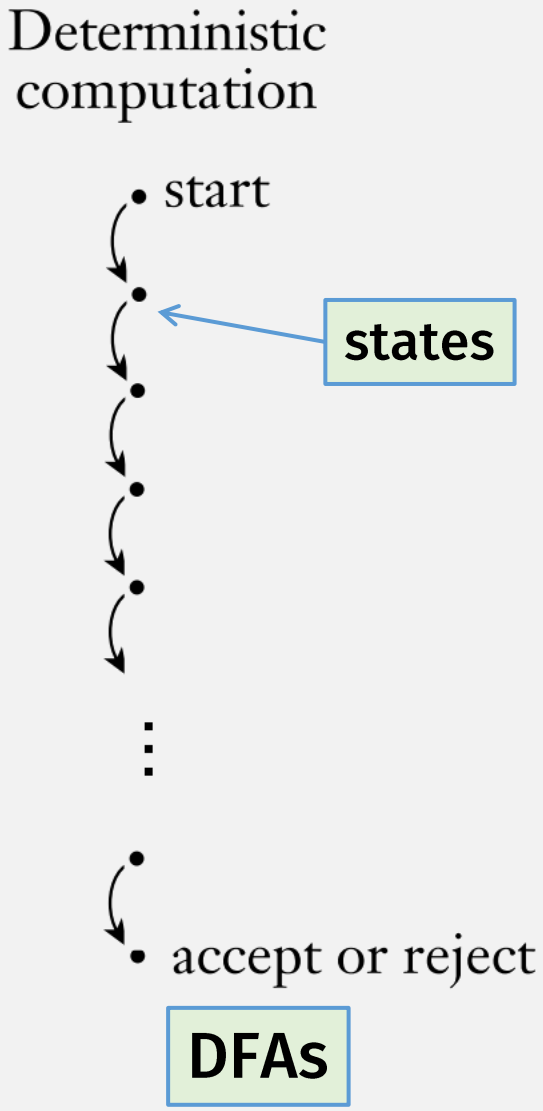
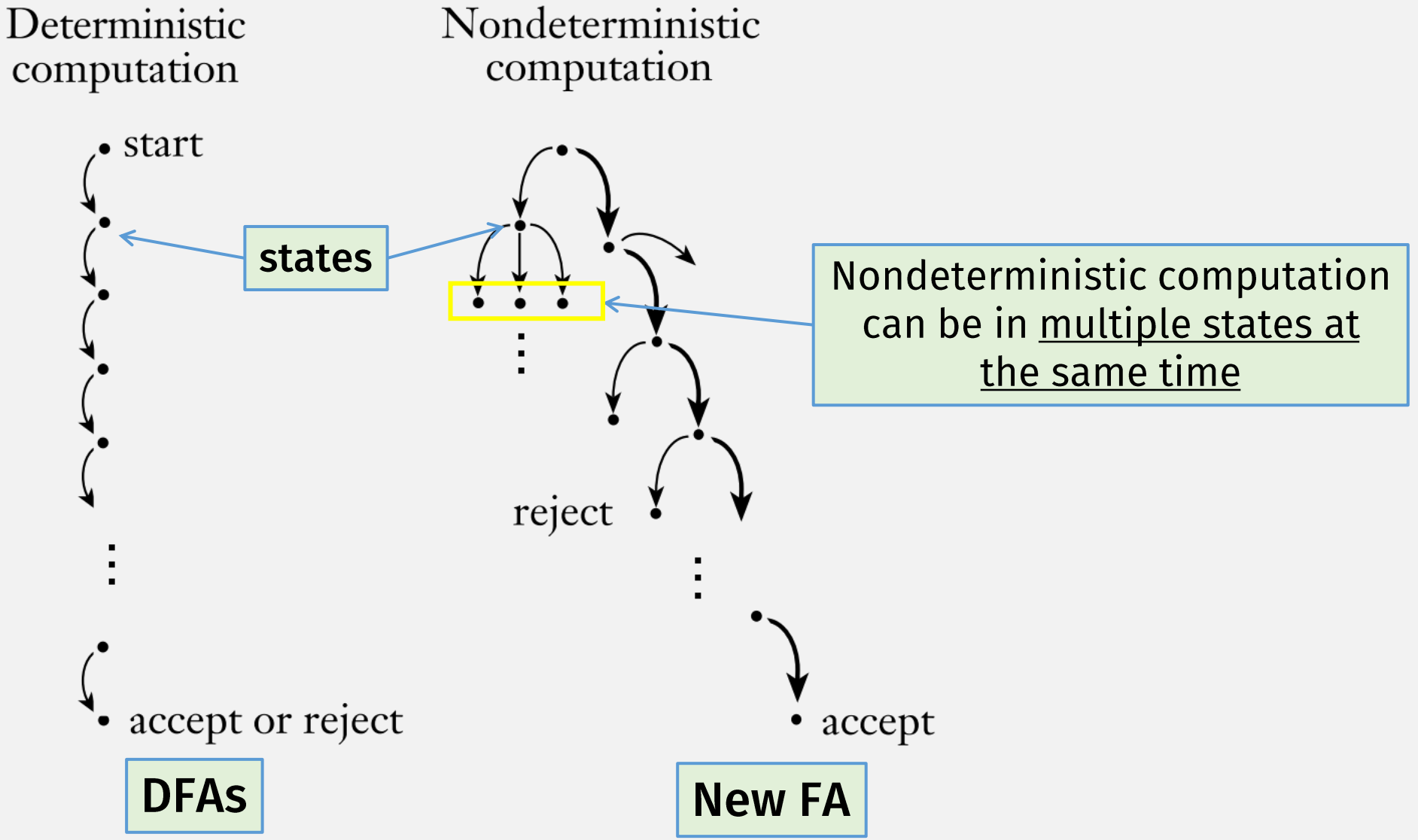
Nondeterminism

