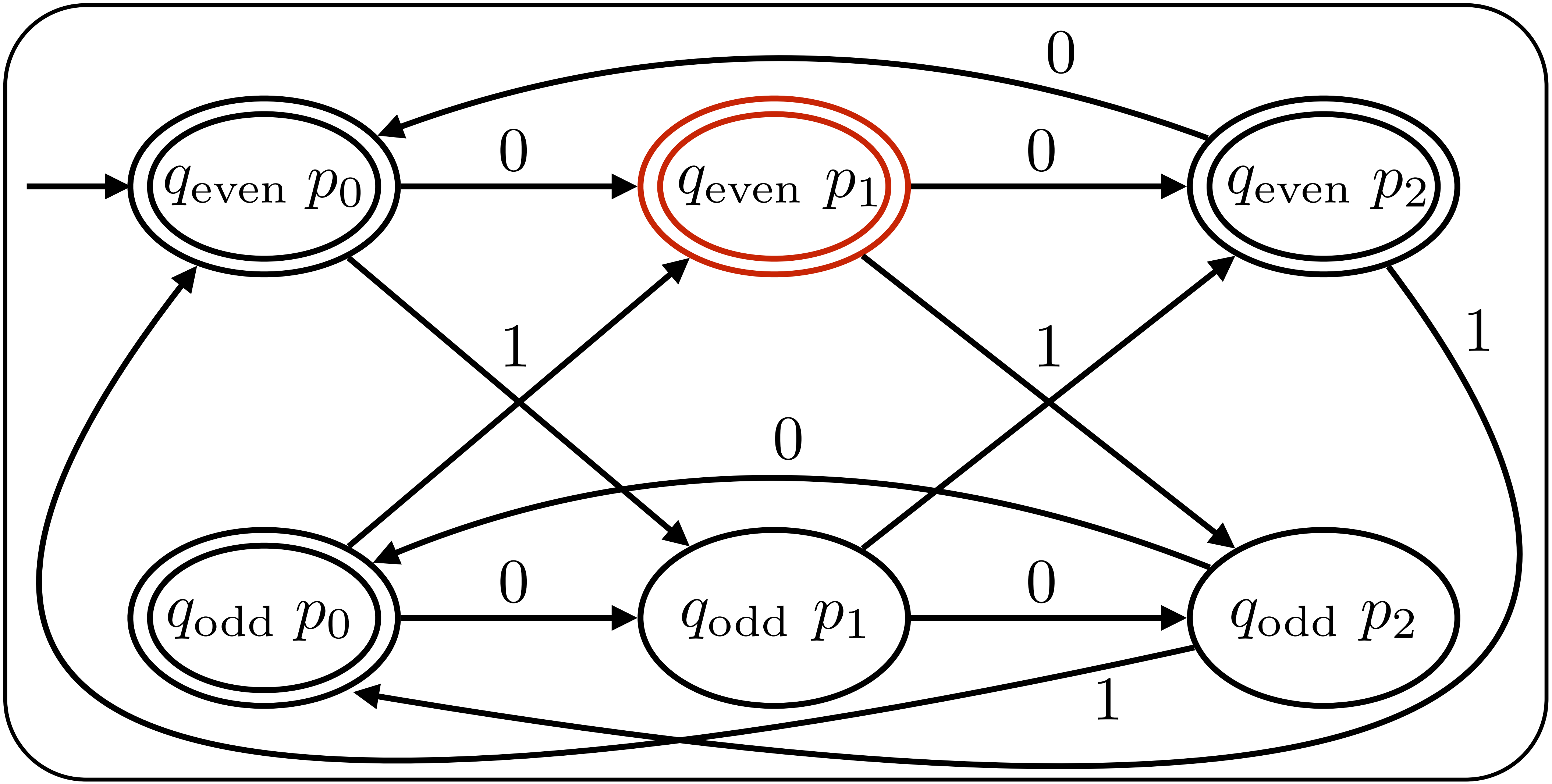
Regular languages are closed under union

