形式语言与计算复杂性

第 2 章 Automata and Languages

2.1 Regular Languages

黄文超

https://faculty.ustc.edu.cn/huangwenchao

$\longrightarrow$ 教学课程 $\longrightarrow$ 形式语言与计算复杂性

Outline

Outline

问: What to *begin* w.r.t theory of computation?
答: Define computer

问: How to define a computer?
答: Use an *idealized computer* called a *computational model*

问: How to define a computational model?
答: Let's define a *simplest* one *first*.

问: What is the simplest computational model?
答: *finite state machine* or ***finite automaton***

问: How to differentiate computational models, e.g., *finite automaton*?
答: Define Languages, e.g., ***Regular Language***

问: What to *begin* w.r.t theory of computation?
答: Define computer

问: How to define a computer?
答: Use an *idealized computer* called a *computational model*

问: How to define a computational model?
答: Let's define a *simplest* one *first*.

问: What is the simplest computational model?
答: *finite state machine* or ***finite automaton***

问: How to differentiate computational models, e.g., *finite automaton*?
答: Define Languages, e.g., ***Regular Language***

问: What to *begin* w.r.t theory of computation?
答: Define computer

问: How to define a computer?
答: Use an *idealized computer* called a *computational model*

问: How to define a computational model?
答: Let's define a *simplest* one *first*.

问: What is the simplest computational model?
答: *finite state machine* or ***finite automaton***

问: How to differentiate computational models, e.g., *finite automaton*?
答: Define Languages, e.g., ***Regular Language***

问: What to *begin* w.r.t theory of computation?
答: Define computer

问: How to define a computer?
答: Use an *idealized computer* called a *computational model*

问: How to define a computational model?
答: Let's define a *simplest* one *first*.

问: What is the simplest computational model?
答: *finite state machine* or ***finite automaton***

问: How to differentiate computational models, e.g., *finite automaton*?
答: Define Languages, e.g., ***Regular Language***

问: What to *begin* w.r.t theory of computation?
答: Define computer

问: How to define a computer?
答: Use an *idealized computer* called a *computational model*

问: How to define a computational model?
答: Let's define a *simplest* one *first*.

问: What is the simplest computational model?
答: *finite state machine* or ***finite automaton***

问: How to differentiate computational models, e.g., *finite automaton*?
答: Define Languages, e.g., ***Regular Language***

Outline

Outline

问: *Informally*, what *are* finite automata?
答: Good models for computers with an *extremely limited amount of memory*.

问: What can a computer do with such a small memory?
答: Many useful things!

问: Any example?
答: an automatic door

问: *Informally*, what *are* finite automata?
答: Good models for computers with an *extremely limited amount of memory*.

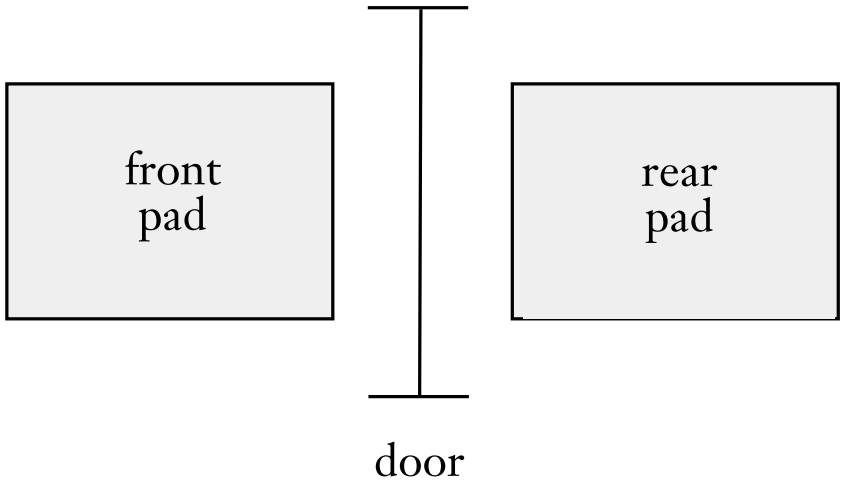
问: What can a computer do with such a small memory?
答: Many useful things!

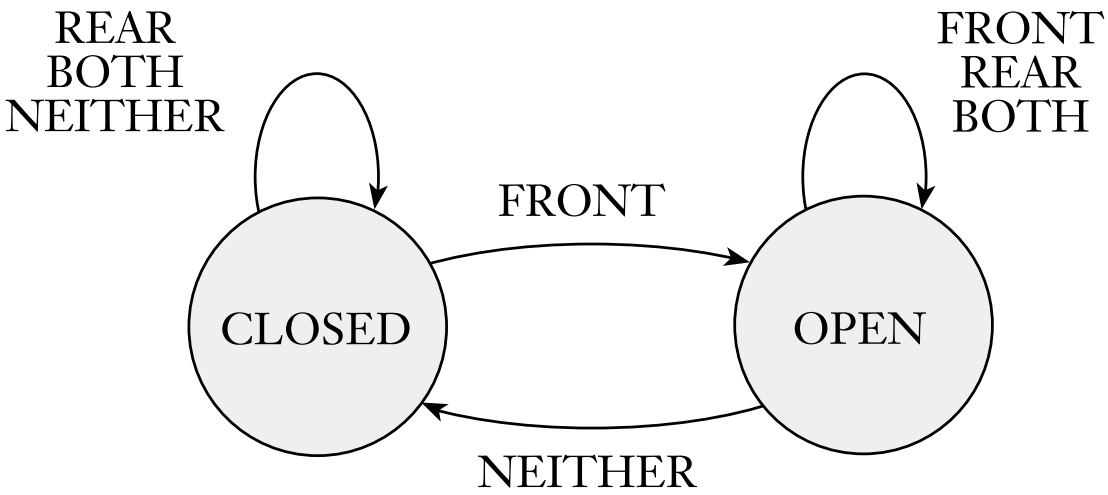
问: Any example?
答: an automatic door

问: *Informally*, what *are* finite automata?
答: Good models for computers with an *extremely limited amount of memory*.

问: What can a computer do with such a small memory?
答: Many useful things!

问: Any example?
答: an automatic door



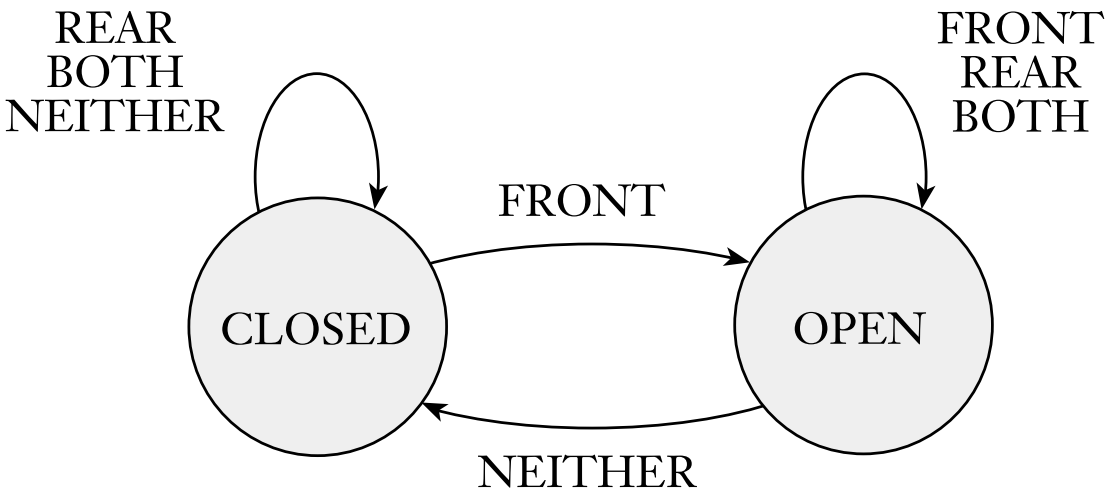


input signal

state	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

问: So, does this finite automaton require a large memory?

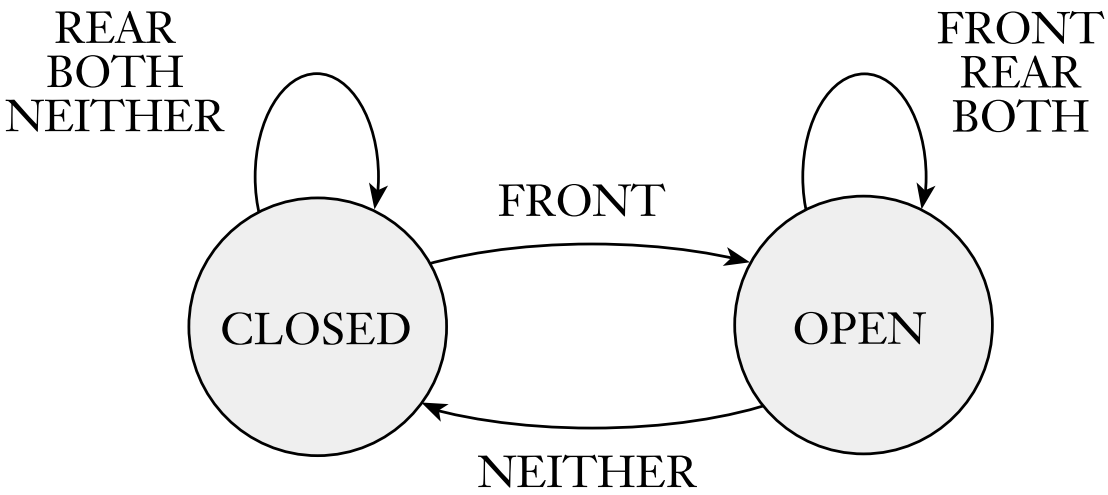
答: Of course No.



input signal

state	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

问: So, does this finite automaton require a large memory?
答: Of course No.



input signal

state	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

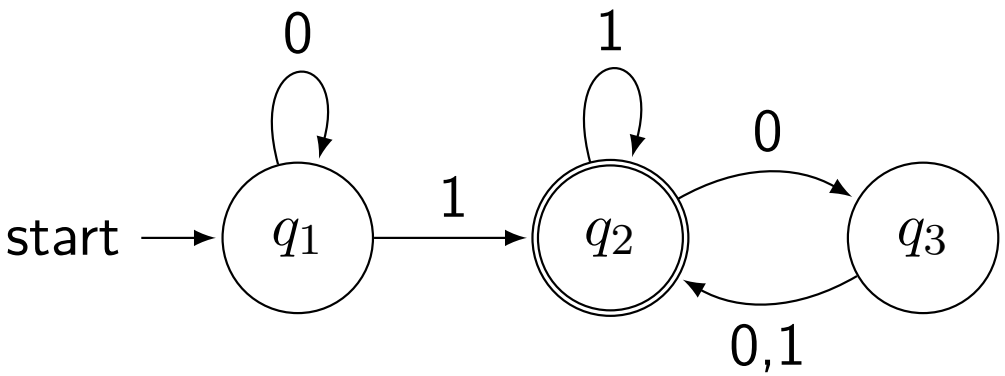
问: So, does this finite automaton require a large memory?

答: Of course No.

定义: 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*,
3. $\delta : Q \times \Sigma \to Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.



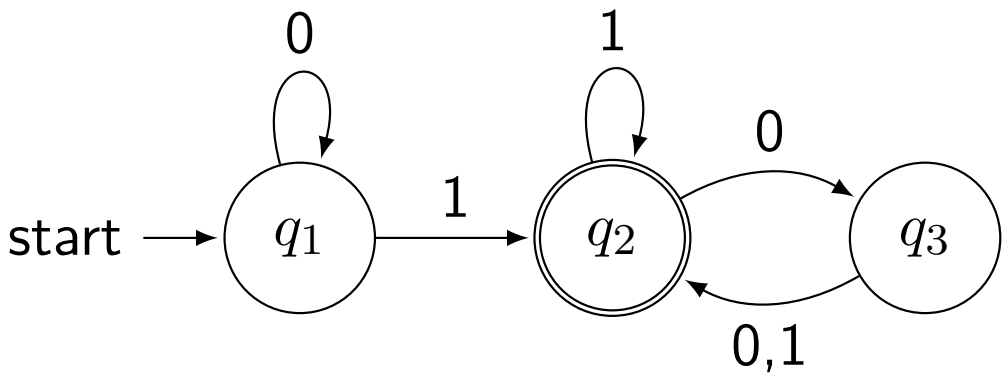
- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- start state $= q_1$
- $F = \{q_2\}$

定义

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. $\delta : Q \times \Sigma \to Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

- $\delta(q_1, 0) = q_1$
- $\delta(q_1, 1) = q_2$
- $\delta(q_2, 0) = q_3$
- $\delta(q_2, 1) = q_2$
- $\delta(q_3, 0) = q_2$
- $\delta(q_3, 1) = q_2$



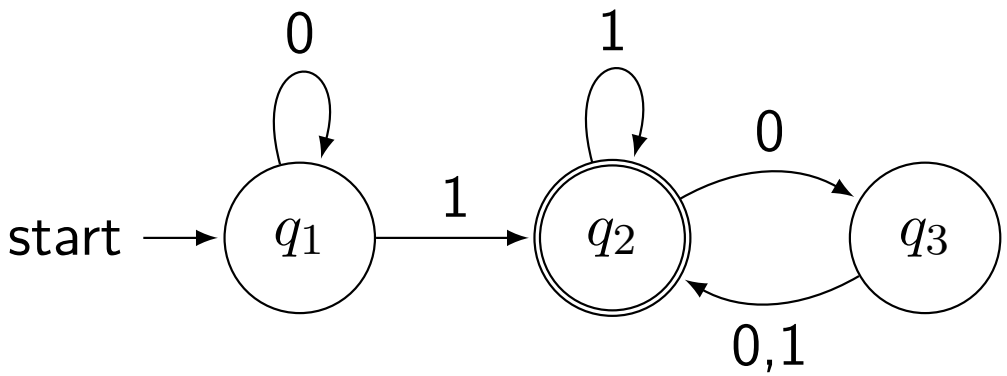
- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- start state $= q_1$
- $F = \{q_2\}$

定义

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. $\delta : Q \times \Sigma \to Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

- $\delta(q_1, 0) = q_1$
- $\delta(q_1, 1) = q_2$
- $\delta(q_2, 0) = q_3$
- $\delta(q_2, 1) = q_2$
- $\delta(q_3, 0) = q_2$
- $\delta(q_3, 1) = q_2$



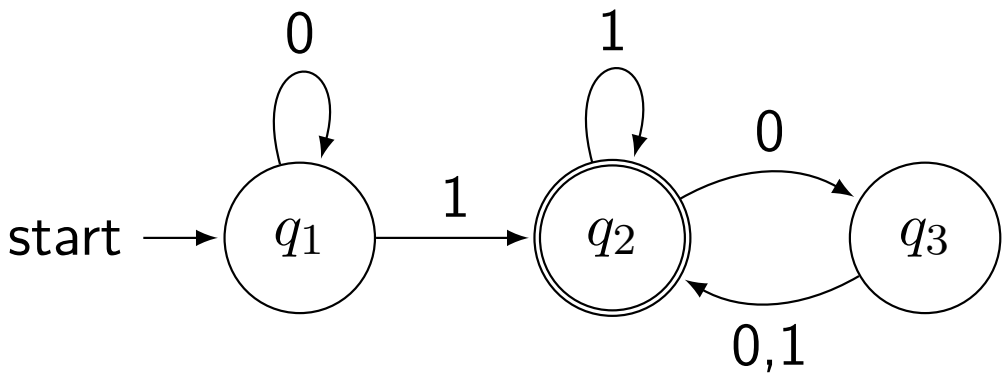
- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- start state $= q_1$
- $F = \{q_2\}$

定义

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. $\delta : Q \times \Sigma \to Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

- $\delta(q_1, 0) = q_1$
- $\delta(q_1, 1) = q_2$
- $\delta(q_2, 0) = q_3$
- $\delta(q_2, 1) = q_2$
- $\delta(q_3, 0) = q_2$
- $\delta(q_3, 1) = q_2$



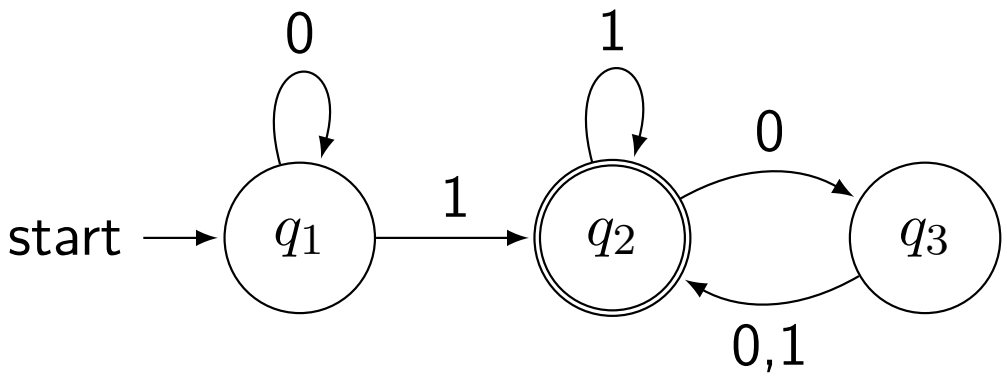
- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- start state $= q_1$
- $F = \{q_2\}$

定义

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. $\delta : Q \times \Sigma \to Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

- $\delta(q_1, 0) = q_1$
- $\delta(q_1, 1) = q_2$
- $\delta(q_2, 0) = q_3$
- $\delta(q_2, 1) = q_2$
- $\delta(q_3, 0) = q_2$
- $\delta(q_3, 1) = q_2$



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

定义

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. $\delta : Q \times \Sigma \to Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

- $\delta(q_1, 0) = q_1$
- $\delta(q_1, 1) = q_2$
- $\delta(q_2, 0) = q_3$
- $\delta(q_2, 1) = q_2$
- $\delta(q_3, 0) = q_2$
- $\delta(q_3, 1) = q_2$

问: Then, what can a finite automaton do?
答: Solve certain problems.

问: Which problems specifically?
答: Define problems first.

问: Then, what can a finite automaton do?
答: Solve certain problems.

问: Which problems specifically?
答: Define problems first.

问: How to define a problem?

Given a *boolean* function $f : S \to \{0, 1\}$, and $x \in S$, compute $f(x)$.

- How to define S and f ?

问: How to further define a problem?

Given a set $A \subseteq S$, and $x \in S$, compute whether $x \in A$.

- $x \in A \Rightarrow f(x) = 1$ and $x \notin A \Rightarrow f(x) = 0$

$f(x)=1$

$f(x)=0$

问: How to define a problem?

Given a *boolean* function $f : S \to \{0, 1\}$, and $x \in S$, compute $f(x)$.

- How to define S and f ?

问: How to further define a problem?

Given a set $A \subseteq S$, and $x \in S$, compute whether $x \in A$.

- $x \in A \Rightarrow f(x) = 1$ and $x \notin A \Rightarrow f(x) = 0$

问: How to define a problem?