Deterministic vs Nondeterministic



Deterministic vs Nondeterministic



Previously

DFAs: The Formal Definition

DEFINITION

deterministic

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***,
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***.

Deterministic Finite Automata (DFA)

Nondeterministic Finite Automata (NFA)

DEFINITION

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Compare with DFA:

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***,
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***.

Difference

Power set, i.e. a transition results in set of states

Power Sets

- A **power set** is the set of all subsets of a set
- Example: S = {a, b, c}
- Power set of S =
  - { { }, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c} }
  - Note: includes the empty set!

Nondeterministic Finite Automata (NFA)

DEFINITION

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

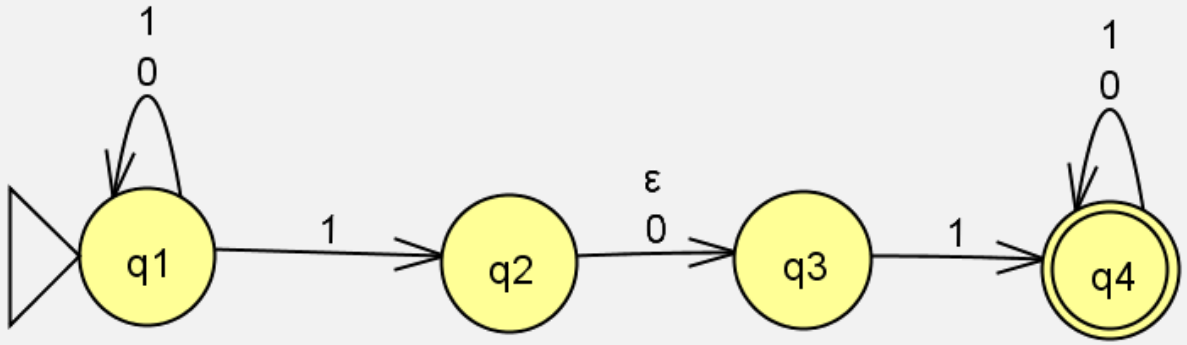
Transition label can be "empty", i.e., machine can transition without reading input

$$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$$

CAREFUL:
ε symbol is reused here, as a transition label (ie, an argument to δ)
- **It's not the empty string!**
- And it's (still) not a character in the alphabet Σ!

