# CS420

## Introduction to the Theory of Computation

### Lecture 2: An algebra of automatons

Tiago Cogumbreiro

# Today we will introduce...

- Standard operations on languages (union, concatenation, exponentiation, kleene star)
- The nil automaton
- The empty automaton
- The character automaton
- The union automaton

Section 1.1

# Formal definition of a Finite Automaton

## Definition 1.5

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

1. $Q$ is a finite set called states
2. $\Sigma$ is a finite set called alphabet
3. $\delta\colon Q \times \Sigma \to Q$ is the transition function
   *($\delta$ takes a state and an alphabet and produces a state)*
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accepted states

> A formal definition is a precise mathematical language. In this example, item declares a name and possibly some constraint, e.g., $q_0 \in Q$ is saying that $q_0$ **must** be in set $Q$. These constraints are visible in the code in the form of assertions.

# Formal declaration of our running example

Let the running example be the following finite automaton $M_{turnstile}$

$$(\{\texttt{Open}, \texttt{Close}\}, \{\texttt{Neither}, \texttt{Front}, \texttt{Rear}, \texttt{Both}\}, \delta, \texttt{Close}, \{\texttt{Close}\})$$

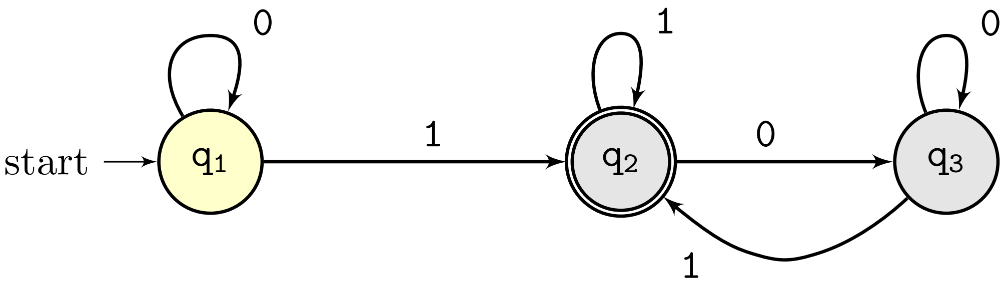
where

$$\delta(\texttt{Close}, \texttt{Front}) = \texttt{Open}$$
$$\delta(\texttt{Open}, \texttt{Neither}) = \texttt{Close}$$
$$\delta(q, i) = q$$

**Facts**

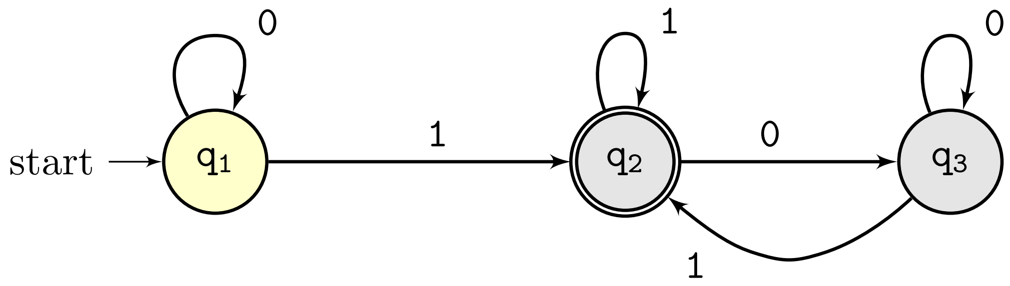
- $M_{turnstile}$ accepts [Front, Neither]
- $M_{turnstile}$ rejects [Rear, Front, Front]
- $M_{turnstile}$ accepts [Rear, Front, Rear, Neither, Rear]

# Example




States?

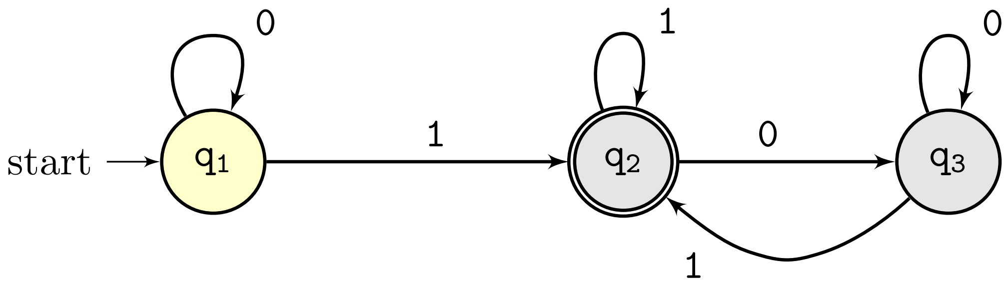
# Example




States? $Q = \{q_1, q_2, q_3\}$

Alphabet?

# Example

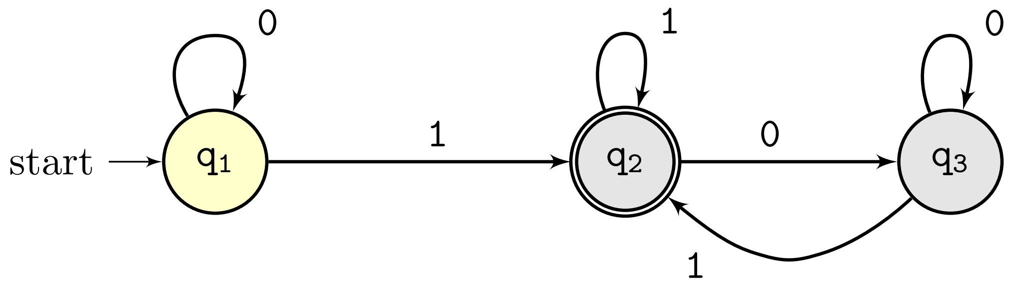



States? $Q = \{q_1, q_2, q_3\}$

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

Transition table $\delta$?