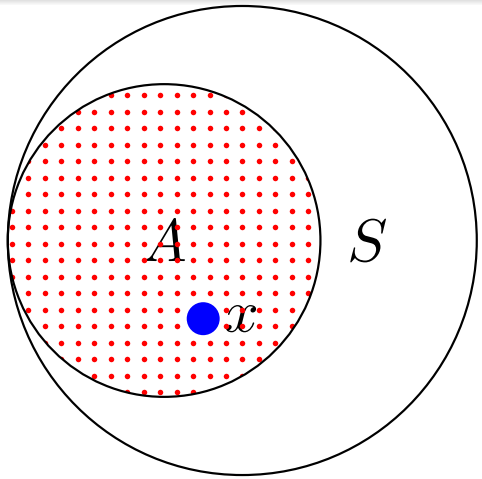
Given a *boolean* function $f : S \to \{0, 1\}$, and $x \in S$, compute $f(x)$.

- How to define S and f ?

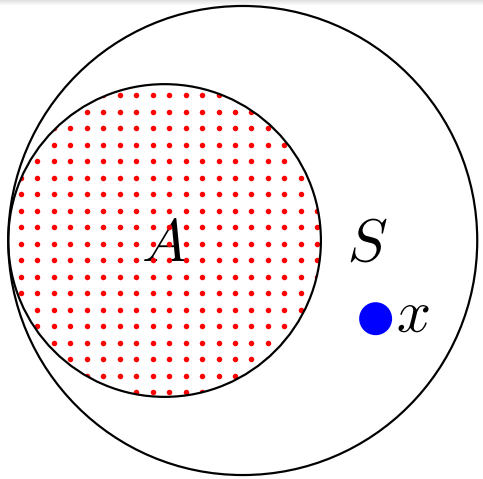
问: How to further define a problem?

Given a set $A \subseteq S$, and $x \in S$, compute whether $x \in A$.

- $x \in A \Rightarrow f(x) = 1$ and $x \notin A \Rightarrow f(x) = 0$



$f(x)=1$



$f(x)=0$

问: So, what can a finite automaton do?

1. define a *machine* M, e.g., a *finite automaton*
2. define S as the set of all strings with alphabet Σ
   - e.g., $S = \{0, 1, 00, 01, 000, 001, 010, 011, \dots\}$
3. define *accept state*
   - e.g., q_2 is an accept state
4. define ***language*** of M, i.e., $L(M) = \{s \in S \mid M \text{ accepts } s\}$
5. 答: Given $x \in S$, M can compute whether $x \in A$, where $A = L(M)$

问 1: Any examples for *finite automata* and their *languages*?
答: 见下页
问 2: What can a finite automaton do w.r.t languages.
答: 见 Jump to the answer

问: So, what can a finite automaton do?

1. define a *machine* M, e.g., a *finite automaton*
2. define S as the set of all strings with alphabet Σ
   - e.g., $S = \{0, 1, 00, 01, 000, 001, 010, 011, \dots\}$
3. define *accept state*
   - e.g., q_2 is an accept state
4. define ***language*** of M, i.e., $L(M) = \{s \in S \mid M \text{ accepts } s\}$
5. 答: Given $x \in S$, M can compute whether $x \in A$, where $A = L(M)$

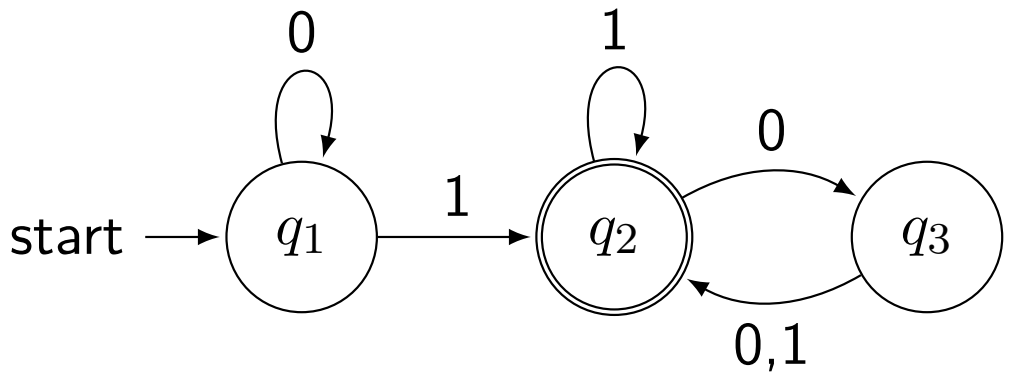
问 1: Any examples for *finite automata* and their *languages*?
答: 见下页
问 2: What can a finite automaton do w.r.t languages.
答: 见 Jump to the answer

问: So, what can a finite automaton do?

1. define a *machine* M, e.g., a *finite automaton*
2. define S as the set of all strings with alphabet Σ
   - e.g., $S = \{0, 1, 00, 01, 000, 001, 010, 011, \dots\}$
3. define *accept state*
   - e.g., q_2 is an accept state
4. define ***language*** of M, i.e., $L(M) = \{s \in S \mid M \text{ accepts } s\}$
5. 答: Given $x \in S$, M can compute whether $x \in A$, where $A = L(M)$

问 1: Any examples for *finite automata* and their *languages*?
答: 见下页
问 2: What can a finite automaton do w.r.t languages.
答: 见 Jump to the answer

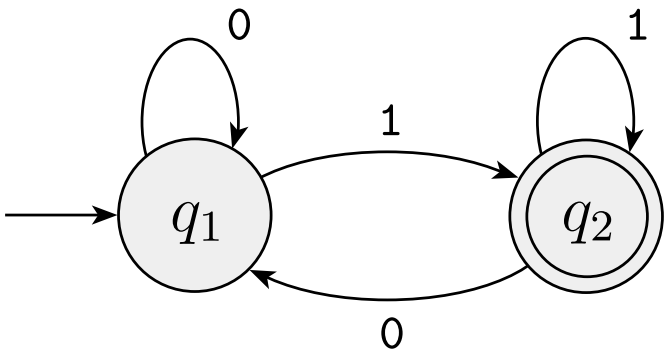
问 1: Any examples for *finite automata* and their *languages*?
例 1: For *finite automaton* M_1:



let $A = \{w \mid w$ contains at least one 1 and an even number of 0s follow the last $1\}$. Then, $L(M_1) = A$

问 1: Any examples for *finite automata* and their *languages*?

例 2: For *finite automaton* M_2

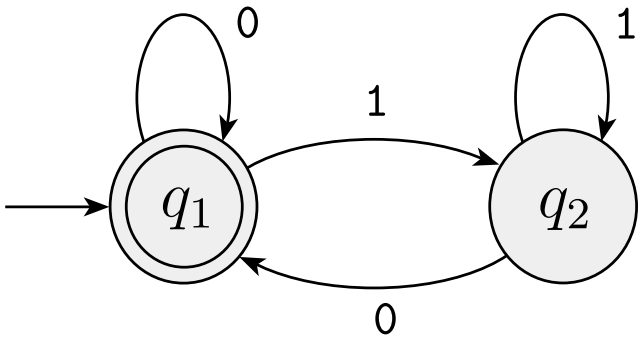


$\Sigma = \{0, 1\}$

$L(M_2) = \{w \mid w \text{ ends in a } 1\}$

问 1: Any examples for *finite automata* and their *languages*?

例 3: For *finite automaton* M_3

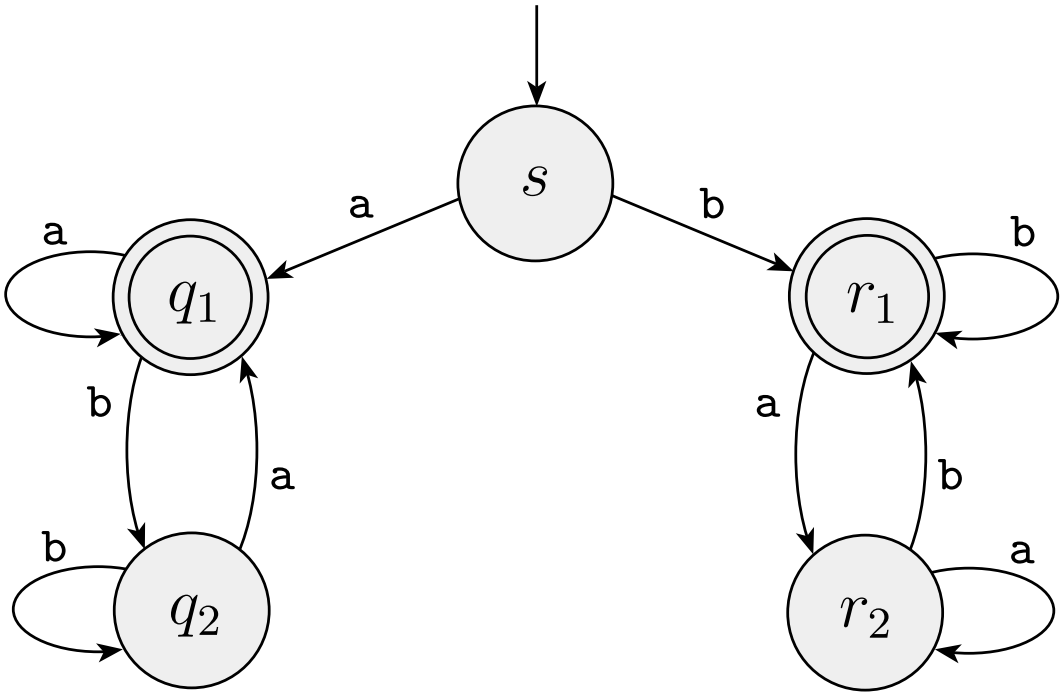


$\Sigma = \{0, 1\}$

$L(M_3) = \{w \mid w$ is the empty string ε or ends in a $0\}$

问 1: Any examples for *finite automata* and their *languages*?

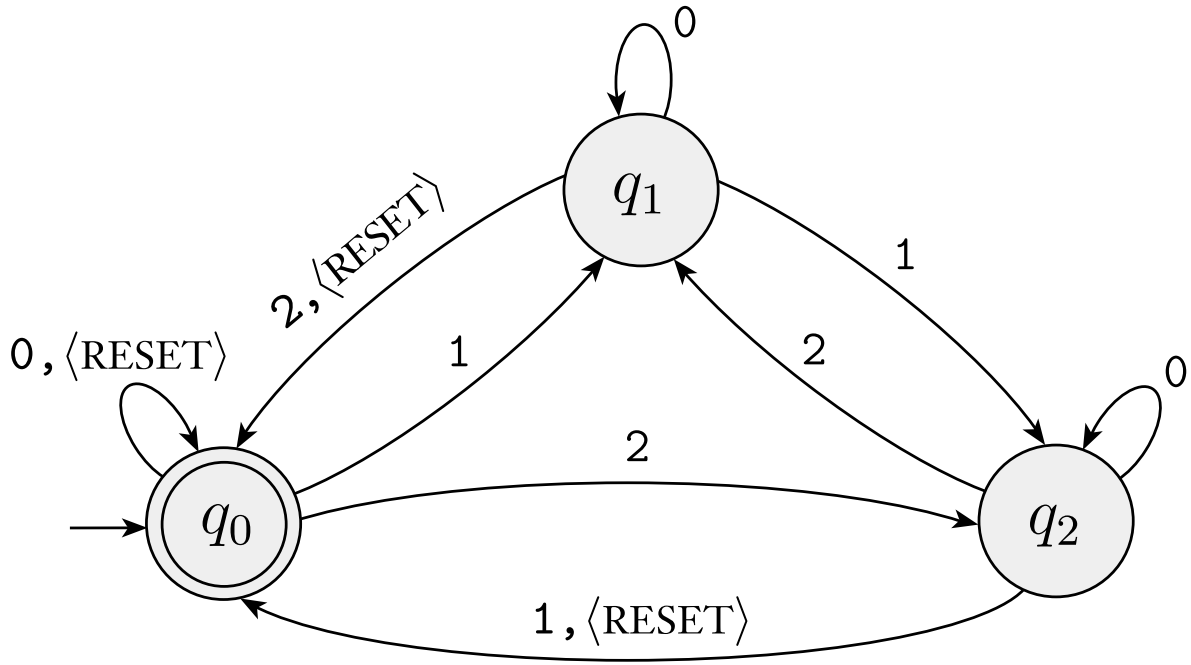
例 4: For *finite automaton* M_4



$\Sigma = \{a, b\}$

$L(M_4) = \{w \mid w \text{ is the string that starts and ends with the same symbol}\}$

问 1: Any examples for *finite automata* and their *languages*?
例 5: For *finite automaton* M_5



$\Sigma = \{0, 1, 2, \langle \text{RESET} \rangle\}$
$L(M_5) = \{w \mid w$ is the string that the sum of the symbols is a multiple of 3$\}$

回顾: 问: So, what can a finite automaton do?

1. define *finite automaton*
2. define *language* of M
3. 答: Given $x \in S$, M can compute whether $x \in A$, where $A = L(M)$

定义: Recognize

M *recognizes* language A, if $A = \{w \mid M \text{ accepts } w\}$

定义: Regular language

A language is called a ***regular language*** if some *finite automaton recognizes* it. See Corollary

例: Regular languages: $L(M_1)$, $L(M_2)$, $L(M_3)$, $L(M_4)$, $L(M_5)$

回顾: 问: So, what can a finite automaton do?

1. define *finite automaton*
2. define *language* of M
3. 答: Given $x \in S$, M can compute whether $x \in A$, where $A = L(M)$

定义: Recognize

M *recognizes* language A, if $A = \{w \mid M \text{ accepts } w\}$

定义: Regular language

A language is called a ***regular language*** if some *finite automaton recognizes* it. See Corollary

例: Regular languages: $L(M_1)$, $L(M_2)$, $L(M_3)$, $L(M_4)$, $L(M_5)$

回顾: 问: So, what can a finite automaton do?

1. define *finite automaton*
2. define *language* of M
3. 答: Given $x \in S$, M can compute whether $x \in A$, where $A = L(M)$

定义: Recognize

M *recognizes* language A, if $A = \{w \mid M \text{ accepts } w\}$

定义: Regular language

A language is called a ***regular language*** if some *finite automaton recognizes* it. See Corollary

例: Regular languages: $L(M_1)$, $L(M_2)$, $L(M_3)$, $L(M_4)$, $L(M_5)$

回顾: 问: So, what can a finite automaton do?

1. define *finite automaton*
2. define *language* of M
3. 答: Given $x \in S$, M can compute whether $x \in A$, where $A = L(M)$

定义: Recognize

M *recognizes* language A, if $A = \{w \mid M \text{ accepts } w\}$

定义: Regular language

A language is called a ***regular language*** if some *finite automaton recognizes* it. See Corollary

例: Regular languages: $L(M_1)$, $L(M_2)$, $L(M_3)$, $L(M_4)$, $L(M_5)$

定义: Recognize

M *recognizes* language A, if $A = \{w \mid M \text{ accepts } w\}$

定义: Regular language

A language is called a ***regular language*** if some *finite automaton recognizes* it.

问: What can a finite automaton do w.r.t languages? ▸ Back to the problem

答: A *finite automaton* can *recognize* a *regular language*.

问 1: Can a finite automaton recognize *all* languages?

答: Of course not. One is *non-regular* language. ▸ Part.4 Non-regular Languages

问 2: What are the properties of regular languages (i.e., finite automata)?

答: 见下页

定义: Recognize

M *recognizes* language A, if $A = \{w \mid M \text{ accepts } w\}$

定义: Regular language

A language is called a ***regular language*** if some *finite automaton recognizes* it.

问: What can a finite automaton do w.r.t languages? ▸ Back to the problem

答: A *finite automaton* can *recognize* a *regular language*.

问 1: Can a finite automaton recognize *all* languages?

答: Of course not. One is *non-regular* language. ▸ Part.4 Non-regular Languages

问 2: What are the properties of regular languages (i.e., finite automata)?

答: 见下页

定义: Recognize

M *recognizes* language A, if $A = \{w \mid M \text{ accepts } w\}$

定义: Regular language

A language is called a ***regular language*** if some *finite automaton recognizes* it.

问: What can a finite automaton do w.r.t languages? ▸ Back to the problem
答: A *finite automaton* can *recognize* a *regular language*.

问 1: Can a finite automaton recognize *all* languages?
答: Of course not. One is *non-regular* language. ▸ Part.4 Non-regular Languages

问 2: What are the properties of regular languages (i.e., finite automata)?
答: 见下页

定义: Recognize

M *recognizes* language A, if $A = \{w \mid M \text{ accepts } w\}$

定义: Regular language

A language is called a ***regular language*** if some *finite automaton recognizes* it.

问: What can a finite automaton do w.r.t languages? ▸ Back to the problem
答: A *finite automaton* can *recognize* a *regular language*.

问 1: Can a finite automaton recognize *all* languages?
答: Of course not. One is *non-regular* language. ▸ Part.4 Non-regular Languages

问 2: What are the properties of regular languages (i.e., finite automata)?
答: 见下页

问: What are the properties of regular languages (i.e., finite automata)?
答:

- Preparation ▸ Part.1
  - Design finite automata
  - Define *regular operations*
- Nondeterminism ▸ Part.2
  - Definition of *NFAs* and *DFAs*
  - *Equivalence* of NFAs and DFAs
  - *Closure* under Regular Operations
- Relations to *regular expressions* ▸ Part.3

Outline

(1) Design finite automata

例: Suppose that is $\Sigma = \{0, 1\}$, and the language consists of all strings with an *odd number* of 1s.

(1) Design finite automata
例: Suppose that is $\Sigma = \{0, 1\}$, and the language consists of all strings with an *odd number* of 1s.

Step 1: Once you have *determined* the *necessary information* to remember about the string as it is being read, you *represent* this *information* as a finite list of possibilities.

1. even
2. odd

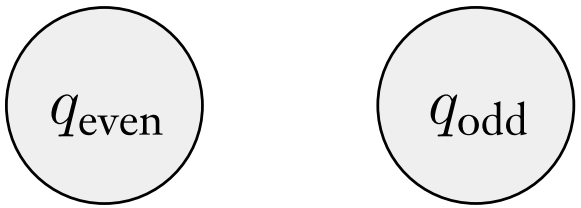
(1) Design finite automata

例: Suppose that is $\Sigma = \{0, 1\}$, and the language consists of all strings with an *odd number* of 1s.

Step 1: Once you have *determined* the *necessary information* to remember about the string as it is being read, you *represent* this *information* as a finite list of possibilities.

1. even
2. odd

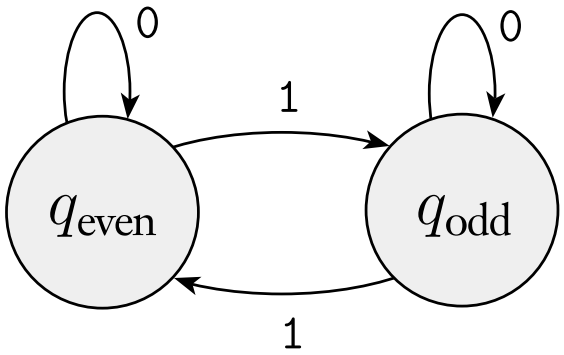
Step 2: Assign a state to each of the possibilities



(1) Design finite automata

例: Suppose that is $\Sigma = \{0, 1\}$, and the language consists of all strings with an *odd number* of 1s.

