Introduction to the Theory of Computation

Set 2 — Regular Languages (1)

Languages

- **Alphabet**
  - Finite collection of objects (denoted Σ)
- **String**
  - Concatenation of 0 or more elements of an alphabet
- **Language**
  - Collection of strings
    - Σ^* is the set of all strings over Σ (including ε)

$\varepsilon \triangleq$ *the empty string*

`ε.length()==0`

Deterministic Finite Automata (DFA)

- **Method for modeling computers with limited memory**
  - Language recognizer
- **Idea**
  - Keep track of current state
    - Events cause movement from one state to another

Next…

- Formally describe DFA’s
- Interpret DFA’s

Example — Combination Lock

- **There are four buttons for user input**

  🐸, 🚗, 🪑, 🔓 (frog, car, chair, unlock)

- **The lock will open if and only if the buttons are pressed in the correct order**

- **If the unlocking sequence is exactly length 3, there are 256 possible sequences**

  {🐸🐸🐸, 🐸🚗🐸, 🪑🔓🚗, …}

  In general, for a sequence length k of B buttons, there are B^k unique sequences

Example — Combination Lock

- There are four actions

  🐸, 🚗, 🪑, 🔓 (push *frog, car, chair,* or *unlock*)

- The lock can be in one of these 4 states
  - RESET — Ready to recognize combination
  - SEEN_FIRST — First correct action
  - SEEN_SECOND — First+second correct actions
  - UNLOCKED — Correct action sequence

Example — Combination 🔓🪑🔓

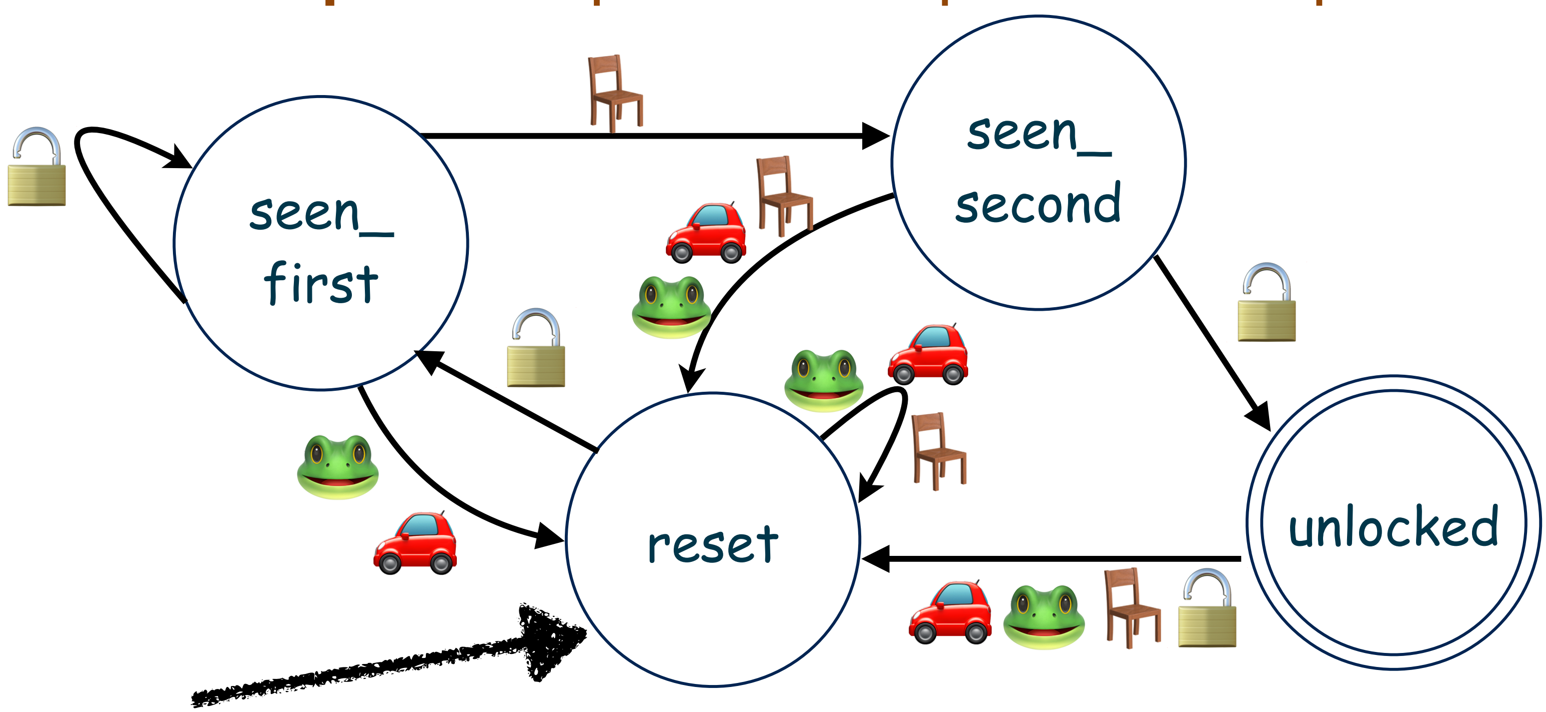
State table

Event State	🐸	🚗	🪑	🔓
reset	reset	reset	reset	seen_first
seen_first	reset	reset	seen_second	seen_first
seen_second	reset	reset	reset	unlocked
unlocked	reset	reset	reset	reset

Example — Combination 🔓🪑🔓

State table

State \ Event	🐸	🚗	🪑	🔓
reset	reset	reset	reset	seen_first
seen_first	reset	reset	seen_second	seen_first
seen_second	reset	reset	reset	unlocked
unlocked	reset	reset	reset	reset



Deterministic Finite Automaton (DFA)

[Formal Definition]

A deterministic finite automaton (DFA) is a 5-tuple, (Q,Σ,δ,q_0,F), where

- **Q is a finite set called the states**
- **Σ is a finite set called the alphabet**
- **$\delta : Q \times \Sigma \to Q$ is the transition function**
  (δ *corresponds to the example state change function)*
- **q_0 is the start state, and**
- **$F \subseteq Q$ is the set of accept states (also called final states).**

