Finite Automata

204213 Theory of Computation

Jittat Fakcharoenphol

Kasetsart University

July 6, 2021

Outline

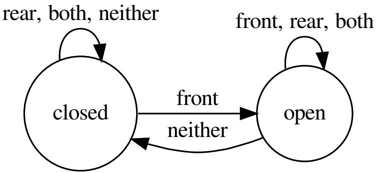
An automatic door

- Recall our automatic door example from last time?

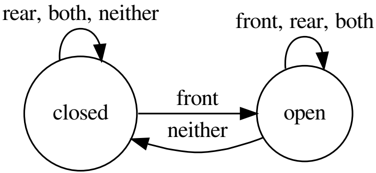
An automatic door

- Recall our automatic door example from last time?
- Let's see a simulation.

What did you see?

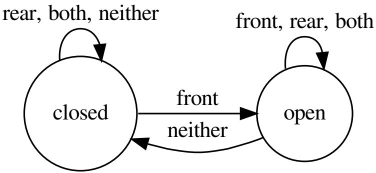


What did you see?



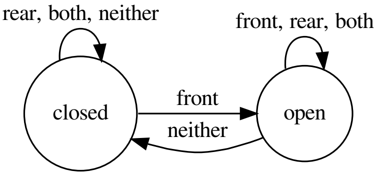
- There are two **states**:

What did you see?



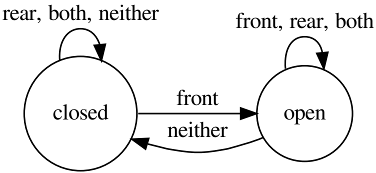
- There are two **states**: closed and open

What did you see?



- There are two **states**: closed and open
- There are 4 possible inputs, and the state of the machine changes (or remains) after each input.

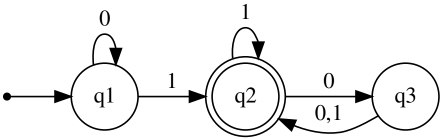
What did you see?



- There are two **states**: closed and open
- There are 4 possible inputs, and the state of the machine changes (or remains) after each input.
- See that in table form:

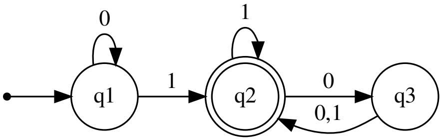
	neither	front	rear	both
closed	closed	open	closed	closed
open	closed	open	open	open

Another automaton: M_1



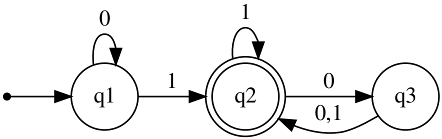
- This is the **state diagram** of M_1.

Another automaton: M_1



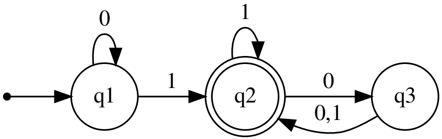
- This is the **state diagram** of M_1.
- There are 3 states: q_1, q_2, q_3.

Another automaton: M_1



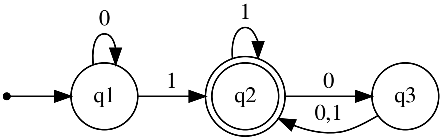
- This is the **state diagram** of M_1.
- There are 3 states: q_1, q_2, q_3.
- q_1 is the **start state**. (see the arrow?)

Another automaton: M_1



- This is the **state diagram** of M_1.
- There are 3 states: q_1, q_2, q_3.
- q_1 is the **start state**. (see the arrow?)
- q_2 is the **accept state**. (see the double circle)

Another automaton: M_1



- This is the **state diagram** of M_1.
- There are 3 states: q_1, q_2, q_3.
- q_1 is the **start state**. (see the arrow?)
- q_2 is the **accept state**. (see the double circle)
- Arrows are **transitions**.

Formal definition: why?

Formal definition: why?

Formal definition gives

- Precision

Formal definition: why?