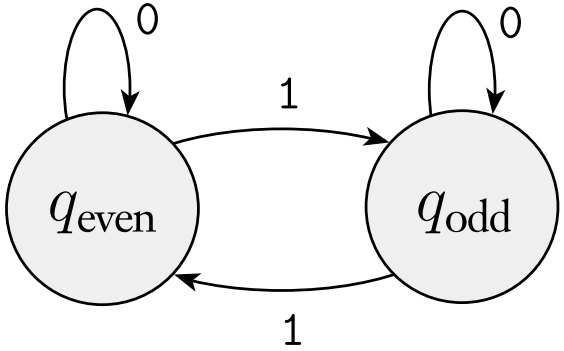
Step 3: Assign the transitions



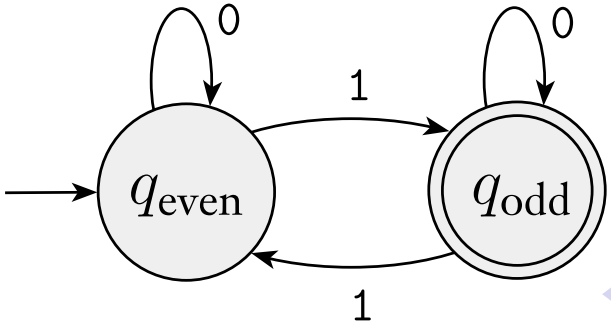
(1) Design finite automata

例: Suppose that is $\Sigma = \{0, 1\}$, and the language consists of all strings with an *odd number* of 1s.

Step 3: Assign the transitions



Step 4: Add the start and accept states



(2) Define regular operations

定义: Regular Operations

Let A and B be languages. We define the *regular operations* ***union***, ***concatenation***, and ***star*** as follows:

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

例

Let $\Sigma = \{a, b, \dots, z\}$. If $A = \{\text{good}, \text{bad}\}$ and $B = \{\text{boy}, \text{girl}\}$, then

- $A \cup B = \{\text{good, bad, boy, girl}\}$
- $A \circ B = \{\text{goodboy, goodgirl, badboy, badgirl}\}$
- $A^* = \{\varepsilon, \text{good,bad,goodgood,goodbad,badgood,badbad,}$ $\text{goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, } \dots\}$

(2) Define regular operations

定义: Regular Operations

Let A and B be languages. We define the *regular operations* ***union***, ***concatenation***, and ***star*** as follows:

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

例

Let $\Sigma = \{a, b, \dots, z\}$. If $A = \{\text{good}, \text{bad}\}$ and $B = \{\text{boy}, \text{girl}\}$, then

- $A \cup B = \{\text{good, bad, boy, girl}\}$
- $A \circ B = \{\text{goodboy, goodgirl, badboy, badgirl}\}$
- $A^* = \{\varepsilon, \text{good,bad,goodgood,goodbad,badgood,badbad,}$ $\text{goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, } \dots\}$

定义: Closed

A collection of objects is *closed* under some *operation*,
if applying that operation to members of the collection returns an *object still in the collection*.

例

Let $\mathcal{N} = \{1, 2, 3, \dots\}$ be the set of natural numbers. When we say that $\mathcal{N}$ is *closed under multiplication*, we mean that for any x and y in $\mathcal{N}$, the product $x \times y$ also is in $\mathcal{N}$

定理

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

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$

The class of regular languages is *closed* under the *concatenation* operation.

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$

定义: Closed

A collection of objects is *closed* under some *operation*,
if applying that operation to members of the collection returns an *object still in the collection*.

例

Let $\mathcal{N} = \{1, 2, 3, \dots\}$ be the set of natural numbers. When we say that $\mathcal{N}$ is *closed under multiplication*, we mean that for any x and y in $\mathcal{N}$, the product $x \times y$ also is in $\mathcal{N}$

定理

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

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$

The class of regular languages is *closed* under the *concatenation* operation.

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$

定义: Closed

A collection of objects is *closed* under some *operation*,
if applying that operation to members of the collection returns an *object still in the collection*.

例

Let $\mathcal{N} = \{1, 2, 3, \dots\}$ be the set of natural numbers. When we say that $\mathcal{N}$ is *closed under multiplication*, we mean that for any x and y in $\mathcal{N}$, the product $x \times y$ also is in $\mathcal{N}$

定理

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

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$

The class of regular languages is *closed* under the *concatenation* operation.

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$

定理: Closure under Union

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

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$

证明: (Proof by Construction)

Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$,
and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$
Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
2. Σ, the alphabet, is the same as in M_1 and M_2
3. $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
4. q_0 is the pair (q_1, q_2)
5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

定理: Closure under Union

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

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$

证明: (Proof by Construction)

Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$,
and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$
Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
2. Σ, the alphabet, is the same as in M_1 and M_2
3. $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
4. q_0 is the pair (q_1, q_2)
5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

定理: Closure under Union

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

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$

证明: (Proof by Construction)

Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$,
and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$
Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
2. Σ, the alphabet, is the same as in M_1 and M_2
3. $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
4. q_0 is the pair (q_1, q_2)
5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

定理: Closure under Union

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

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$

证明: (Proof by Construction)

Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$,
and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$
Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
2. Σ, the alphabet, is the same as in M_1 and M_2
3. $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
4. q_0 is the pair (q_1, q_2)
5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

定理: Closure under Union

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

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$

证明: (Proof by Construction)

Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$,
and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$
Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
2. Σ, the alphabet, is the same as in M_1 and M_2
3. $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
4. q_0 is the pair (q_1, q_2)
5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

定理: Closure under Union

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

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$

证明: (Proof by Construction)

Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$,
and M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$
Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
2. Σ, the alphabet, is the same as in M_1 and M_2
3. $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
4. q_0 is the pair (q_1, q_2)
5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$

定理: Closure under Concatenation

The class of regular languages is *closed* under the *concatenation* operation.

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$

证明: (Proof by Construction)

新问题: On constructing δ, can the input be broken into two pieces?

- M_1 accepts the first piece
- M_2 accepts the second piece

Not now

解决方法:

- Introducing *Nondeterminism*, and *NFA* (见下页)
- Prove the *Equivalence* of NFAs and DFAs See: Equivalence
- Prove the *Closure* See: Proof-of-closure

定理: Closure under Concatenation

The class of regular languages is *closed* under the *concatenation* operation.

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$

证明: (Proof by Construction)

新问题: On constructing δ, can the input be broken into two pieces?

- M_1 accepts the first piece
- M_2 accepts the second piece

Not now

解决方法:

- Introducing *Nondeterminism*, and *NFA* (见下页)
- Prove the *Equivalence* of NFAs and DFAs See: Equivalence
- Prove the *Closure* See: Proof-of-closure

定理: Closure under Concatenation

The class of regular languages is *closed* under the *concatenation* operation.

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$

证明: (Proof by Construction)

新问题: On constructing δ, can the input be broken into two pieces?

- M_1 accepts the first piece
- M_2 accepts the second piece

Not now

解决方法:

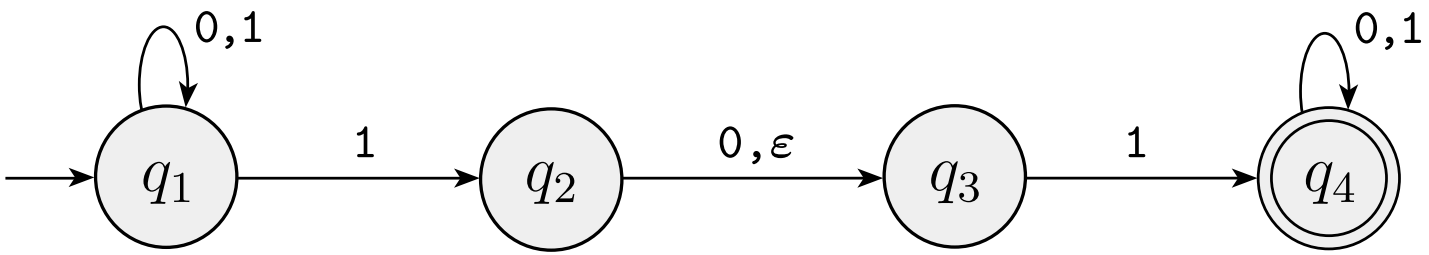
- Introducing *Nondeterminism*, and *NFA* (见下页)
- Prove the *Equivalence* of NFAs and DFAs See: Equivalence
- Prove the *Closure* See: Proof-of-closure

Outline

定义: Deterministic, Nondeterministic, DFA, NFA

When the machine is in a given state and reads the next input symbol

- if we know what the next state will be, it is *determined*. We call this *deterministic* computation. The machine is a *deterministic finite automaton*, ***DFA***
- otherwise, i.e., if *several choices* may exist for the next state, the machine is a *nondeterministic finite automaton*, ***NFA***.

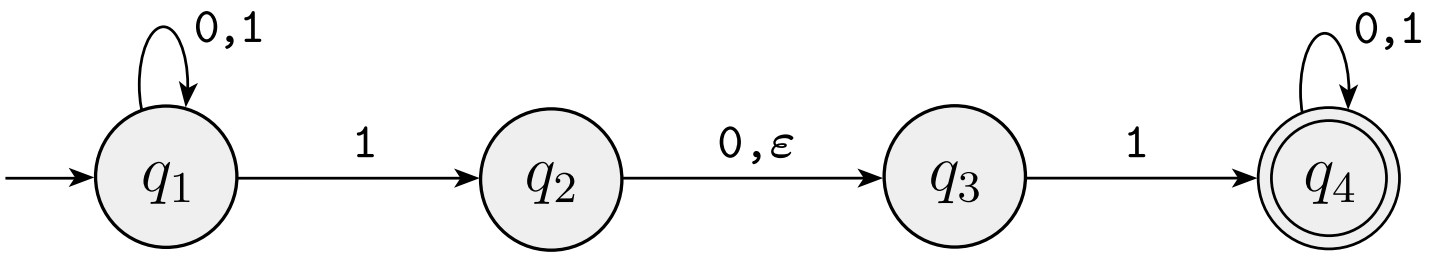


Observations on an NFA:

- In an NFA, a state may have *zero*, one, or *many* exiting arrows for *each alphabet symbol*
- This NFA has an arrow with the label ε

问: How does an NFA compute, on receiving a symbol?

- *splits* into *multiple copies* of itself and follows all the possibilities in parallel
  - If there are subsequent choices, the machine splits again.
  - If there is no choice, the copy dies
  - if *any one* is in an *accept state*, the NFA accepts the input string

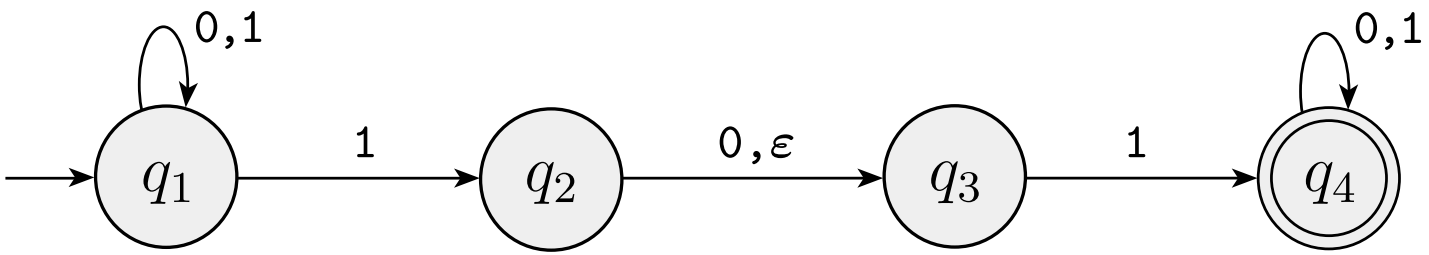


Observations on an NFA:

- In an NFA, a state may have *zero*, one, or *many* exiting arrows for *each alphabet symbol*
- This NFA has an arrow with the label ε

问: How does an NFA compute, on receiving a symbol?

- *splits* into *multiple copies* of itself and follows all the possibilities in parallel
  - If there are subsequent choices, the machine splits again.
  - If there is no choice, the copy dies
  - if *any one* is in an *accept state*, the NFA accepts the input string

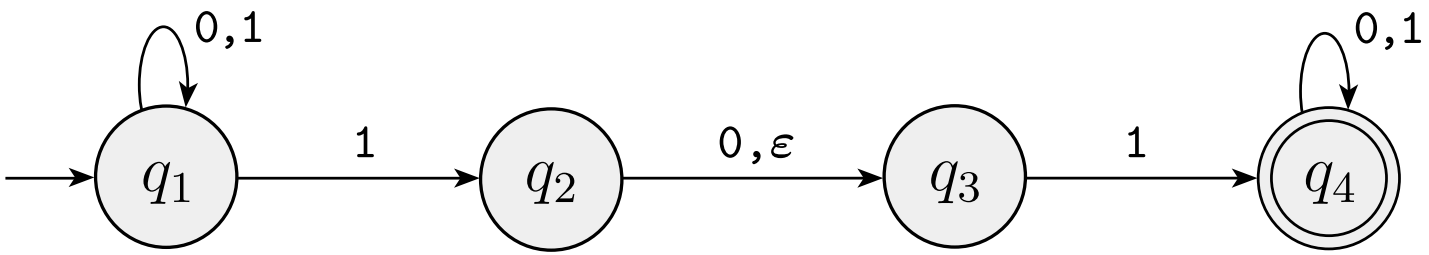


Observations on an NFA:

- In an NFA, a state may have *zero*, one, or *many* exiting arrows for *each alphabet symbol*
- This NFA has an arrow with the label ε

问: How does an NFA compute, on receiving a symbol?

- *splits* into *multiple copies* of itself and follows all the possibilities in parallel
  - If there are subsequent choices, the machine splits again.
  - If there is no choice, the copy dies
  - if *any one* is in an *accept state*, the NFA accepts the input string

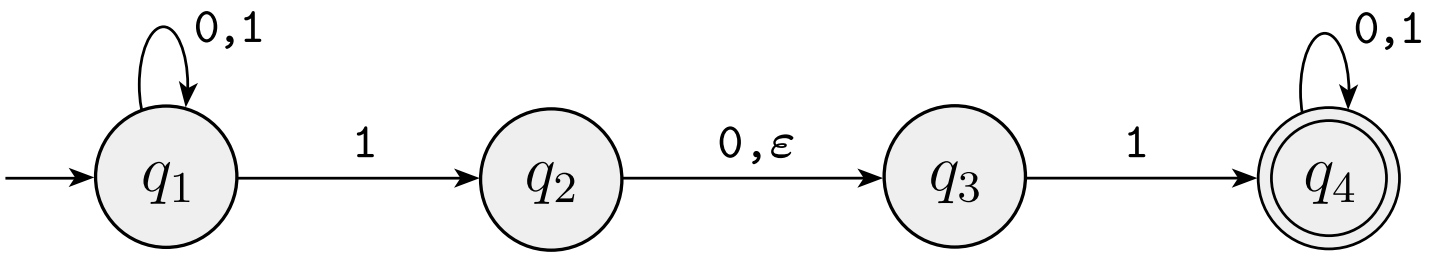


Observations on an NFA:

- In an NFA, a state may have *zero*, one, or *many* exiting arrows for *each alphabet symbol*
- This NFA has an arrow with the label ε

问: How does an NFA compute, on receiving a symbol?

- *splits* into *multiple copies* of itself and follows all the possibilities in parallel
  - If there are subsequent choices, the machine splits again.
  - If there is no choice, the copy dies
  - if *any one* is in an *accept state*, the NFA accepts the input string

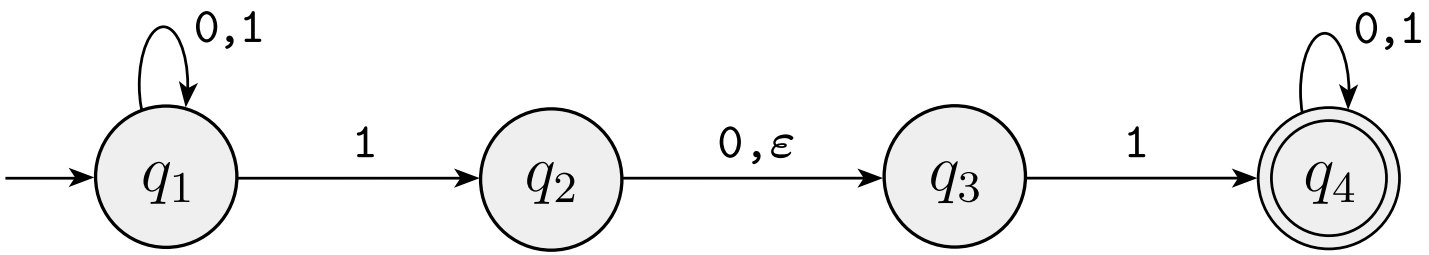


Observations on an NFA:

- In an NFA, a state may have *zero*, one, or *many* exiting arrows for *each alphabet symbol*
- This NFA has an arrow with the label ε

问: How does an NFA compute, on receiving a symbol?

- *splits* into *multiple copies* of itself and follows all the possibilities in parallel
  - If there are subsequent choices, the machine splits again.
  - If there is no choice, the copy dies
  - if *any one* is in an *accept state*, the NFA accepts the input string

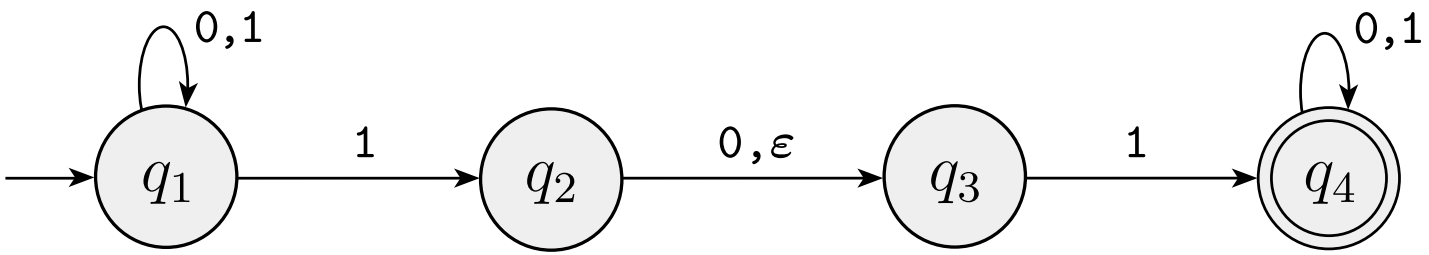


Observations on an NFA:

- In an NFA, a state may have *zero*, one, or *many* exiting arrows for *each alphabet symbol*
- This NFA has an arrow with the label ε

问: How does an NFA compute, on receiving a symbol?