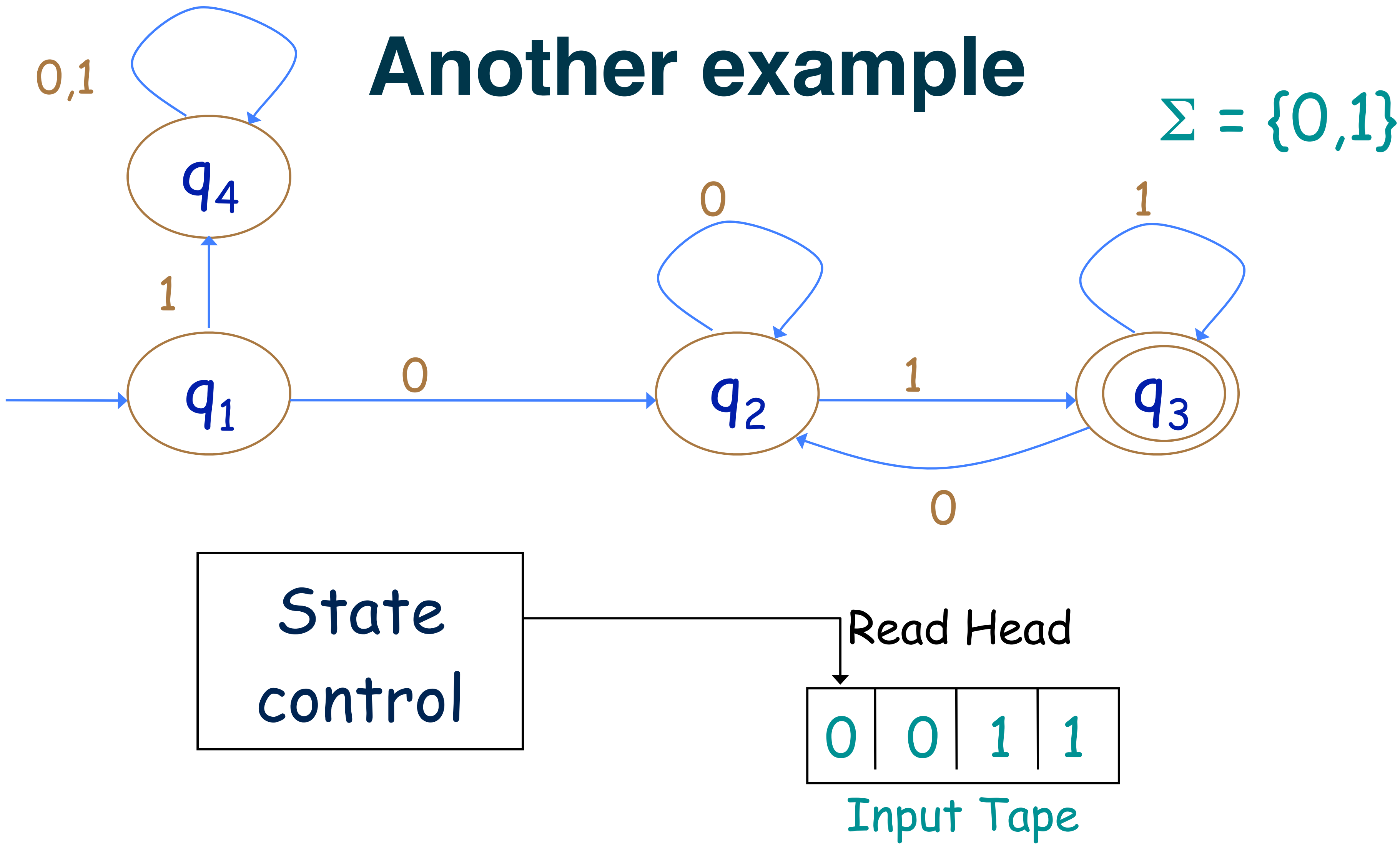
Example

From previous example

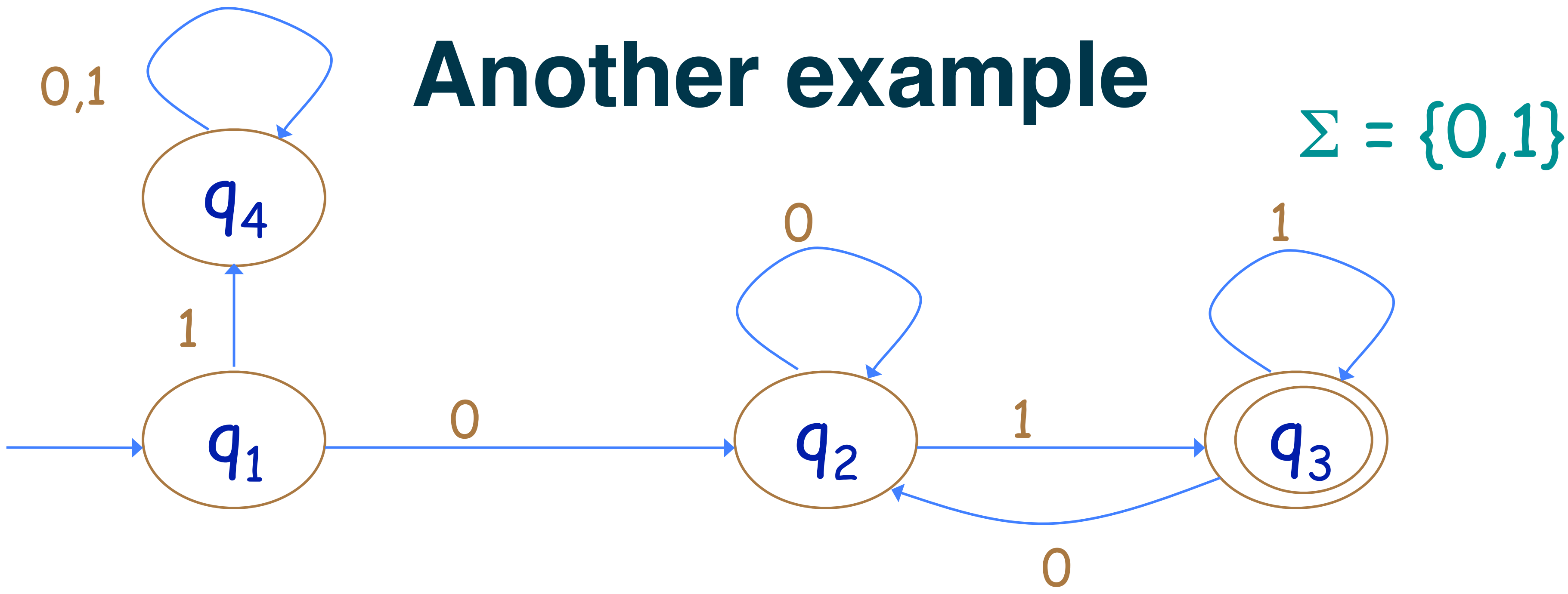
- Q = {Reset, Seen_First, Seen_Second, Unlocked}
- Σ = {🐸, 🚗, 🪑, 🔓}
- δ = *The state table we constructed*
- q_0 = Reset
- F = {Unlocked}

Q states
Σ alphabet
δ transition function
q_0 start state
F accept states

Another example



Another example



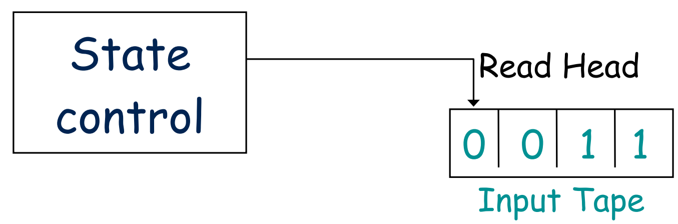
$Q = \{q_1, q_2, q_3, q_4\}$

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

$\delta =$ ...in a moment...

