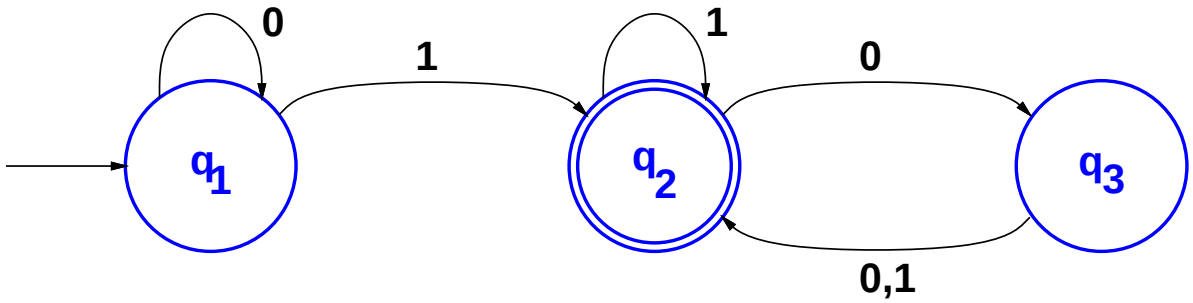
DFA Formal Definition (reminder)

A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, 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,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states.

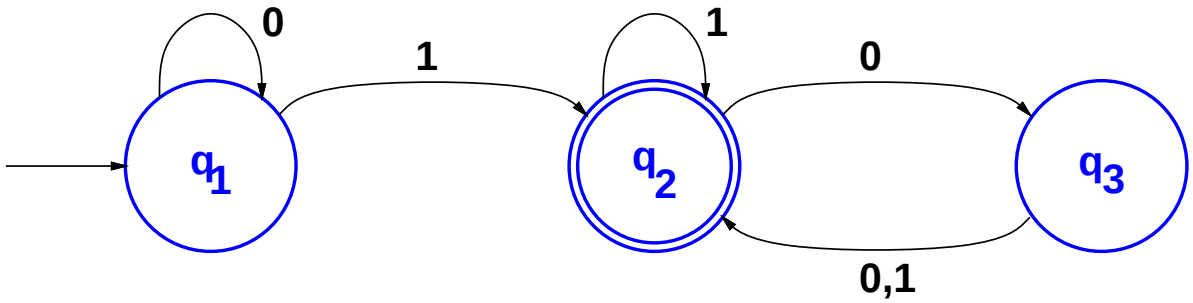
Back to M_1



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

- $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$,

Back to M_1

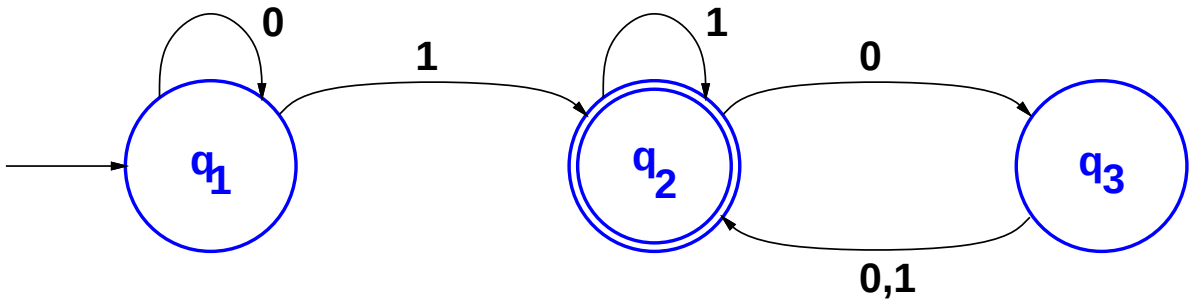


$M_1 = (Q, \Sigma, \delta, q_1, F)$ where

- $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$,
- the transition function δ is

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

Back to M_1



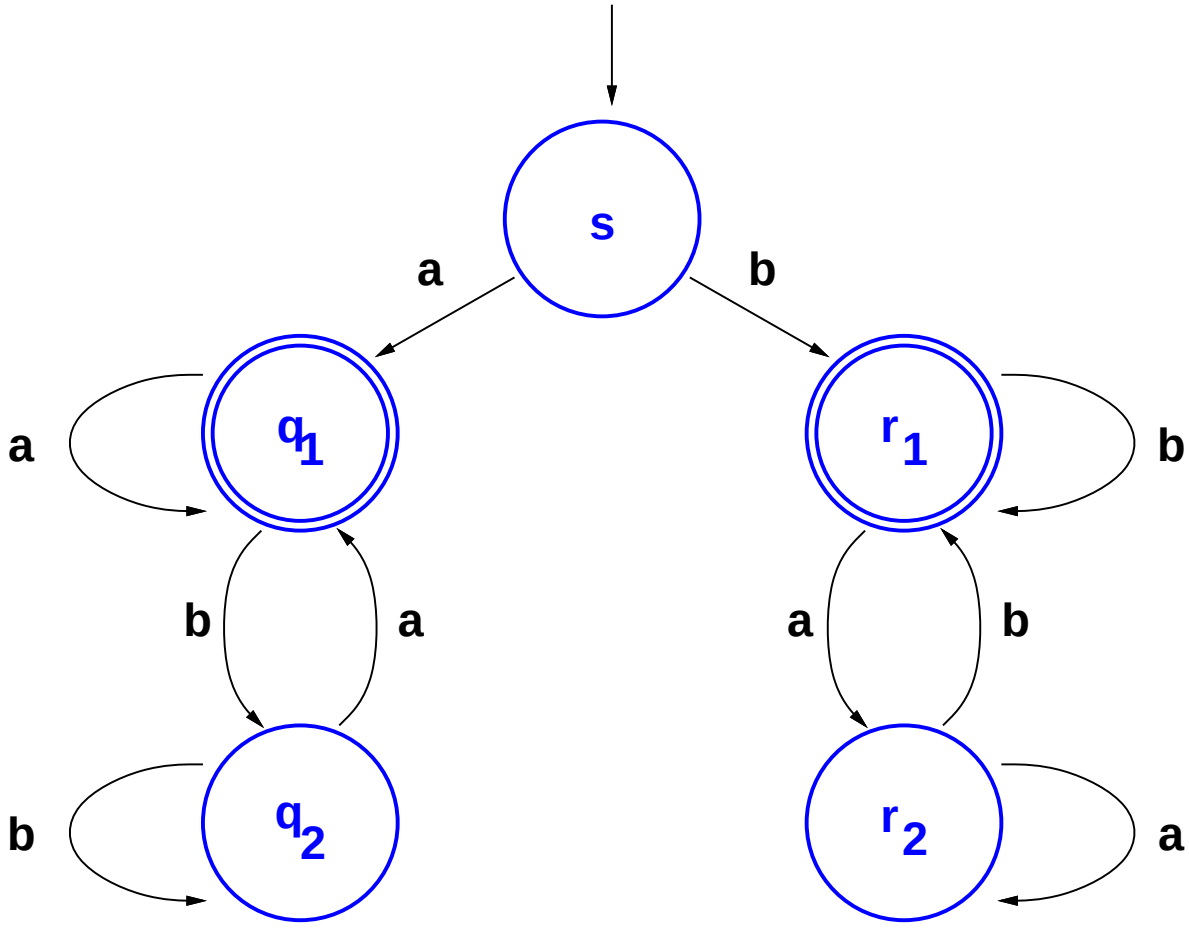
$M_1 = (Q, \Sigma, \delta, q_1, F)$ where

- $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$,
- the transition function δ is

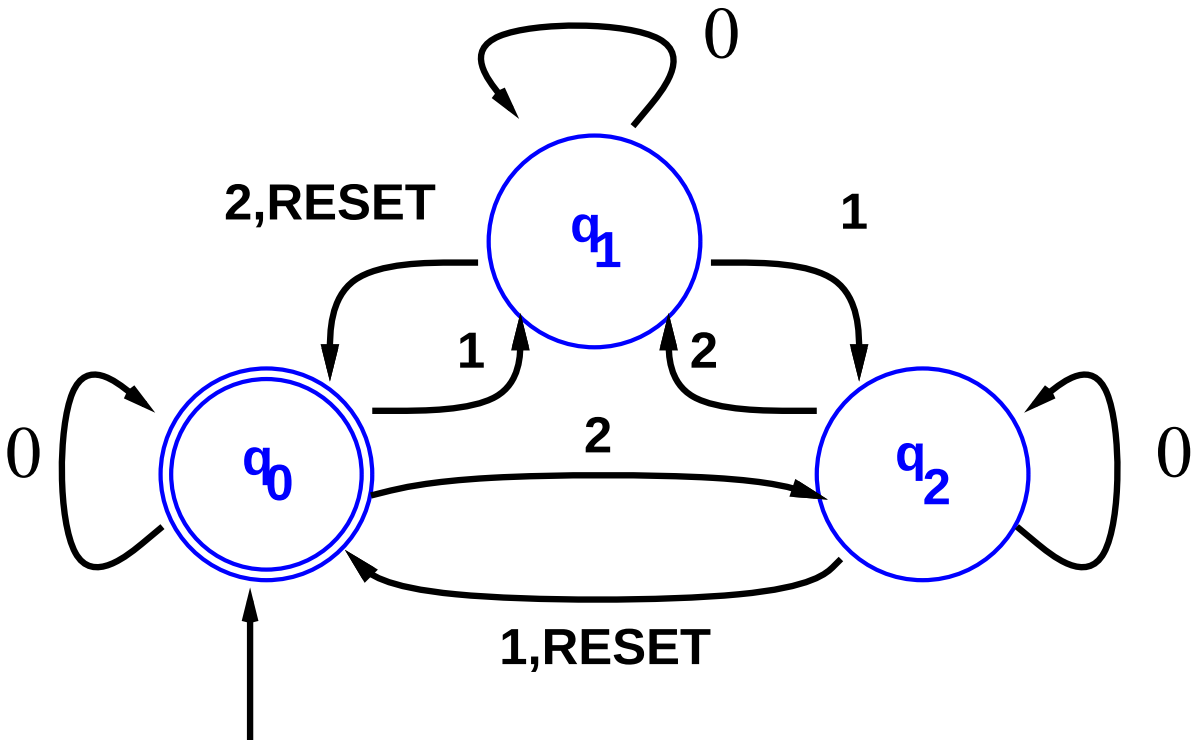
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

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

Another Example



And Yet Another Example



A Formal Model of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and
- let $w = w_1 w_2 \cdots w_n$ be a string over Σ.