# Example



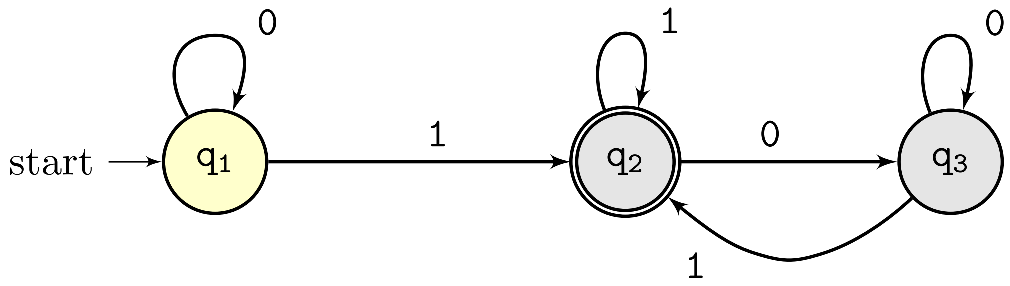

States? $Q = \{q_1, q_2, q_3\}$

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

Transition table $\delta$?

| ***(prev)*** | ***0*** | ***1*** |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_3$ | $q_2$ |

# Example




States? $Q = \{q_1, q_2, q_3\}$

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

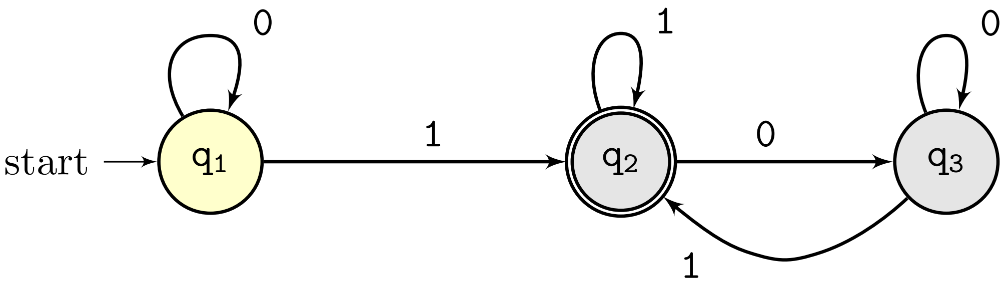
Transition table $\delta$?

| ***(prev)*** | ***0*** | ***1*** |
|---|---|---|
| $q_1$ | $q_1$ | $q_2$ |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | $q_3$ | $q_2$ |

Finite Automaton:

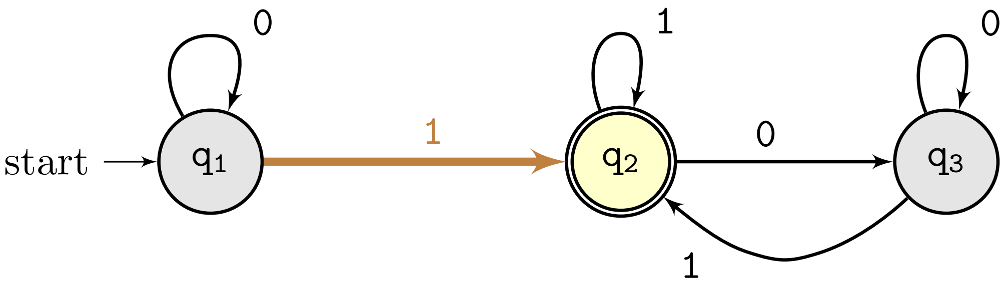
$$(\{q_1, q_2, q_3\}, \{0, 1\}, q_1, \{q_2\})$$

# Example



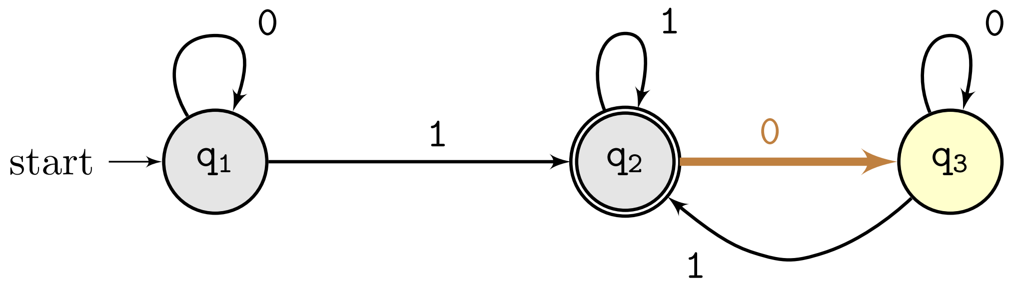

[1, 0, 1, 1]

# Example



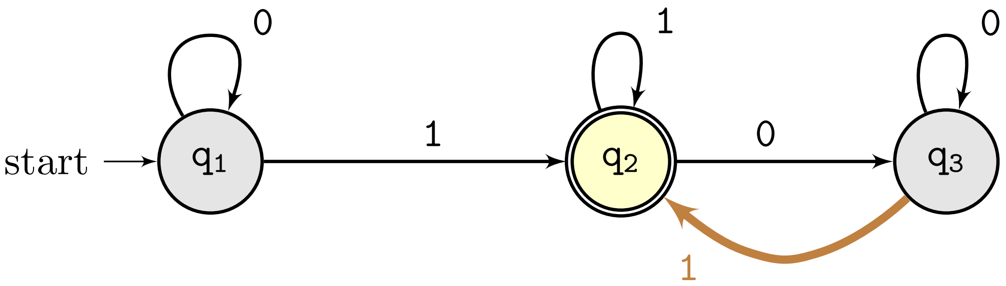

[**1**, 0, 1, 1]