NFA Example

- Come up with a formal description of the following NFA:



DEFINITION

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

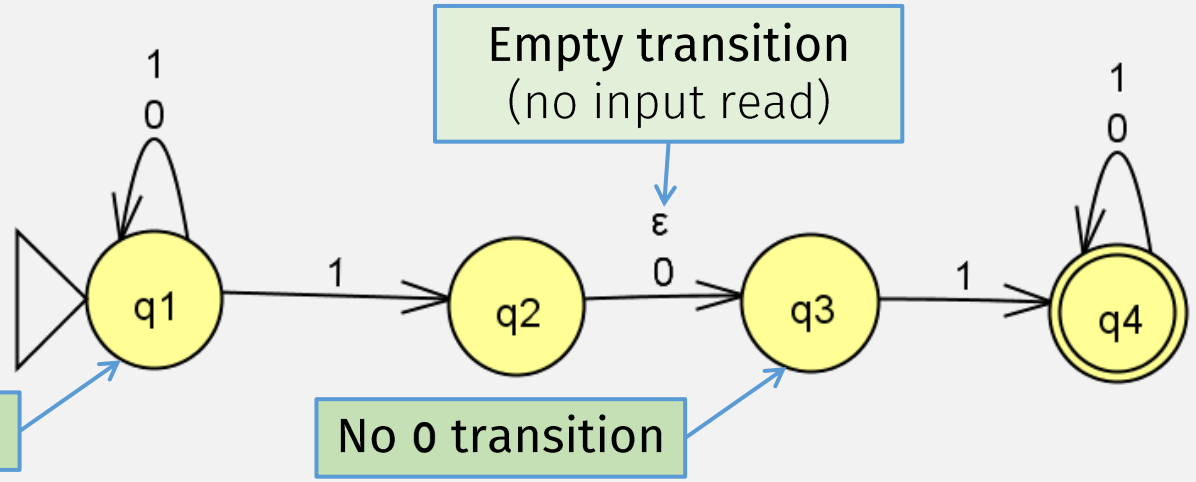
$$\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$$

Empty transition (no input read)

Result of transition is a set

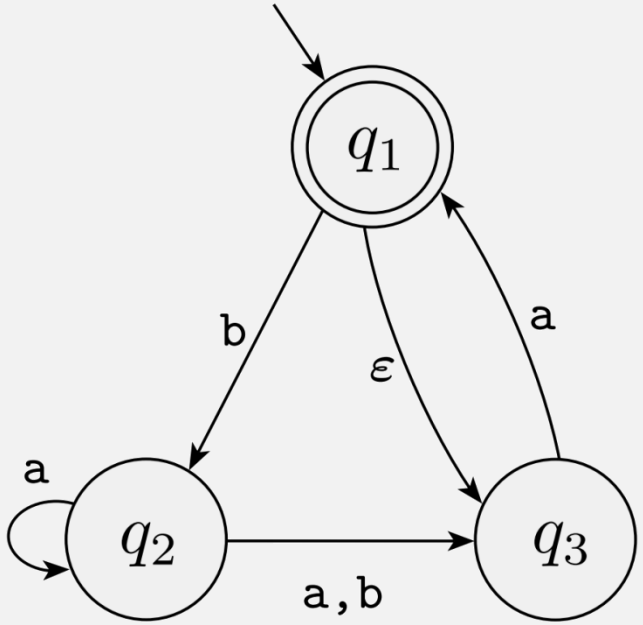
	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

4. q_1 is the start state, and
5. $F = \{q_4\}$.