Formal definition gives

- Precision
- Notation

Transition function: δ

- The rule for moving.

Transition function: δ

- The rule for moving.
- If you are in state q, after receiving 1 as an input, go to state p:

Transition function: δ

- The rule for moving.
- If you are in state q, after receiving 1 as an input, go to state p:

$$\delta(q, 1) = p$$

Transition function: δ

- The rule for moving.
- If you are in state q, after receiving 1 as an input, go to state p:

$$\delta(q, 1) = p$$

- δ is a function from

Transition function: δ

- The rule for moving.
- If you are in state q, after receiving 1 as an input, go to state p:

$$\delta(q, 1) = p$$

- δ is a function from the set of states and

Transition function: δ

- The rule for moving.
- If you are in state q, after receiving 1 as an input, go to state p:

$$\delta(q, 1) = p$$

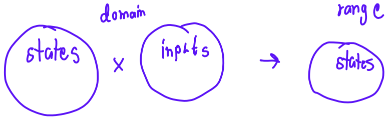
- δ is a function from the set of states and the set of possible inputs to

Transition function: δ

- The rule for moving.
- If you are in state q, after receiving 1 as an input, go to state p:

$$\delta(q, 1) = p$$

- δ is a function from the set of states and the set of possible inputs to the set of states.

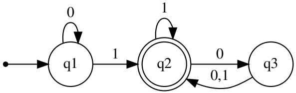


Definition [finite automaton]

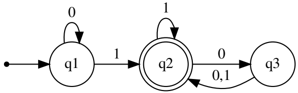
A **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

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

Formal definition of M_1



Formal definition of M_1



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

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

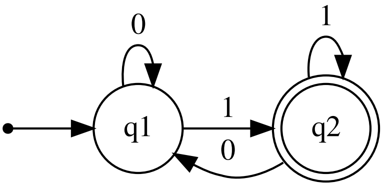
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

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

Finite automaton M_2

$M_2 = (\{q_1, q_2\}, \{0,1\}, \delta, q_1, \{q_2\})$ โดย δ นิยามเป็น

δ	0	1
q_1	q_1	q_2
q_2	q_1	q_2



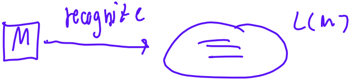
$L(M_2) = \{w \mid$ w เป็นชุด string ที่ลงท้ายด้วย 1 $\}$

Language of a machine

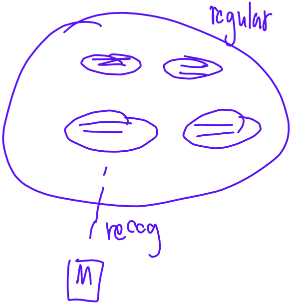
set of string

- A set A of strings is called the **language of machine** M if A is the set of all strings that M accepts.

Language of a machine



- A set A of strings is called the **language of machine** M if A is the set of all strings that M accepts.
- We write $L(M) = A$.
- We also say that M **recognizes** A.



ของ A

machine

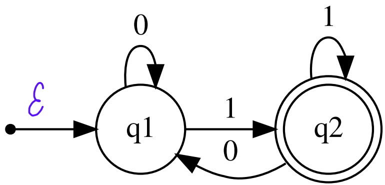
Language of machine M_1

- Let $A = \{w | w$ contains at least one 1 and an even number of 0's follow the last 1$\}$.
- $L(M_1) = A$

Language of machine M_1

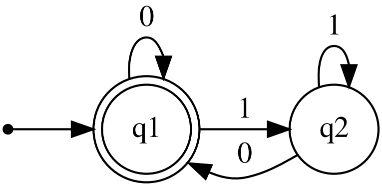
- Let $A = \{w | w$ contains at least one 1 and an even number of 0's follow the last 1$\}$.
- $L(M_1) = A$
- Or, we can say that M_1 recognizes A.