# Example



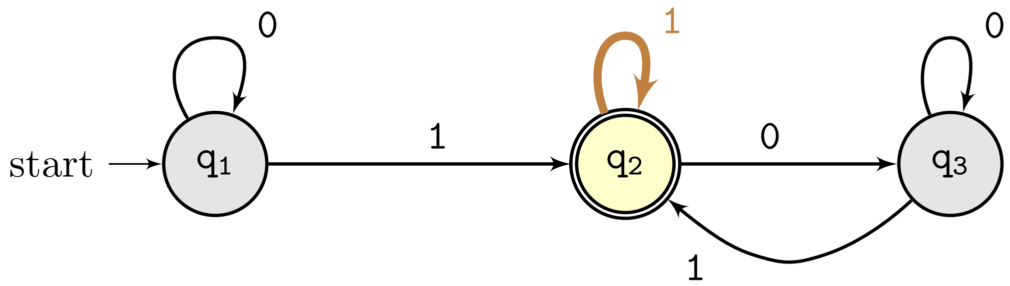

[1, **0**, 1, 1]

# Example




[1, 0, **1**, 1]

# Example




[1, 0, 1, **1**]

What are the set of inputs accepted by this automaton?

# What are the set of inputs accepted by this automaton?

**Answer:** Strings terminating in 1

# The language of a machine

## Definition: language of a machine

1. We define $\mathrm{L}(M)$ to be the set of all strings accepted by finite automaton $M$.
2. Let $A = \mathrm{L}(M)$, we say that the finite automaton $M$ **recognizes** the set of strings $A$.

# The language of a machine

## Definition: language of a machine

1. We define $\mathrm{L}(M)$ to be the set of all strings accepted by finite automaton $M$.
2. Let $A = \mathrm{L}(M)$, we say that the finite automaton $M$ **recognizes** the set of strings $A$.

**Notes**

- The language is the *set* of all possible alphabet-sequences recognized by a finite automaton
- Since $\mathrm{L}(M)$ is a **total** function, then the language recognized by a machine always exists and is unique
- A language may be empty
- We **cannot** write a program that returns the language of an arbitrary finite automaton. Why? **Because the language set may be infinite. How could a program return $\Sigma^\star$?**

A total function is defined for all inputs.

# From a language to an automaton

## Define a finite automaton that recognizes `{[h,i], [h,o]}`

- What is the alphabet of the finite automaton?

# From a language to an automaton

## Define a finite automaton that recognizes `{[h,i], [h,o]}`

- What is the alphabet of the finite automaton? $\{h, i, o\}$
- What are the states of the finite automaton?

# From a language to an automaton

## Define a finite automaton that recognizes `{[h,i], [h,o]}`

- What is the alphabet of the finite automaton? $\{h, i, o\}$
- What are the states of the finite automaton?

1. We need an initial state, say $q_1$ that reads h and moves it to $q_2$

# From a language to an automaton

## Define a finite automaton that recognizes `{[h,i], [h,o]}`

- What is the alphabet of the finite automaton? $\{h, i, o\}$
- What are the states of the finite automaton?

1. We need an initial state, say $q_1$ that reads h and moves it to $q_2$
2. We need another state, say $q_3$, after reading h it reads `i` (from $q_2$)

# From a language to an automaton

## Define a finite automaton that recognizes `{[h,i], [h,o]}`

- What is the alphabet of the finite automaton? $\{h, i, o\}$
- What are the states of the finite automaton?

1. We need an initial state, say $q_1$ that reads `h` and moves it to $q_2$
2. We need another state, say $q_3$, after reading `h` it reads `i` (from $q_2$)
3. We need another state, say $q_4$, that reads `o` after having read `h` (from $q_2$)

# From a language to an automaton

## Define a finite automaton that recognizes `{[h,i], [h,o]}`

- What is the alphabet of the finite automaton? $\{h, i, o\}$
- What are the states of the finite automaton?

1. We need an initial state, say $q_1$ that reads `h` and moves it to $q_2$
2. We need another state, say $q_3$, after reading `h` it reads `i` (from $q_2$)
3. We need another state, say $q_4$, that reads `o` after having read `h` (from $q_2$)

- What are the accepted states?

# From a language to an automaton

## Define a finite automaton that recognizes `{[h,i], [h,o]}`

- What is the alphabet of the finite automaton? $\{h, i, o\}$
- What are the states of the finite automaton?

1. We need an initial state, say $q_1$ that reads `h` and moves it to $q_2$
2. We need another state, say $q_3$, after reading `h` it reads `i` (from $q_2$)
3. We need another state, say $q_4$, that reads `o` after having read `h` (from $q_2$)

- What are the accepted states? $\{q_2, q_3\}$
- What happens if we read $h$ in $q_3$?