We say that M accepts w if there is a sequence of states $r_0, \ldots, r_n$ ($r_i \in Q$) such that

- $r_0 = q_0$

A Formal Model of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and
- let $w = w_1 w_2 \cdots w_n$ be a string over Σ.

We say that M accepts w if there is a sequence of states $r_0, \ldots, r_n$ ($r_i \in Q$) such that

- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}$, $0 \leq i < n$

A Formal Model of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and
- let $w = w_1 w_2 \cdots w_n$ be a string over Σ.

We say that M accepts w if there is a sequence of states $r_0, \ldots, r_n$ ($r_i \in Q$) such that

- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}$, $0 \leq i < n$
- $r_n \in F$

The Regular Operations

Let A and B be languages.

The union operation:

$$A \cup B = \{x | x \in A \text{ or } x \in B\}$$

The concatenation operation:

$$A \circ B = \{xy | x \in A \text{ and } y \in B\}$$

The star operation:

$$A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \text{ and each } x_i \in A\}$$

The Regular Operations – Examples

Let $A = \{\text{good}, \text{bad}\}$ and $B = \{\text{boy}, \text{girl}\}$.

Union

$$A \cup B = \{\text{good}, \text{bad}, \text{boy}, \text{girl}\}$$

Concatenation

$$A \circ B = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$$

Star

$$A^* = \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badbad}, \text{badgood}, \ldots\}$$

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach to Proof:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach to Proof:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.

Attempted Proof Idea:

- first simulate M_1, and
- if M_1 doesn't accept, then simulate M_2.

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach to Proof:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.

Attempted Proof Idea:

- first simulate M_1, and
- if M_1 doesn't accept, then simulate M_2.

What's wrong with this?

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach to Proof:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.

Attempted Proof Idea:

- first simulate M_1, and
- if M_1 doesn't accept, then simulate M_2.

What's wrong with this?

Fix: Simulate both machines simultaneously.

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Define M as follows (M will accept $L_1 \cup L_2$):

- $Q = Q_1 \times Q_2$.
- Σ is the same.

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Define M as follows (M will accept $L_1 \cup L_2$):

- $Q = Q_1 \times Q_2$.
- Σ is the same.
- For each $(r_1, r_2) \in Q$ and $a \in \Sigma$,
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Define M as follows (M will accept $L_1 \cup L_2$):

- $Q = Q_1 \times Q_2$.
- Σ is the same.
- For each $(r_1, r_2) \in Q$ and $a \in \Sigma$,
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Define M as follows (M will accept $L_1 \cup L_2$):

- $Q = Q_1 \times Q_2$.
- Σ is the same.
- For each $(r_1, r_2) \in Q$ and $a \in \Sigma$,
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$
- $F = \{(r_1, r_2) | r_1 \in F_1$ or $r_2 \in F_2\}$. ♣

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Define M as follows (M will accept $L_1 \cup L_2$):

- $Q = Q_1 \times Q_2$.
- Σ is the same.
- For each $(r_1, r_2) \in Q$ and $a \in \Sigma$,
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$
- $F = \{(r_1, r_2) | r_1 \in F_1$ or $r_2 \in F_2\}$. ♣

(hey, why not choose $F = F_1 \times F_2$?)

What About Concatenation?

Thm: If L_1, L_2 are regular languages, so is $L_1 \circ L_2$.

Example: $L_1 = \{\text{good}, \text{bad}\}$ and $L_2 = \{\text{boy}, \text{girl}\}$.

$$L_1 \circ L_2 = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$$

What About Concatenation?

Thm: If L_1, L_2 are regular languages, so is $L_1 \circ L_2$.

Example: $L_1 = \{\text{good}, \text{bad}\}$ and $L_2 = \{\text{boy}, \text{girl}\}$.

$$L_1 \circ L_2 = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$$

This is much harder to prove.

Idea: Simulate M_1 for a while, then switch to M_2.

What About Concatenation?

Thm: If L_1, L_2 are regular languages, so is $L_1 \circ L_2$.

Example: $L_1 = \{\text{good}, \text{bad}\}$ and $L_2 = \{\text{boy}, \text{girl}\}$.

$$L_1 \circ L_2 = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$$

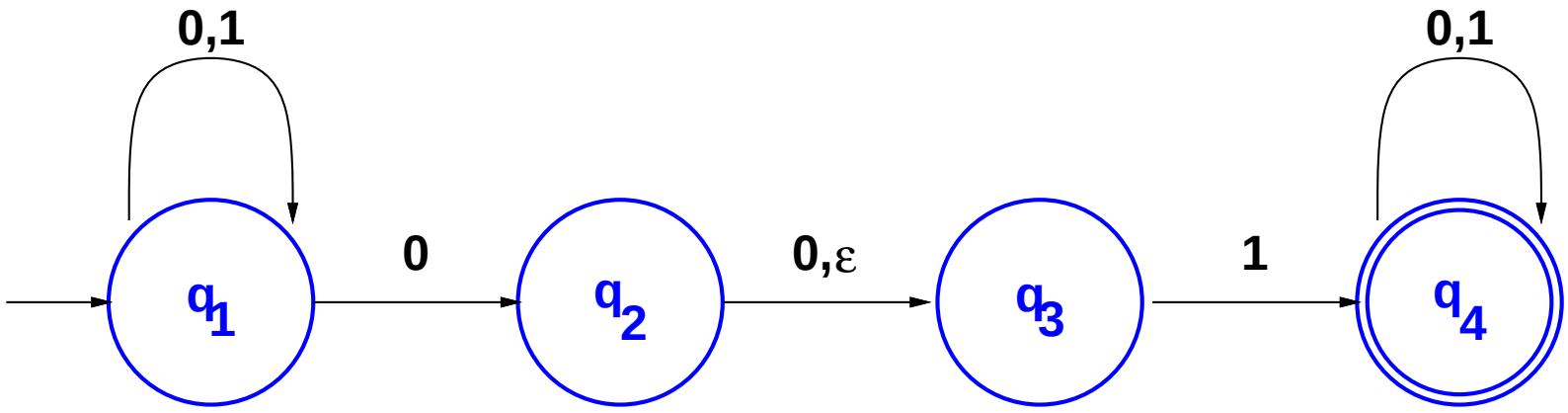
This is much harder to prove.

Idea: Simulate M_1 for a while, then switch to M_2.

Problem: But when do you switch?

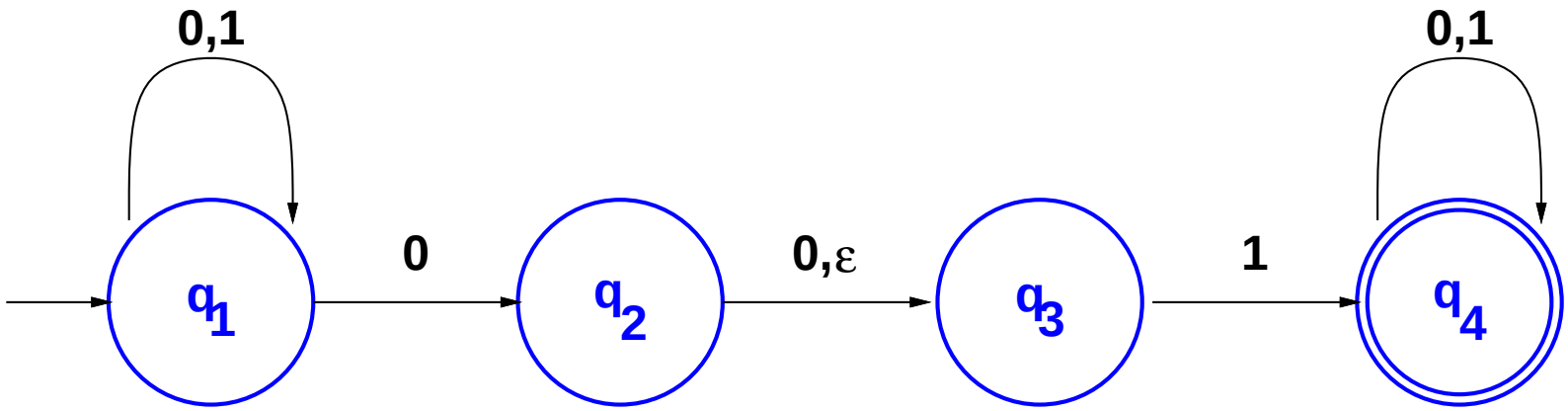
This leads us into non-determinism.

Non-Deterministic Finite Automata



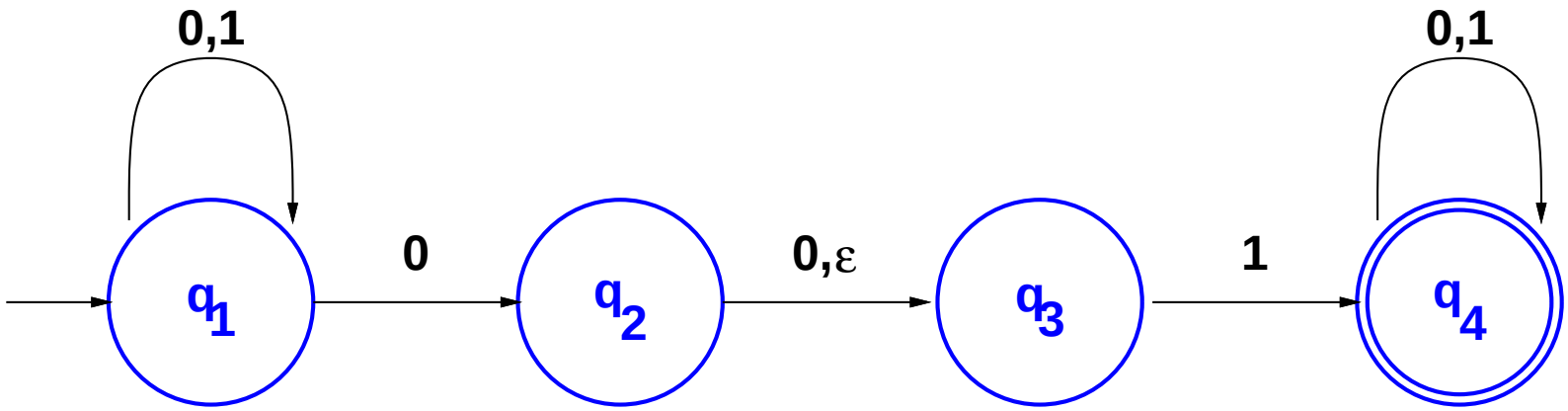
- an NFA may have more than one transition labeled with a certain symbol,