Input: 101001



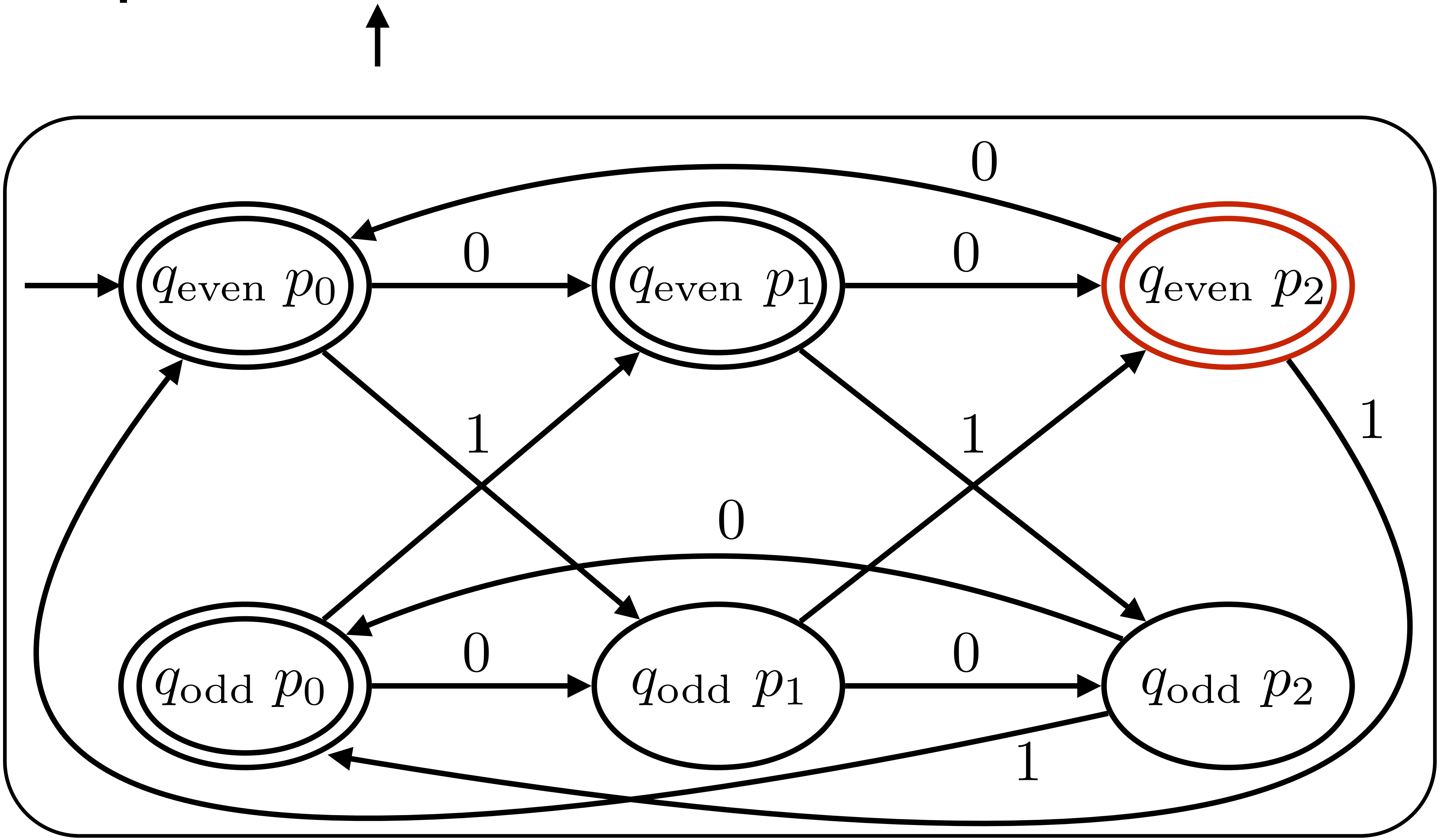
Regular languages are closed under union