Finite automaton M_2



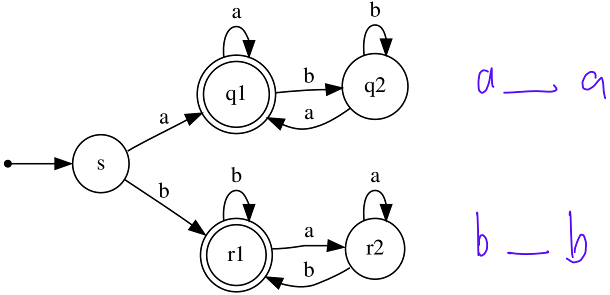
What is the language that M_2 recognizes?

Finite automaton M_3



What is the language that M_3 recognizes?

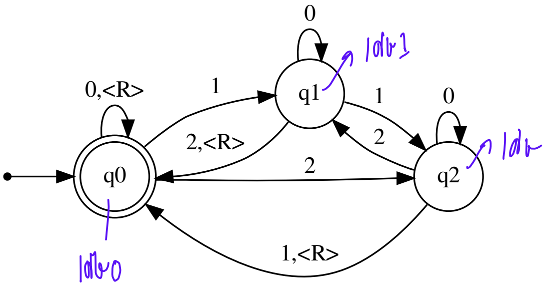
Finite automaton M_4



What is the language that M_4 recognizes?

Finite automaton M_5

$L(M_5) = \{ w \mid w$ เป็นสตริงตัวเลขที่รวมกันแล้วหาร 3 ลงตัว $\}$



What is the language that M_5 recognizes?

Formal definition of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string over alphabet Σ. M **accepts** w if

Formal definition of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string over alphabet Σ. ***M* accepts *w*** if there exists a sequence of states $r_0, r_1, \ldots, r_n$ in Q such that

Formal definition of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string over alphabet Σ. M **accepts** w if there exists a sequence of states $r_0, r_1, \ldots, r_n$ in Q such that

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$, and ← รับประกันว่าเกิด states ที่เกิดขึ้นจริง
3. $r_n \in F$.

Definition [regular language]

- **M recognizes language A** if $A = \{w | M \text{ accepts } w\}$.

Definition [regular language]

- M **recognizes language** A if $A = \{w | M \text{ accepts } w\}$.
- A language is called a **regular language** if some finite automaton recongnizes it.

set of language

{"01", "11"}

Designing finite automata

Tips:

- Pretending that you are the automaton.

Designing finite automata

Tips:

- Pretending that you are the automaton.
- You get one input at a time.

Designing finite automata

Tips:

- Pretending that you are the automaton.
- You get one input at a time.
- Think about what you have to remember to make decision correctly. (That would be a set of states.)

Practice

Language consisting of all strings with an odd number of 1's.

Building more complex finite automata

- Let $\Sigma = \{0, 1, 2\}$.
- Can you build a finite automaton M_3 that accepts all strings whose sums are divisible by 3?

Building more complex finite automata

- Let $\Sigma = \{0, 1, 2\}$.
- Can you build a finite automaton M_3 that accepts all strings whose sums are divisible by 3?
- Can you build a finite automaton M_5 that accepts all strings whose sums are divisible by 5?

Building more complex finite automata

- Let $\Sigma = \{0, 1, 2\}$.
- Can you build a finite automaton M_3 that accepts all strings whose sums are divisible by 3?
- Can you build a finite automaton M_5 that accepts all strings whose sums are divisible by 5?
- Can you build a finite automaton M_5 that accepts all strings whose sums are divisible by 3 or 5?

$L_3 \cup L_5$ eg. regular operations

Construction from smaller building boxes

This is one of important ideas in computer science.

Regular operations

- Operations for “manipulating” languages.
- A toolbox for

Regular operations

- Operations for "manipulating" languages.
- A toolbox for
  - building more complex machines.

Regular operations

- Operations for “manipulating” languages.
- A toolbox for
  - building more complex machines.
  - reasoning about larger (?) classes of machines.

Regular operations

- Operations for “manipulating” languages.
- A toolbox for
  - building more complex machines.
  - reasoning about larger (?) classes of machines.
  - some fun (?).

A set and operations

- A collection of objects is **closed under some operation** if applying that operation to objects in that set only result in object in that set.