Non-Deterministic Finite Automata



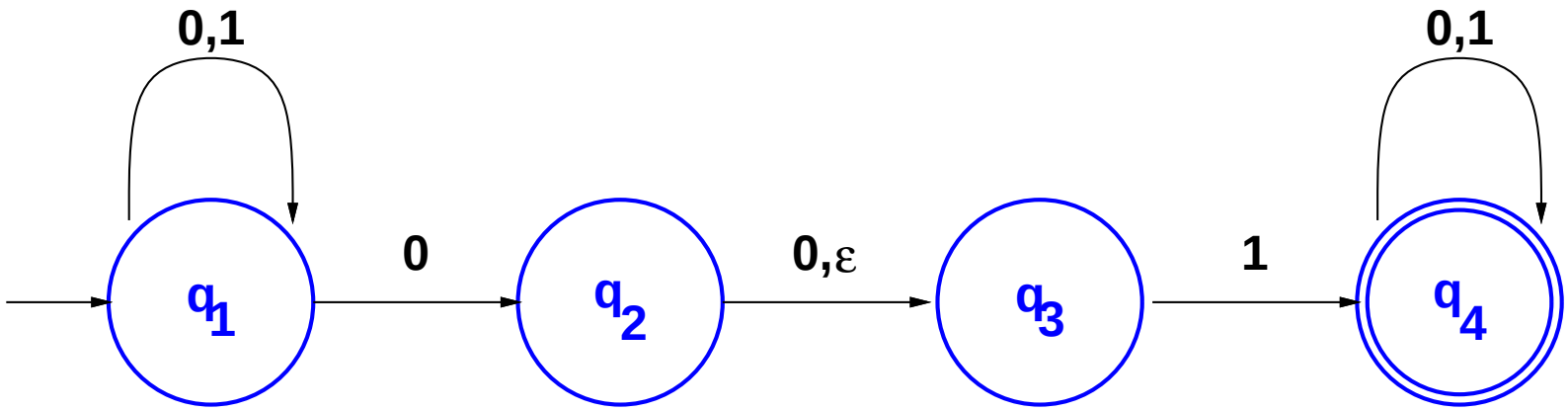
- an NFA may have more than one transition labeled with a certain symbol,
- an NFA may have no transitions labeled with a certain symbol, and

Non-Deterministic Finite Automata



- an NFA may have more than one transition labeled with a certain symbol,
- an NFA may have no transitions labeled with a certain symbol, and
- transitions may be labeled with ε, the empty string.

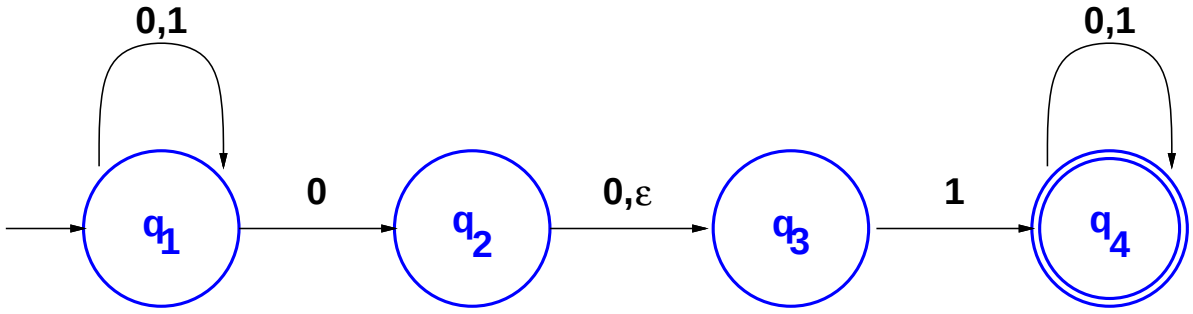
Non-Deterministic Finite Automata



- an NFA may have more than one transition labeled with a certain symbol,
- an NFA may have no transitions labeled with a certain symbol, and
- transitions may be labeled with ε, the empty string.

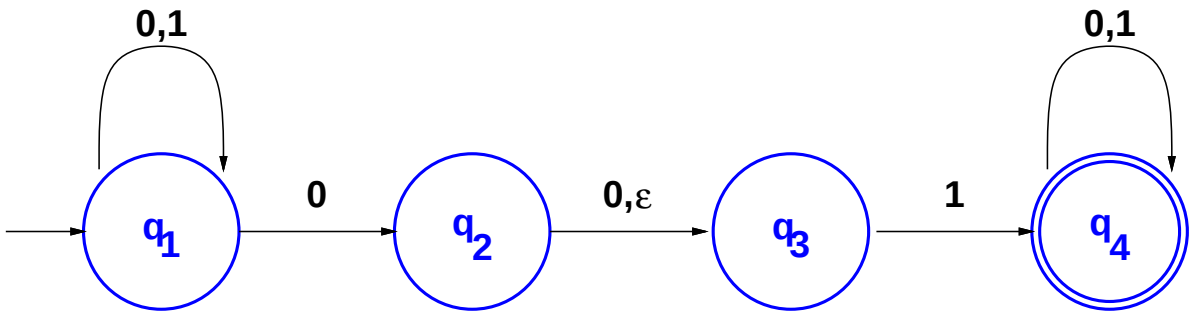
Comment: Every DFA is also a non-deterministic finite automata (NFA).

Non-Deterministic Computation



What happens when more than one transition is possible?

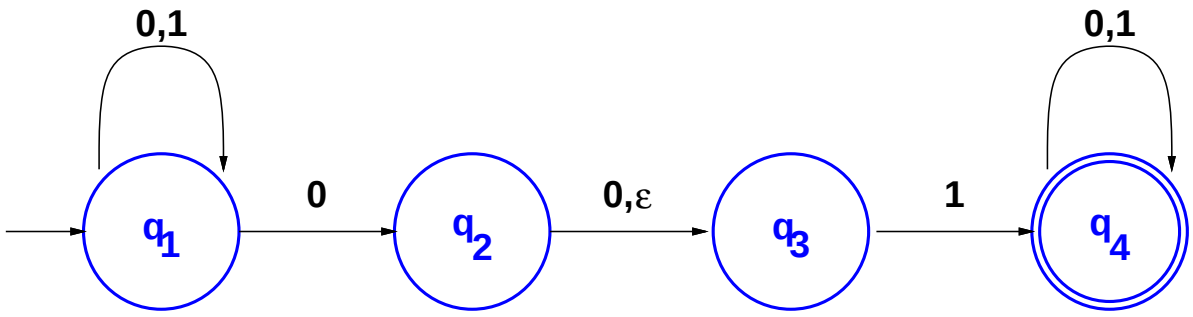
Non-Deterministic Computation



What happens when more than one transition is possible?

- the machine “splits” into multiple copies
- each branch follows one possibility
- together, branches follow all possibilities.
- If the input doesn’t appear, that branch “dies”.
- Automaton accepts if some branch accepts.

Non-Deterministic Computation

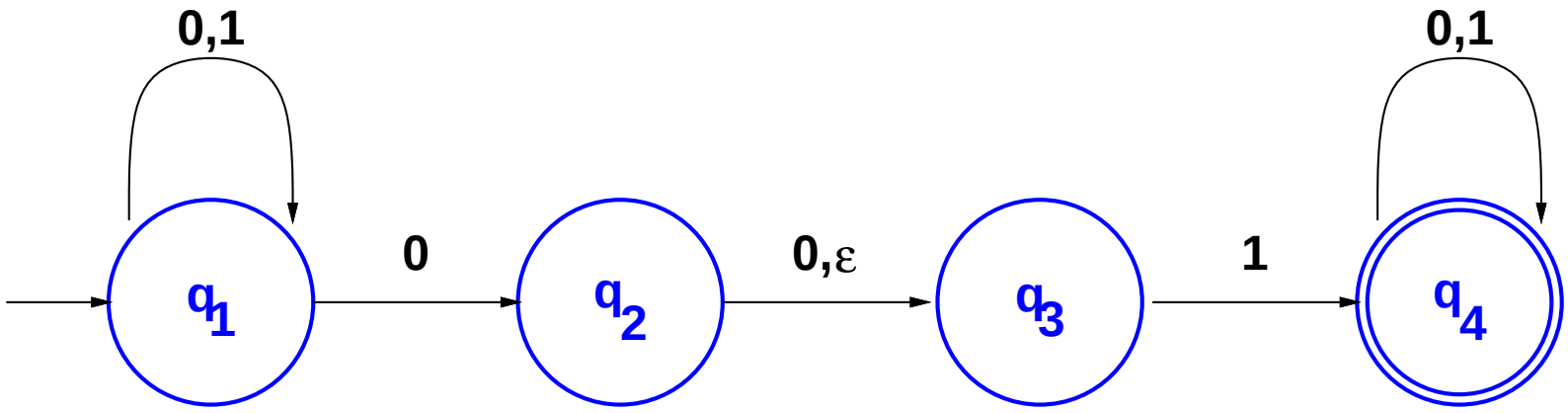


What happens when more than one transition is possible?

- the machine "splits" into multiple copies
- each branch follows one possibility
- together, branches follow all possibilities.
- If the input doesn't appear, that branch "dies".
- Automaton accepts if some branch accepts.