# From a language to an automaton

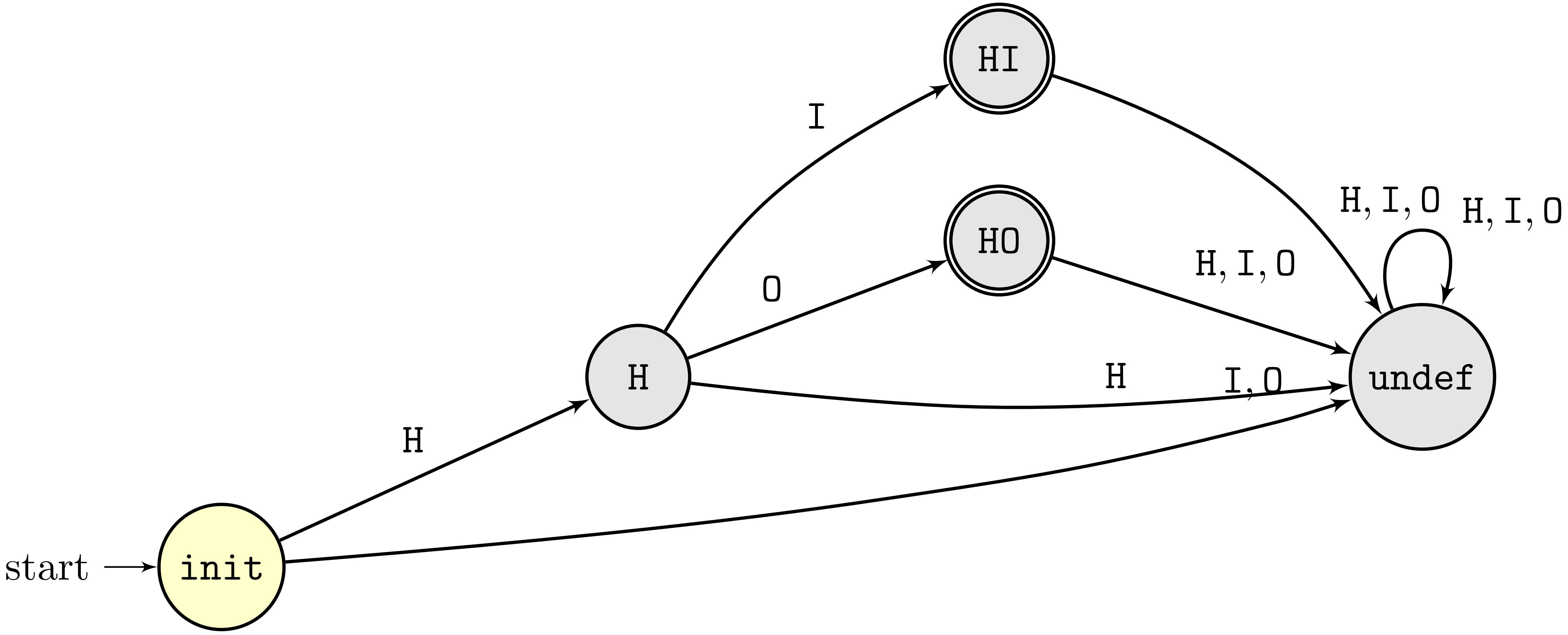
## Define a finite automaton that recognizes `{[h,i], [h,o]}`

- What is the alphabet of the finite automaton? $\{h, i, o\}$
- What are the states of the finite automaton?

1. We need an initial state, say $q_1$ that reads `h` and moves it to $q_2$
2. We need another state, say $q_3$, after reading `h` it reads `i` (from $q_2$)
3. We need another state, say $q_4$, that reads `o` after having read `h` (from $q_2$)

- What are the accepted states? $\{q_2, q_3\}$
- What happens if we read $h$ in $q_3$? We need a "reject" state, say $q_5$, that every unexpected letter takes us to.

# Example



# Standard operations on languages

# Standard operations on languages

1. union (since a language is a set of strings, we can use the union of two languages)
2. concatenation
3. the Kleene star

# Concatennation

- $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$

## Examples

1. $\{a, aa\} \cdot \{b, bb\} =$

# Concatennation

- $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$

## Examples

1. $\{a, aa\} \cdot \{b, bb\} = \{ab, aab, abb, aabb\}$
2. $\{a, aa, aaa\} \cdot \{b, bb\} =$

# Concatennation

- $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$

## Examples

1. $\{a, aa\} \cdot \{b, bb\} = \{ab, aab, abb, aabb\}$
2. $\{a, aa, aaa\} \cdot \{b, bb\} = \{ab, abb, aab, aabb, aaab, aaabb\}$
3. $\{a, aa, aaa\} \cdot \emptyset =$

# Concatennation

- $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$

## Examples

1. $\{a, aa\} \cdot \{b, bb\} = \{ab, aab, abb, aabb\}$
2. $\{a, aa, aaa\} \cdot \{b, bb\} = \{ab, abb, aab, aabb, aaab, aaabb\}$
3. $\{a, aa, aaa\} \cdot \emptyset = \emptyset$
4. $\{a, aa, aaa\} \cdot \{\epsilon\} =$

# Concatennation

- $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \land w_2 \in L_2\}$

## Examples

1. $\{a, aa\} \cdot \{b, bb\} = \{ab, aab, abb, aabb\}$
2. $\{a, aa, aaa\} \cdot \{b, bb\} = \{ab, abb, aab, aabb, aaab, aaabb\}$
3. $\{a, aa, aaa\} \cdot \emptyset = \emptyset$
4. $\{a, aa, aaa\} \cdot \{\epsilon\} = \{a, aa, aaa\}$

# Exponentiation

- $L^0 = \{\epsilon\}$
- $L^{n+1} = L \cdot (L^n)$

Alternatively:

- $L^n = \{w^n \mid w \in L\}$

## Examples

- $\{a, b\}^0 = \{\epsilon\}$
- $\{a, b\}^1 = \{a, b\}$
- $\{a, b\}^2 = \{aa, ab, ba, bb\}$

# The Kleene star

- $L^\star = \{w^n \mid w \in L \land n \geq 0\}$

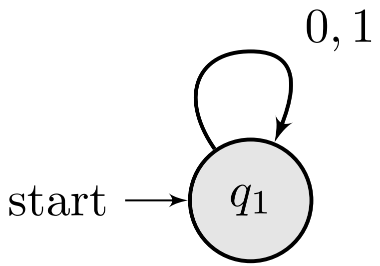
## Examples

- $\{a\}^\star = \{w \mid \text{words that only contain } a\}$
- $\{a, b\}^\star = \{a, b\}^0 \cup \{a, b\}^1 \cup \{a, b\}^2 \cup \cdots \cup \{a, b\}^n$