Input: 101001



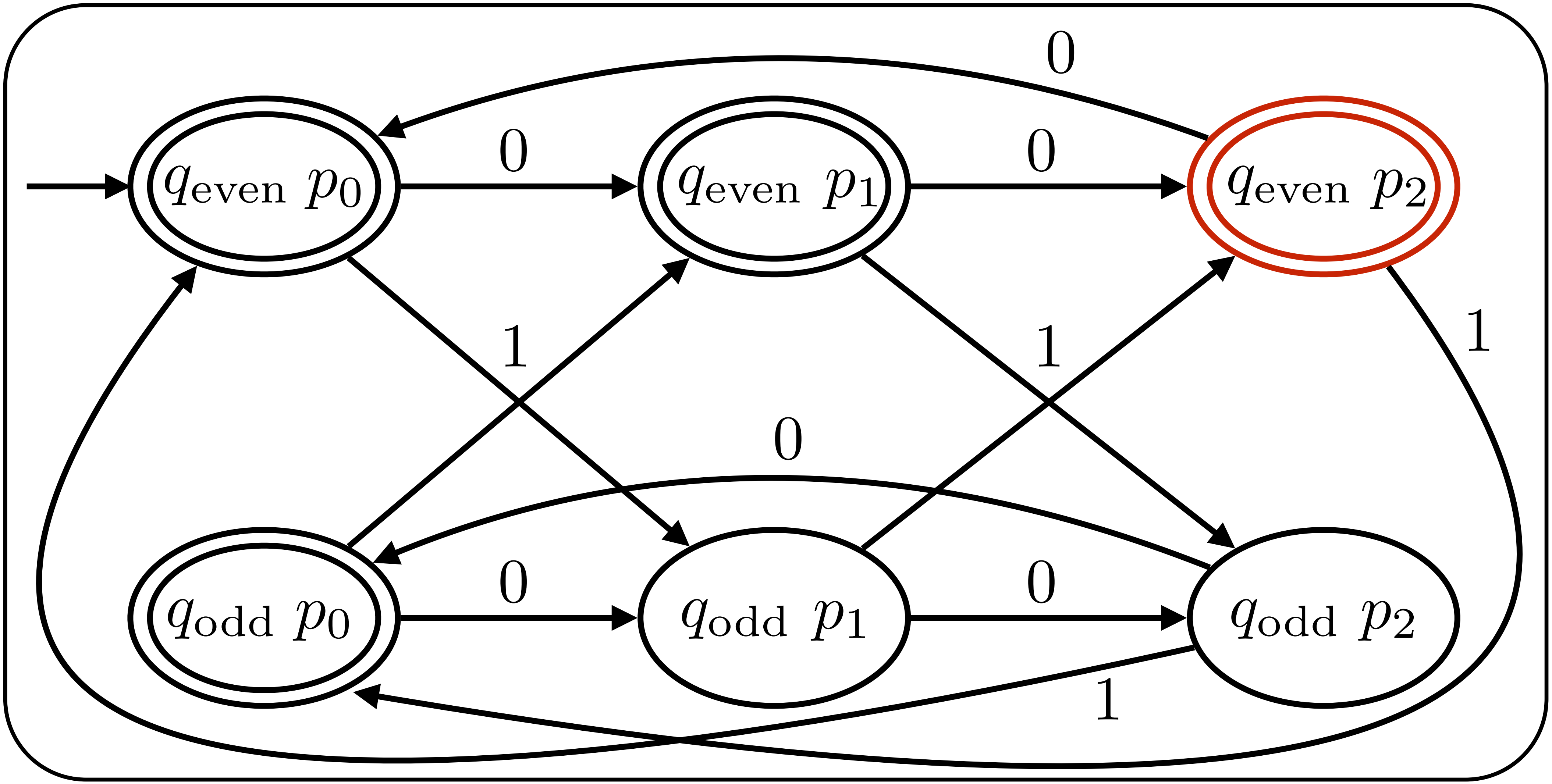
Regular languages are closed under union