In-class Exercise

- Come up with a formal description for the following NFA
  - $\Sigma = \{ \texttt{a}, \texttt{b} \}$



DEFINITION

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

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

In-class Exercise Solution

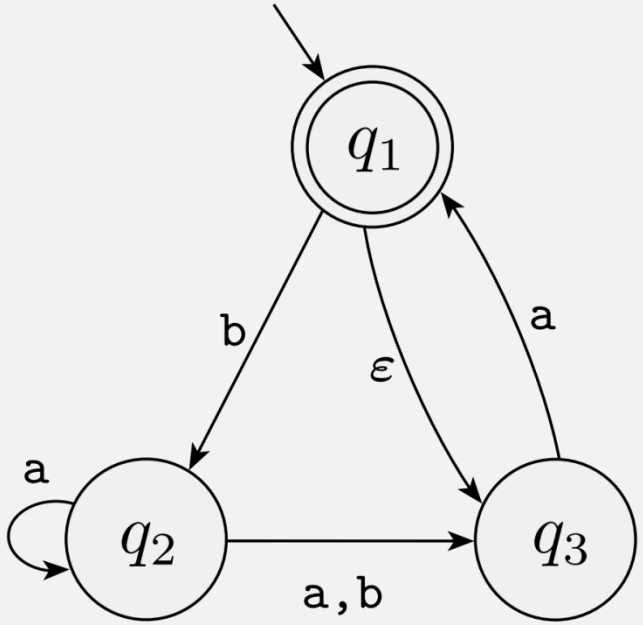
Let $N = (Q, \Sigma, \delta, q_0, F)$

- $Q = \{ q_1, q_2, q_3 \}$
- $\Sigma = \{ \mathtt{a}, \mathtt{b} \}$
- δ ...
- $q_0 = q_1$
- $F = \{ q_1 \}$

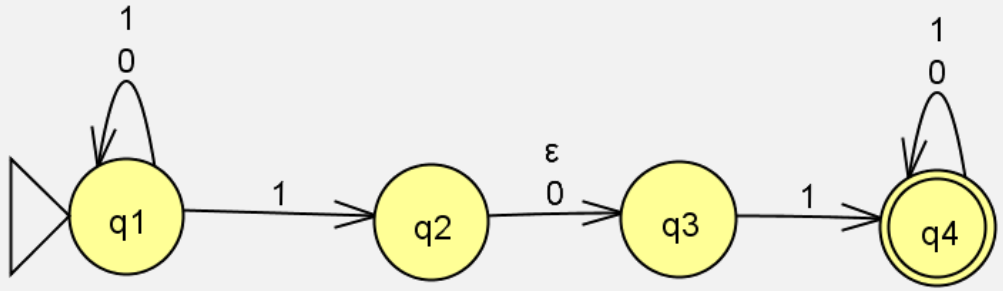
$$\delta(q_1, \mathtt{a}) = \{\}$$
$$\delta(q_1, \mathtt{b}) = \{q_2\}$$
$$\delta(q_1, \varepsilon) = \{q_3\}$$
$$\delta(q_2, \mathtt{a}) = \{q_2, q_3\}$$
$$\delta(q_2, \mathtt{b}) = \{q_3\}$$
$$\delta(q_2, \varepsilon) = \{\}$$
$$\delta(q_3, \mathtt{a}) = \{q_1\}$$
$$\delta(q_3, \mathtt{b}) = \{\}$$
$$\delta(q_3, \varepsilon) = \{\}$$

Differences with DFA?