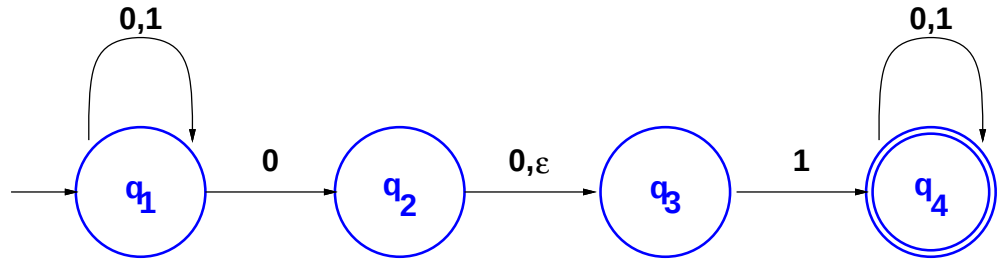
What does an ε transition do?

Non-Deterministic Computation

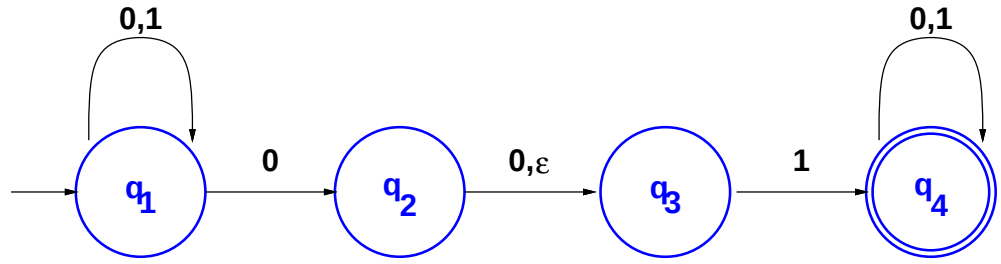


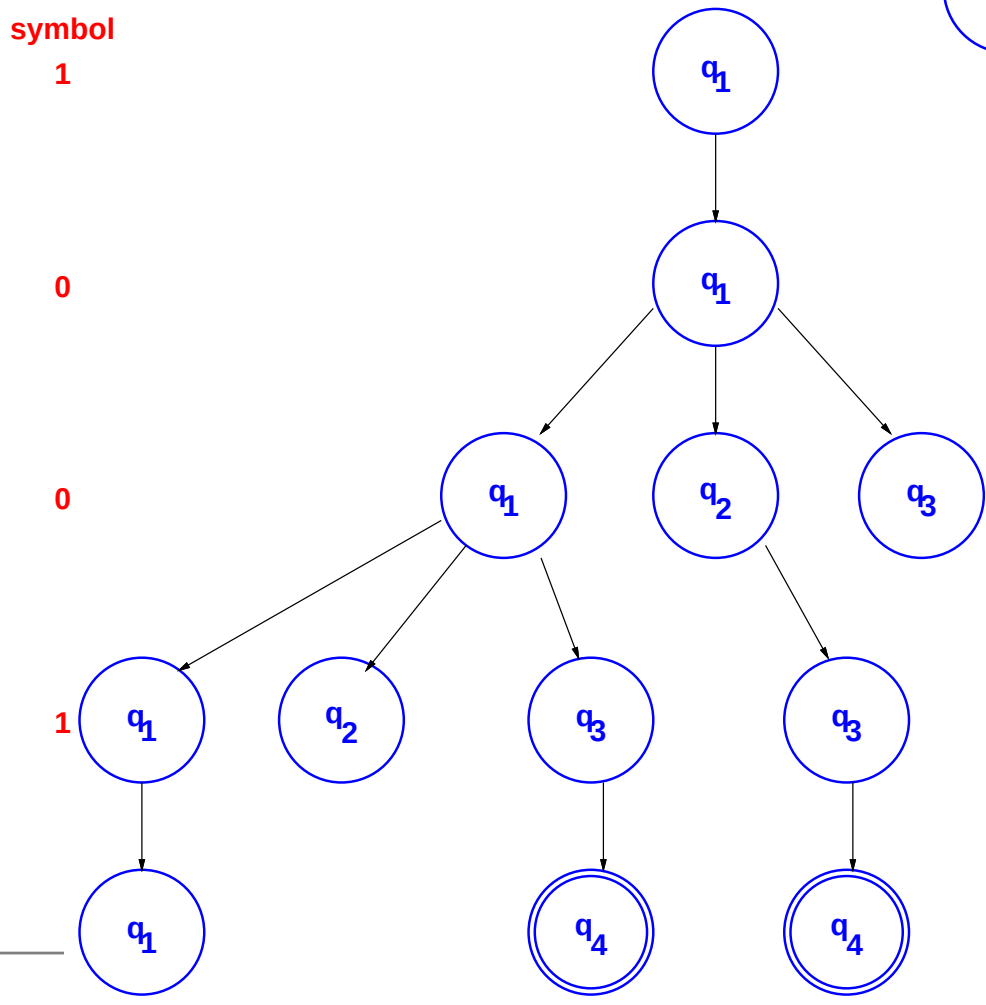
What happens on string 1001?

The String 1001



The String 1001





Why Non-Determinism?

Theorem: Deterministic and non-deterministic finite automata accept exactly the same set of languages.

Why Non-Determinism?

Theorem: Deterministic and non-deterministic finite automata accept exactly the same set of languages.

Q.: So why do we need them?

Why Non-Determinism?

Theorem: Deterministic and non-deterministic finite automata accept exactly the same set of languages.

Q.: So why do we need them?

A.: NFAs are usually easier to design than equivalent DFAs.

Why Non-Determinism?

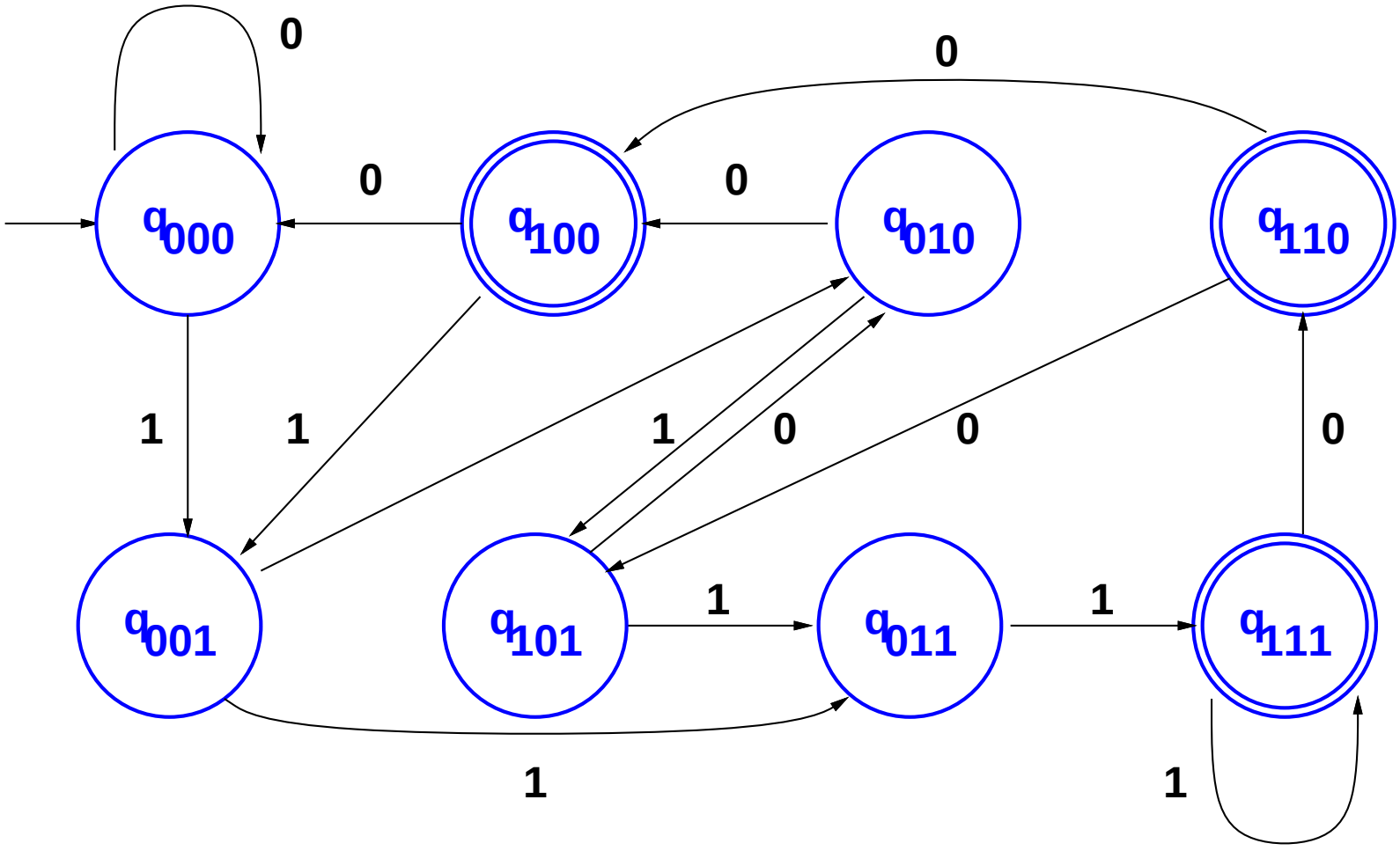
Theorem: Deterministic and non-deterministic finite automata accept exactly the same set of languages.

Q.: So why do we need them?

A.: NFAs are usually easier to design than equivalent DFAs.

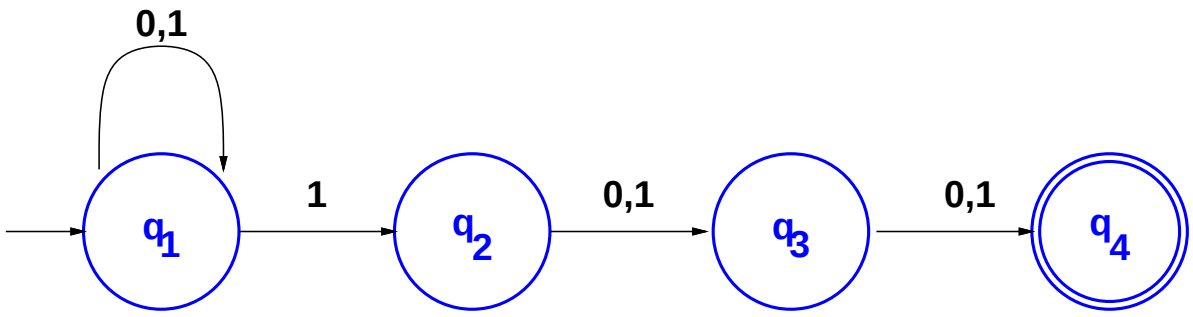
Example: Design a finite automaton that accepts all strings with a 1 in their third-to-the-last position?