Input: 101001



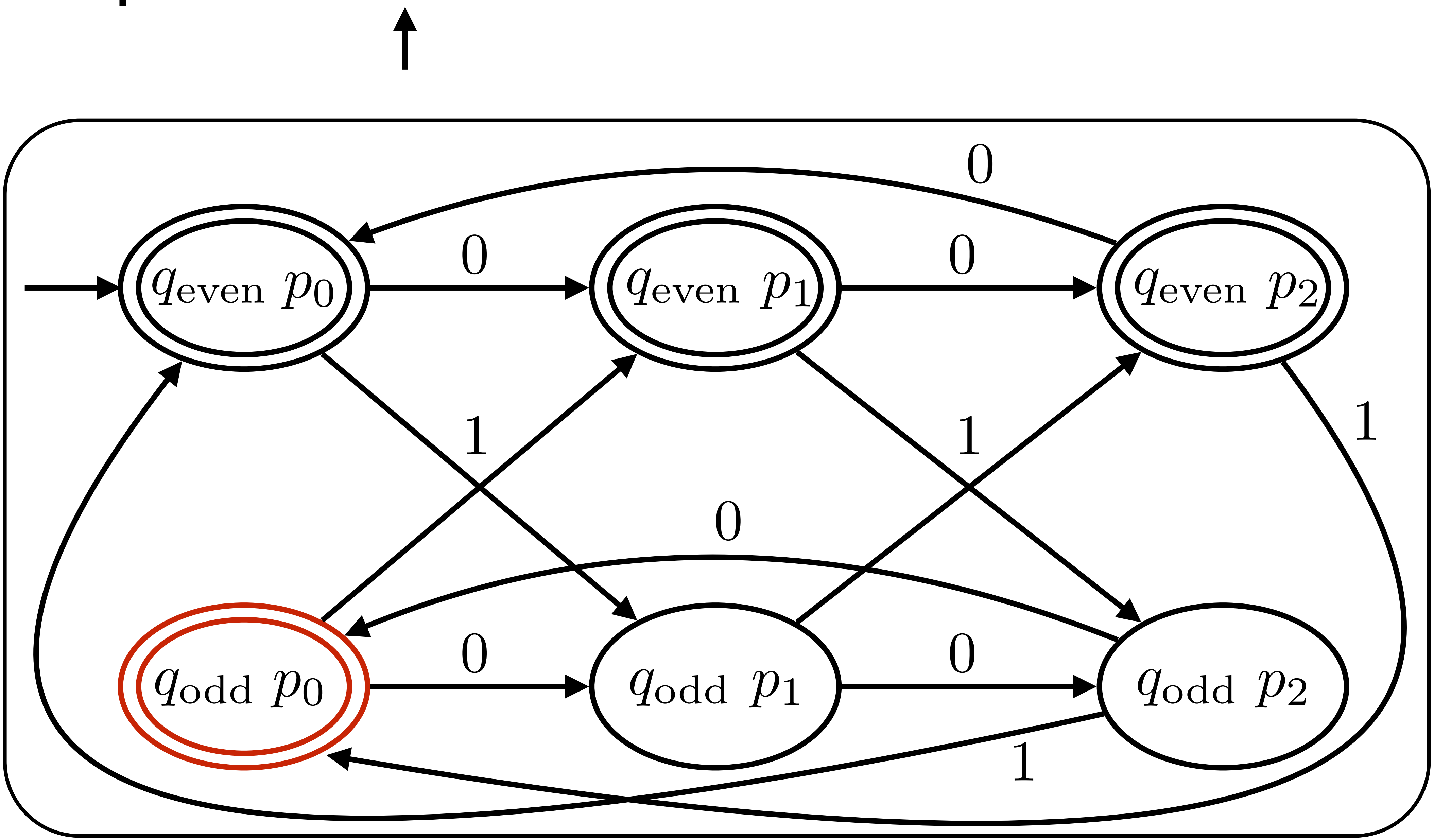
Regular languages are closed under union