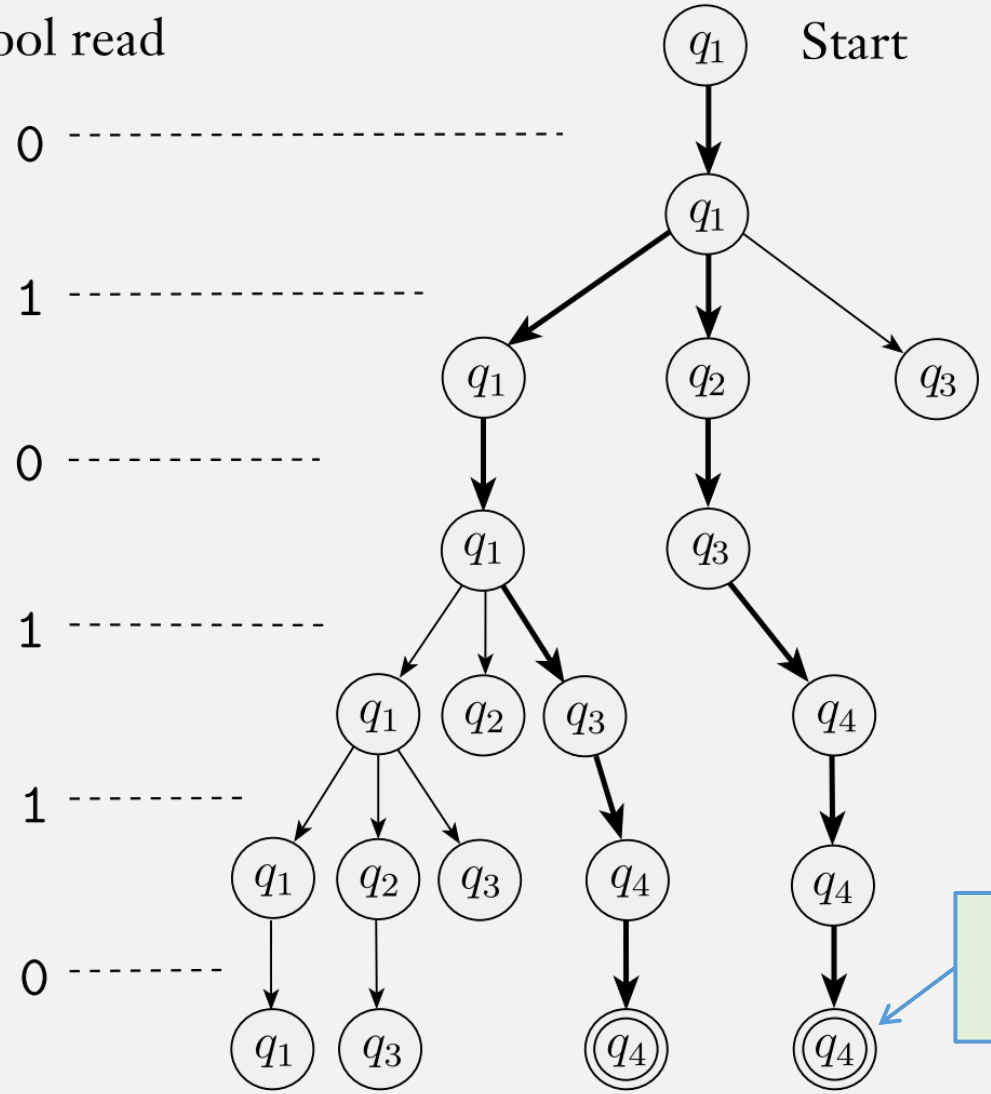
- δ output is a set
- state doesn't need transition for every alphabet symbol
- state can have multiple transitions for one symbol
- can have "empty" transitions (δ output is empty set)



NFA Computation (JFLAP demo): **010110**



NFA Computation Sequence



NFA **accepts** input if at least one path ends in accept state

Each step can branch into multiple states at the same time!

So this is an **accepting computation**

Flashback

DFA Computation Rules

Informally

Given
- A **DFA** (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

A **DFA computation** (~ "Program run"):
- **Start** in ***start state***

- **Repeat**:
  - **Read 1 char** from **Input,** and
  - **Change state** according to ***transition rules***

Result of **computation:**
- **Accept** if **last state** is ***Accept state***
- **Reject** otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

A DFA **computation** is a sequence of states:

- specified by $\hat{\delta}(q_0, w)$ where:

- M **accepts** w if $\hat{\delta}(q_0, w) \in F$
- M **rejects** otherwise

Flashback

DFA Computation Rules

Informally

Given

- A **DFA** (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

A **DFA** computation (~ "Program run"):

- Start in *start state*

- Repeat:
  - Read **1 char** from **Input,** and
  - Change state according to *transition rules*

Result of **computation:**

- Accept if **last state** is *Accept state*
- Reject otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

A **DFA computation** is a sequence of states:

- specified by $\hat{\delta}(q_0, w)$ where:

- M **accepts** w if $\hat{\delta}(q_0, w) \in F$
- M **rejects** otherwise

Ignoring ε transitions, for now!