A Deterministic Automaton



(there are a few errors, *e.g.* q_{101} should be an accept state, but overall it is OK.)

A Non-Deterministic Automaton



- "Guesses" which symbol is third from the last, and
- checks that it's a 1.

NFA – Formal Definition

Transition function δ is going to be different.

- $\mathcal{P}(Q)$ is the powerset of Q.
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

NFA – Formal Definition

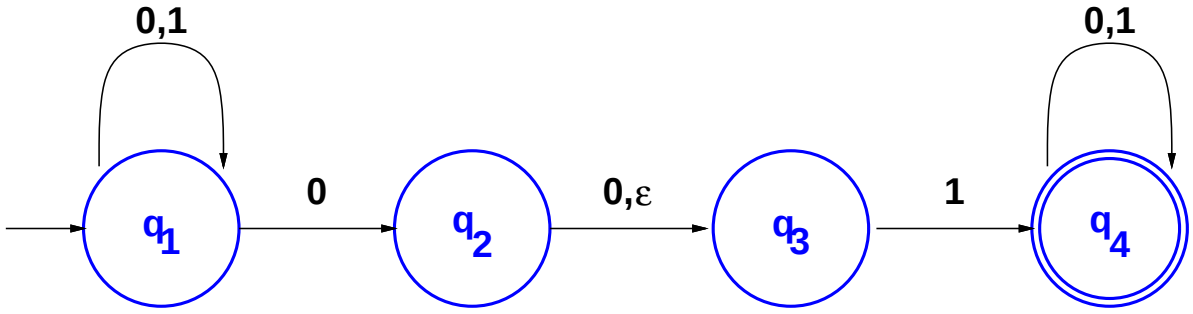
Transition function δ is going to be different.

- $\mathcal{P}(Q)$ is the powerset of Q.
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

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

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

Example

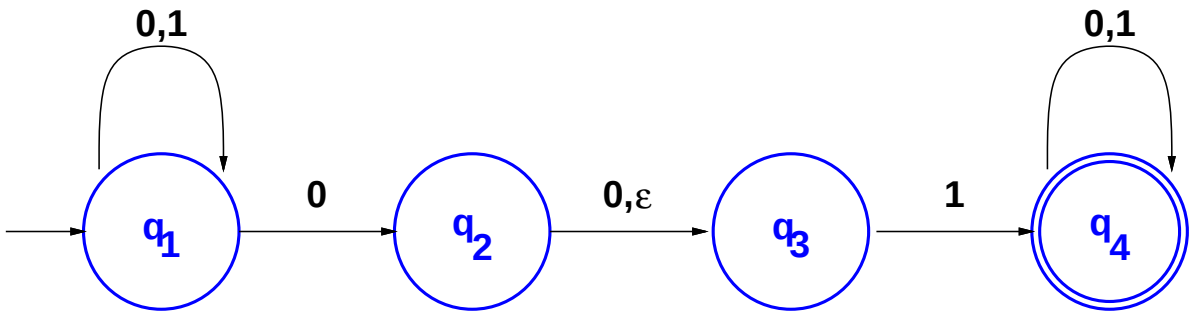


$N_1 = (Q, \Sigma, \delta, q_1, F)$

where

- $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$,

Example



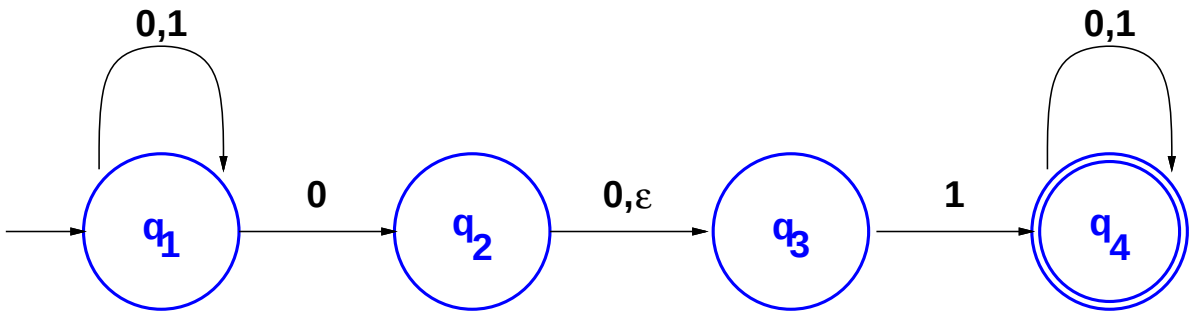
$N_1 = (Q, \Sigma, \delta, q_1, F)$

where

- $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$,
- δ is

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

Example



$N_1 = (Q, \Sigma, \delta, q_1, F)$

where

- $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$,
- δ is

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

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

Formal Model of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA, and
- w be a string over Σ_ε that has the form $y_1 y_2 \cdots y_m$ where $y_i \in \Sigma_\varepsilon$.
- u be the string over Σ obtained from w by omitting all occurances of ε.

Formal Model of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA, and
- w be a string over Σ_ε that has the form $y_1 y_2 \cdots y_m$ where $y_i \in \Sigma_\varepsilon$.
- u be the string over Σ obtained from w by omitting all occurances of ε.

Suppose there is a sequence of *states* (in Q), $r_0, \ldots, r_n$, such that

- $r_0 = q_0$
- $\delta(r_i, y_{i+1}) \in r_{i+1}$, $0 \leq i < n$
- $r_n \in F$

Formal Model of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA, and
- w be a string over Σ_ε that has the form $y_1 y_2 \cdots y_m$ where $y_i \in \Sigma_\varepsilon$.
- u be the string over Σ obtained from w by omitting all occurances of ε.

Suppose there is a sequence of *states* (in Q), $r_0, \ldots, r_n$, such that

- $r_0 = q_0$
- $\delta(r_i, y_{i+1}) \in r_{i+1}$, $0 \leq i < n$
- $r_n \in F$

Then we say that M accepts u.

Equivalence of NFA's and DFA's

- Given an an NFA, N then we construct a DFA, M, that accepts the same language.

Equivalence of NFA's and DFA's

- Given an an NFA, N then we construct a DFA, M, that accepts the same language.
- To begin with, we make things easier by ignoring ε transitions.

Equivalence of NFA's and DFA's

- Given an an NFA, N then we construct a DFA, M, that accepts the same language.
- To begin with, we make things easier by ignoring ε transitions.
- Make DFA simulate all possible NFA states.

Equivalence of NFA's and DFA's

- Given an an NFA, N then we construct a DFA, M, that accepts the same language.
- To begin with, we make things easier by ignoring ε transitions.
- Make DFA simulate all possible NFA states.
- As consequence of the construction, if the NFA has k states, the DFA has 2^k states.

Equivalence of NFA's and DFA's

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA accepting A.

Construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$.

- $Q' = \mathcal{P}(Q)$.

Equivalence of NFA's and DFA's

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA accepting A.

Construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$.

- $Q' = \mathcal{P}(Q)$.
- For $R \in Q'$ and $a \in \Sigma$, let

$$\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \text{ for some } r \in R\}$$

Equivalence of NFA's and DFA's

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA accepting A.

Construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$.

- $Q' = \mathcal{P}(Q)$.
- For $R \in Q'$ and $a \in \Sigma$, let

$$\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \text{ for some } r \in R\}$$

- $q_0' = \{q_0\}$

Equivalence of NFA's and DFA's

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA accepting A.

Construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$.

- $Q' = \mathcal{P}(Q)$.
- For $R \in Q'$ and $a \in \Sigma$, let

$$\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \text{ for some } r \in R\}$$

- $q_0' = \{q_0\}$
- $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$

Dealing with ε-Transitions

For any state R of M, define $E(R)$ to be the collection of states reachable from R by ε transitions only.

$E(R) = \{q \in Q | q$ can be reached from some $r \in R$ by 0 or more ε transitions$\}$

Dealing with ε-Transitions

For any state R of M, define $E(R)$ to be the collection of states reachable from R by ε transitions only.

$E(R) = \{q \in Q | q$ can be reached from some $r \in R$ by 0 or more ε transitions$\}$

Define transition function:

$\delta'(R, a) = \{q \in Q |$ there is some $r \in R$ such that $q \in E(\delta(r, a))\}$

Dealing with ε-Transitions

For any state R of M, define $E(R)$ to be the collection of states reachable from R by ε transitions only.

$E(R) = \{q \in Q | q$ can be reached from some $r \in R$ by 0 or more ε transitions$\}$

Define transition function:

$\delta'(R, a) = \{q \in Q |$ there is some $r \in R$ such that $q \in E(\delta(r, a))\}$

Change start state to

$$q_0' = E(\{q_0\})$$

Regular Languages, Revisited

By definition, a language is regular if it is accepted by some DFA.

Regular Languages, Revisited

By definition, a language is regular if it is accepted by some DFA.

Corollary: A language is regular if and only if it is accepted by some NFA.

Regular Languages, Revisited

By definition, a language is regular if it is accepted by some DFA.

Corollary: A language is regular if and only if it is accepted by some NFA.

This is an alternative way of characterizing regular languages.