A set and operations

- A collection of objects is **closed under some operation** if applying that operation to objects in that set only result in object in that set.
- E.g., a set of natural number $\mathcal{N}$ is closed under multiplication.

Definition [regular operations]

For a language A and B, the regular operations **union**, **concatenation**, and **star** can be defined as follows.

- Union: $A \cup B = \{x | x \in A \ \text{ or } \ x \in B\}$
- Concatenation: $A \circ B = \{xy | x \in A \ \text{ and } \ y \in B\}$ เอา string มาต่อกัน
- Star: $A^* = \{x_1 x_2 \cdots x_k | k \geq 0 \ \text{ and each } \ x_i \in A\}$

↳ ทำซ้ำ, 0 ต่อก็ได้

Can regular operations make regular languages irregular?

- Irregular languages?

Can regular operations make regular languages irregular?

- Irregular languages?
    - A language is irregular if there is no finite automaton recognizing it.

Can regular operations make regular languages irregular?

- Irregular languages?
  - A language is irregular if there is no finite automaton recognizing it.
- Look at each regular operation, starting with union.

Can regular operations make regular languages irregular?

- Irregular languages?
  - A language is irregular if there is no finite automaton recognizing it.
- Look at each regular operation, starting with union.
- If A_1 and A_2 are regular, is $A_1 \cup A_2$ regular?

Can regular operations make regular languages irregular?

- Irregular languages?
  - A language is irregular if there is no finite automaton recognizing it.
- Look at each regular operation, starting with union.
- If A_1 and A_2 are regular, is $A_1 \cup A_2$ regular?
- How can we answer that?

M_1 M_2

M_1 recognize A_1

M_2 —— A_2

Can regular operations make regular languages irregular?

- Irregular languages?
  - A language is irregular if there is no finite automaton recognizing it.
- Look at each regular operation, starting with union.
- If A_1 and A_2 are regular, is $A_1 \cup A_2$ regular?
- How can we answer that?
- Go back to the definition of regular languages.
  - **Goal:** to show that there exists a finite automaton recognizing $A_1 \cup A_2$,

Can regular operations make regular languages irregular?

- Irregular languages?
  - A language is irregular if there is no finite automaton recognizing it.
- Look at each regular operation, starting with union.
- If A_1 and A_2 are regular, is $A_1 \cup A_2$ regular?
- How can we answer that?
- Go back to the definition of regular languages.
  - **Goal:** to show that there exists a finite automaton recognizing $A_1 \cup A_2$,
  - **Given that:** there are finite automata M_1 and M_2 such that M_1 recognizes A_1 and M_2 that recognizes A_2.

Union operation

"set"

Theorem

The class of regular languages is closed under the union operation

Union operation

Theorem

The class of regular languages is closed under the union operation

Approach for proving it.

- Proof by construction.

Union operation

Theorem

The class of regular languages is closed under the union operation

Approach for proving it.

- Proof by construction.
- We know that there are finite automata M_1 that recognizes A_1 and M_2 that recognizes A_2.

Union operation

Theorem

The class of regular languages is closed under the union operation

Approach for proving it.

- Proof by construction.
- We know that there are finite automata M_1 that recognizes A_1 and M_2 that recognizes A_2.
- We shall construct M that recognizes $A_1 \cup A_2$.

How can we show that?

- Again, pretend that you are a finite automaton trying to recognizes $A_1 \cup A_2$.

How can we show that?

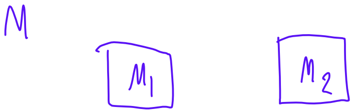
- Again, pretend that you are a finite automaton trying to recognizes $A_1 \cup A_2$.
  - Arrh.. doesn't seem to help.

How can we show that?

- Again, pretend that you are a finite automaton trying to recognizes $A_1 \cup A_2$.
  - Arrh.. doesn't seem to help.
- Let's try again:

How can we show that?