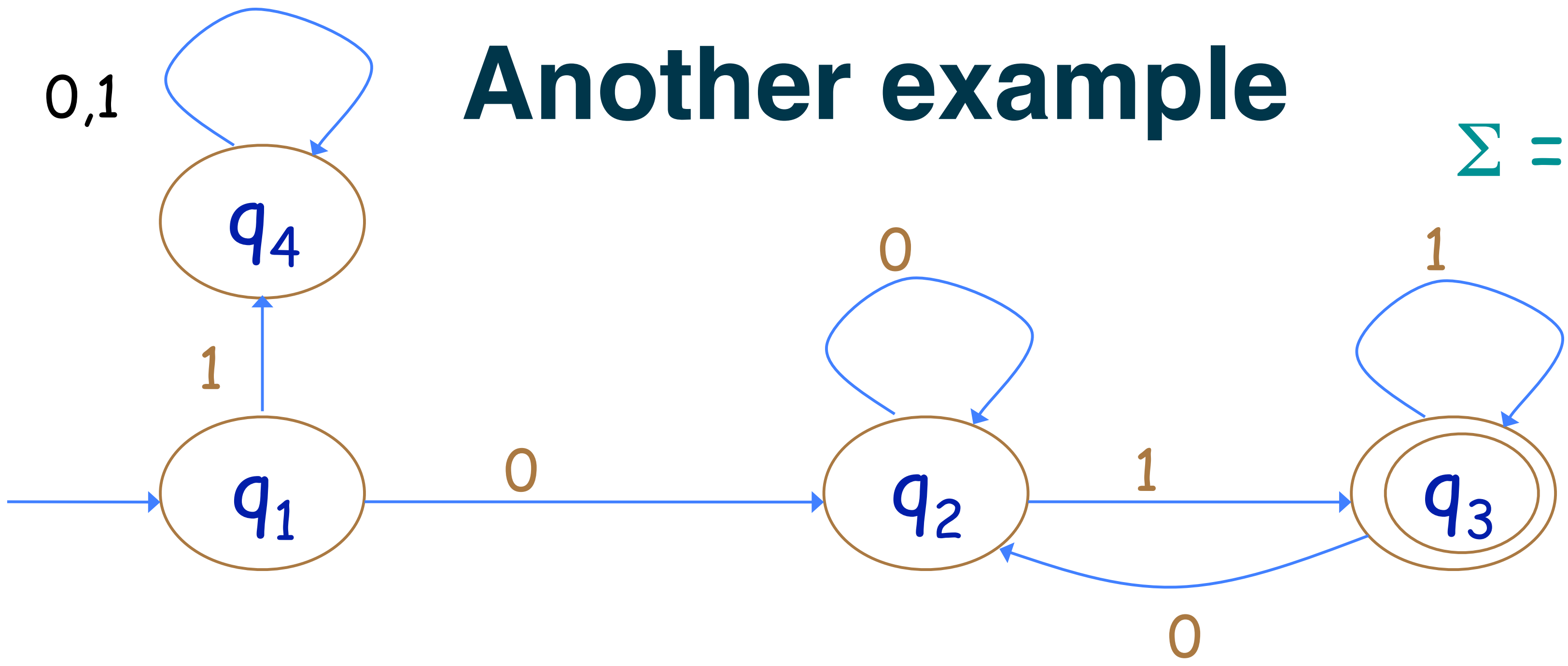
$q_0 = q_1$

$F = \{q_3\}$



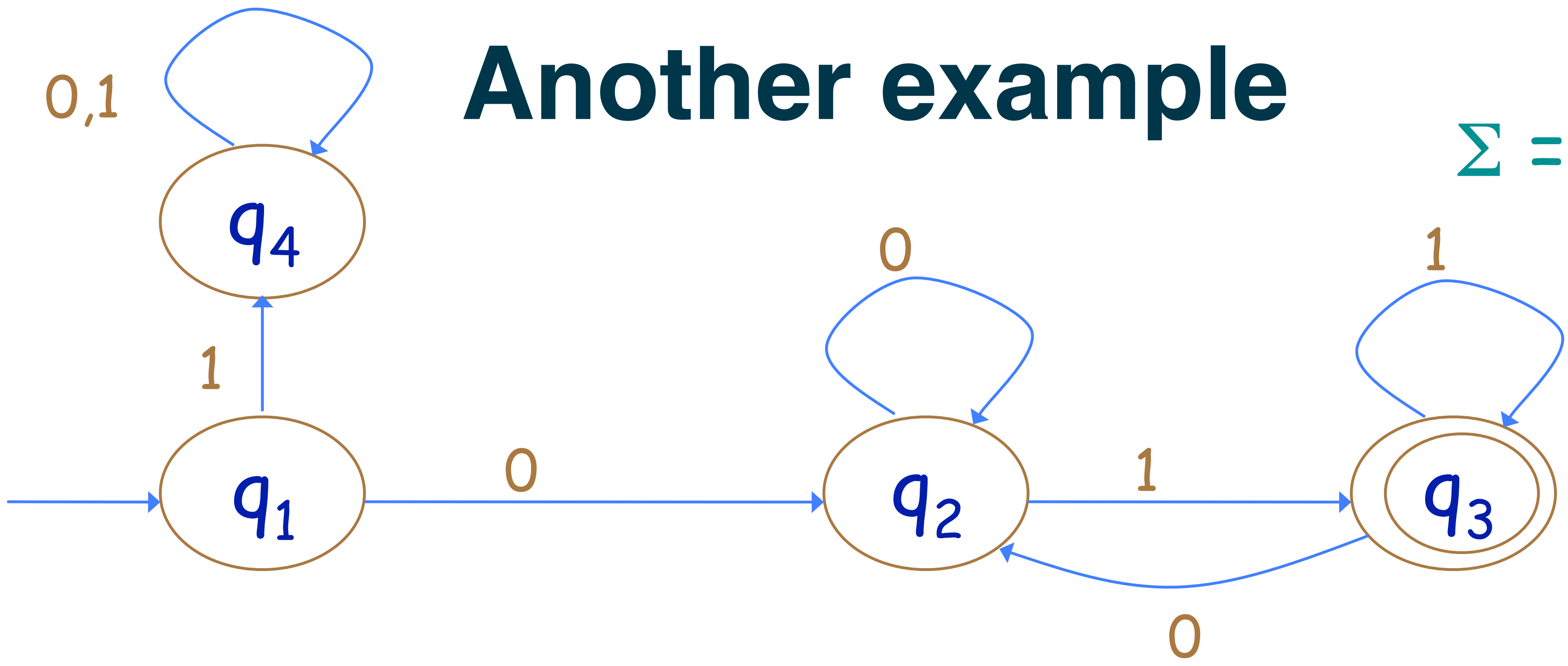
Another example

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



State table	0	1
q_1	q_2	q_4
q_2	q_2	q_3
q_3	q_2	q_3
q_4	q_4	q_4

Another example



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

Informal description of the strings accepted by this DFA

All strings of 0's and 1's beginning with a 0 and ending with a 1

Collaborative Exercises