Regular Languages, Revisited

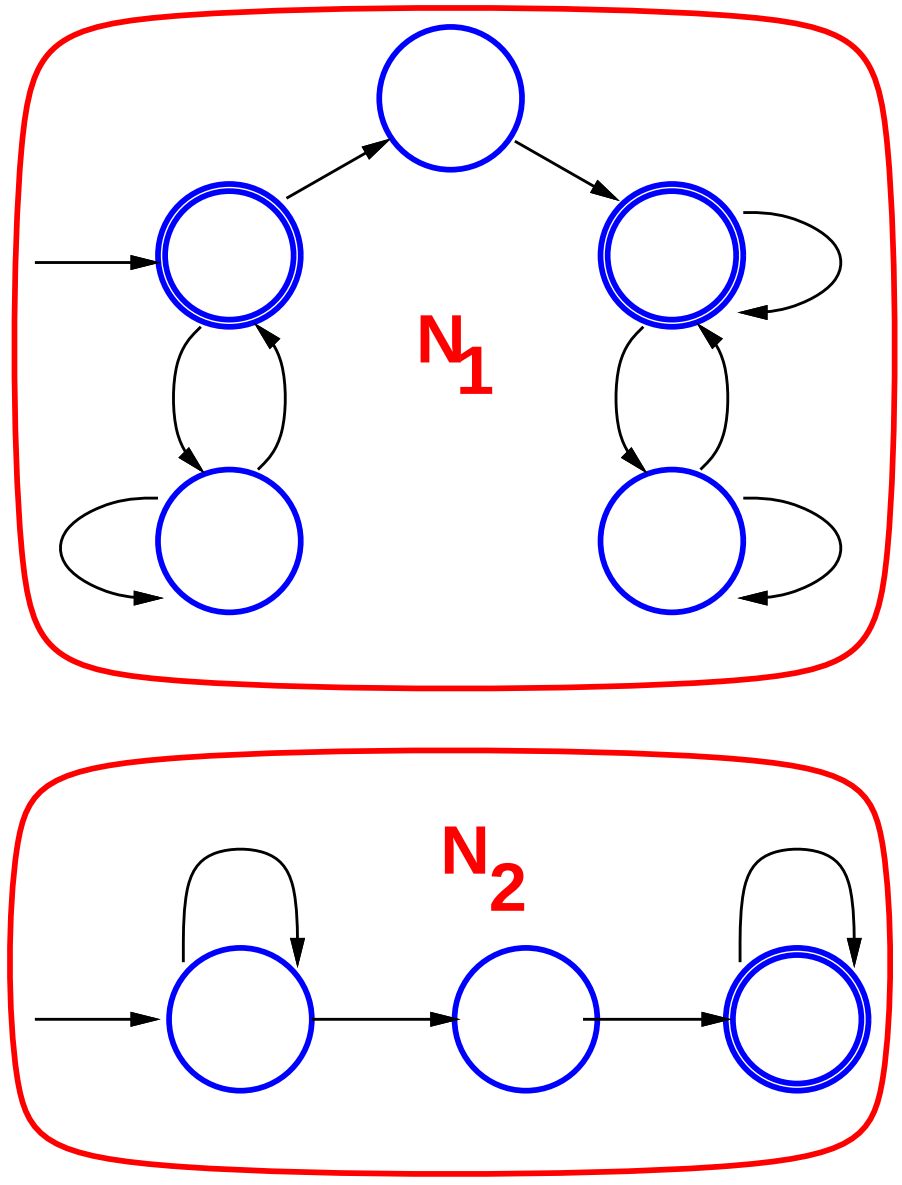
By definition, a language is regular if it is accepted by some DFA.

Corollary: A language is regular if and only if it is accepted by some NFA.

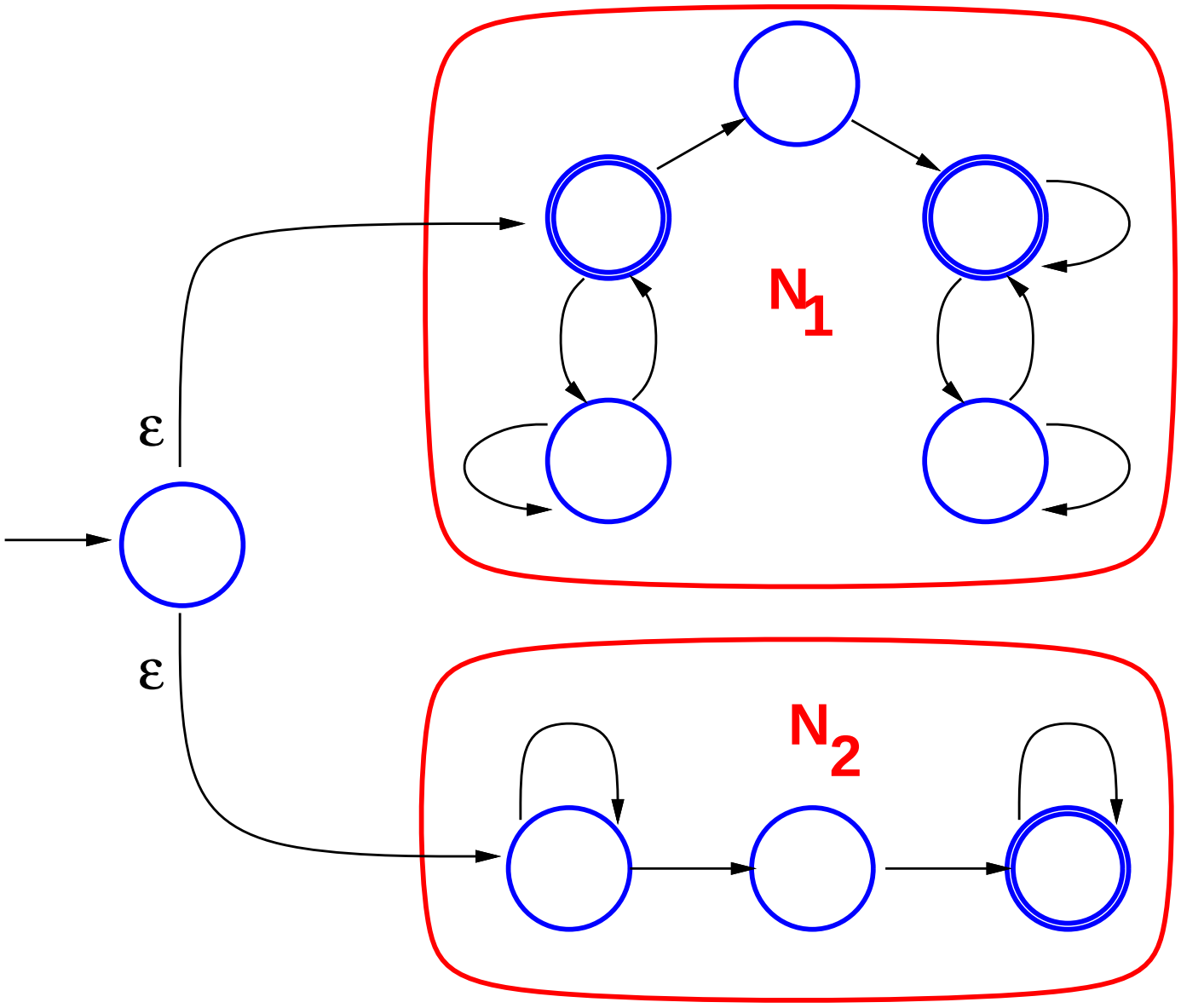
This is an alternative way of characterizing regular languages.

We will now use the equivalence to show that regular languages are closed under the regular operations (union, concatenation, star).

Regular Languages Closed Under Union



Regular Languages Closed Under Union



Regular Languages Closed Under Union

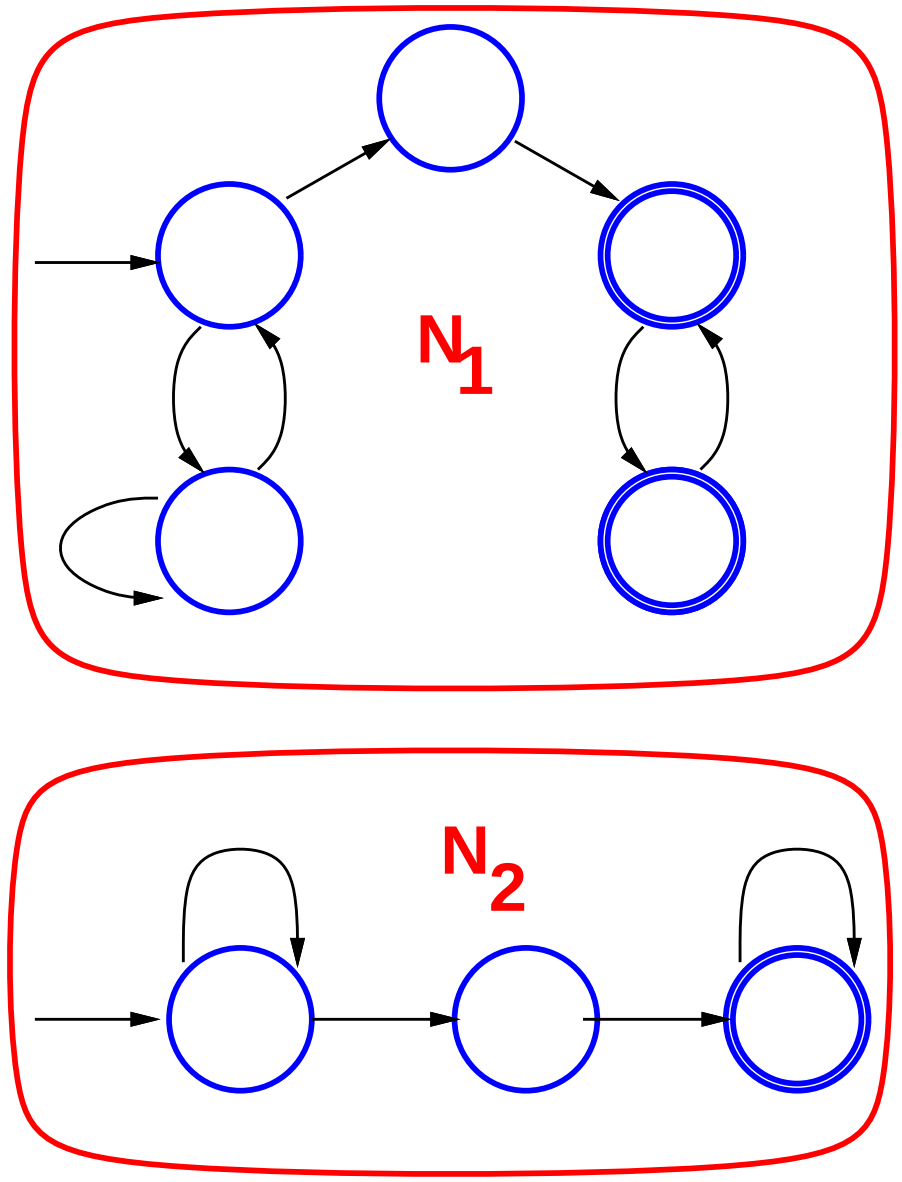
Suppose

- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept L_1, and
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accept L_2.