- *splits* into *multiple copies* of itself and follows all the possibilities in parallel
  - If there are subsequent choices, the machine splits again.
  - If there is no choice, the copy dies
  - if *any one* is in an *accept state*, the NFA accepts the input string



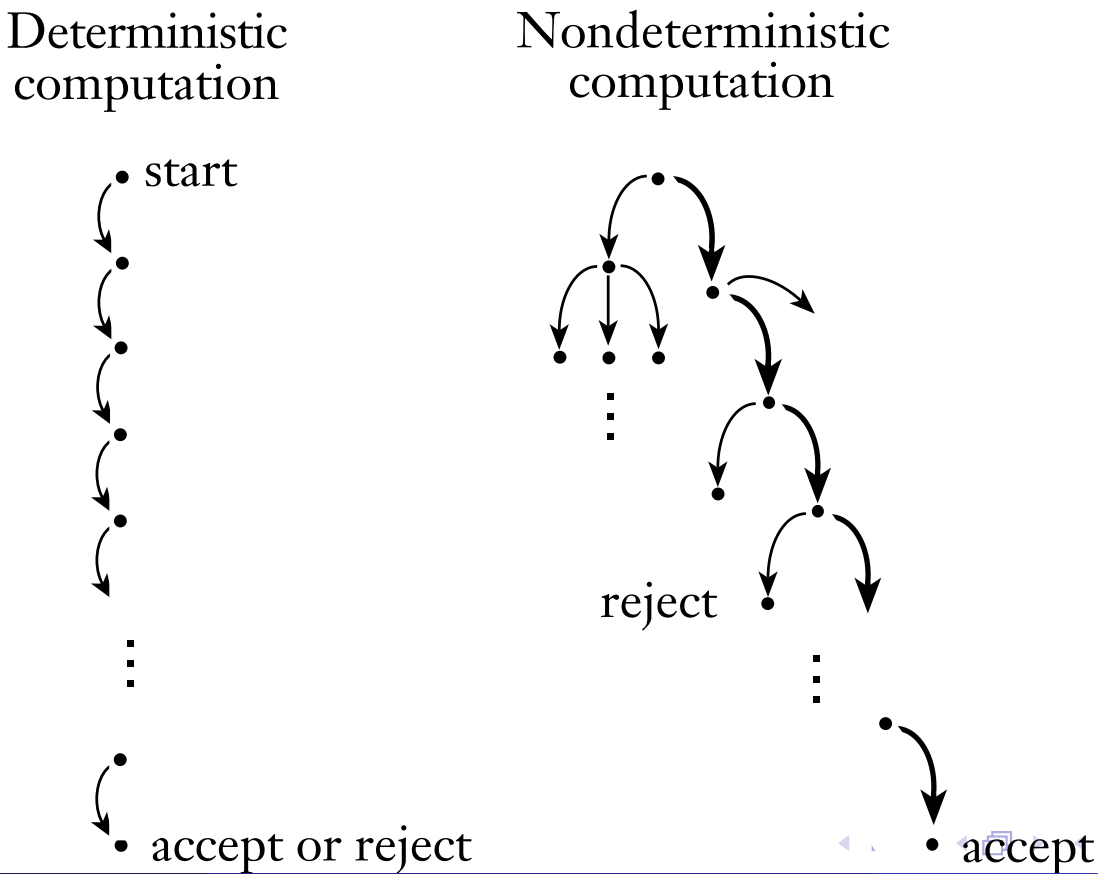
Observations on an NFA:

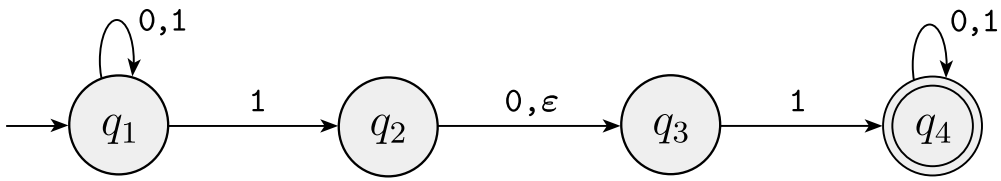
- In an NFA, a state may have *zero*, one, or *many* exiting arrows for *each alphabet symbol*
- This NFA has an arrow with the label ε

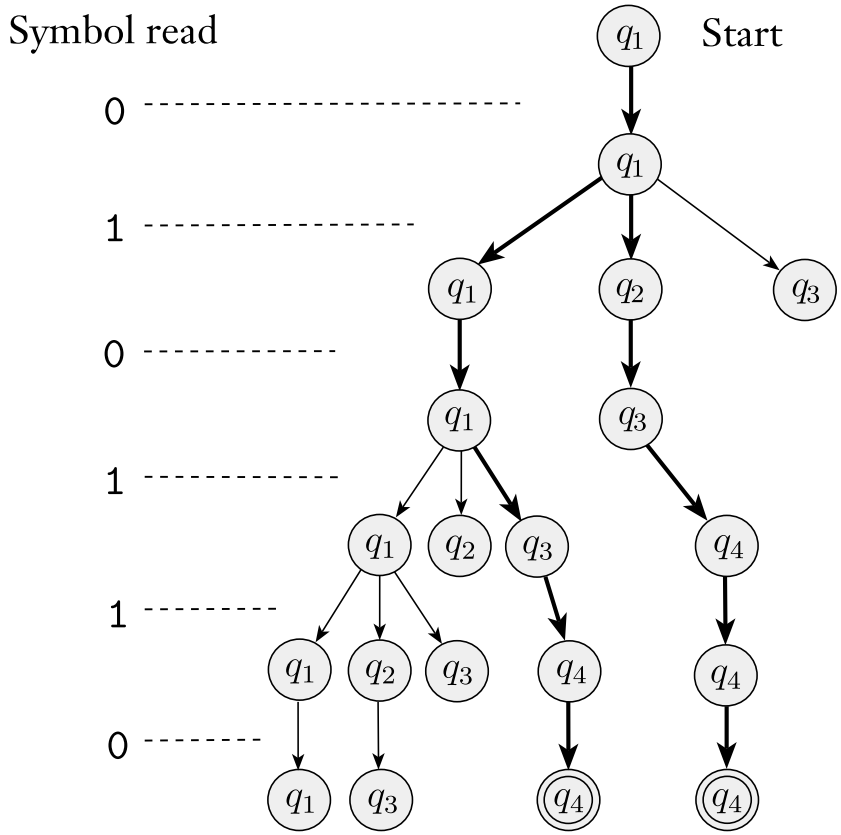
问: How does an NFA compute, on receiving a symbol?

- *splits* into *multiple copies* of itself and follows all the possibilities in parallel
  - If there are subsequent choices, the machine splits again.
  - If there is no choice, the copy dies
  - if *any one* is in an *accept state*, the NFA accepts the input string

问: What can *Nondeterminism* be view as? (注: 不仅仅指 NFA)
答 1: a kind of *parallel* computation wherein multiple independent "processes" or "threads" can be running concurrently.
答 2: Nondeterminism may be viewed as a tree of possibilities.

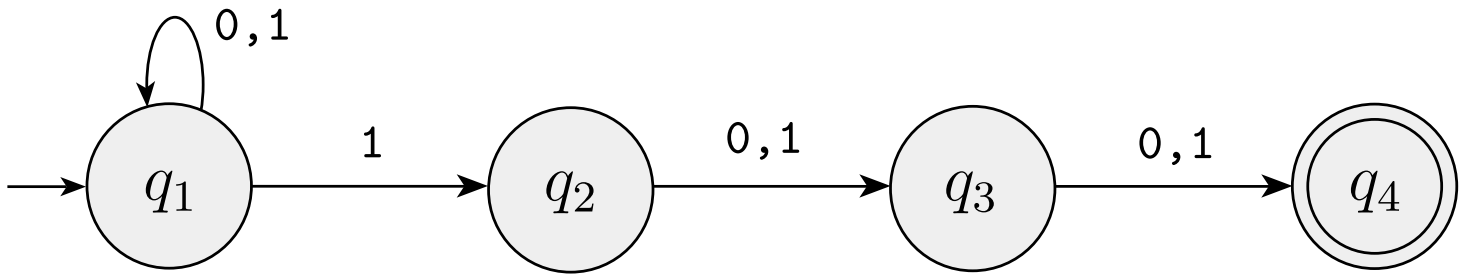



0,1
1
0,ε
1
0,1
q_1
q_2
q_3
q_4

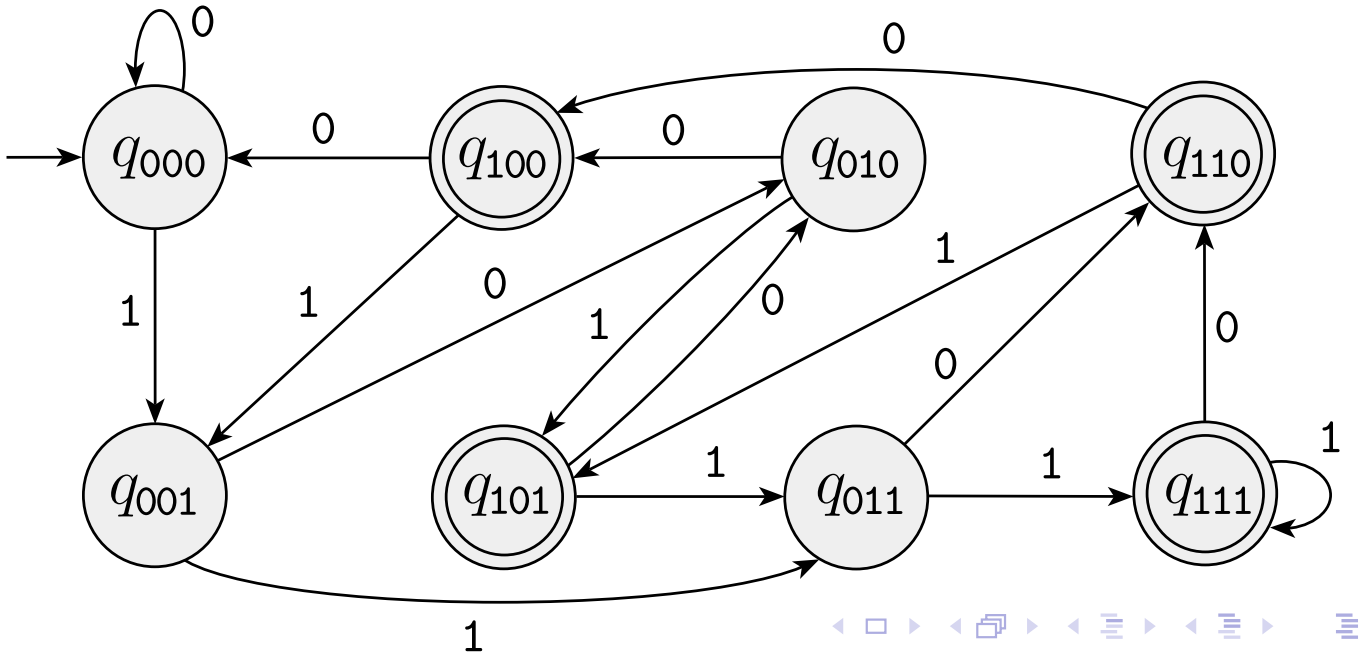

Symbol read
Start
0
1
0
1
1
0
q_1
q_2
q_3
q_4

例: NFA N_2: Let A be the language consisting of all strings over $\{0,1\}$ containing a 1 in the *third position from the end* (e.g., 000100 is in A but 0011 is not).

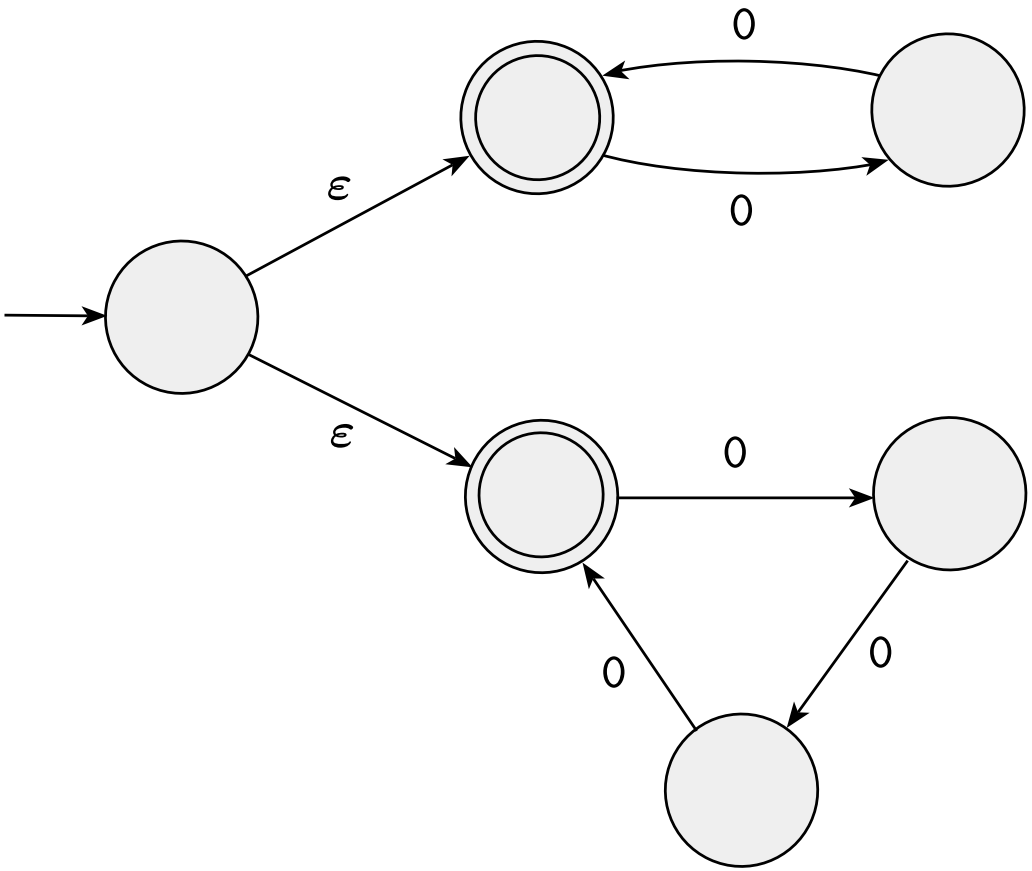
An NFA N_2 recognizing A:



A DFA recognizing A:



例: NFA N_3: For $\Sigma = \{0\}$, it accepts all strings of the form 0^k where k is a *multiple of 2 or 3*.



回顾 Sec1: 定义: Power Set

The *power set* of A is the set of all subsets of A. If A is the set $\{0, 1\}$,

- the power set of A, named as $\mathcal{P}(A)$, is the set $\{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

定义: Nondeterminitic Finite Automaton, NFA

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. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the *transition function*, ▸ DFA
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

Let's focus on δ (见下页)

回顾 Sec1: 定义: Power Set

The *power set* of A is the set of all subsets of A. If A is the set $\{0, 1\}$,

- the power set of A, named as $\mathcal{P}(A)$, is the set $\{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

定义: Nondeterminitic Finite Automaton, NFA

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. $\delta : Q \times \mathbf{\Sigma}_\varepsilon \to \mathcal{P}(Q)$ is the *transition function*, ▸ DFA
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

Let's focus on δ (见下页)

回顾 Sec1: 定义: Power Set

The *power set* of A is the set of all subsets of A. If A is the set $\{0, 1\}$,

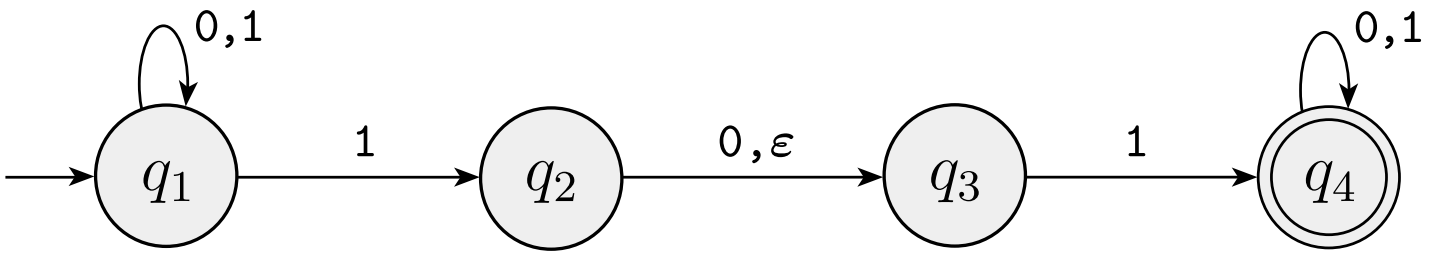
- the power set of A, named as $\mathcal{P}(A)$, is the set $\{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

定义: Nondeterminitic Finite Automaton, NFA

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. $\delta : Q \times \mathbf{\Sigma}_\varepsilon \to \mathcal{P}(Q)$ is the *transition function*, ▸ DFA
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the set of *accept states*.

Let's focus on δ (见下页)



δ is given as

	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	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

问题: *Equivalence* of NFAs and DFAs? See: Why studying this?

定义: Equivalent

Two machines are *equivalent* if they recognize the same language

定理: Equivalence of NFAs and DFAs

Every nondeterministic finite automaton (*NFA*) *has* an *equivalent* deterministic finite automaton (*DFA*).

证明思路: (Proof by Construction)

Convert the *NFA* into an equivalent *DFA* that *simulates* the *NFA*.

问题: *Equivalence* of NFAs and DFAs? See: Why studying this?

定义: Equivalent

Two machines are *equivalent* if they recognize the same language

定理: Equivalence of NFAs and DFAs

Every nondeterministic finite automaton (*NFA*) *has* an *equivalent* deterministic finite automaton (*DFA*).

证明思路: (Proof by Construction)

Convert the *NFA* into an equivalent *DFA* that *simulates* the *NFA*.

问题: *Equivalence* of NFAs and DFAs? See: Why studying this?

定义: Equivalent

Two machines are *equivalent* if they recognize the same language

定理: Equivalence of NFAs and DFAs

Every nondeterministic finite automaton (*NFA*) *has* an *equivalent* deterministic finite automaton (*DFA*).

证明思路: (Proof by Construction)

Convert the *NFA* into an equivalent *DFA* that *simulates* the *NFA*.

问题: *Equivalence* of NFAs and DFAs? See: Why studying this?

定义: Equivalent

Two machines are *equivalent* if they recognize the same language

定理: Equivalence of NFAs and DFAs

Every nondeterministic finite automaton (*NFA*) *has* an *equivalent* deterministic finite automaton (*DFA*).

证明思路: (Proof by Construction)

Convert the *NFA* into an equivalent *DFA* that *simulates* the *NFA*.