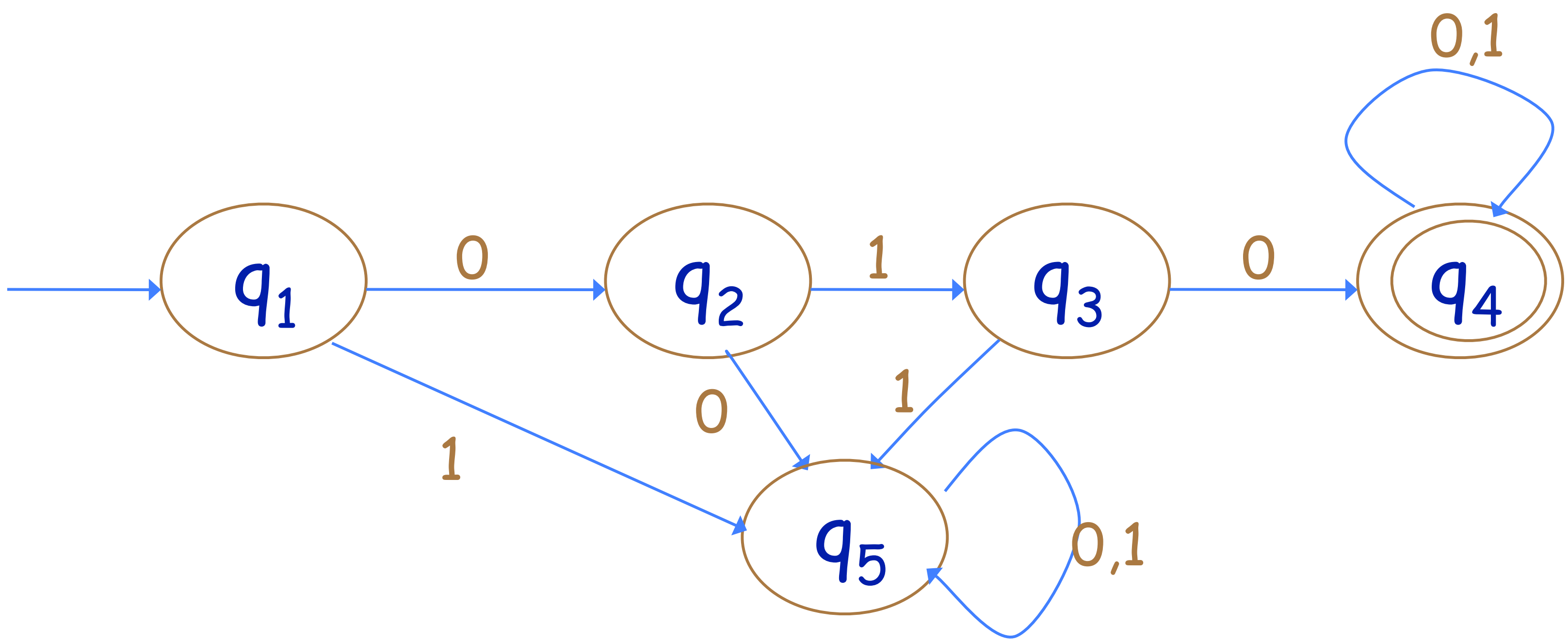
Formally describe the DFA illustrated

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

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

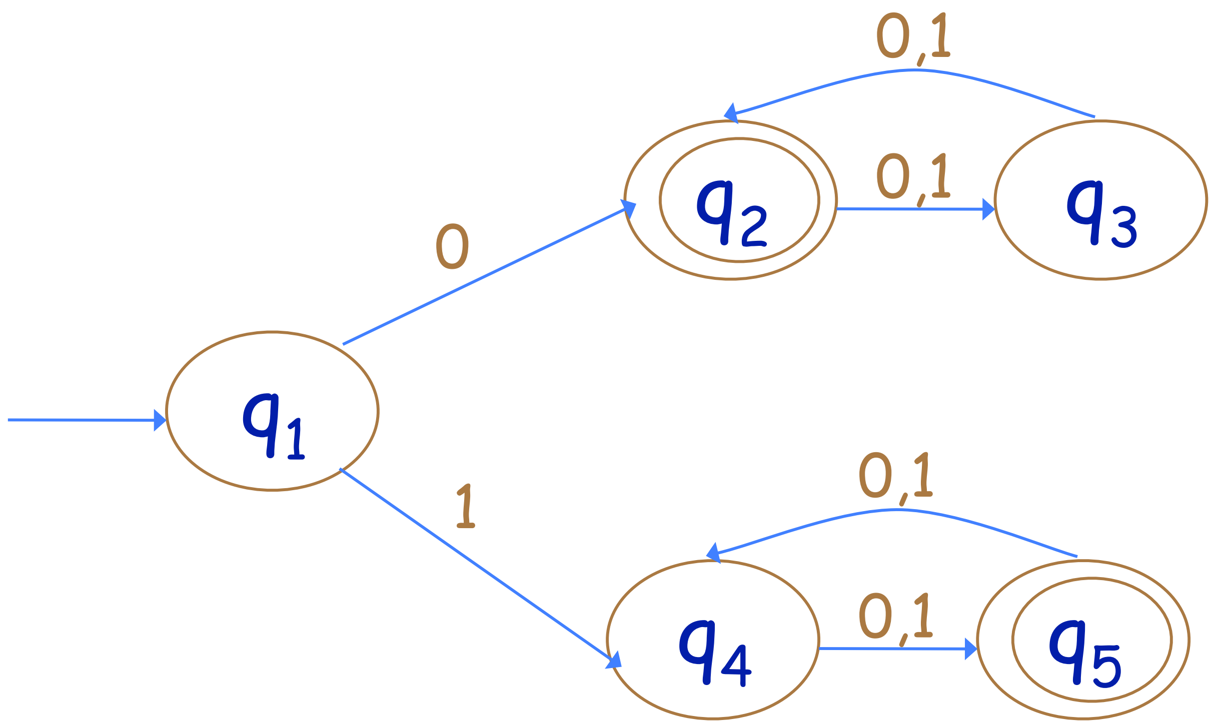
Include informal description

DFA 1



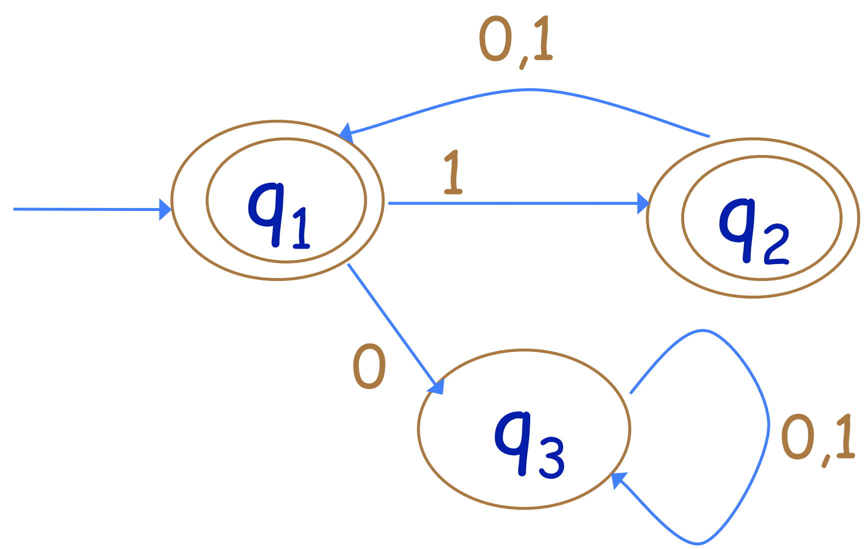
Hint: What strings doesn't this DFA accept?

DFA 2



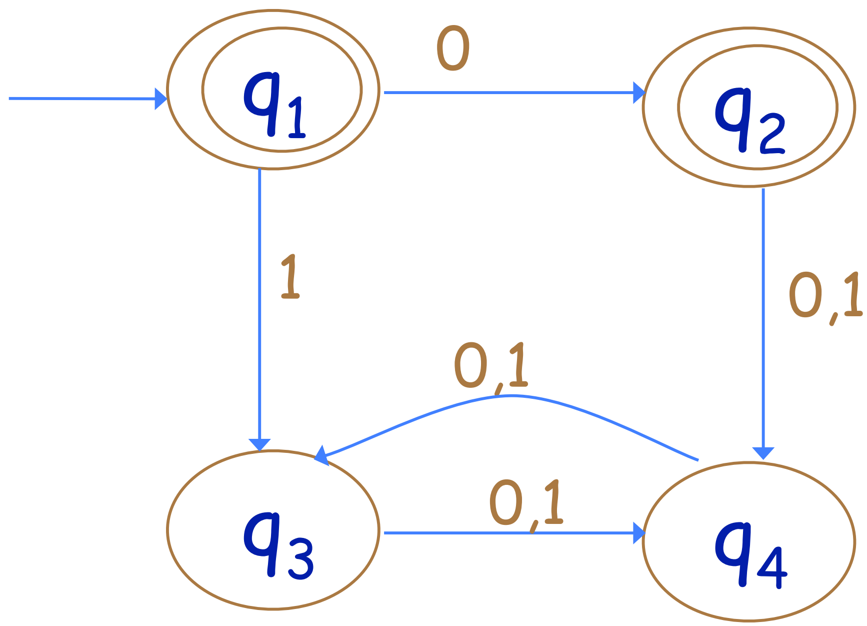
Hint: String length counts.

DFA 3



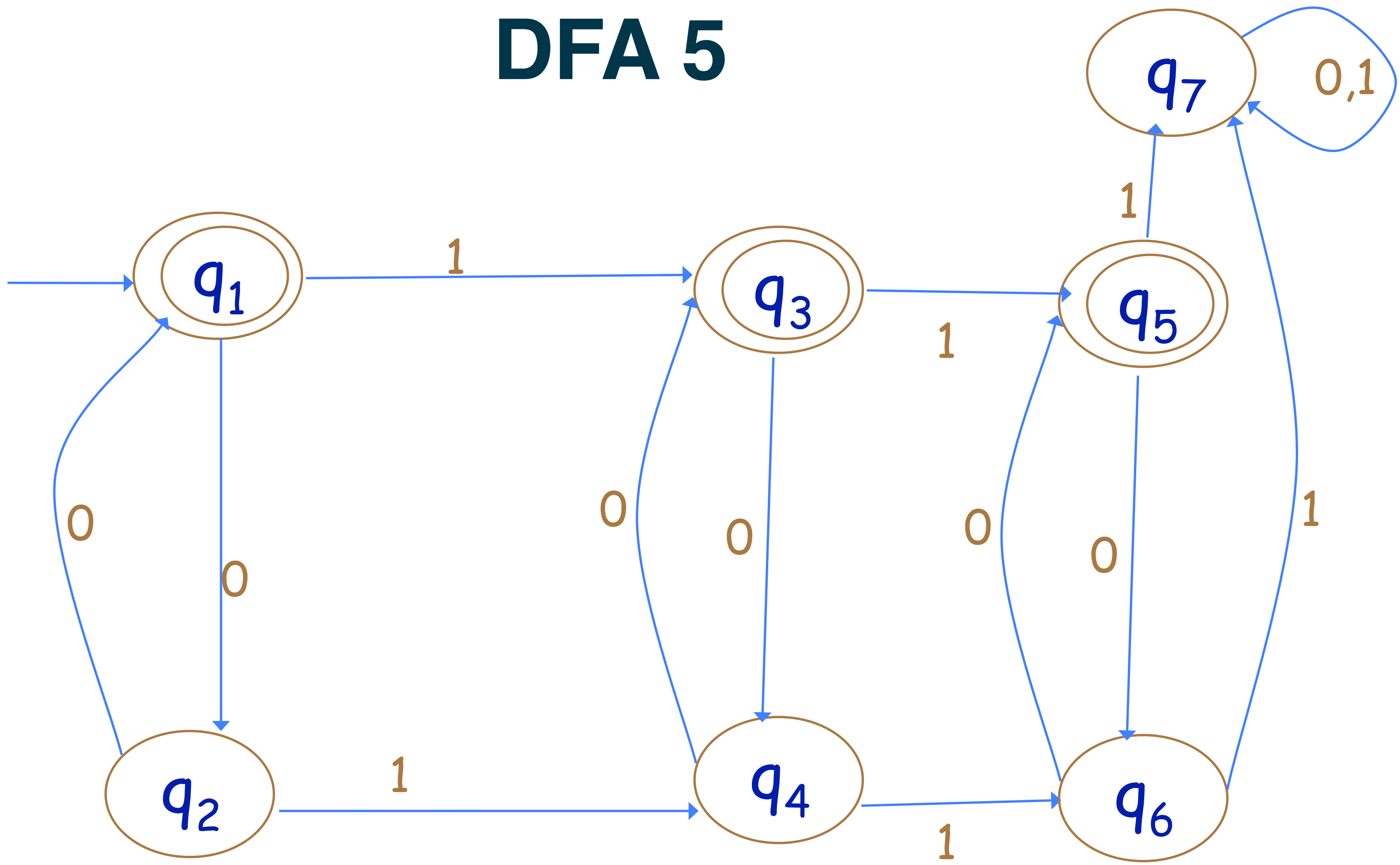
Hint: Symbol position counts.