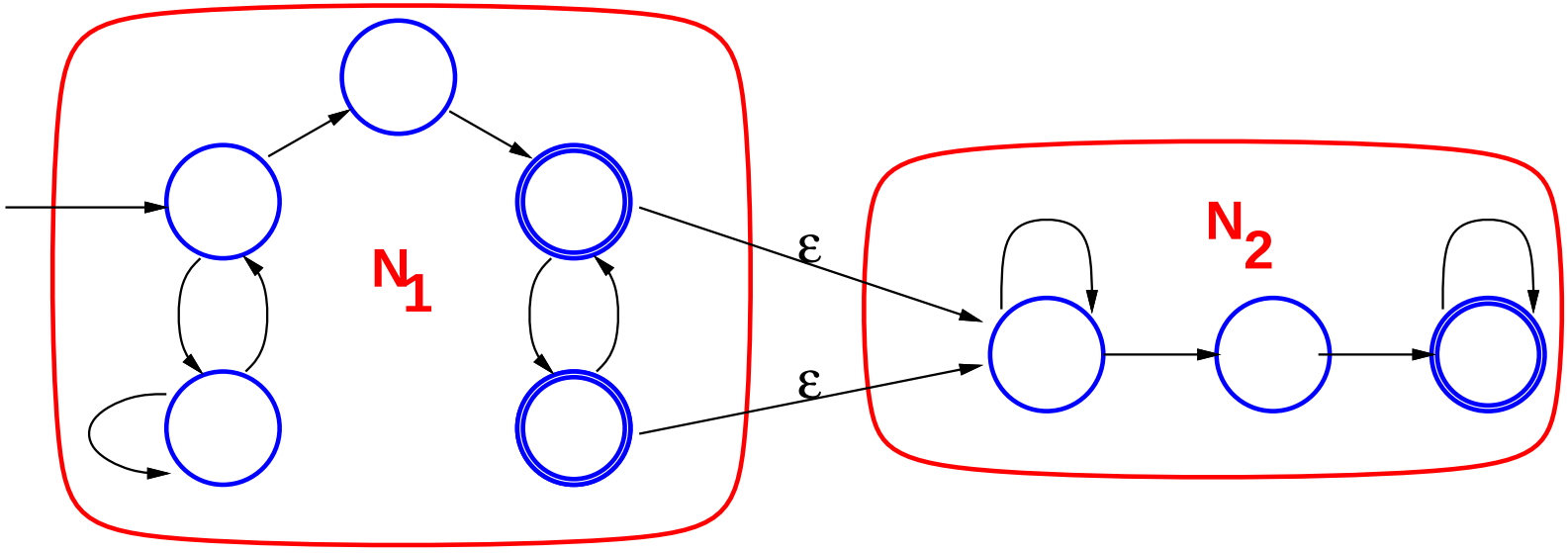
Define $N = (Q, \Sigma, \delta, q_0, F)$:

- $Q = q_0 \cup Q_1 \cup Q_2$
- Σ is the same, q_0 is the start state
- $F = F_1 \cup F_2$
- $$\delta'(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

Regular Languages Closed UnderConcatenation



Regular LanguagesClosed Under Concatenation



Regular LanguagesClosed Under Concatenation

Suppose

- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept L_1, and
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accept L_2.

Define $N = (Q, \Sigma, \delta, q_1, F_2)$:

- $Q = Q_1 \cup Q_2$

Regular LanguagesClosed Under Concatenation

Suppose

- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept L_1, and
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accept L_2.

Define $N = (Q, \Sigma, \delta, q_1, F_2)$:

- $Q = Q_1 \cup Q_2$
- q_1 is the start state of N

Regular LanguagesClosed Under Concatenation

Suppose

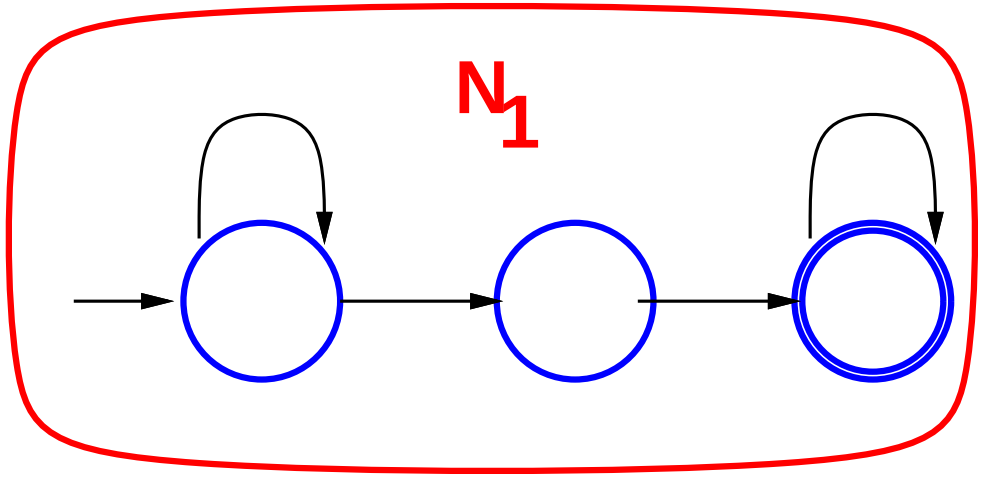
- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accept L_1, and
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accept L_2.

Define $N = (Q, \Sigma, \delta, q_1, F_2)$:

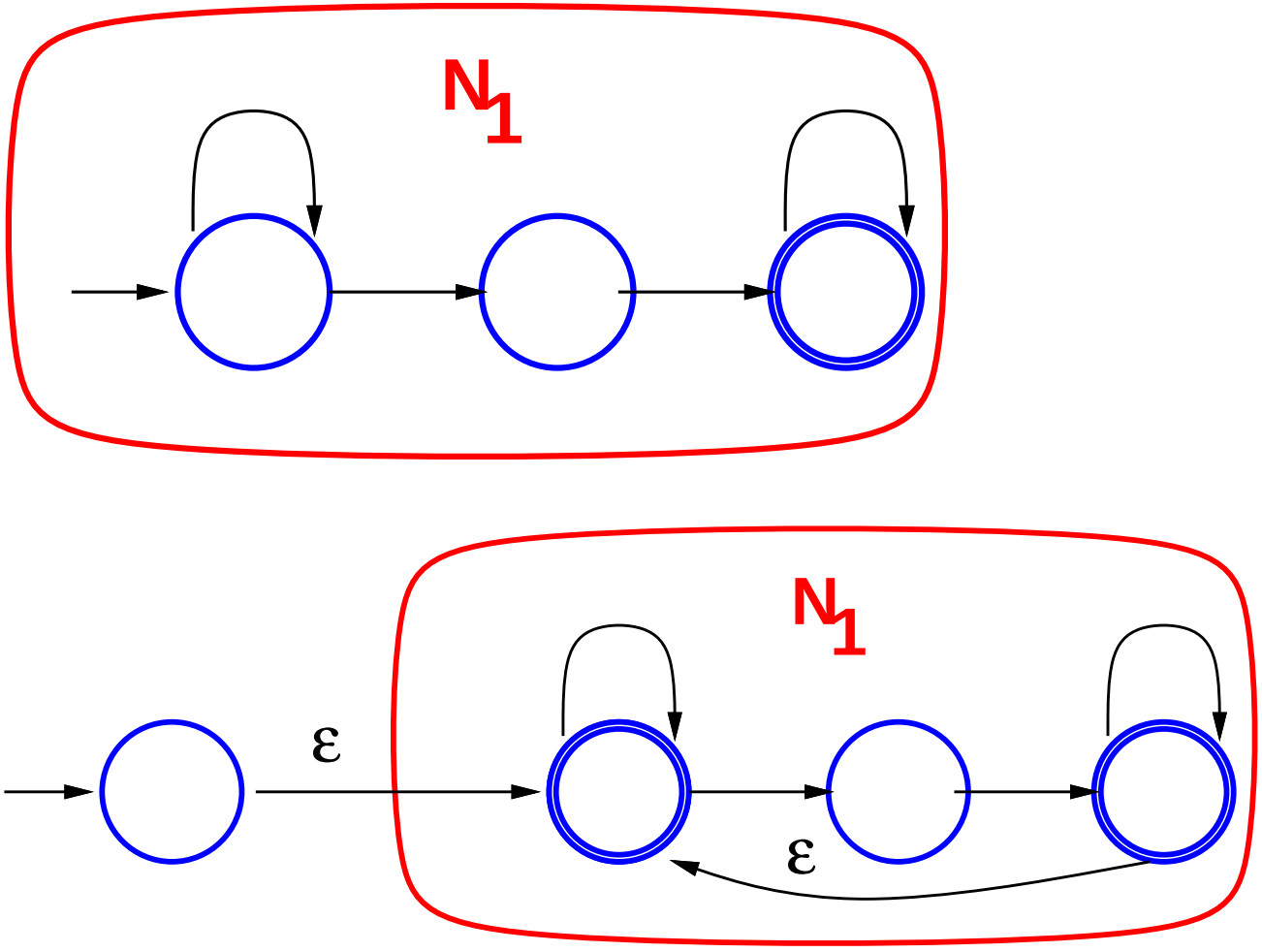
- $Q = Q_1 \cup Q_2$
- q_1 is the start state of N
- F_2 is the set of accept states of N

$$\delta'(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F \\ \delta_1(q,a) & q \in Q_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