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Case 2: N has ε arrows

- For any state R of M, define $E(R) = \{q \mid q \text{ can be reached from } R$ by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Case 2: N has ε arrows

- For any state R of M, define $E(R) = \{q \mid q \text{ can be reached from } R$ by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Case 2: N has ε arrows

- For any state R of M, define $E(R) = \{q \mid q \text{ can be reached from } R$ by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Case 2: N has ε arrows

- For any state R of M, define $E(R) = \{q \mid q$ can be reached from R by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Case 2: N has ε arrows

- For any state R of M, define $E(R) = \{q \mid q \text{ can be reached from } R$ by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Case 2: N has ε arrows

- For any state R of M, define $E(R) = \{q \mid q$ can be reached from R by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Case 2: N has ε arrows

- For any state R of M, define $E(R) = \{q \mid q \text{ can be reached from } R$ by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Case 2: N has ε arrows

- For any state R of M, define $E(R) = \{q \mid q \text{ can be reached from } R$ by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

Case 2: N has ε arrows

- For any state R of M, define $E(R) = \{q \mid q$ can be reached from R by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

证明: Equivalence of NFAs and DFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.
We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A ▸ Example

Case 1: N has no ε arrows

1. $Q' = \mathcal{P}(Q)$
2. $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$

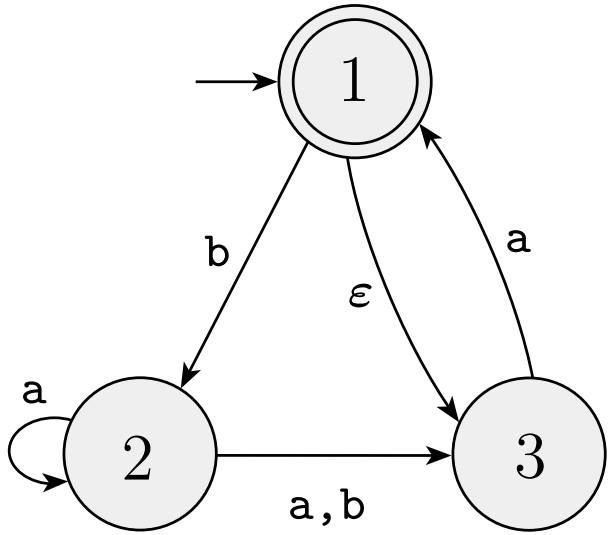
Case 2: N has ε arrows

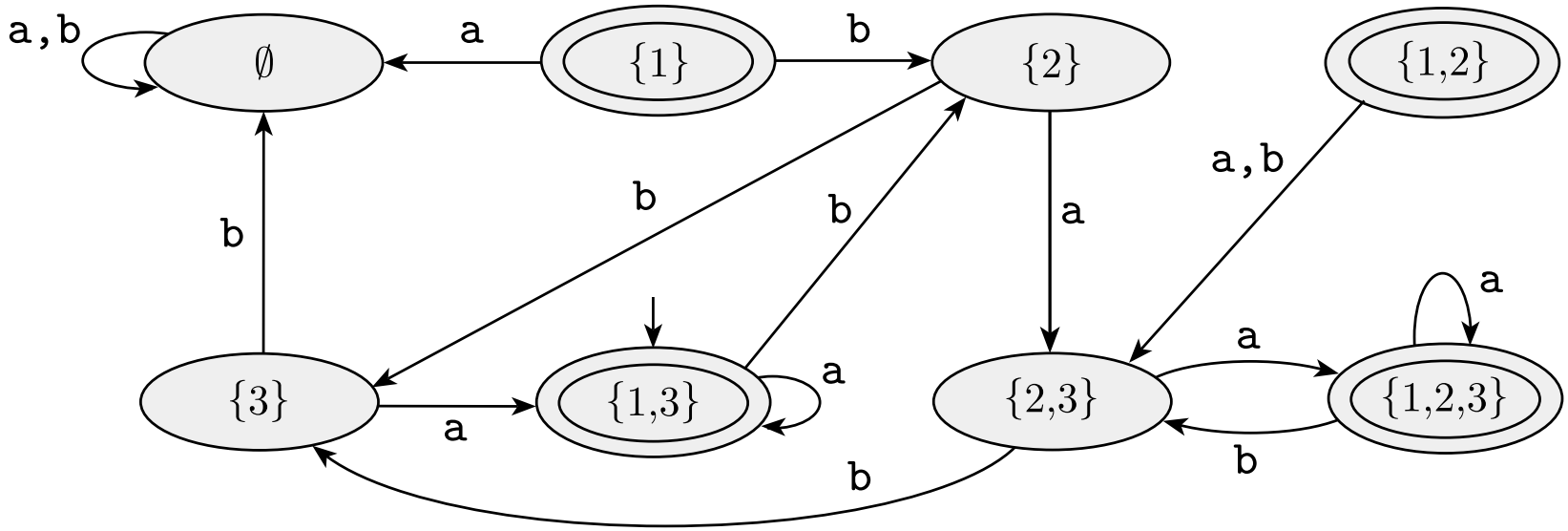
- For any state R of M, define $E(R) = \{q \mid q$ can be reached from R by traveling along *0 or more* ε arrows$\}$
- $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for some r} \in R\}$
- $q_0' = E(\{q_0\})$

推论

A language is regular if and only if some *nondeterministic* finite automaton recognizes it. See Def. Regular

- 例: From NFA to DFA



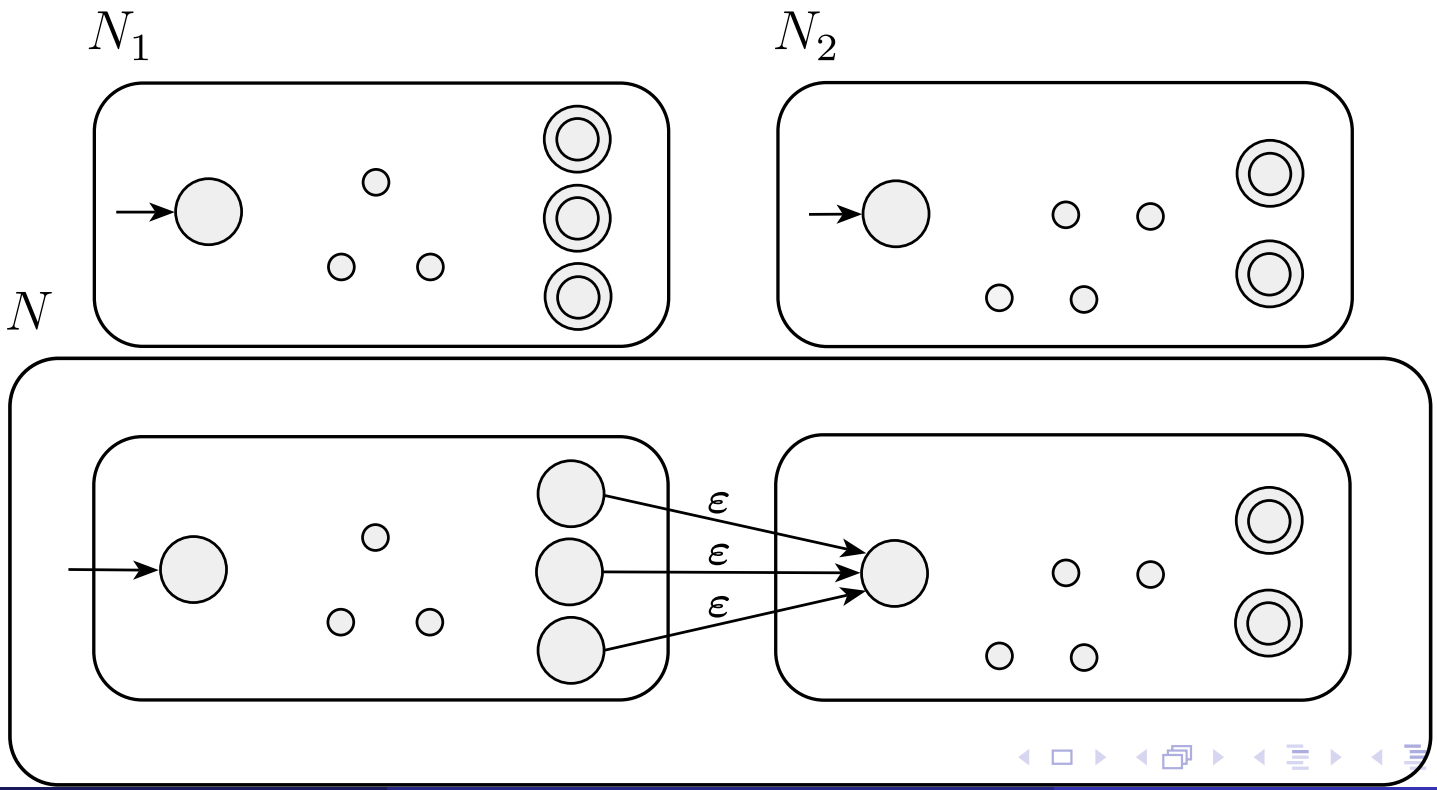


问题: Prove closure under the regular operations. See: Why studying this?

定理: Closure under Concatenation Operation

The class of regular languages is closed under the *concatenation* operation.

- In other words, if A_1 and A_2 are regular languages, so is $A_1 \circ A_2$



证明: (Proof by Construction)

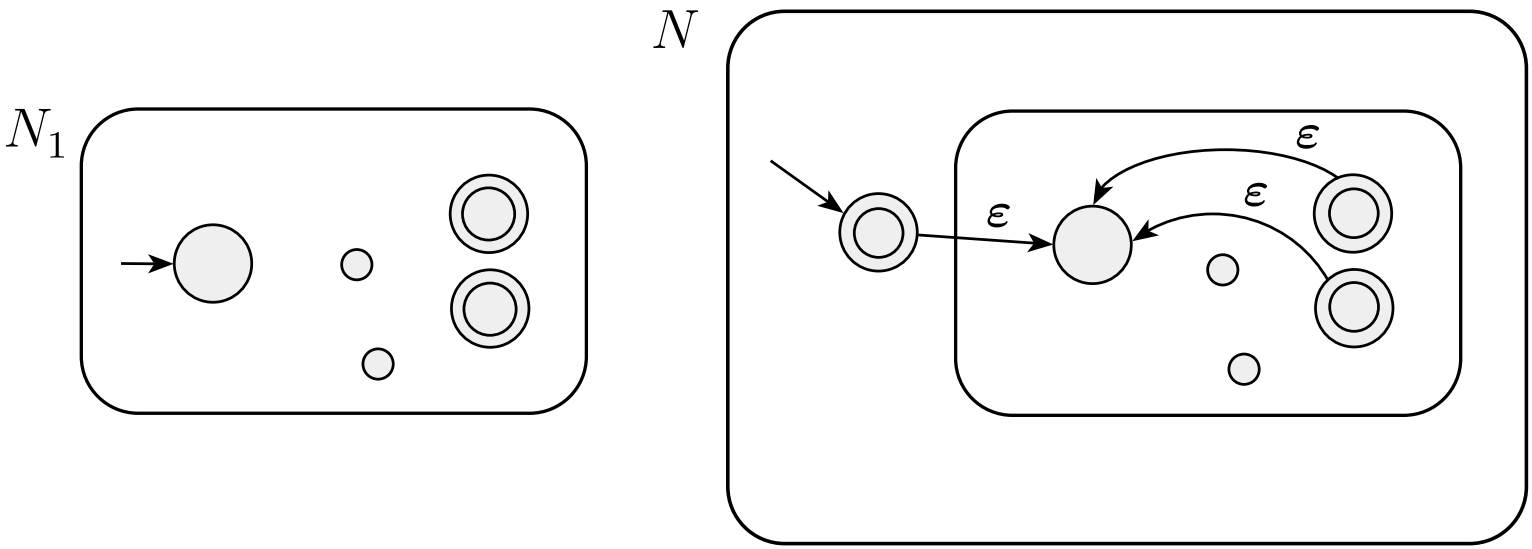
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1,
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2
Construct $N = (Q, \Sigma, \delta, q_1, F_2)$

- $Q = Q_1 \cup Q_2$
- Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

定理: Closure under Star Operation

The class of regular languages is closed under the *star* operation.



证明

略

总结:

定理: Closure under Regular Operation

The class of regular languages is closed under the regular operations.

- union
- star
- concatenation

问题: What can *regular operations* and *their theorems* be used for?

答: Regular Expressions (见下页), and See Lemma 1 for Equivalence

总结:

定理: Closure under Regular Operation

The class of regular languages is closed under the regular operations.

- union
- star
- concatenation

问题: What can *regular operations* and *their theorems* be used for?
答: Regular Expressions (见下页), and See Lemma 1 for Equivalence

Outline

问题: Relations to *regular expressions* (正则表达式)

- 问题 1: What are regular expressions
- 问题 2: Property: Equivalence to Finite automata

问题 1: What are regular expressions
答: We can use the *regular operations* to build up expressions describing *languages*, which are called *regular expressions*

Once the syntax of a programming language has been described with a *regular expression* in terms of its tokens, automatic systems can generate the *lexical analyzer*

问题: Relations to *regular expressions* (正则表达式)

- 问题 1: What are regular expressions
- 问题 2: Property: Equivalence to Finite automata

问题 1: What are regular expressions
答: We can use the *regular operations* to build up expressions describing *languages*, which are called *regular expressions*

Once the syntax of a programming language has been described with a *regular expression* in terms of its tokens, automatic systems can generate the *lexical analyzer*

问题: Relations to *regular expressions* (正则表达式)

- 问题 1: What are regular expressions
- 问题 2: Property: Equivalence to Finite automata

问题 1: What are regular expressions
答: We can use the *regular operations* to build up expressions describing *languages*, which are called *regular expressions*

Once the syntax of a programming language has been described with a *regular expression* in terms of its tokens, automatic systems can generate the *lexical analyzer*

问题 1: What are regular expressions

例 1 (解释) :

$$(0 \cup 1)0^*$$

- The symbols 0 and 1 are shorthand for the sets $\{0\}$ and $\{1\}$
- $(0 \cup 1)$ means $(\{0\} \cup \{1\})$
- The part 0^* means $\{0\}^*$
- $(0 \cup 1)0^*$ is shorthand for $(0 \cup 1) \circ 0^*$

问题 1: What are regular expressions

例 2（分析）：

$$(0 \cup 1)^*$$

The *value* of this expression is the *language* consisting of all possible strings of 0s and 1s

- If $\Sigma = \{0, 1\}$, Σ can be shorthand for regular expression $(0 \cup 1)$
- Σ^* describes the language consisting of all strings over that alphabet

正式解答: Formal Definition（见下页）

问题 1: What are regular expressions
例 2（分析）：

$$(0 \cup 1)^*$$

The *value* of this expression is the *language* consisting of all possible strings of 0s and 1s

- If $\Sigma = \{0, 1\}$, Σ can be shorthand for regular expression $(0 \cup 1)$
- Σ^* describes the language consisting of all strings over that alphabet

正式解答: Formal Definition（见下页）

问题 1: What are regular expressions
例 2（分析）：

$$(0 \cup 1)^*$$

The *value* of this expression is the *language* consisting of all possible strings of 0s and 1s

- If $\Sigma = \{0, 1\}$, Σ can be shorthand for regular expression $(0 \cup 1)$
- Σ^* describes the language consisting of all strings over that alphabet

正式解答: Formal Definition（见下页）

问题 1: What are regular expressions

定义: Regular Expressions

Say that R is a regular expression if R is

1. $\{a\}$ for some a in the alphabet Σ,
2. $\{\varepsilon\}$
3. $\emptyset$
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions
6. (R_1^*), where R_1 is a regular expression

问题 1: What are regular expressions

例（Language）：

In the following instances, we assume that the alphabet Σ is $\{0, 1\}$

- $0^*10^* = \{w \mid w \text{ contains a single } 1\}$
- $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$
- $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \{\varepsilon\}$

问题 1: What are regular expressions

例（Language）：

In the following instances, we assume that the alphabet Σ is $\{0, 1\}$

- $0^*10^* = \{w \mid w \text{ contains a single } 1\}$
- $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$
- $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \{\varepsilon\}$

问题 1: What are regular expressions

例（Language）：

In the following instances, we assume that the alphabet Σ is $\{0, 1\}$

- $0^*10^* = \{w \mid w \text{ contains a single } 1\}$
- $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$
- $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \{\varepsilon\}$

问题 1: What are regular expressions

例（Language）：

In the following instances, we assume that the alphabet Σ is $\{0, 1\}$

- $0^*10^* = \{w \mid w \text{ contains a single } 1\}$
- $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$
- $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \{\varepsilon\}$

问题 1: What are regular expressions

例（Language）：

In the following instances, we assume that the alphabet Σ is $\{0, 1\}$

- $0^*10^* = \{w \mid w \text{ contains a single } 1\}$
- $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$
- $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \{\varepsilon\}$

问题 1: What are regular expressions

例（Language）：

In the following instances, we assume that the alphabet Σ is $\{0, 1\}$

- $0^*10^* = \{w \mid w \text{ contains a single } 1\}$
- $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$
- $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \{\varepsilon\}$

问题 1: What are regular expressions

例（Language）：

In the following instances, we assume that the alphabet Σ is $\{0, 1\}$

- $0^*10^* = \{w \mid w \text{ contains a single } 1\}$
- $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$
- $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$
- $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \{\varepsilon\}$

问题 2: Property: Equivalence to Finite automata

引理 1

If a language is described by a regular expression, then it is regular

证明: (Proof by construction)

- *Case 1*: $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$
- *Case 2*: $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$
- *Case 3*: $R = \emptyset$. Then $L(R) = \emptyset$
- *Case 4*: $R = R_1 \cup R_2$; *Case 5*: $R = R_1 \circ R_2$; *Case 6*: $R = R_1^*$

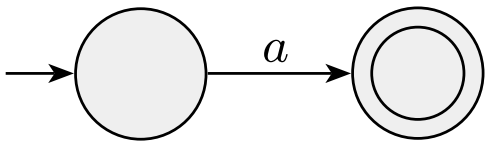
问题 2: Property: Equivalence to Finite automata

引理 1

If a language is described by a regular expression, then it is regular

证明: (Proof by construction)

- *Case 1*: $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$.
- *Case 2*: $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$
- *Case 3*: $R = \emptyset$. Then $L(R) = \emptyset$
- *Case 4*: $R = R_1 \cup R_2$; *Case 5*: $R = R_1 \circ R_2$; *Case 6*: $R = R_1^*$



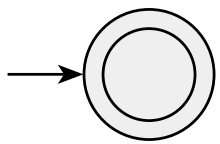
问题 2: Property: Equivalence to Finite automata

引理 1

If a language is described by a regular expression, then it is regular

证明: (Proof by construction)

- *Case 1*: $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$
- *Case 2*: $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$, and the following NFA recognizes $L(R)$.
- *Case 3*: $R = \emptyset$. Then $L(R) = \emptyset$
- *Case 4*: $R = R_1 \cup R_2$; *Case 5*: $R = R_1 \circ R_2$; *Case 6*: $R = R_1^*$



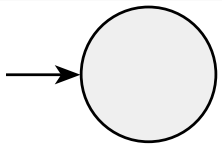
问题 2: Property: Equivalence to Finite automata