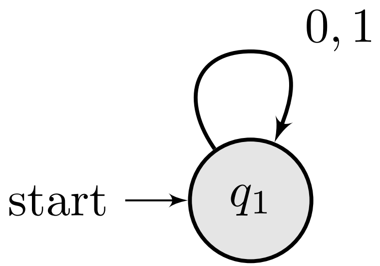
# The nil automaton

$$L(\mathbf{nil}_{\Sigma}) = \emptyset$$

# The nil automaton



# The nil automaton




- Note the absence of accepted states

```
def make_nil(alphabet):
  Q1 = "q_1"
  def transition(q, a):
    return q
  return DFA([Q1], alphabet, transition, Q1, lambda x: False)
```

# The empty automaton

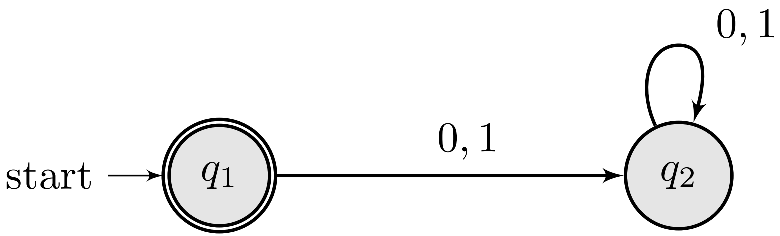
$$L(\mathbf{empty}_\Sigma) = \{\epsilon\}$$

# The empty automaton

Build an automaton that *only* accepts the empty string $\epsilon$. You can imagine it to be akin to the zero of finite automatons.

# The empty automaton

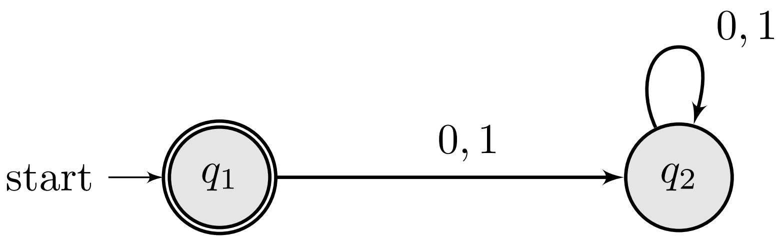
> Build an automaton that *only* accepts the empty string $\epsilon$. You can imagine it to be akin to the zero of finite automatons.




```python
def make_empty(alphabet):
  return DFA(
    states = ["q_1", "q_2"],
    alphabet = alphabet,
    transition_func = lambda q, a: "q_2",
    start_state = "q_1",
    accepted_states = lambda x: x == "q_1")
```

# The empty automaton

We define function $\mathrm{zero}$ that takes an alphabet $\Sigma$ as input and outputs an automation that only accepts the empty string whose alphabet is $\Sigma$.




$$\mathbf{empty}_\Sigma = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_1\})$$

where

$$\delta(q, i) = q_2$$

# The character-automaton

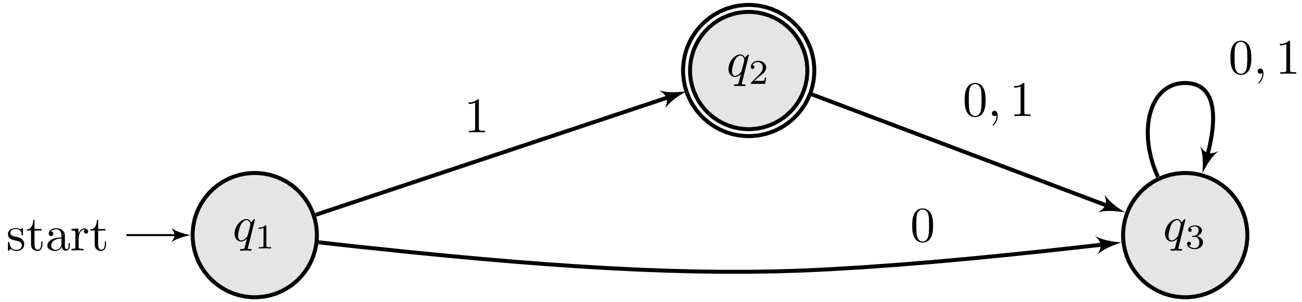
$$L(\mathbf{char}(a)) = \{a\}$$

# The character automaton

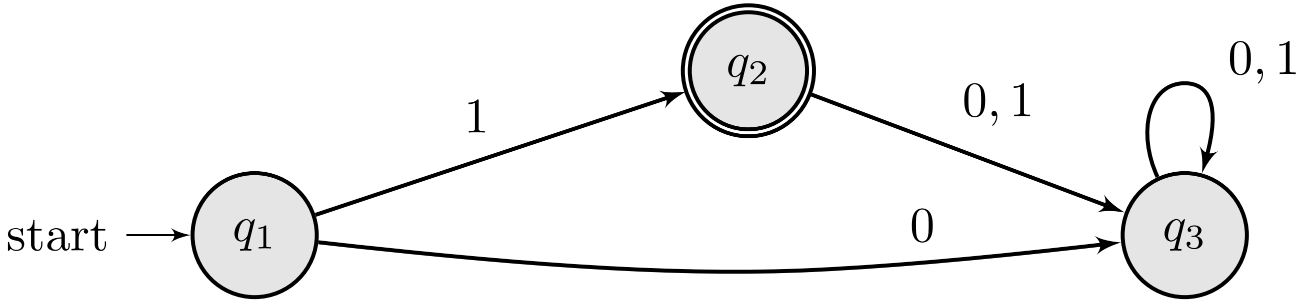
> Given some character `c`, build an automaton that only accepts string `[c]`. This is akin to the numeral 1.