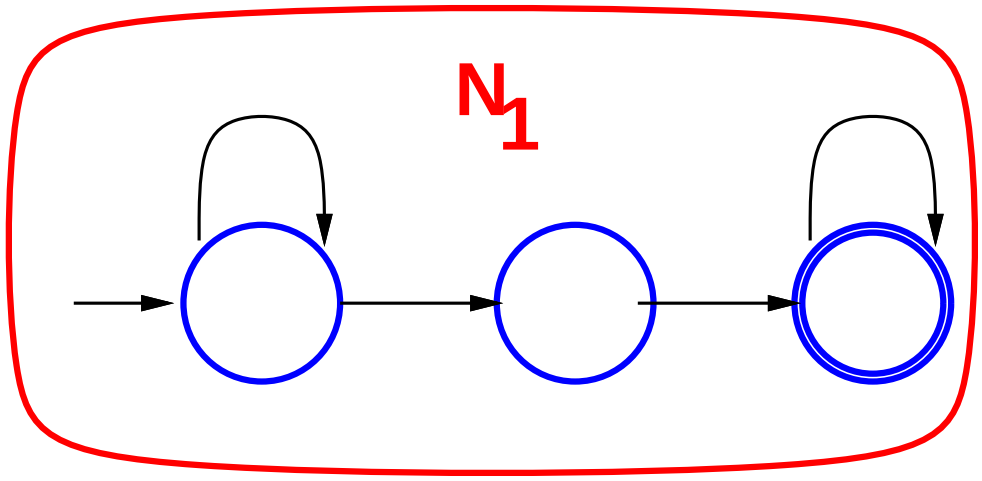
Regular Languages Closed Under Star

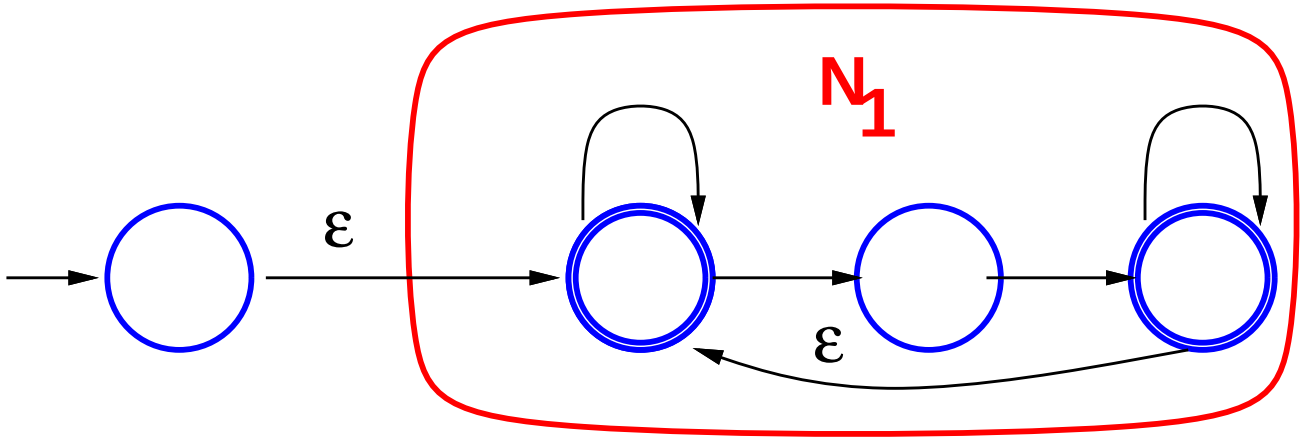


Regular Languages Closed Under Star



Regular Languages Closed Under Star





Oops - bad construction. How do we fix it?

Regular Languages Closed Under Star

Suppose $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1.
Define $N = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0\} \cup Q_1$

Regular Languages Closed Under Star

Suppose $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1.
Define $N = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0\} \cup Q_1$
- q_0 is the new start state.

Regular Languages Closed Under Star

Suppose $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1.
Define $N = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0\} \cup Q_1$
- q_0 is the new start state.
- $F = \{q_0 \cup F_1\}$

$$\delta'(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,\varepsilon) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

Summary

- Regular languages are closed under

Summary

- Regular languages are closed under
  - union

Summary

- Regular languages are closed under
  - union
  - concatenation

Summary

- Regular languages are closed under
  - union
  - concatenation
  - star

Summary

- Regular languages are closed under
  - union
  - concatenation
  - star
- Non-deterministic finite automata

Summary

- Regular languages are closed under
  - union
  - concatenation
  - star
- Non-deterministic finite automata
  - are equivalent to deterministic finite automata

Summary

- Regular languages are closed under
  - union
  - concatenation
  - star
- Non-deterministic finite automata
  - are equivalent to deterministic finite automata
  - but much easier to use in some proofs and constructions.

Regular Expressions

A notation for building up languages by describing them as expressions, *e.g.* $(0 \cup 1)0^*$.

- 0 and 1 are shorthand for $\{0\}$ and $\{1\}$

Regular Expressions

A notation for building up languages by describing them as expressions, *e.g.* $(0 \cup 1)0^*$.

- 0 and 1 are shorthand for $\{0\}$ and $\{1\}$
- so $(0 \cup 1) = \{0, 1\}$.

Regular Expressions

A notation for building up languages by describing them as expressions, *e.g.* $(0 \cup 1)0^*$.

- 0 and 1 are shorthand for $\{0\}$ and $\{1\}$
- so $(0 \cup 1) = \{0, 1\}$.
- 0^* is shorthand for $\{0\}^*$.

Regular Expressions

A notation for building up languages by describing them as expressions, *e.g.* $(0 \cup 1)0^*$.

- 0 and 1 are shorthand for $\{0\}$ and $\{1\}$
- so $(0 \cup 1) = \{0, 1\}$.
- 0^* is shorthand for $\{0\}^*$.
- concatenation, like multiplication, is implicit, so 0^*10^* is shorthand for the set of all strings over $\Sigma = \{0, 1\}$ having exactly a single 1.

Regular Expressions

A notation for building up languages by describing them as expressions, *e.g.* $(0 \cup 1)0^*$.

- 0 and 1 are shorthand for $\{0\}$ and $\{1\}$
- so $(0 \cup 1) = \{0, 1\}$.
- 0^* is shorthand for $\{0\}^*$.
- concatenation, like multiplication, is implicit, so 0^*10^* is shorthand for the set of all strings over $\Sigma = \{0, 1\}$ having exactly a single 1.

Q.: What does $(0 \cup 1)0^*$ stand for?

Regular Expressions

A notation for building up languages by describing them as expressions, *e.g.* $(0 \cup 1)0^*$.

- 0 and 1 are shorthand for $\{0\}$ and $\{1\}$
- so $(0 \cup 1) = \{0, 1\}$.
- 0^* is shorthand for $\{0\}^*$.
- concatenation, like multiplication, is implicit, so 0^*10^* is shorthand for the set of all strings over $\Sigma = \{0, 1\}$ having exactly a single 1.

Q.: What does $(0 \cup 1)0^*$ stand for?

Remark: Regular expressions are often used in text editors or shell scripts.

More Examples

Let Σ be an alphabet.

- The regular expression Σ is the language of one-symbol strings.
- Σ^* is all strings.
- Σ^*1 all strings ending in 1.
- $0\Sigma^* \cup \Sigma^*1$ strings starting with 0 or ending in 1.

More Examples

Let Σ be an alphabet.

- The regular expression Σ is the language of one-symbol strings.
- Σ^* is all strings.
- Σ^*1 all strings ending in 1.
- $0\Sigma^* \cup \Sigma^*1$ strings starting with 0 or ending in 1.

Just like in arithmetic, operations have precedence:

- star first
- concatenation next
- union last
- parentheses used to change usual order

Regular Expressions – Formal Definition

Syntax: R is a regular expression if R is of form

- a for some $a \in \Sigma$

Regular Expressions – Formal Definition

Syntax: R is a regular expression if R is of form

- a for some $a \in \Sigma$
- ε

Regular Expressions – Formal Definition

Syntax: R is a regular expression if R is of form

- a for some $a \in \Sigma$
- ε
- $\emptyset$

Regular Expressions – Formal Definition

Syntax: R is a regular expression if R is of form

- a for some $a \in \Sigma$
- ε
- $\emptyset$
- $(R_1 \cup R_2)$ for regular expressions R_1 and R_2

Regular Expressions – Formal Definition

Syntax: R is a regular expression if R is of form

- a for some $a \in \Sigma$
- ε
- $\emptyset$
- $(R_1 \cup R_2)$ for regular expressions R_1 and R_2
- $(R_1 \circ R_2)$ for regular expressions R_1 and R_2

Regular Expressions – Formal Definition

Syntax: R is a regular expression if R is of form

- a for some $a \in \Sigma$
- ε
- $\emptyset$
- $(R_1 \cup R_2)$ for regular expressions R_1 and R_2
- $(R_1 \circ R_2)$ for regular expressions R_1 and R_2
- (R_1^*) for regular expression R_1

Regular Expressions – Formal Definition

Let $L(R)$ be the language denoted by regular expression R.

R	$L(R)$
a	$\{a\}$
ε	$\{\varepsilon\}$
$(R_1 \cup R_2)$	$L(R_1) \cup L(R_2)$
$(R_1 \circ R_2)$	$L(R_1) \circ L(R_2)$
$(R_1)^*$	$L(R_1)^*$

Regular Expressions – Formal Definition

Let $L(R)$ be the language denoted by regular expression R.

R	$L(R)$
a	$\{a\}$
ε	$\{\varepsilon\}$
$(R_1 \cup R_2)$	$L(R_1) \cup L(R_2)$
$(R_1 \circ R_2)$	$L(R_1) \circ L(R_2)$
$(R_1)^*$	$L(R_1)^*$

Q.: What's the difference between $\emptyset$ and ε?

Regular Expressions – Formal Definition

Let $L(R)$ be the language denoted by regular expression R.

R	$L(R)$
a	$\{a\}$
ε	$\{\varepsilon\}$
$(R_1 \cup R_2)$	$L(R_1) \cup L(R_2)$
$(R_1 \circ R_2)$	$L(R_1) \circ L(R_2)$
$(R_1)^*$	$L(R_1)^*$

Q.: What's the difference between $\emptyset$ and ε?

Q.: Isn't this definition circular?

Remarkable Fact

Thm.: A language, L, is described by a regular expression, R, if and only if L is regular.

Remarkable Fact

Thm.: A language, L, is described by a regular expression, R, if and only if L is regular.

$\Longrightarrow$ construct an NFA accepting R.

Remarkable Fact

Thm.: A language, L, is described by a regular expression, R, if and only if L is regular.

$\Longrightarrow$ construct an NFA accepting R.

$\Longleftarrow$ Given a regular language, L, construct an equivalent regular expression.