- Again, pretend that you are a finite automaton trying to recognizes $A_1 \cup A_2$.
  - Arrh.. doesn't seem to help.
- Let's try again: suppose that we want to recognize $A_1 \cup A_2$ and also have M_1 and M_2 standing in front of us.



How can we show that?

- Again, pretend that you are a finite automaton trying to recognizes $A_1 \cup A_2$.
  - Arrh.. doesn't seem to help.
- Let's try again: suppose that we want to recognize $A_1 \cup A_2$ and also have M_1 and M_2 standing in front of us.
  - Can we use them to recognize $A_1 \cup A_2$?

Given

We have to be formal:

Given

We have to be formal:

- Machine $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizing A_1
- Machine $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizing A_2

Machine M recognizing $A_1 \cup A_2$

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

- $Q = Q_1 \times Q_2$,

Machine M recognizing $A_1 \cup A_2$

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

- $Q = Q_1 \times Q_2$,
- Σ remains the same,

Machine M recognizing $A_1 \cup A_2$

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

- $Q = Q_1 \times Q_2$,
- Σ remains the same,
- δ is defined as

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

Machine M recognizing $A_1 \cup A_2$

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

- $Q = Q_1 \times Q_2$,
- Σ remains the same,
- δ is defined as

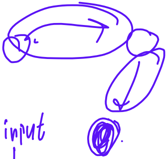
$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)),$$

- $q_0 = (q_1, q_2)$,

Machine M recognizing $A_1 \cup A_2$

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

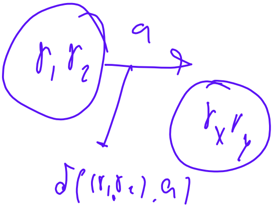
- $Q = Q_1 \times Q_2$,
- Σ remains the same,
- δ is defined as



state input

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)),$$: state ถัดไป

- $q_0 = (q_1, q_2)$,
- $F = \{(r_1, r_2) | \; r_1 \in F_1 \text{ or } r_2 \in F_2\}$



Other regular operations

concatenate

- Can we use the same technique to prove that $A_1 \circ A_2$ is regular?

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.
- This approach cannot be used directly to prove that the set of regular languages is closed under concatenation. Why?

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.
- This approach cannot be used directly to prove that the set of regular languages is closed under concatenation. Why?
  - For string $w \in A_1 \circ A_2$, there exists a pair x and y such that $w = xy$ and $x \in A_1$ and $y \in A_2$.

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.
- This approach cannot be used directly to prove that the set of regular languages is closed under concatenation. Why?
  - For string $w \in A_1 \circ A_2$, there exists a pair x and y such that $w = xy$ and $x \in A_1$ and $y \in A_2$.
  - To construct a finite automaton M for $A_1 \circ A_2$ from M_1 and M_2 that recognize A_1 and A_2 we need to simulate M_1 to the end of x and start simulating M_2 right after that.

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.
- This approach cannot be used directly to prove that the set of regular languages is closed under concatenation. Why?
    - For string $w \in A_1 \circ A_2$, there exists a pair x and y such that $w = xy$ and $x \in A_1$ and $y \in A_2$.
    - To construct a finite automaton M for $A_1 \circ A_2$ from M_1 and M_2 that recognize A_1 and A_2 we need to simulate M_1 to the end of x and start simulating M_2 right after that. **And it is hard to "tell" where x ends.**

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?
  - But guess correctly?

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?
  - But guess correctly?
  - Ummm..

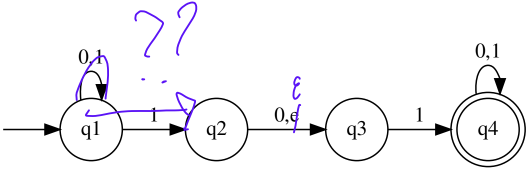
A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?
  - But guess correctly?
  - Ummm.. it definitely can,

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?
  - But guess correctly?
  - Ummm.. it definitely can, in theory.