NFA Computation Rules

Informally

Given

- An NFA (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

An NFA **computation** (~ "Program run"):

- Start in *start state*
- Repeat:
  - Read 1 char from **Input**, and according to *transition rules*

For each "current" state, go to next states

... then combine all "next states"

Result of **computation**:

- Accept if last set of states has accept state
- Reject otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

An **NFA computation** is a ...

- specified by $\hat{\delta}(q_0, w)$ where:

- M **accepts** w if ...
- M **rejects** ...

Ignoring ε transitions, for now!

NFA Computation Rules

Informally

Given
- An NFA (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

A DFA computation (~ "Program run"):
- Start in *start state*
- Repeat:
  - Read 1 char from **Input**, and according to *transition rules*

For each "current" state, go to next states

... then combine all "next states"

Result of **computation**:
- Accept if last set of states has accept state
- Reject otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

An **NFA computation** is a sequence of: sets of states

- specified by $\hat{\delta}(q_0, w)$ where:

???

- M **accepts** w if ...
- M **rejects** ...

Flashback

DFA Multi-Step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \rightarrow Q$$

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - state $q \in Q$ (doesn't have to be an accept state)

Recursive Input Data needs Recursive Function

Base case

A **String** is either:

- the **empty string** (ε), or
- xa (non-empty string) where
  - x is a **string**
  - a is a "**char**" in Σ

Base case $\hat{\delta}(q, \varepsilon) =$

Flashback

DFA Multi-Step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \to Q$$

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - state $q \in Q$ (doesn't have to be an accept state)

(Defined recursively)

Base case $\hat{\delta}(q, \varepsilon) = q$

Recursive Case $\hat{\delta}(q, w'w_n) = \delta(\hat{\delta}(q, w'), w_n)$

where $w' = w_1 \cdots w_{n-1}$

Recursive Input Data needs Recursive Function

A **String** is either:
- the **empty string** (ε), or
- xa (**non-empty string**) where
  - x is a **string**
  - a is a "**char**" in Σ

Recursive case

Recursion on string

"smaller" argument

string

char

Recursion on string

string

char

"second to last" state

Flashback

$\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***

DFA Multi-Step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \rightarrow Q$$

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - state $q \in Q$ (doesn't have to be an accept state)

(Defined recursively)

Base case $\hat{\delta}(q, \varepsilon) = q$

Recursive Case $\hat{\delta}(q, w'w_n) = \delta(\hat{\delta}(q, w'), w_n)$

where $w' = w_1 \cdots w_{n-1}$

Single step from "second to last" state and last char gets to last state

Recursive Input Data needs Recursive Function

A **String** is either:

- the **empty string** (ε), or
- xa (**non-empty string**) where
  - x is a **string**
  - a is a "**char**" in Σ

$\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function

NFA Multi-Step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - states $qs \subseteq Q$

Result is set of states

$\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function

NFA Multi-Step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

Result is set of states

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):

  states $qs \subseteq Q$

(Defined recursively)

Base case $\hat{\delta}(q, \varepsilon) = \{q\}$

Recursively Defined Input needs Recursive Function

Base case

A **String** is either:

- the **empty string** (ε), or
- xa (non-empty string) where
  - x is a **string**
  - a is a "**char**" in Σ

$\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function

NFA Multi-Step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):

  states $qs \subseteq Q$

(Defined recursively)

Base case $\hat{\delta}(q, \varepsilon) = \{q\}$

Recursive Case $\hat{\delta}(q, w'w_n) =$

where $w' = w_1 \cdots w_{n-1}$

Recursion on recursive part

"second to last" set of states

$\hat{\delta}(q, w') = \{q_1, \ldots, q_k\}$

Recursively Defined Input needs Recursive Function

A **String** is either:

- the **empty string** (ε), or
- xa (non-empty string) where
  - x is a **string**
  - a is a "**char**" in Σ

Recursive case

Recursive part

$\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function

NFA Multi-Step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - states $qs \subseteq Q$

(Defined recursively)

Base case $\hat{\delta}(q, \varepsilon) = \{q\}$

Recursive Case $\hat{\delta}(q, w'w_n) = \bigcup_{i=1}^{k} \delta(q_i, w_n)$

where $w' = w_1 \cdots w_{n-1}$

For each "second to last" state, take single step on last char

Recursively Defined Input needs Recursive Function

A **String** is either:

- the **empty string** (ε), or
- xa (**non-empty string**) where
  - x is a **string**
  - a is a "**char**" in Σ

Last char

$$\hat{\delta}(q, w') = \{q_1, \ldots, q_k\}$$

$\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function

NFA Multi-Step Transition Function

$\hat{\delta} : Q \times \Sigma^* \rightarrow$

- Domain (inpu
  - state $q \in$
  - string $w =$
- Range (outpu

  states $qs \subseteq$

(Defined recurs

Given
- An **NFA** (~ a "Program")
- and **Input = string of chars**, e.g. "1101"

A **DFA computation** (~ "Program run"):
- Start in *start state*
- Repeat:
  - Read 1 char from **Input,** and
  - according to *transition rules*

Recursively Defined Input needs

- the **empty string** (ε), or
- xa (non-empty string) where
  - x is a **string**
  - a is a "**char**" in Σ

Still ignoring ε transitions!

For each "current" state, go to next states

... then combine all sets of "next states"

Recursive Case

$$\hat{\delta}(q, w'w_n) = \bigcup_{i=1}^{k} \delta(q_i, w_n)$$

where $w' = w_1 \cdots w_{n-1}$

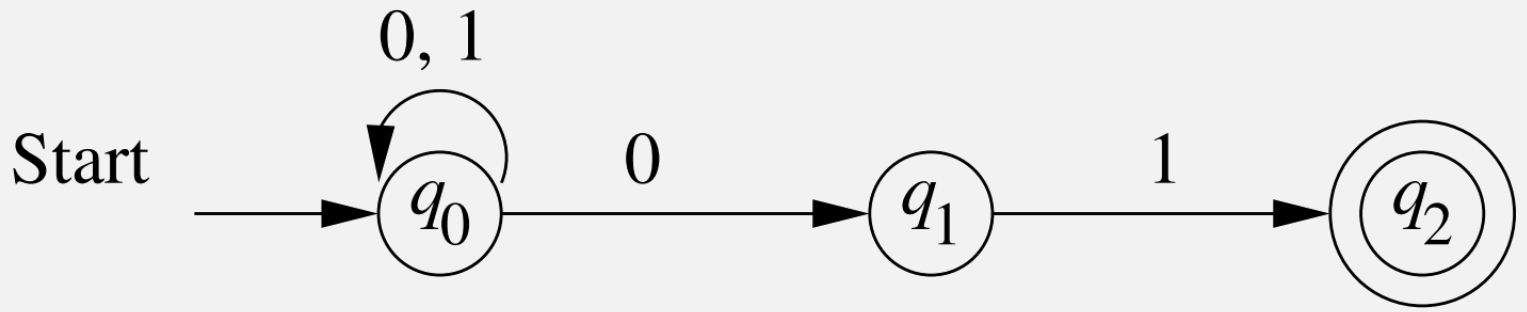
$$\hat{\delta}(q, w') = \{q_1, \ldots, q_k\}$$

NFA Multi-Step δ Example

Base case: $\hat{\delta}(q, \epsilon) = \{q\}$

Recursive case: $\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \delta(q_i, w_n)$

where: $\hat{\delta}(q, w_1 \cdots w_{n-1}) = \{q_1, \ldots, q_k\}$



- $\hat{\delta}(q_0, \epsilon) = \{q_0\}$
- $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$
- $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
- $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