DFA 4



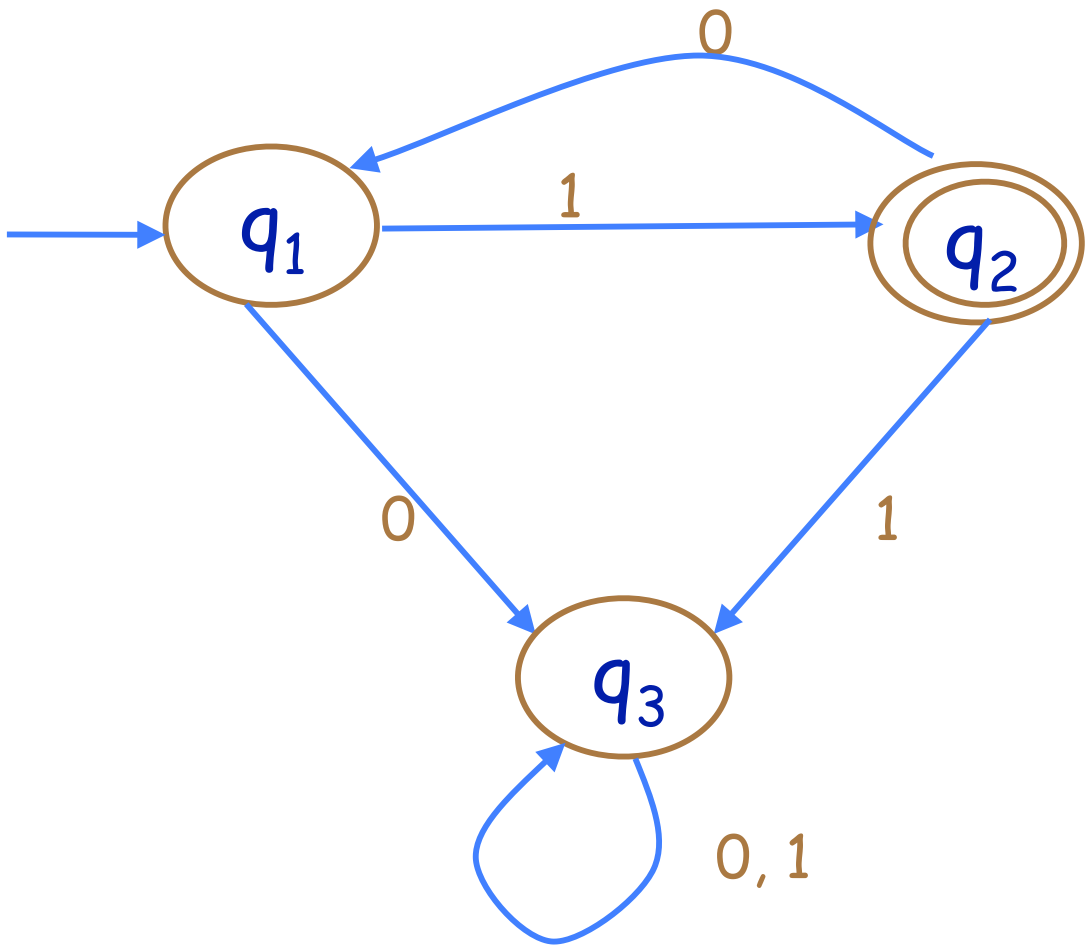
Hint: Can you simplify this DFA?

DFA 5



Hint: For each state, what do you know about how many times each symbol has appeared?

DFA 6



Hint: What happens when you get to q_3?

Formalizing *Computation*

Let $M = (Q,\Sigma,\delta,q_0,F)$ be a finite automaton and let $w = w_1w_2\ldots w_n$ be any string over Σ.

$w_i \in \Sigma$

$\boxed{M \text{ accepts } w}$ if there is a sequence of states $r_0, r_1, \ldots, r_n$, of Q such that

- $r_0 = q_0$

  start in the start state

- $r_i = \delta(r_{i-1}, w_i)$

  the transition function determines each step

- $r_n \in F$

  the last state is one of the final states

Regular Languages

A deterministic finite automaton M **recognizes** the language A if

$$A = \{ w \mid M \text{ accepts } w \}$$

We say A is the **language of** M, $L(M)$

$$L(M) = \{ w \mid M \text{ accepts } w \}$$

Any language recognized by a deterministic finite automaton is called a regular language

Designing Finite Automata

- **Select states specifically to reflect some important concept**
  - **For example…**
    - **even number of 0’s**
    - **odd number of occurrences of the string 010**
- **Ensure this meaning is relevant to the language you are trying to define**
- **Try to get “in the head” of the automaton**

Designing Finite Automata – Examples

1. Design a DFA accepting the following strings over {a}: {a}

2. Design a DFA accepting the following strings over {a}: {ε, a}

3. Design a DFA accepting the following strings over {a}: {ε, a, aa}

4. Design a DFA accepting the following strings over {a}: a*

Designing Finite Automata – Example

Design a DFA accepting all strings over {0,1,2,3} such that the <u>sum</u> of the symbols in the string is equivalent to 2 modulo 4 or 3 modulo 4

Designing Finite Automata – Example

- **What states do we need?**
  - **One state for each value modulo 4**
    - **q1 represents 1 modulo 4**
    - **q2 represents 2 modulo 4**
    - **q3 represents 3 modulo 4**
    - **q4 represents 0 modulo 4**

Designing Finite Automata – Example

Create the state transition table

	0	1	2	3
q_1 (1 mod 4)				
q_2 (2 mod 4)				
q_3 (3 mod 4)				
q_4 (0 mod 4)				

Designing Finite Automata – Example

What elements of the 5-tuple do we know?