# The character automaton

> Given some character `c`, build an automaton that only accepts string `[c]`. This is akin to the numeral 1.



# The character automaton




## Implementation

```
def make_char(alphabet, char):
  return DFA(
    states=["q_1", "q_2", "q_3"],
    alphabet=alphabet,
    transition_func=lambda q, a: "q_2" if q == "q_1" and a == char else "q_3",
    start_state="q_1",
    accepted_states = lambda x: x == "q_2")
```

# The character automaton

We define a function $\mathrm{char}$ that takes an alphabet $\Sigma$, a function $eq$ that tests if two elements of $\Sigma$ are equal, and a character $c \in \Sigma$ as input and outputs an automation that only accepts the string `[c]` and whose alphabet is $\Sigma$.

$$\mathrm{char}_{\Sigma}(c) = (\{q_1, q_2, q_3\}, \Sigma, \delta, q_1, \{q_2\})$$

where

2. $c \in \Sigma$
3. $\delta(q_1, c) = q_2$ (Note: This says that the arguments must be exactly $q_1$ and $c$.)
   $\delta(q, i) = q_3$ (otherwise)

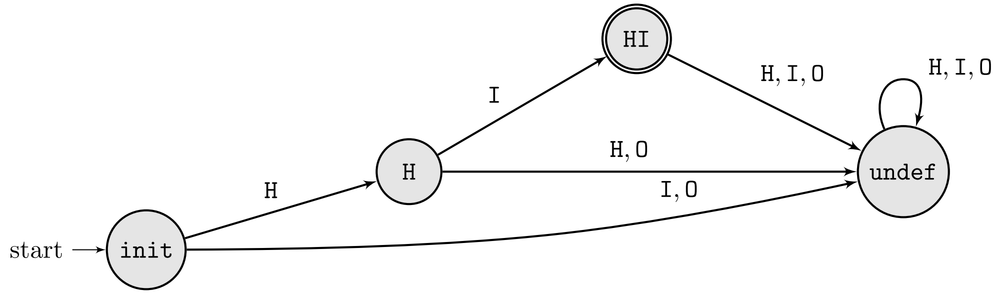
# The union automaton

$$L(\mathbf{union}(M_1, M_2)) = L(M_1) \cup L(M_2)$$

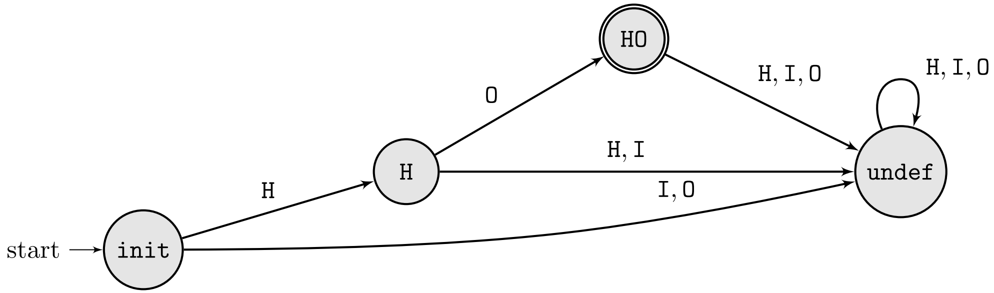
# The union automaton

> Formally, the union of two automatons is defined as the union of the recognized languages.