Input: 101001



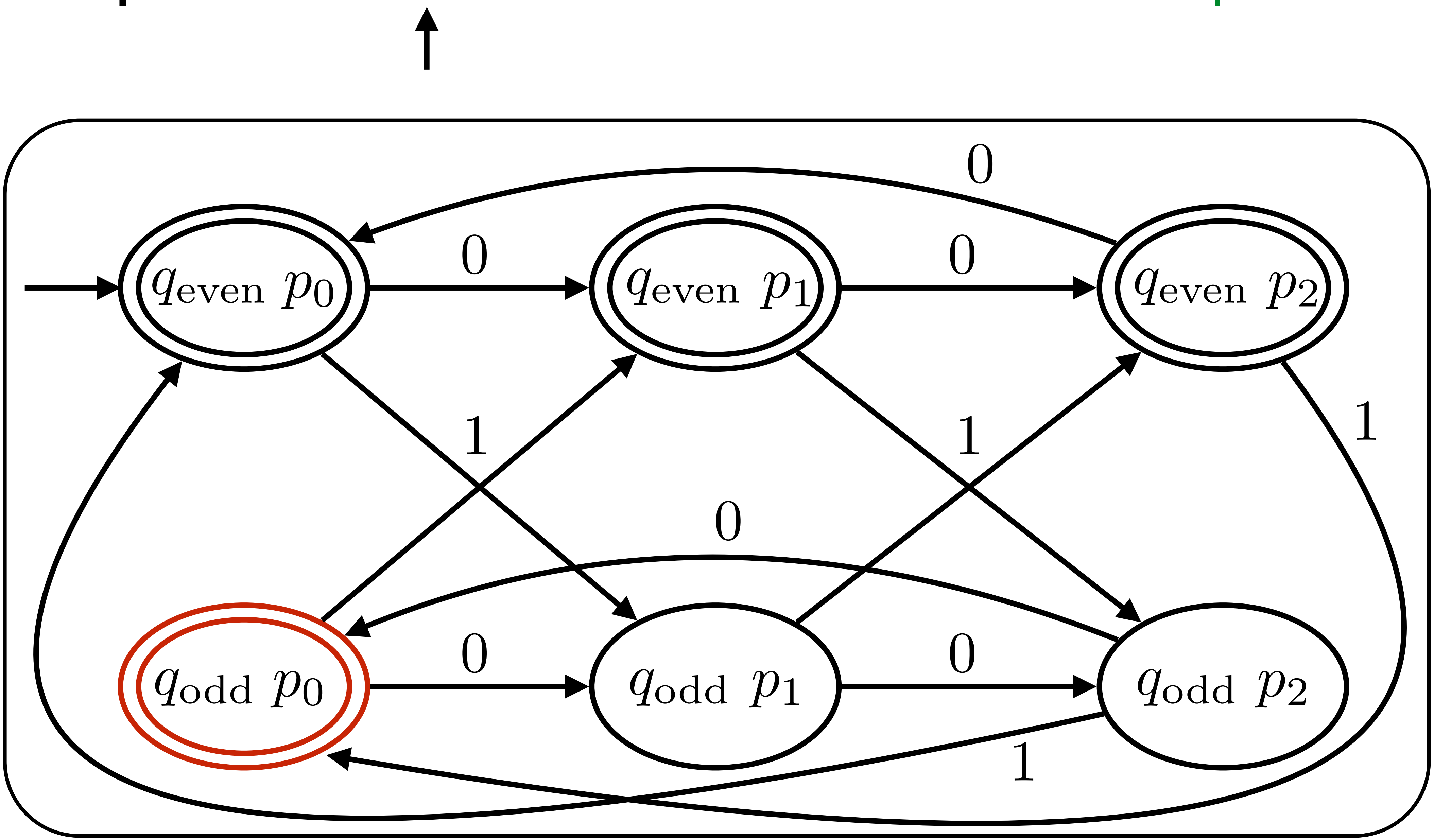
Regular languages are closed under union

Input: 101001



Regular languages are closed under union

Input: 101001 **Decision:** Accept



Regular languages are closed under union

Theorem:

Let Σ be some finite alphabet.

If $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$ are regular, then so is $L_1 \cup L_2$.

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be the decider for L_1 and $M' = (Q', \Sigma, \delta', q_0', F')$ be the decider for L_2.

We construct a DFA $M'' = (Q'', \Sigma, \delta'', q_0'', F'')$ that decides $L_1 \cup L_2$, as follows:

- $Q'' = Q \times Q' = \{(q, q') : q \in Q, q' \in Q'\}$
- $\delta''((q, q'), a) = (\delta(q, a), \delta'(q', a))$
- $q_0'' = (q_0, q_0')$
- $F'' = \{(q, q') : q \in F \text{ or } q' \in F'\}$

Regular languages are closed under union

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be the decider for L_1 and $M' = (Q', \Sigma, \delta', q_0', F')$ be the decider for L_2.

We construct a DFA $M'' = (Q'', \Sigma, \delta'', q_0'', F'')$ that decides $L_1 \cup L_2$, as follows:

- $Q'' = Q \times Q' = \{(q, q') : q \in Q, q' \in Q'\}$
- $\delta''((q, q'), a) = (\delta(q, a), \delta'(q', a))$
- $q_0'' = (q_0, q_0')$
- $F'' = \{(q, q') : q \in F \text{ or } q' \in F'\}$

It remains to show that $L(M'') = L_1 \cup L_2$.

$L(M'') \subseteq L_1 \cup L_2 : \quad \ldots$

$L_1 \cup L_2 \subseteq L(M'') : \quad \ldots$

$\square$

An application of DFAs

String Searching Problem

Input: string T of length n. string w of length k.

Output: Yes/No. Does w occur in T?

Naive algorithm:

About nk steps.

Can we do better?

An application of DFAs

String Searching Problem

Input: string T of length n. string w of length k.

Output: Yes/No. Does w occur in T ?

Automaton solution:

The language $\Sigma^* w \Sigma^*$ is regular.

So there is some DFA M_w that accepts it.

Build M_w and feed it T. Running time: $\sim n$ steps.

Time to build M_w ? Simple alg: $\sim k^3$ steps.

An application of DFAs

String Searching Problem

Input: string T of length n. string w of length k.

Output: Yes/No. Does w occur in T?

Knuth-Morris-Pratt 1977:

$\sim k$ steps to build M_w.

Next Time