引理 1

If a language is described by a regular expression, then it is regular

证明: (Proof by construction)

- *Case 1*: $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$
- *Case 2*: $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$
- *Case 3*: $R = \emptyset$. Then $L(R) = \emptyset$, and the following NFA recognizes L(R)
- *Case 4*: $R = R_1 \cup R_2$; *Case 5*: $R = R_1 \circ R_2$; *Case 6*: $R = R_1^*$



问题 2: Property: Equivalence to Finite automata

引理 1

If a language is described by a regular expression, then it is regular

证明: (Proof by construction)

- *Case 1*: $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$
- *Case 2*: $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$
- *Case 3*: $R = \emptyset$. Then $L(R) = \emptyset$
- *Case 4*: $R = R_1 \cup R_2$; *Case 5*: $R = R_1 \circ R_2$; *Case 6*: $R = R_1^*$, use the Proof steps in ▸ Closure Theorems

问题 2: Property: Equivalence to Finite automata

引理 1

If a language is described by a regular expression, then it is regular

例

Building an NFA from the regular expression $(ab \cup a)^*$ ▸ Closure Theorems

问题 2: Property: Equivalence to Finite automata

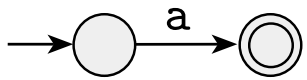
引理 1

If a language is described by a regular expression, then it is regular

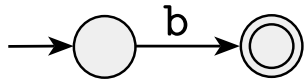
例

Building an NFA from the regular expression $(ab \cup a)^*$ ▸ Closure Theorems

a



b



问题 2: Property: Equivalence to Finite automata

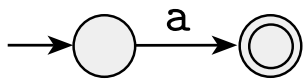
引理 1

If a language is described by a regular expression, then it is regular

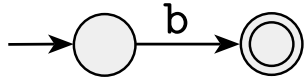
例

Building an NFA from the regular expression $(ab \cup a)^*$ ▸ Closure Theorems

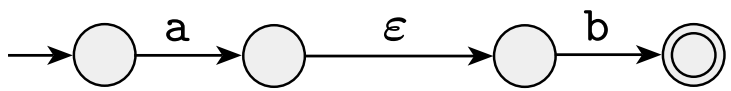
a



b



ab



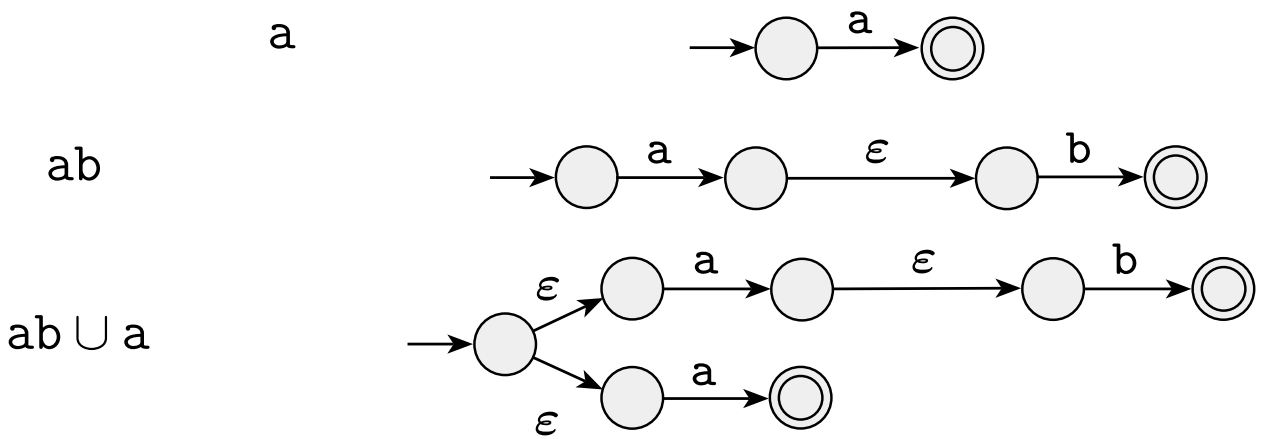
问题 2: Property: Equivalence to Finite automata

引理 1

If a language is described by a regular expression, then it is regular

例

Building an NFA from the regular expression $(ab \cup a)^*$ ▸ Closure Theorems



问题 2: Property: Equivalence to Finite automata

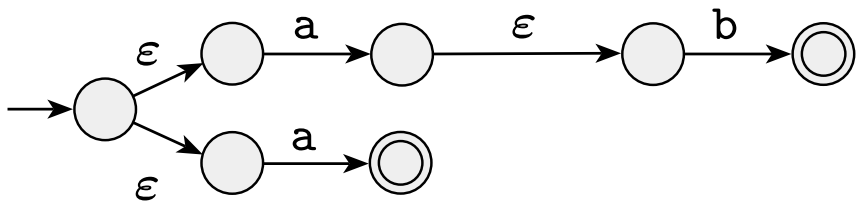
引理 1

If a language is described by a regular expression, then it is regular

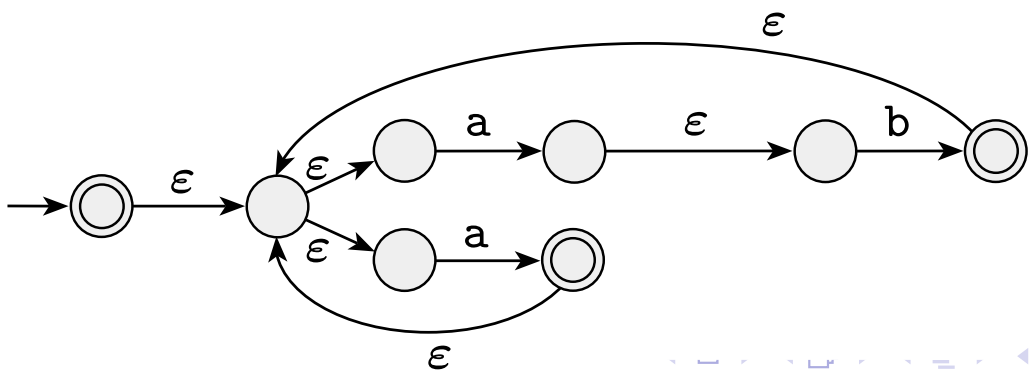
例

Building an NFA from the regular expression $(ab \cup a)^*$ ▸ Closure Theorems

ab ∪ a



(ab ∪ a)*



问题 2: Property: Equivalence to Finite automata

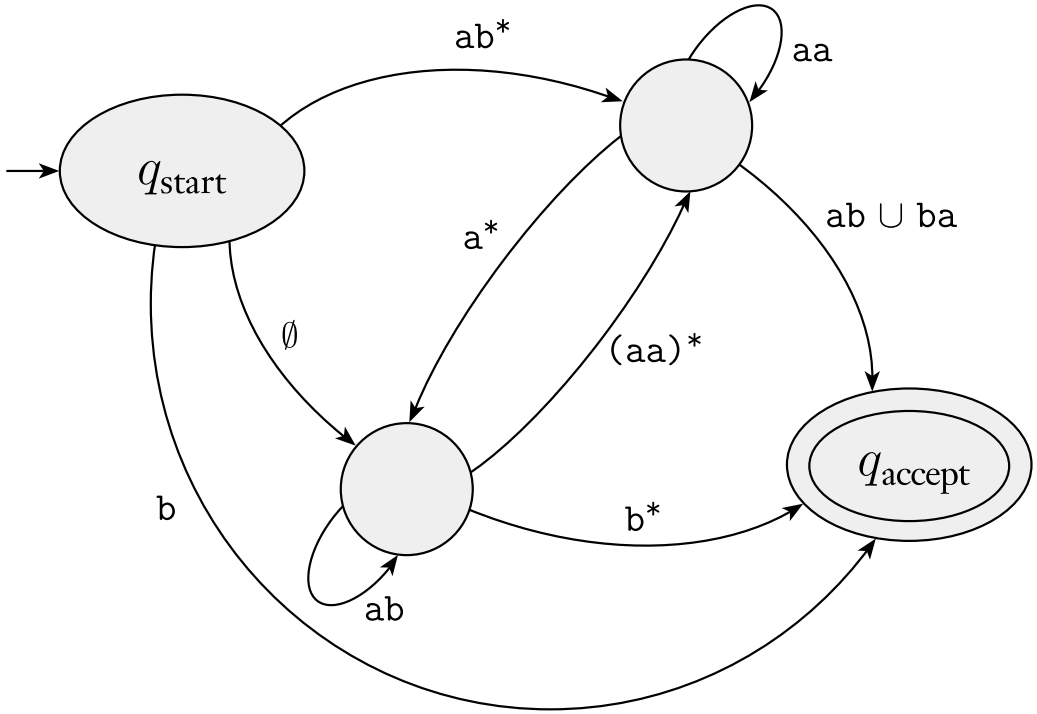
引理 2

If a language is regular, then it is described by a regular expression

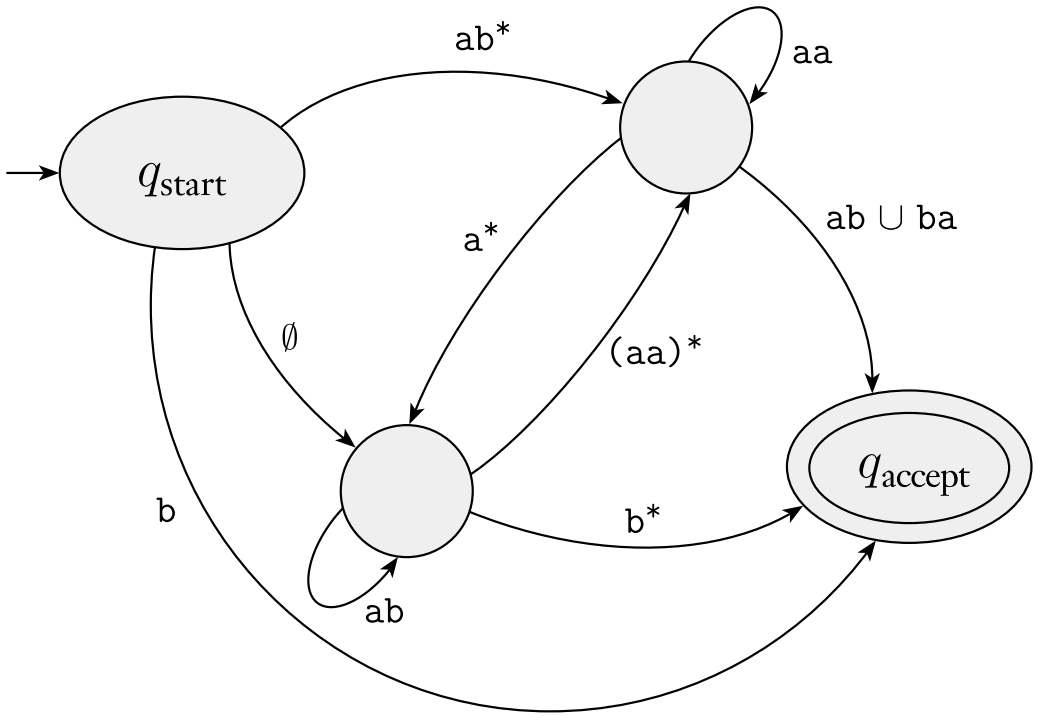
证明思路

- Step 0: Define *GNFA*, *generalized nondeterministic finite automaton*
- Step 1: Convert *DFAs* into *GNFAs*
- Step 2: Convert *GNFAs* into *regular expressions*

Lemma 2 Step 0: Define *GNFA*

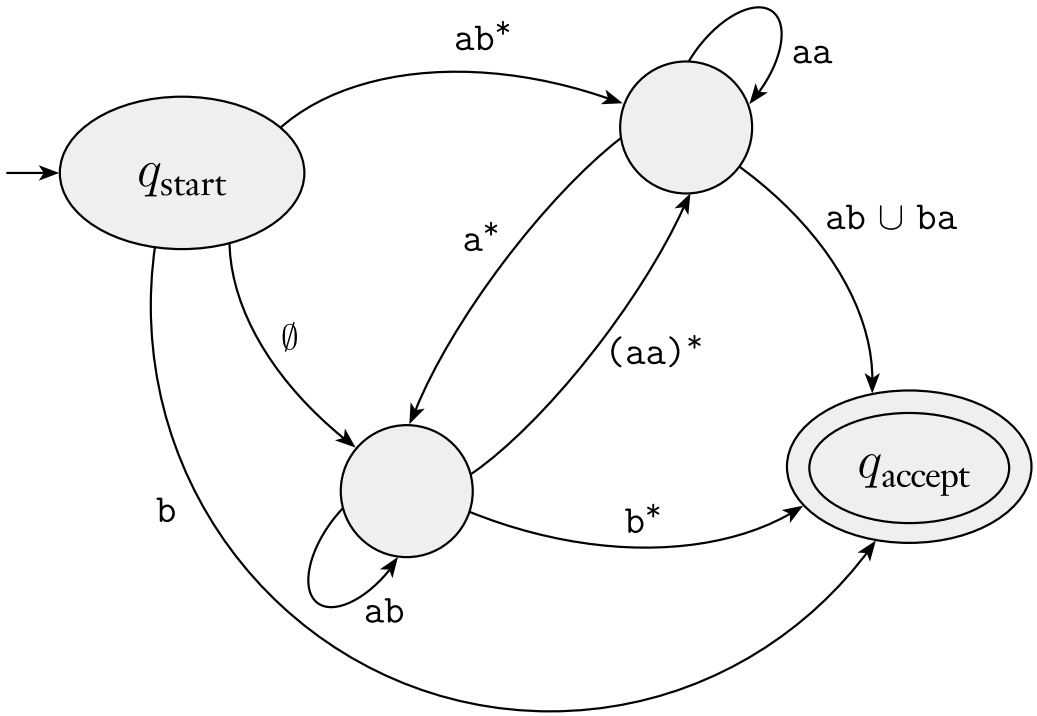


Lemma 2 Step 0: Define *GNFA*



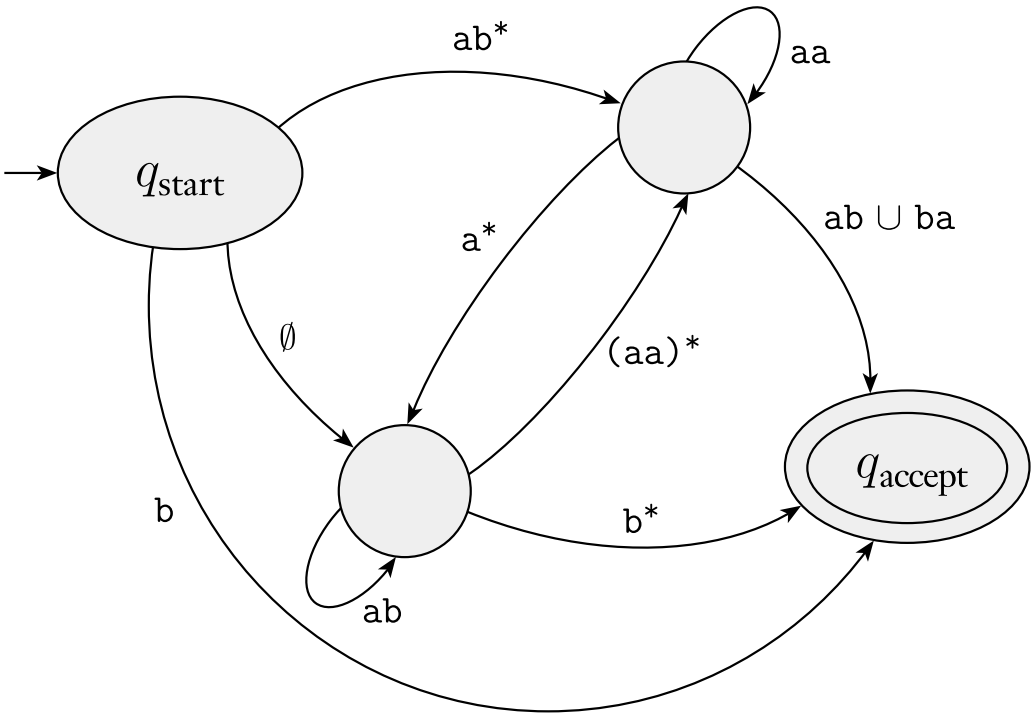
Observation 1: *Transition arrows* may have any *regular expressions* as labels, *instead of* only members of the *alphabet* or ε

Lemma 2 Step 0: Define *GNFA*



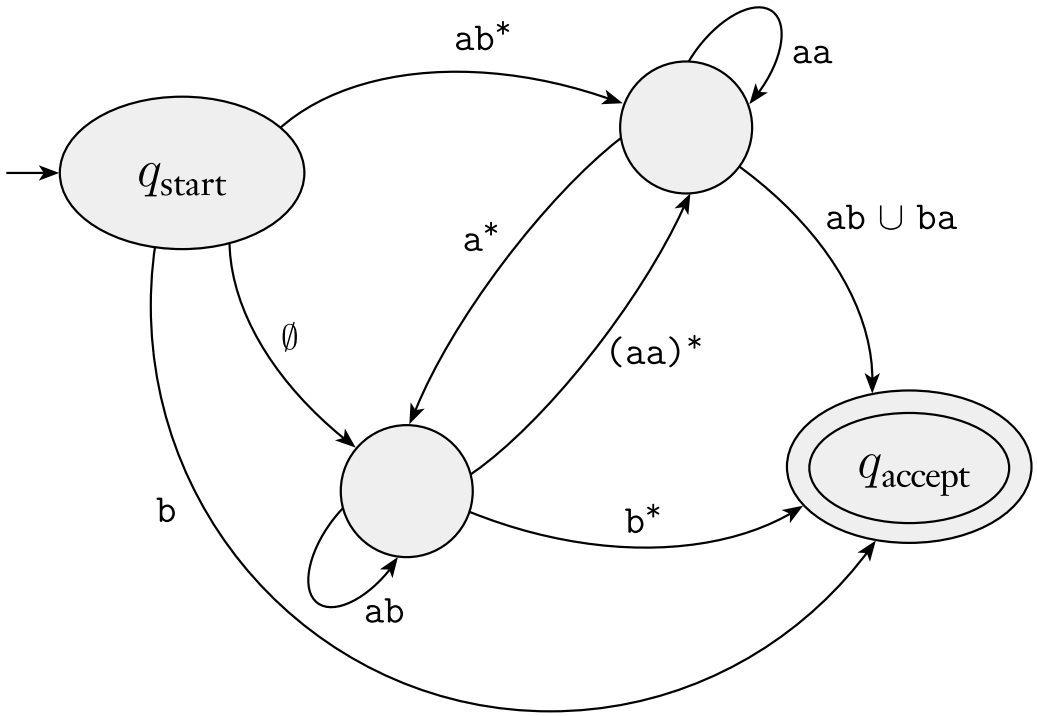
Observation 2: The *start state* has transition arrows *going to every other state*, but *no arrows* coming in *from* any other state

Lemma 2 Step 0: Define *GNFA*



Observation 3: There is only a *single accept state*, and it has arrows coming *in* from *every other state*, but *no arrows* going *to* any other state

Lemma 2 Step 0: Define *GNFA*



Observation 4: *Except* for the *start* and *accept* states, one arrow goes *from every* state *to every* other state, and also from each state *to itself*.