**Definition.** Say $M_1$ recognizes $A_1$ and $M_2$ recognizes $A_2$, then $M_1 \cup M_2$ accepts $A_1 \cup A_2$ .

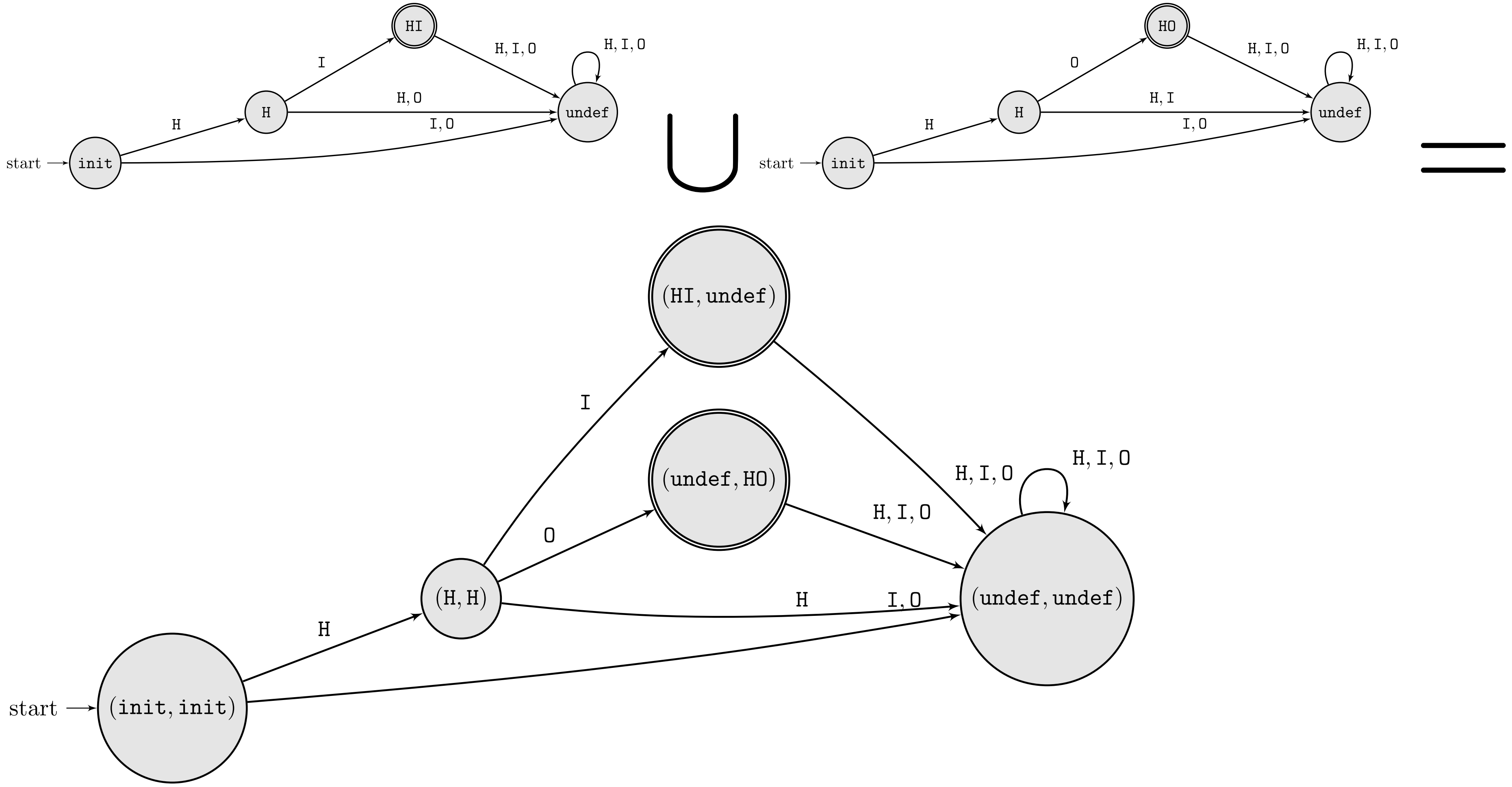



$$\bigcup$$




$$=?$$

# Visually


start
init
H
HI
undef
I
H, I, O
H, O
I, O
∪
HO
O
H, I
=
(init, init)
(H, H)
(HI, undef)
(undef, HO)
(undef, undef)

# Implementing the union of DFAs

```
def union(dfa1, dfa2):
  def transition(q, a):
    return (dfa1.transition_func(q[0], a), dfa2.transition_func(q[1], a))

  def is_final(q):
    return dfa1.accepted_states(q[0]) or dfa2.accepted_states(q[1])

  return DFA(
    states = set(product(dfa1.states, dfa2.states)),
    alphabet = set(dfa1.alphabet).union(dfa2.alphabet),
    transition_func = transition,
    start_state = (dfa1.start_state, dfa2.start_state),
    accepted_states = is_final
  )
```

# Formalizing the union

The union operation is defined as $\mathbf{union}(M_1, M_2) = (Q_{1,2}, \Gamma_1, \delta_{1,2}, q_{1,2}, F_{1,2})$ where

- $M_1 = (Q_1, \Gamma_1, \delta_1, q_1, F_1)$
- $M_2 = (Q_2, \Gamma_2, \delta_2, q_1', F_2)$
- **States:** $Q_{1,2} = Q_1 \times Q_2$
- **Alphabet:** $\Gamma_1 = \Gamma_2$
- **Transition:** $\delta_{1,2}(q, a) = (\delta_1(q|_1, a), \delta_2(q|_2, a))$
- **Initial:** $q_{1,2} = (q_1, q_1')$
- **Final:** $F_{1,2} = \{q \mid q|_1 \in F_1 \vee q|_2 \in F_2\}$

Let notation $q|_1 = x$ be defined when $q = (x, y)$. Let notation $q|_2 = y$ be defined when $q = (x, y)$.

# The concatenation automaton

$$L(\mathbf{concat}(M_1, M_2)) = L(M_1) \cdot L(M_2)$$

Building a concatenation automation is non-trivial!

# Building a concatenation automation is non-trivial!

Idea: new formalism!

(To be continued...)

# Exercise

Draw an automaton that **recognizes** $\{w \mid w \text{ starts with 10 or ends with 01}\}$.

# Exercise

Draw an automaton that **recognizes** $\{w \mid w \text{ starts with 10 or ends with 01}\}$.