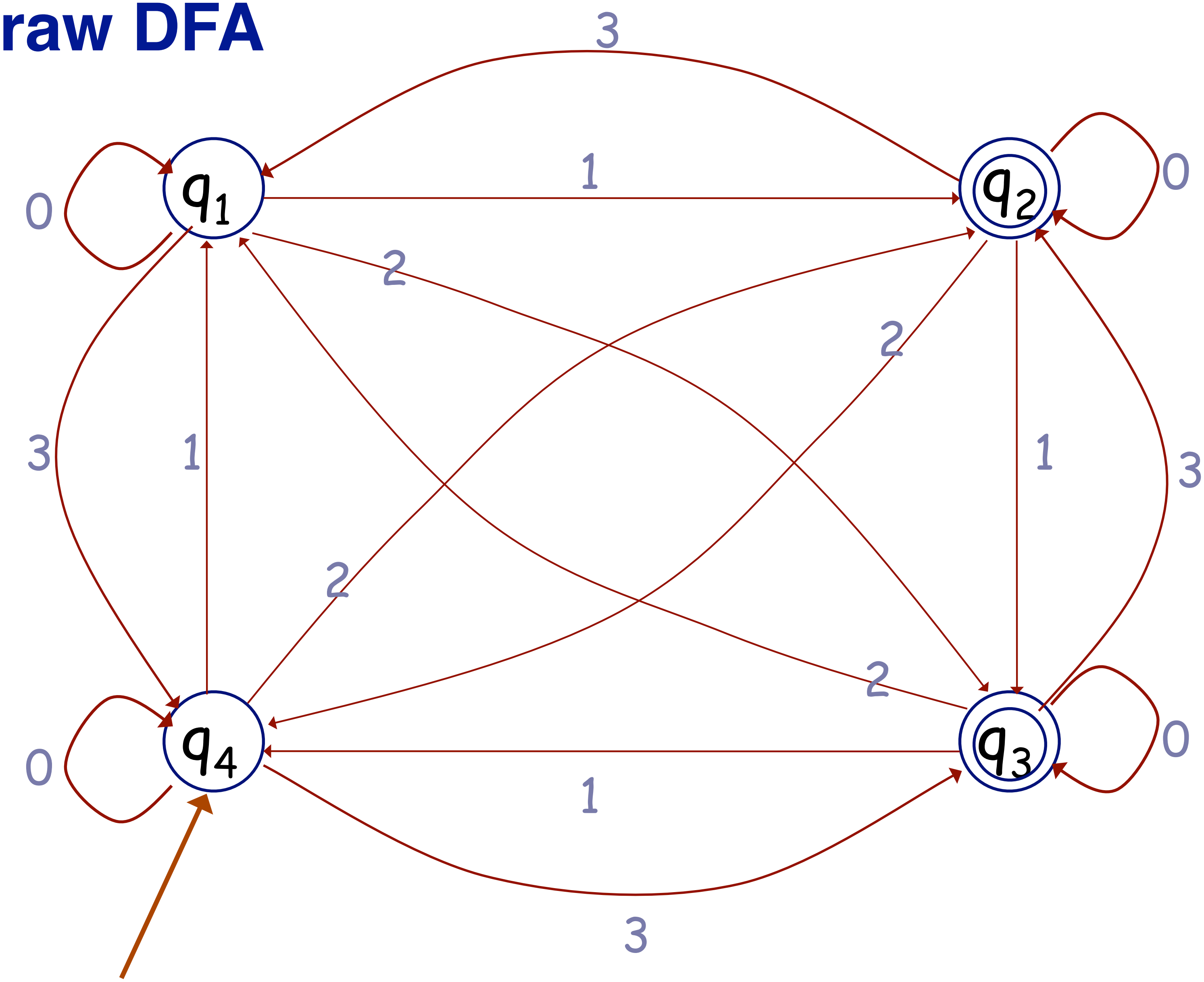
- Q, Σ, and δ

So we still need $q0$ and F

- $q0 = q4$
- $F = \{q2, q3\}$

Designing Finite Automata – Example

Draw DFA



Designing Finite Automata

- Select states specifically to reflect some important concept
- Ensure this meaning is relevant to the language you are trying to define
- Try to get “in the head” of the automaton
- Can also design a DFA by combining two other DFA’s

Combining Regular Languages

We can create a regular language from other regular languages A and B using specific allowable operations called regular operations

- **Union:** $A \cup B$
- **Concatenation:** $A \bullet B$
- **Kleene star:** A^*

Union Is a Regular Operation

Theorem: **The class of regular languages is closed under the union operation**

Proof approach: **Assume A_1 and A_2 are both regular languages with $A_1=L(M_1)$ and $A_2=L(M_2)$ and create a DFA M such that $L(M) = A_1 \cup A_2$**

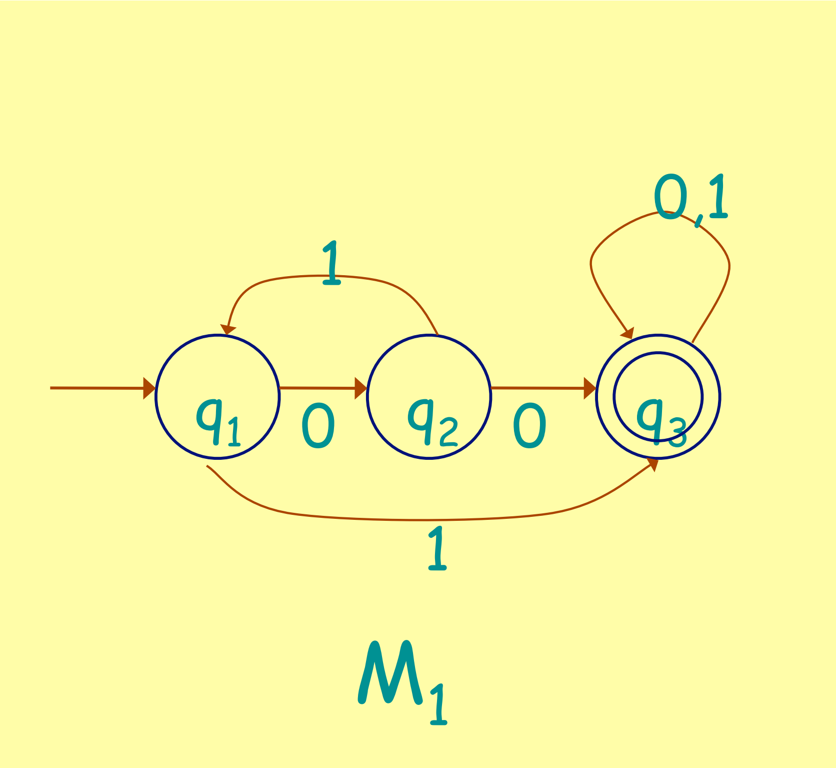
Method: **Proof by construction**

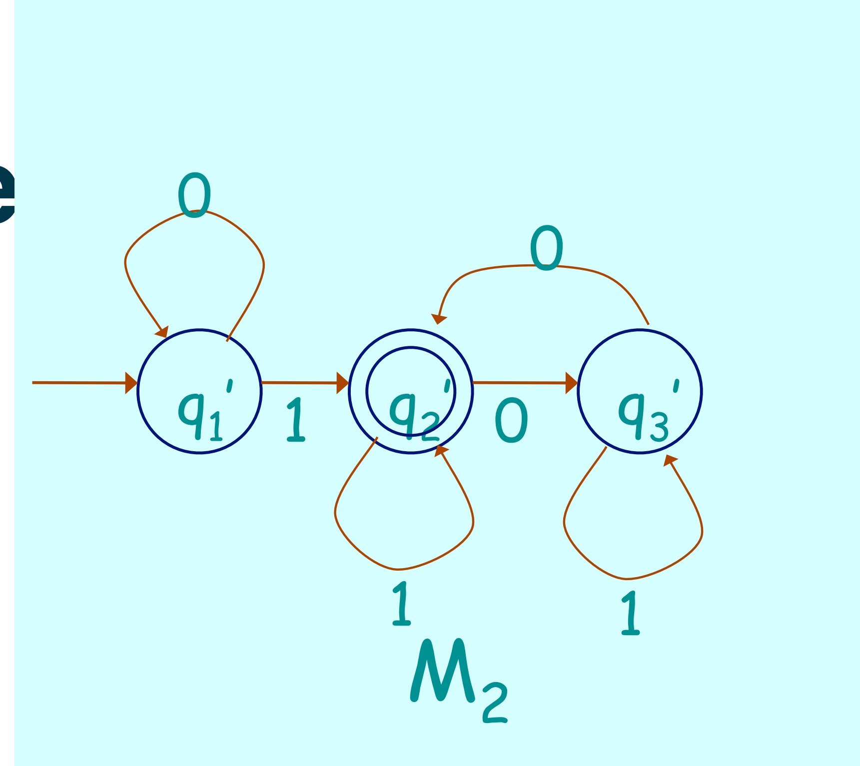
Construction Idea

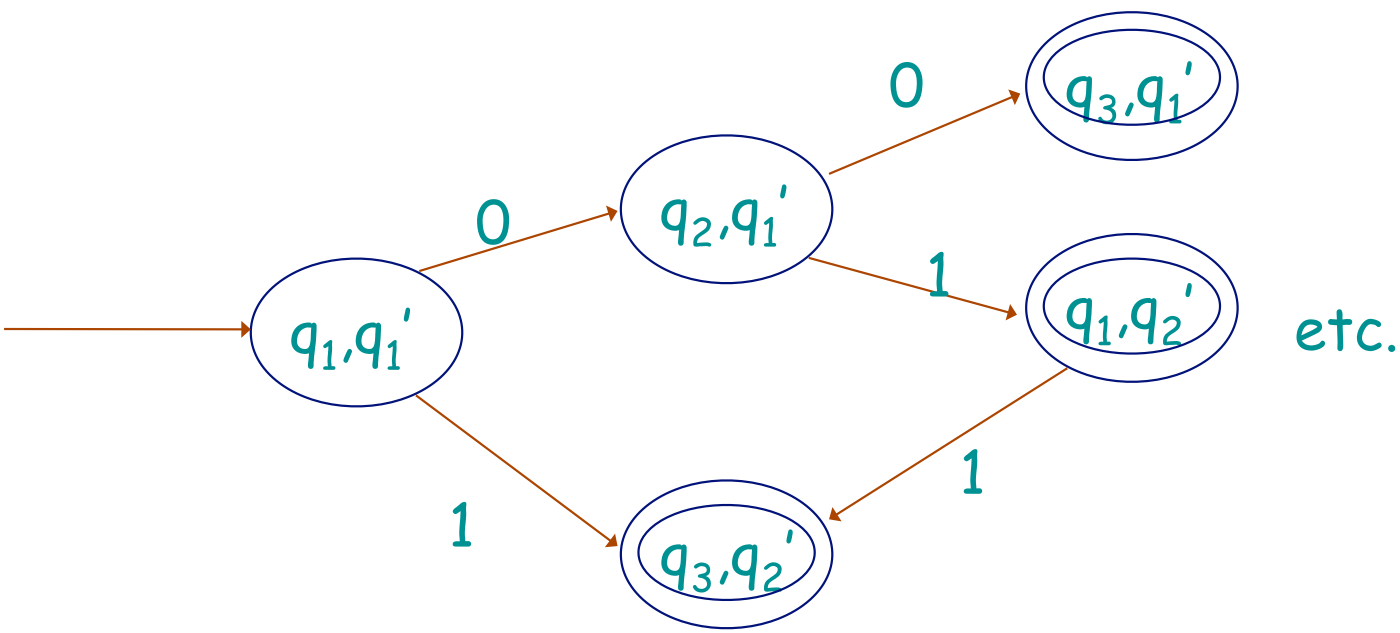
Each state of the new DFA represents both where the same word would be if it was being processed in M_1 and where it would be if it were processed in M_2