Lemma 2 Step 1: Convert a *DFA* into a *GNFA*

- *Add* a *new start state* with an ε arrow *to* the *old start state*
- *Add* a *new accept state* with ε arrows from the *old accept states*
- If there are *multiple arrows* going between the same two states in the *same direction*
  - replace each with a *single arrow* whose *label* is the ***union*** *of the previous labels*.
- *Add arrows* labeled $\emptyset$ between states that had *no arrows*.

Lemma 2 Step 1: Convert a *DFA* into a *GNFA*

- *Add* a *new start state* with an ε arrow *to* the *old start state*
- *Add* a *new accept state* with ε arrows from the *old accept states*
- If there are *multiple arrows* going between the same two states in the *same direction*
  - replace each with a *single arrow* whose *label* is the ***union*** *of the previous labels*.
- *Add arrows* labeled $\emptyset$ between states that had *no arrows*.

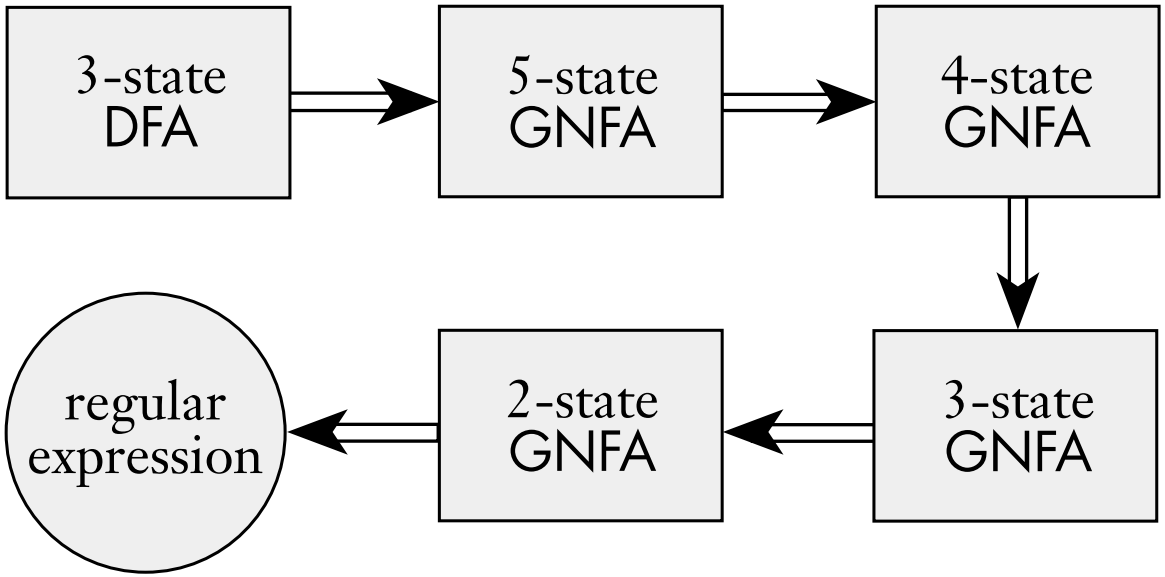
Lemma 2 Step 1: Convert a *DFA* into a *GNFA*

- *Add* a *new start state* with an ε arrow *to* the *old start state*
- *Add* a *new accept state* with ε arrows from the *old accept states*
- If there are *multiple arrows* going between the same two states in the *same direction*
  - replace each with a *single arrow* whose *label* is the ***union*** *of the previous labels*.
- *Add arrows* labeled $\emptyset$ between states that had *no arrows*.

Lemma 2 Step 1: Convert a *DFA* into a *GNFA*

- *Add* a *new start state* with an ε arrow *to* the *old start state*
- *Add* a *new accept state* with ε arrows from the *old accept states*
- If there are *multiple arrows* going between the same two states in the *same direction*
  - replace each with a *single arrow* whose *label* is the ***union*** *of the previous labels*.
- *Add arrows* labeled $\emptyset$ between states that had *no arrows*.

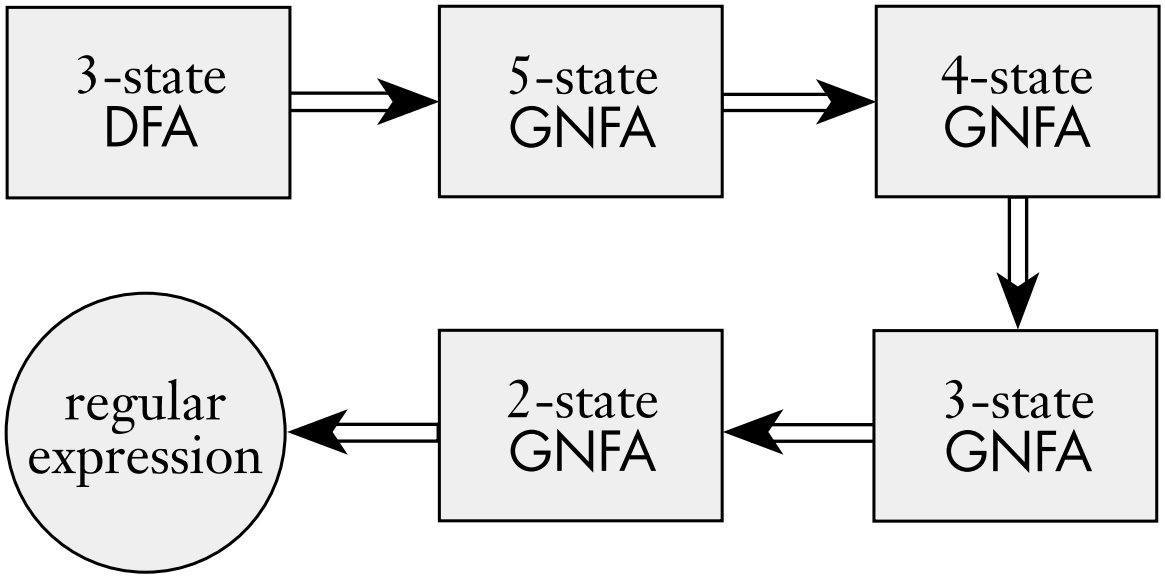
Lemma 2 Step 2: Convert *GNFAs* into *regular expressions*



Method: constructing an equivalent GNFA with one fewer state

- *selecting* a state
- *ripping* it *out* of the machine
- *repairing* the remainder so that the same language is still recognized.

Lemma 2 Step 2: Convert *GNFAs* into *regular expressions*



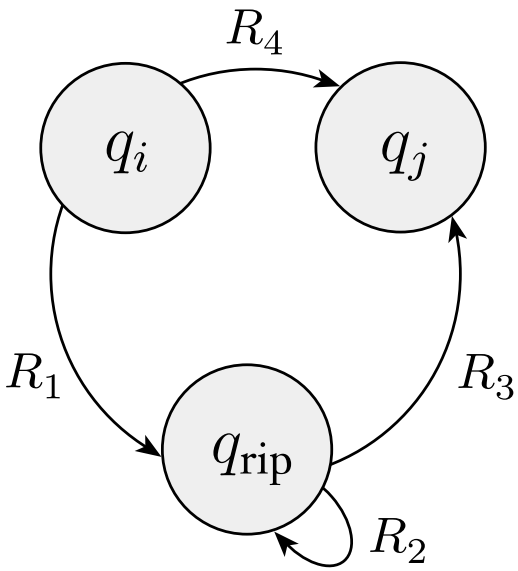
Method: constructing an equivalent GNFA with one fewer state

- *selecting* a state
- *ripping* it *out* of the machine
- *repairing* the remainder so that the same language is still recognized.

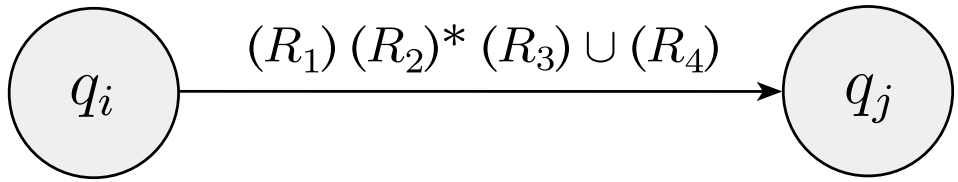
Lemma 2 Step 2: Convert *GNFAs* into *regular expressions*

Method: constructing an equivalent GNFA with one fewer state

- *selecting* a state
- *ripping* it *out* of the machine
- *repairing* the remainder so that the same language is still recognized.

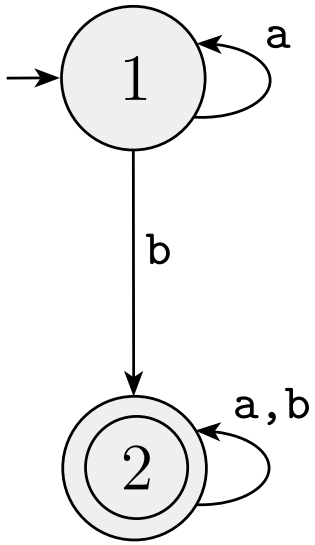


before

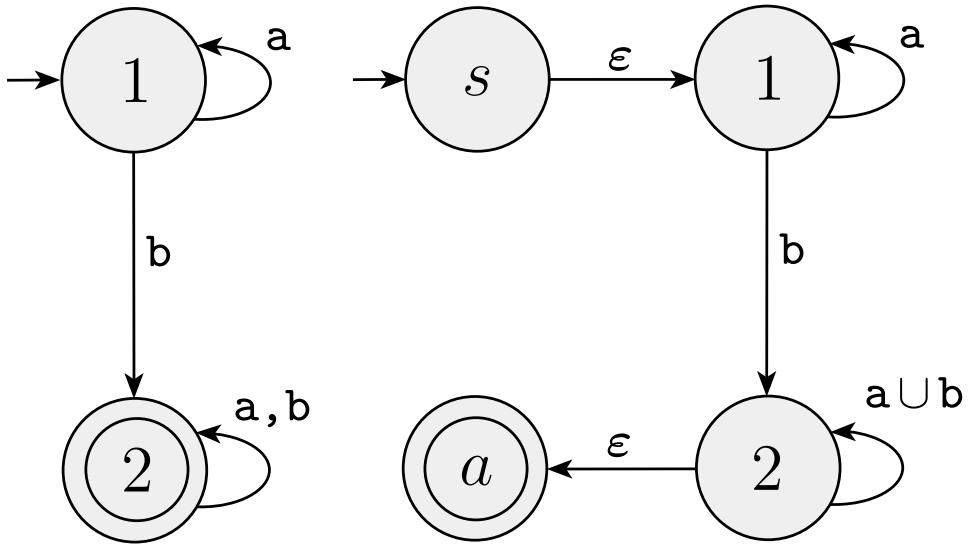


after

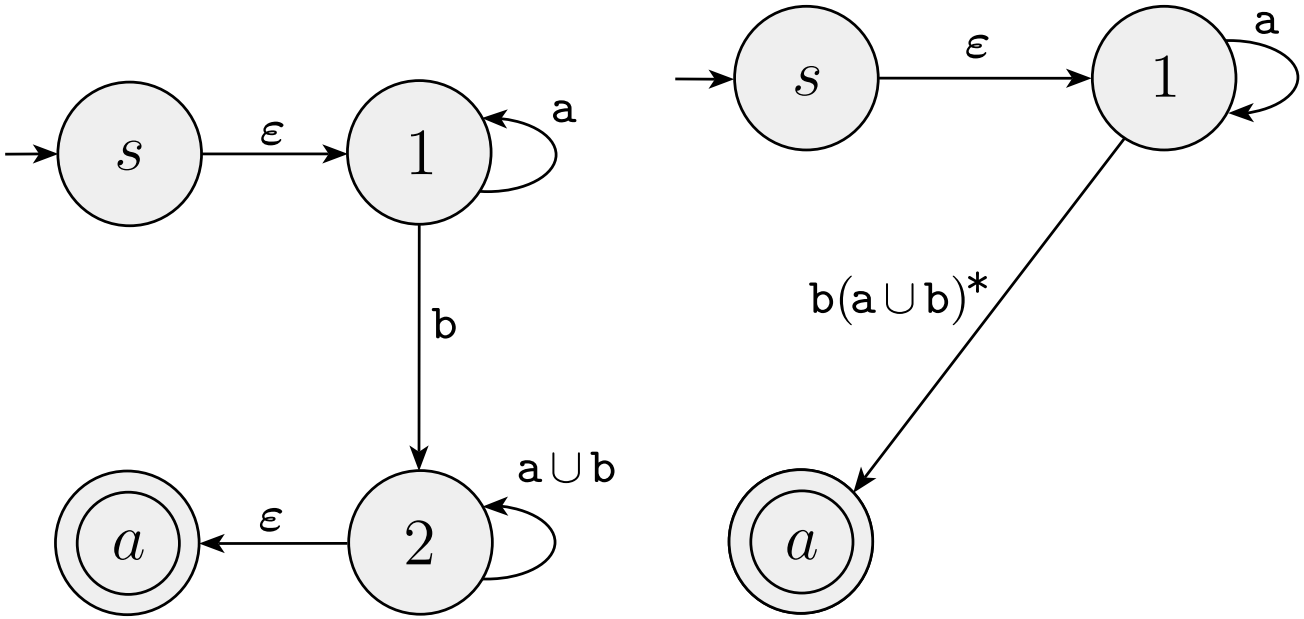
Lemma 2 例 1：From a DFA to a regular expression ▸ step 1 ▸ step 2



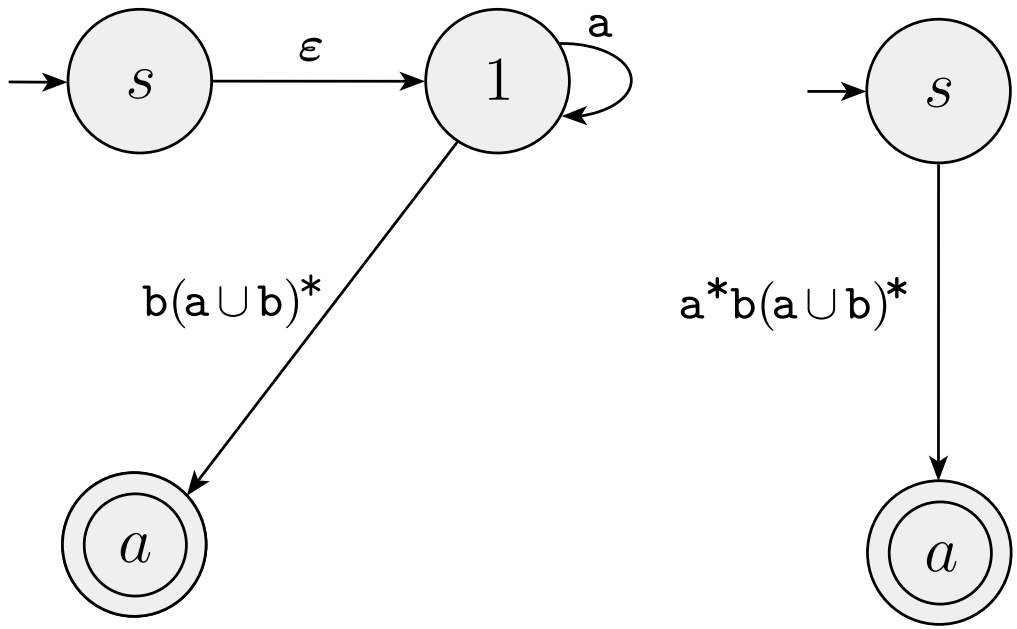
Lemma 2 例 1：From a DFA to a regular expression ▸ step 1 ▸ step 2



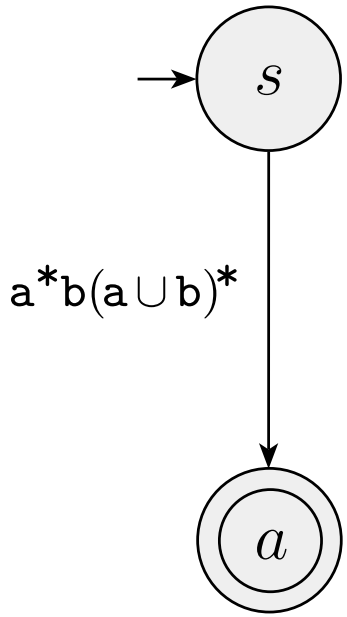
Lemma 2 例 1：From a DFA to a regular expression ▸ step 1 ▸ step 2



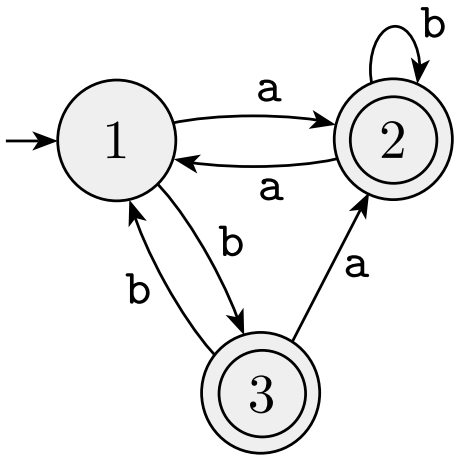
Lemma 2 例 1：From a DFA to a regular expression ▸ step 1 ▸ step 2



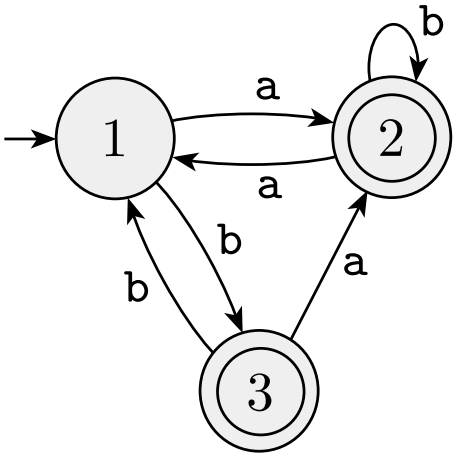
Lemma 2 例 1：From a DFA to a regular expression ▸ step 1 ▸ step 2