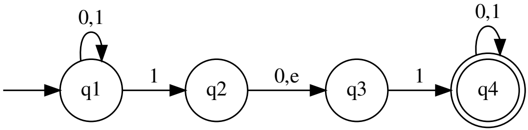
Example: Nondeterministic Finite Automaton N_1



เกดความสับสน

Note: e in the figure is ε.

Differences



- Duplicate symbols
- Missing symbols
- Empty string: ε

Deterministic and Nondeterministic Finite Automata

- Previously, we only consider finite automata whose next states are **determined** by their input alphabet and their current states.

ชัดเจน ไม่กำกวม

Deterministic and Nondeterministic Finite Automata

- Previously, we only consider finite automata whose next states are **determined** by their input alphabet and their current states.
- Computation where each next step is fully determined is called **deterministic** computation.

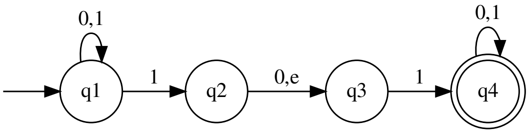
Deterministic and Nondeterministic Finite Automata

- Previously, we only consider finite automata whose next states are **determined** by their input alphabet and their current states.
- Computation where each next step is fully determined is called **deterministic** computation.
- On the other hand, in **nondeterministic** computation, many choices may exist.

Deterministic and Nondeterministic Finite Automata

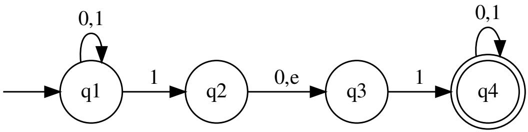
- Previously, we only consider finite automata whose next states are **determined** by their input alphabet and their current states.
- Computation where each next step is fully determined is called **deterministic** computation.
- On the other hand, in **nondeterministic** computation, many choices may exist.
- Therefore, we have deterministic finite automata (**DFA**) and nondeterministic finite automata (**NFA**).

How does N_1 compute?



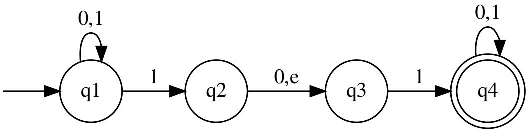
At any point where there are many choices for the next step, the machine **splits** itself into many copies and follow all possible steps in parallel.

How does N_1 compute?



At any point where there are many choices for the next step, the machine **splits** itself into many copies and follow all possible steps in parallel.
Think about *Kage Bunshin no Jutsu!*.

How does N_1 compute?



At any point where there are many choices for the next step, the machine **splits** itself into many copies and follow all possible steps in parallel.

Think about *Kage Bunshin no Jutsu!*.

See simulation.

Rules for computation of nondeterministic finite automata

- If there are many choices, split.

Rules for computation of nondeterministic finite automata

- If there are many choices, split.
- Copies die if they can't move according to the input.

Rules for computation of nondeterministic finite automata

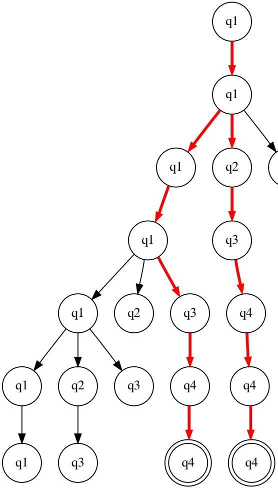
- If there are many choices, split.
- Copies die if they can't move according to the input.
- When to accept a string:

Rules for computation of nondeterministic finite automata

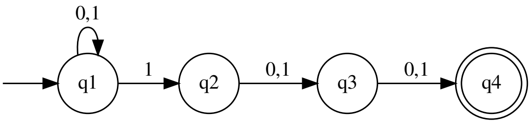
- If there are many choices, split.
- Copies die if they can't move according to the input.
- When to accept a string:
    - At the end of the input, if **any** of the copies is in an accept state, it **accept** the input.

N_1 on 010110

N_1 on 010110



NFA N_2: what are the strings accepted by N_2?



NFA N_3: what are the strings accepted by N_3?

Let $\{0\}$ be the alphabet for N_3.