- Keep track of the progress of the string in both DFA's simultaneously

Example







Maximum number of states?

9

Product of number of states in M_1 and in M_2

Union Is a Regular Operation

Theorem: The class of regular languages is closed under the union operation

Proof approach: Assume A_1 and A_2 are both regular languages with $A_1 = L(M_1)$ and $A_2 = L(M_2)$ and create a DFA M such that $L(M) = A_1 \cup A_2$

Method: Proof by construction

Formally defining M

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

- $Q = Q_1 \times Q_2$

  Q_1 and Q_2 are the states in machines M_1 and M_2, respectively

- $\Sigma = \Sigma_1 \cup \Sigma_2$

  Σ_1 and Σ_2 are the alphabets for machines M_1 and M_2, respectively

- $\delta((r_1,r_2),a) = (\delta_1(r_1,a), \delta_2(r_2,a))$

  δ_1 and δ_2 are the state transition functions for machines M_1 and M_2, respectively

Formally defining M

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

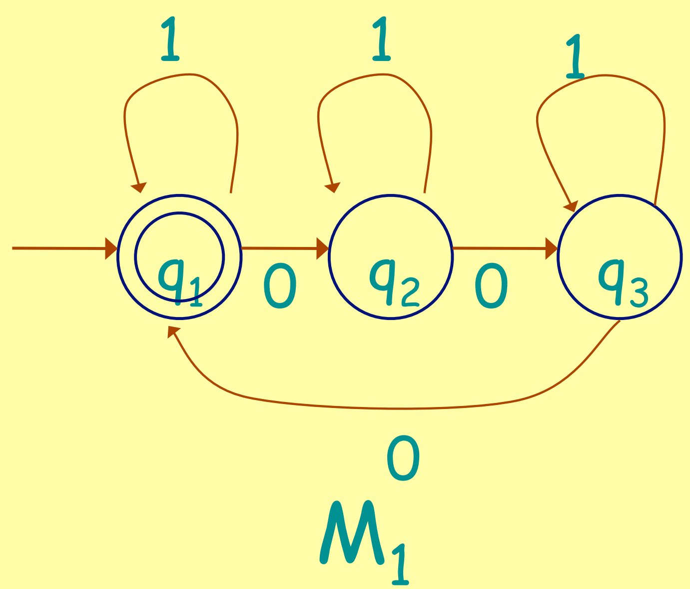
- $q_0 = (r_1, r_2)$

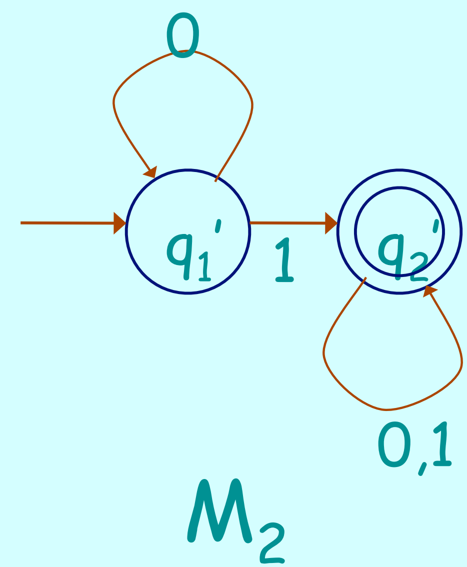
  r_1 and r_2 are the starting states in machines M_1 and M_2, respectively

- $F = \{(r_1,r_2) \mid r_1 \in F_1$ **or** $r_2 \in F_2\}$

  F_1 and F_2 are the accepting states for machines M_1 and M_2, respectively

Another Example





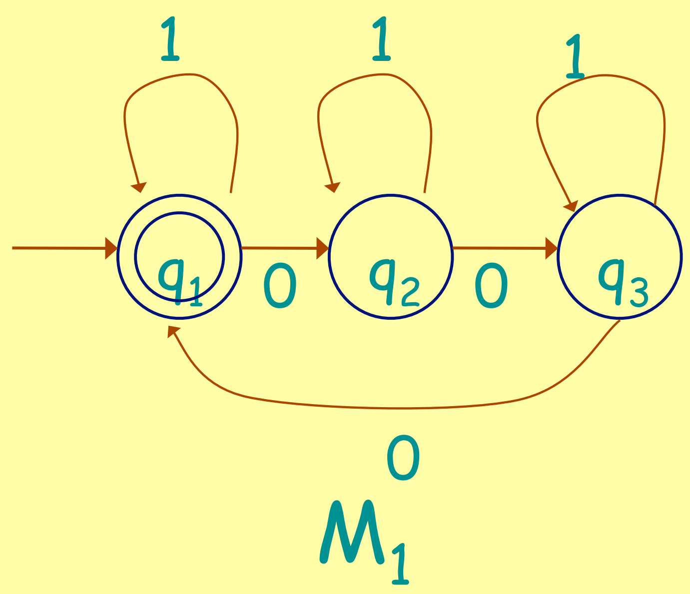
$Q = \{(q_1,q_1'), (q_1,q_2'), (q_2,q_1'), (q_2,q_2'), (q_3,q_1'), (q_3,q_2')\}$