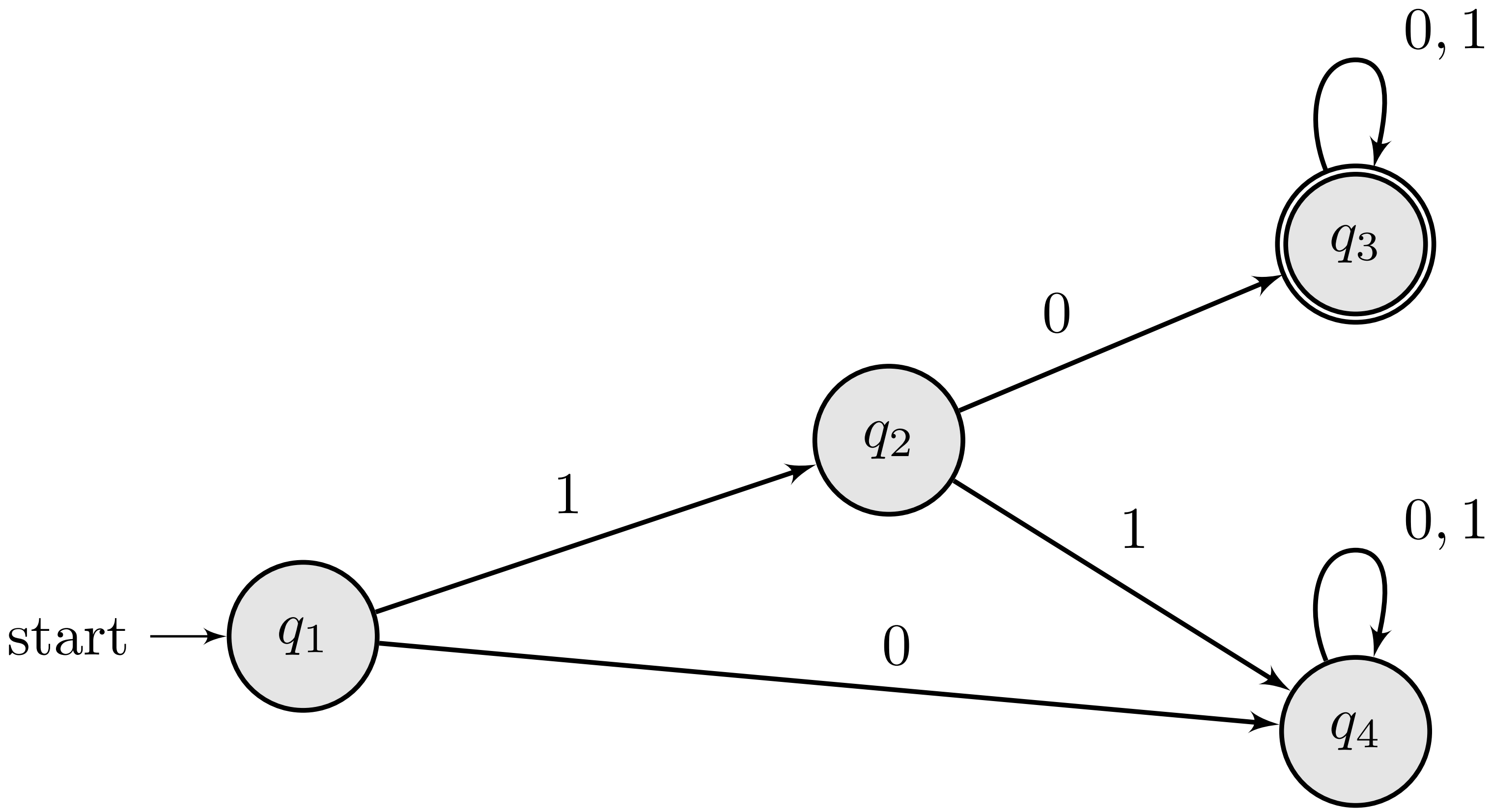
Idea: separate into two languages and then apply the union automaton

# Exercise

Draw an automaton that **recognizes** $\{w \mid w \text{ starts with } 10\}$.

# Exercise

Draw an automaton that **recognizes** $\{w \mid w \text{ starts with } 10\}$.

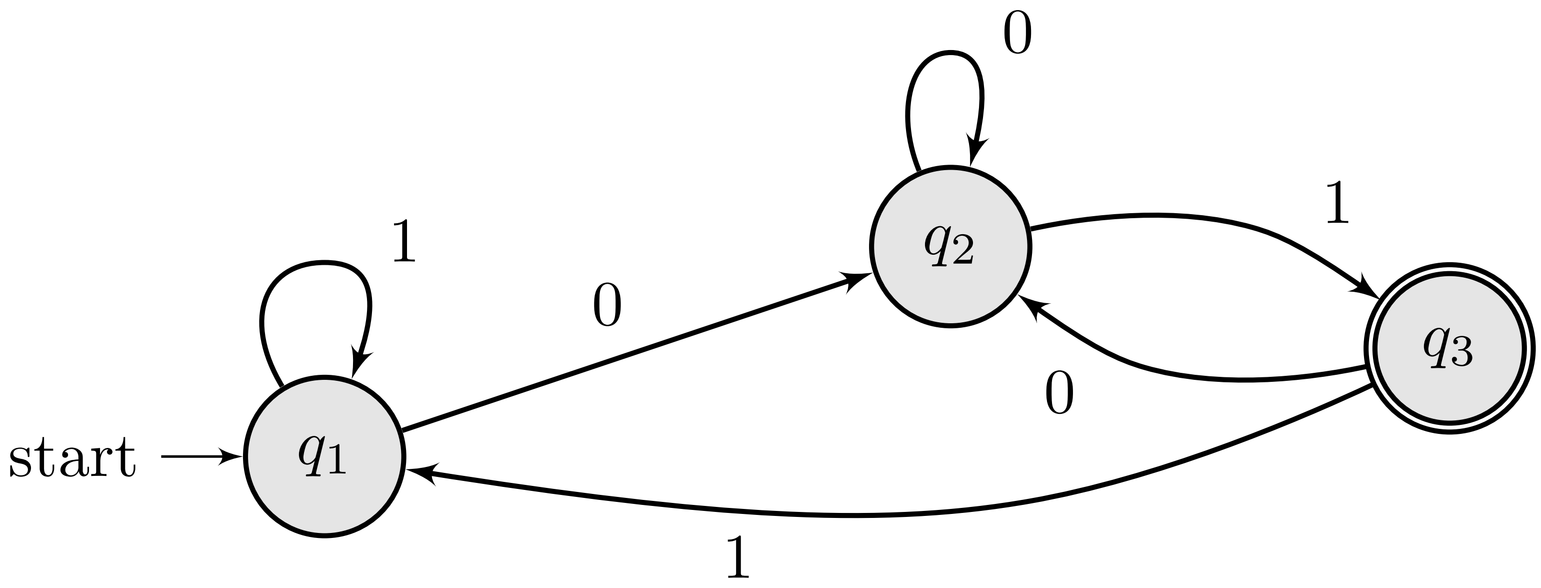


# Exercise

Draw an automaton that **recognizes** $\{w \mid w \text{ ends with } 01\}$.

# Exercise

Draw an automaton that **recognizes** $\{w \mid w \text{ ends with } 01\}$.



# Exercise

Draw an automaton that **recognizes** $\{w \mid w \text{ starts with 10 or ends with 01}\}$.