We haven't considered **empty transitions!**

Combine result of recursive call with "last step"

Adding Empty Transitions

- Define **the set** $\varepsilon\text{-REACHABLE}(q)$
  - … to be all states reachable from q via zero or more empty transitions

(Defined recursively)

- Base case: $q \in \varepsilon\text{-REACHABLE}(q)$

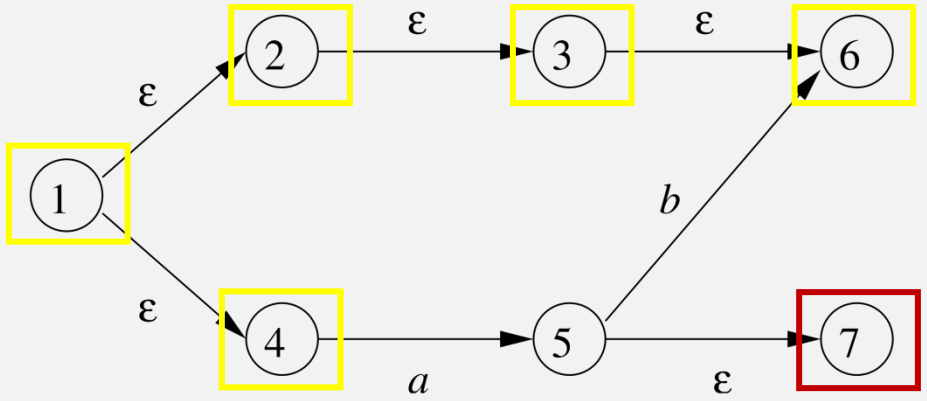
- Recursive case:

$$\varepsilon\text{-REACHABLE}(q) = \{r \mid p \in \varepsilon\text{-REACHABLE}(q) \text{ and } r \in \delta(p, \varepsilon)\}$$

A state is in the reachable set if …

… there is an empty transition to it from another state in the reachable set

ε-REACHABLE Example



$$\varepsilon\text{-REACHABLE}(1) = \{1, 2, 3, 4, 6\}$$

Handling ε transitions now!

NFA Multi-Step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$$

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - states $qs \subseteq Q$

where $w' = w_1 \cdots w_{n-1}$

$$\hat{\delta}(q, w') = \{q_1, \ldots, q_k\}$$

$$\bigcup_{i=1}^{k} \delta(q_i, w_n) = \{r_1, \ldots, r_\ell\}$$

(Defined recursively)

Base case $\hat{\delta}(q, \varepsilon) = \varepsilon\text{-REACHABLE}(q)$

Recursive Case $\hat{\delta}(q, w'w_n) =$

Handling ε transitions now!

NFA Multi-Step Transition Function

$$\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$$

- Domain (inputs):
  - state $q \in Q$ (doesn't have to be start state)
  - string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range (output):
  - states $qs \subseteq Q$

All chars except last

"second to last" set of states

$$\text{where } w' = w_1 \cdots w_{n-1}$$

$$\hat{\delta}(q, w') = \{q_1, \ldots, q_k\}$$

$$\bigcup_{i=1}^{k} \delta(q_i, w_n) = \{r_1, \ldots, r_\ell\}$$

"last" set of states (no ε)

(Defined recursively)

Base case $\hat{\delta}(q, \varepsilon) = \varepsilon\text{-REACHABLE}(q)$

Recursive Case $\hat{\delta}(q, w'w_n) = \bigcup_{j=1}^{\ell} \varepsilon\text{-REACHABLE}(r_j)$

Summary: NFA vs DFA Computation

DFAs

- Can only be in one state
- Transition:
  - Must read 1 char
- Acceptance:
  - If final state is accept state

NFAs

- Can be in multiple states
- Transition
  - Has empty transitions
- Acceptance:
  - If one of final states is accept state