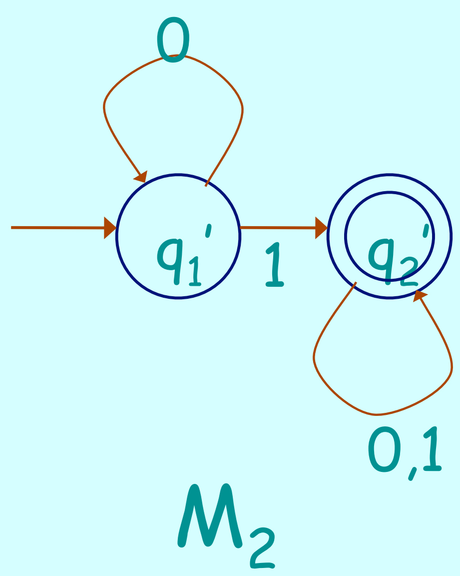
$\Sigma = \{0,1\}$

$q_0 = (q_1,q_1')$

$F = \{(q_1,q_1'), (q_1,q_2'), (q_2,q_2'), (q_3,q_2')\}$

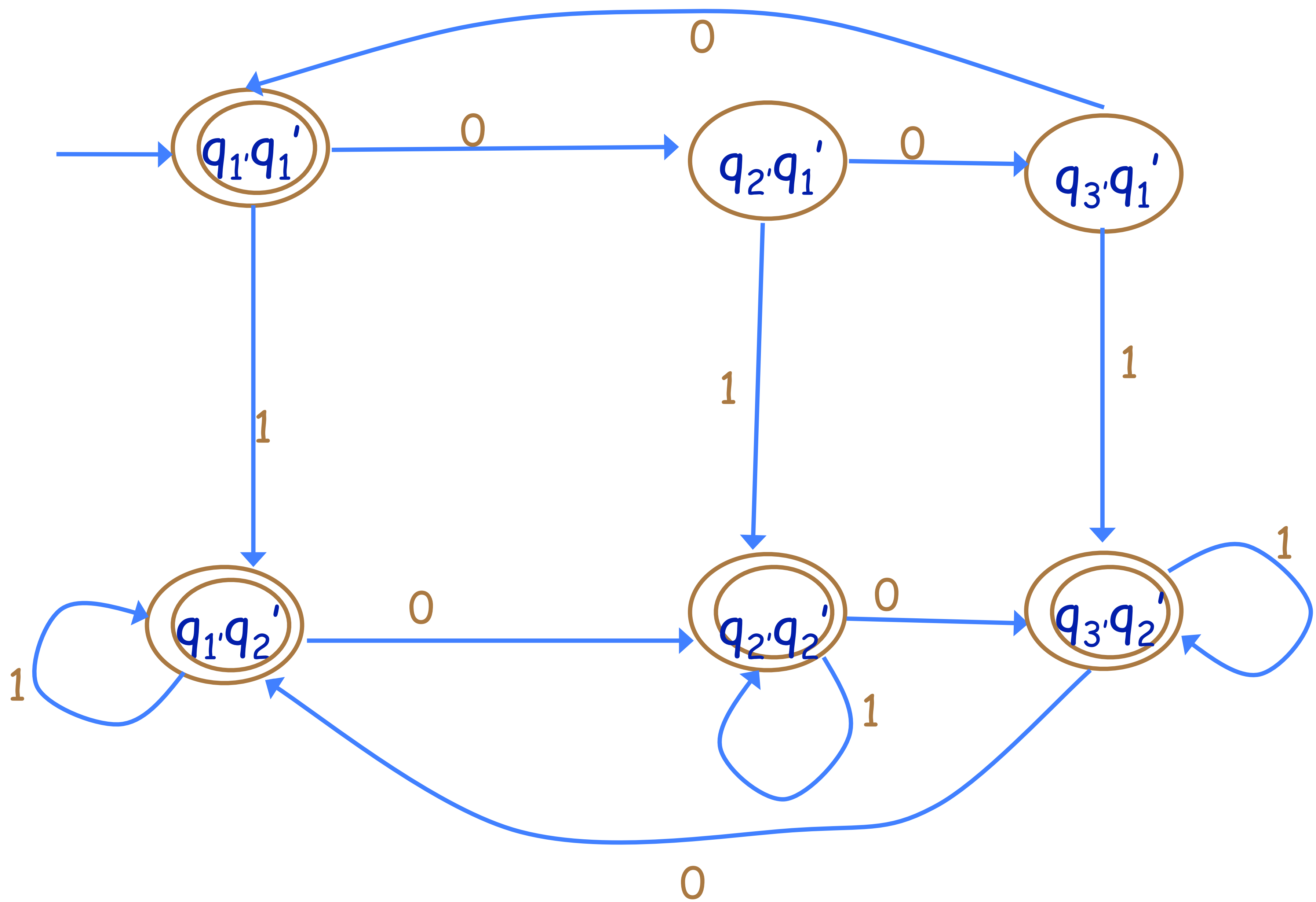
Another Example





δ	0	1
(q_1,q_1')	(q_2,q_1')	(q_1,q_2')
(q_1,q_2')	(q_2,q_2')	(q_1,q_2')
(q_2,q_1')	(q_3,q_1')	(q_2,q_2')
(q_2,q_2')	(q_3,q_2')	(q_2,q_2')
(q_3,q_1')	(q_1,q_1')	(q_3,q_2')
(q_3,q_2')	(q_1,q_2')	(q_3,q_2')

Another Example



Concatenation is a Regular Operation

Theorem: The class of regular languages is closed under the concatenation operation

Proof approach: Assume A_1 and A_2 are both regular languages with $A_1=L(M_1)$ and $A_2=L(M_2)$ then create a DFA M such that $L(M) = A_1 \bullet A_2$

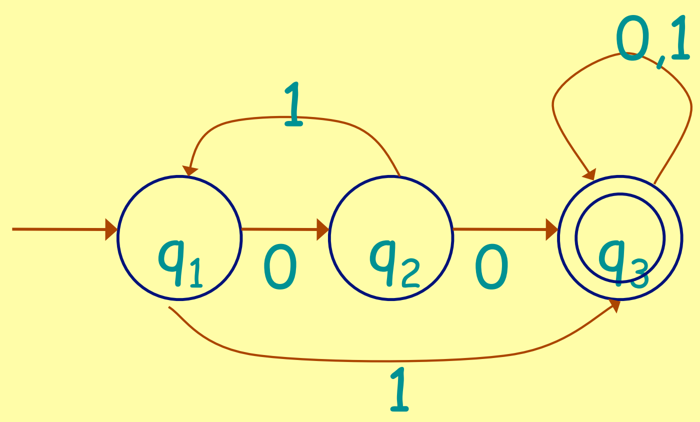
Method: Proof by construction

Construction Idea

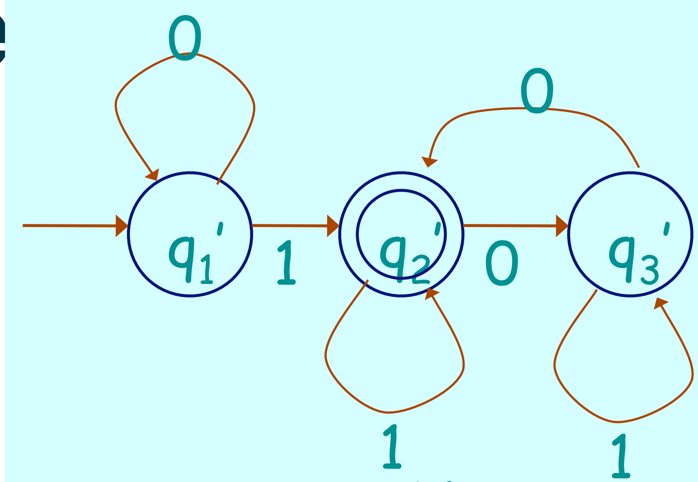
Every accepting state in M_1 has a copy of M_2 "tacked on"

Problem:
If we tack a copy of M_2 on at each accepting states, we lose the deterministic property

Example

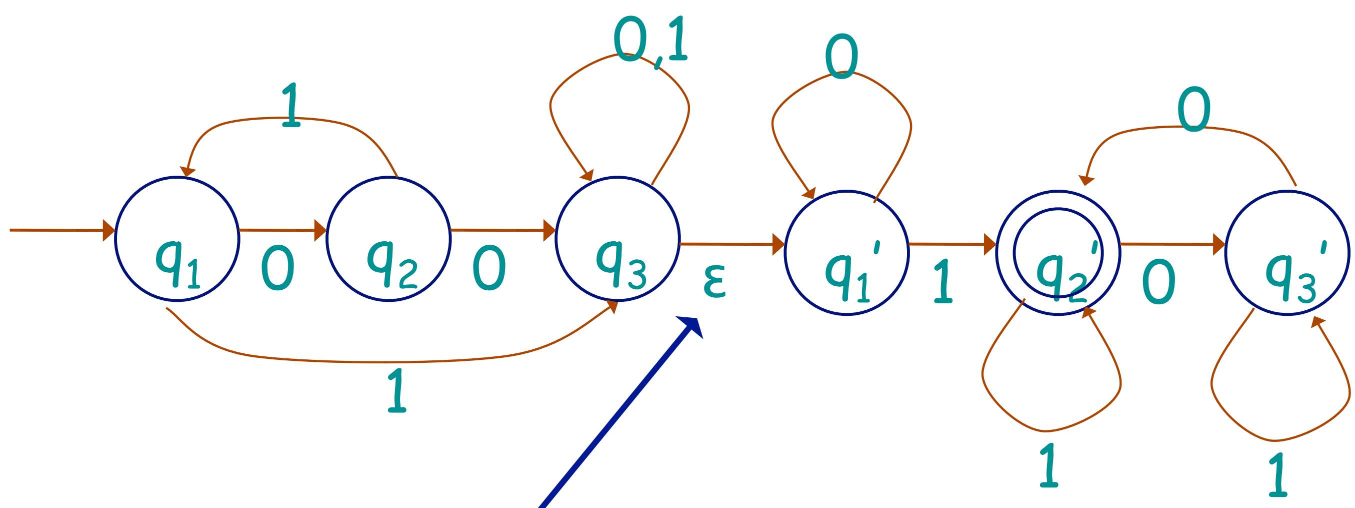


M_1



M_2

Find M such that $L(M) = L(M_1) \bullet L(M_2)$



Can jump to q_1' nondeterministically