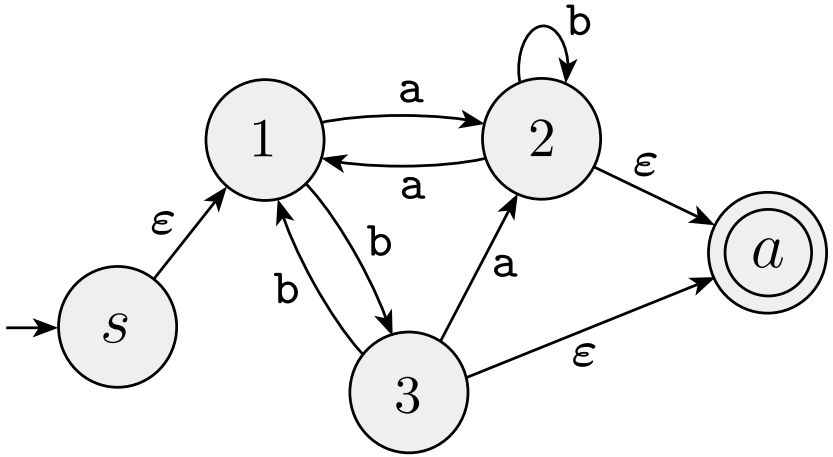


Lemma 2 例 2：From a DFA to a regular expression ▸ step 1 ▸ step 2

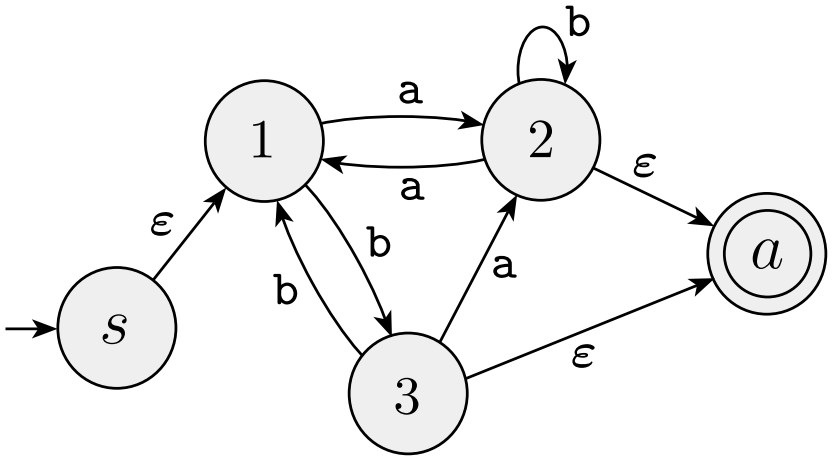


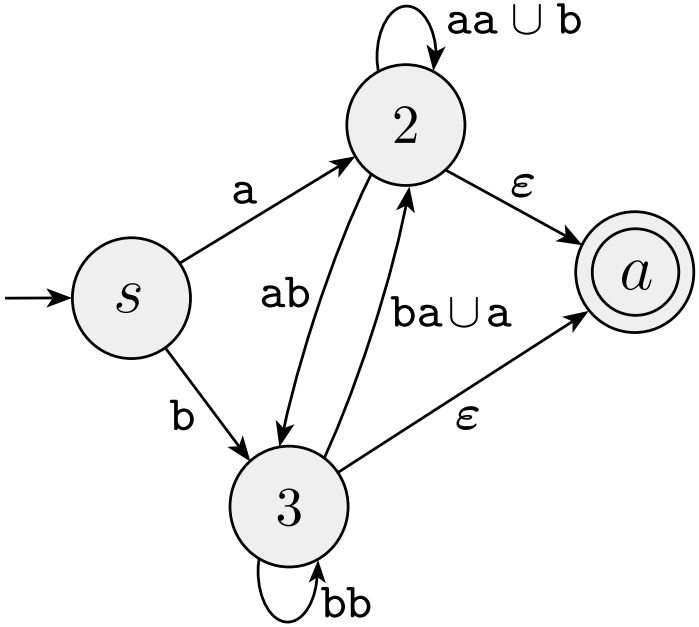
Lemma 2 例 2：From a DFA to a regular expression ▸ step 1 ▸ step 2



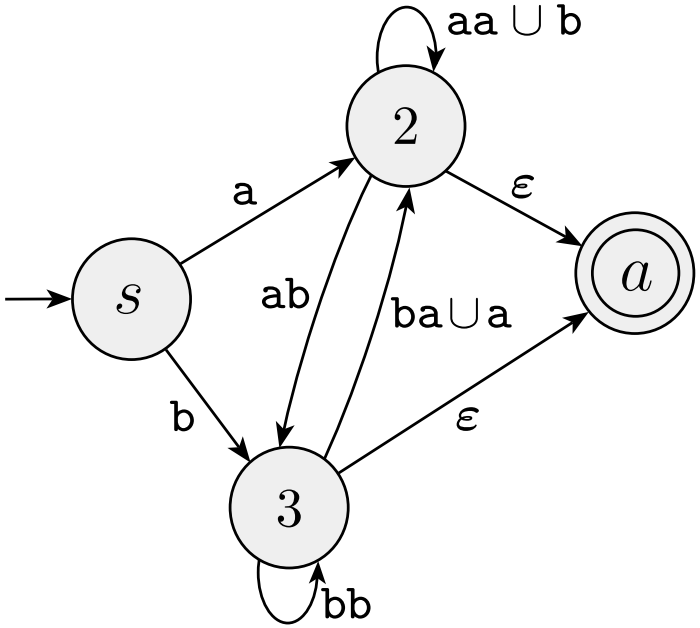


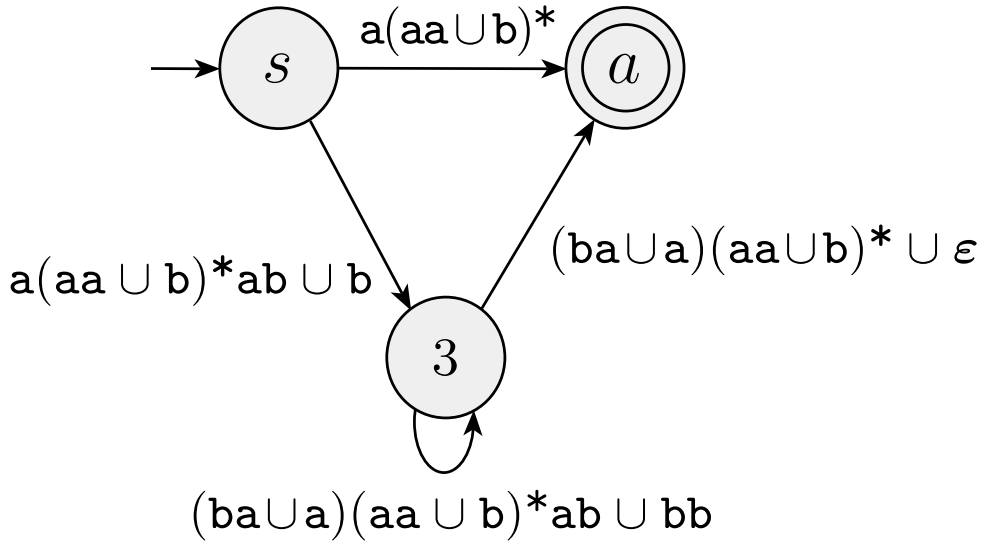
Lemma 2 例 2：From a DFA to a regular expression ▸ step 1 ▸ step 2



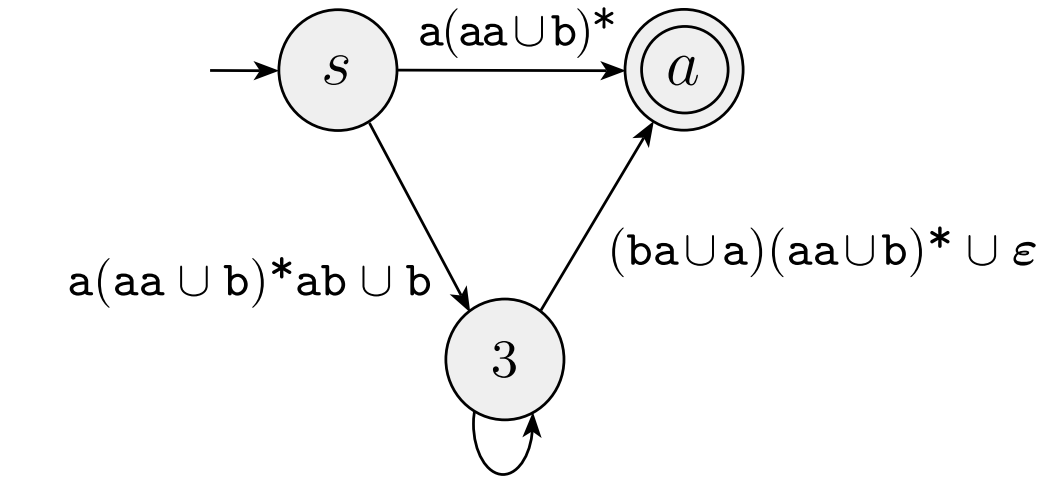


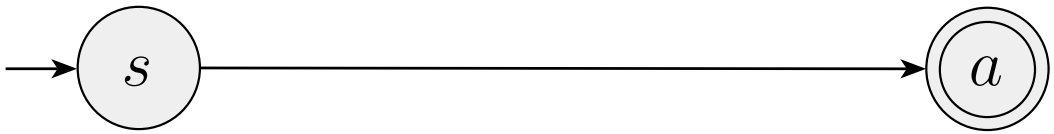
Lemma 2 例 2：From a DFA to a regular expression ▸ step 1 ▸ step 2





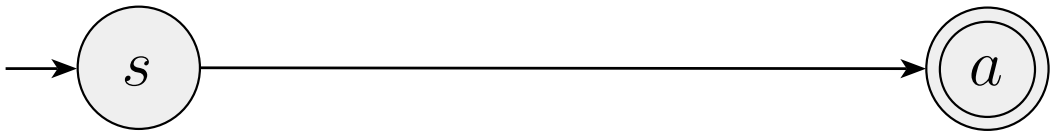
Lemma 2 例 2：From a DFA to a regular expression ▸ step 1 ▸ step 2





$$(a(aa\cup b)^*ab\cup b)((ba\cup a)(aa\cup b)^*ab\cup bb)^*((ba\cup a)(aa\cup b)^*\cup\varepsilon)\cup a(aa\cup b)^*$$

Lemma 2 例 2：From a DFA to a regular expression ▸ step 1 ▸ step 2



$$(\mathtt{a}(\mathtt{aa}\cup\mathtt{b})^*\mathtt{ab}\cup\mathtt{b})((\mathtt{ba}\cup\mathtt{a})(\mathtt{aa}\cup\mathtt{b})^*\mathtt{ab}\cup\mathtt{bb})^*((\mathtt{ba}\cup\mathtt{a})(\mathtt{aa}\cup\mathtt{b})^*\cup\varepsilon)\cup\mathtt{a}(\mathtt{aa}\cup\mathtt{b})^*$$

总结: See: why prove this

引理 1

If a language is described by a *regular expression*, then it is *regular*

引理 2

If a language is *regular*, then it is described by a *regular expression*

定理

A language is *regular*, iff, it is described by a *regular expression*

Outline

问: Can a finite automaton recognize *all* languages?
答: Of course not.

问: Any counterexample?
答: $B = \{0^n 1^n \mid n \geq 0\}$

- the machine ***seems*** to need to *remember how many* 0s have been seen so far as it reads the input
  - Cannot: $C = \{w \mid \text{w has an equal number of 0s and 1s}\}$
  - *Can*: $D = \{w \mid w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

问: How to name the class of languages?
答: *Non-regular* languages.

问: How to prove that a language is *non-regular*
答: 见下页

问: Can a finite automaton recognize *all* languages?
答: Of course not.

问: Any counterexample?
答: $B = \{0^n 1^n \mid n \geq 0\}$

- the machine ***seems*** to need to *remember how many* 0s have been seen so far as it reads the input
  - Cannot: $C = \{w \mid$ w has an equal number of 0s and 1s$\}$
  - *Can*: $D = \{w \mid w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

问: How to name the class of languages?
答: *Non-regular* languages.

问: How to prove that a language is *non-regular*
答: 见下页

问: Can a finite automaton recognize *all* languages?
答: Of course not.

问: Any counterexample?
答: $B = \{0^n 1^n \mid n \geq 0\}$

- the machine ***seems*** to need to *remember how many* 0s have been seen so far as it reads the input
  - Cannot: $C = \{w \mid$ w has an equal number of 0s and 1s$\}$
  - *Can*: $D = \{w \mid w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

问: How to name the class of languages?
答: *Non-regular* languages.

问: How to prove that a language is *non-regular*
答: 见下页

问: Can a finite automaton recognize *all* languages?
答: Of course not.

问: Any counterexample?
答: $B = \{0^n 1^n \mid n \geq 0\}$

- the machine ***seems*** to need to *remember how many* 0s have been seen so far as it reads the input
  - Cannot: $C = \{w \mid$ w has an equal number of 0s and 1s$\}$
  - *Can*: $D = \{w \mid w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

问: How to name the class of languages?
答: *Non-regular* languages.

问: How to prove that a language is *non-regular*
答: 见下页

问: Can a finite automaton recognize *all* languages?
答: Of course not.

问: Any counterexample?
答: $B = \{0^n 1^n \mid n \geq 0\}$

- the machine ***seems*** to need to *remember how many* 0s have been seen so far as it reads the input
  - Cannot: $C = \{w \mid$ w has an equal number of 0s and 1s$\}$
  - *Can*: $D = \{w \mid w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

问: How to name the class of languages?
答: *Non-regular* languages.

问: How to prove that a language is *non-regular*
答: 见下页

问: How to prove that a language is *non-regular*
准备:

术语: pumping lemma, pumping length

- *pumping lemma*: A theorem, stating that all regular languages have a *special property*.
- *pumping length*: The property states that *all strings* in the language can be *pumped*, if they are at least as *long* as a certain *special value*, called the *pumping length*.
  - That means each such *string contains* a *section* that can be *repeated* any number of times with the *resulting* string *remaining* in the language.

定理: Pumping Lemma for regular languages

If A is a *regular language*, then there is a *number* p (the *pumping length*), where if s is any string in A of *length at least* p, then s may be *divided* into *three pieces*, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$
2. $|y| > 0$
3. $|xy| \leq p$

证明思路

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

- $p =$ *number of states* of M
- any string s in A of length *at least* p may be broken into the three pieces xyz

定理: Pumping Lemma for regular languages

If A is a *regular language*, then there is a *number* p (the *pumping length*), where if s is any string in A of *length at least* p, then s may be *divided* into *three pieces*, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$
2. $|y| > 0$
   - Either x or z may be ε, but $y \neq \varepsilon$
3. $|xy| \leq p$

证明思路

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

- $p =$ *number of states* of M
- any string s in A of length *at least* p may be broken into the three pieces xyz

定理: Pumping Lemma for regular languages

If A is a *regular language*, then there is a *number* p (the *pumping length*), where if s is any string in A of *length at least* p, then s may be *divided* into *three pieces*, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$
2. $|y| > 0$
3. $|xy| \leq p$
   - The pieces x and y together have length at most p

证明思路

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

- $p =$ *number of states* of M
- any string s in A of length *at least* p may be broken into the three pieces xyz

定理: Pumping Lemma for regular languages

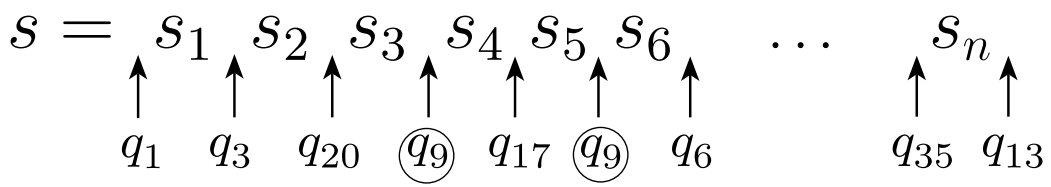
If A is a *regular language*, then there is a *number* p (the *pumping length*), where if s is any string in A of *length at least* p, then s may be *divided* into *three pieces*, $s = xyz$, satisfying the following conditions:

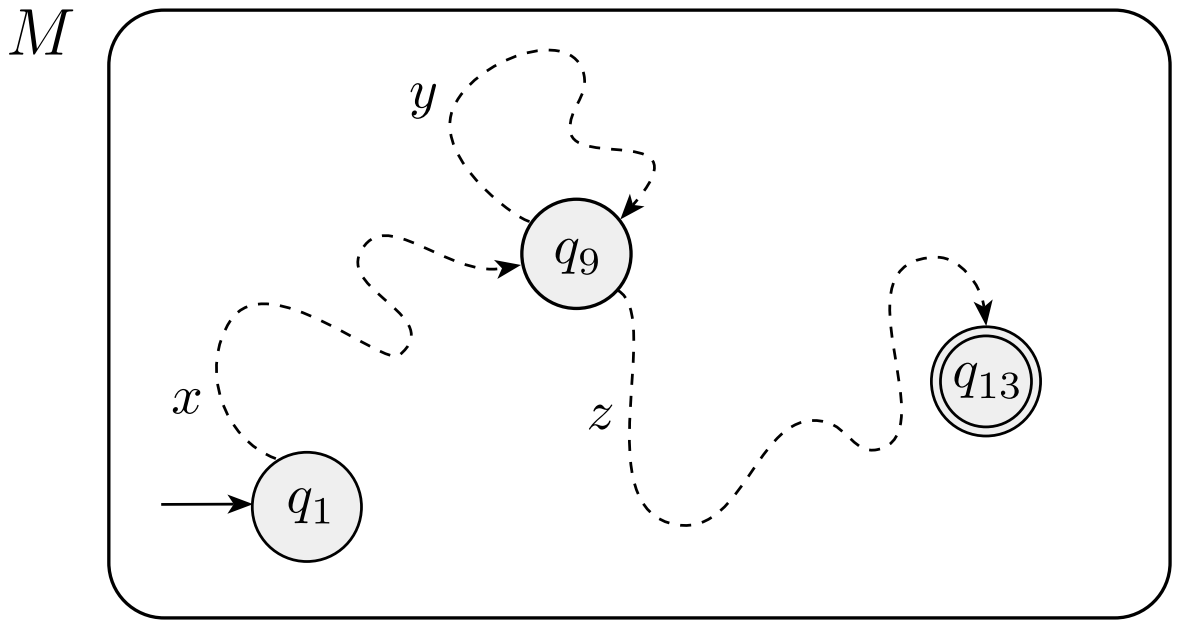
1. for each $i \geq 0$, $xy^iz \in A$
2. $|y| > 0$
3. $|xy| \leq p$

证明思路

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

- $p =$ *number of states* of M
- any string s in A of length *at least* p may be broken into the three pieces xyz





- Piece x is the part of s appearing before q_9
- Piece y is the part between the two appearances of q_9
- Piece z is the remaining part of s

定理: Pumping Lemma for regular languages

If A is a *regular language*, then ... ▸ Pumping Lemma for Regular Languages

证明思路 (续)

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

- $p =$ *number of states* of M
- any string s in A of length *at least* p may be broken into the three pieces xyz
- If s in A has length at least p, consider the *sequence* of states that M goes through when computing with input s. ▸ Example
  - The *length of sequence* is greater than p
  - The sequence must contain a *repeated* state.
  - Then break s according to the *repeated state*

定理: Pumping Lemma for regular languages

If A is a *regular language*, then ... ▸ Pumping Lemma for Regular Languages

证明思路 (续)

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

- $p =$ *number of states* of M
- any string s in A of length *at least* p may be broken into the three pieces xyz
- If s in A has length at least p, consider the *sequence* of states that M goes through when computing with input s. ▸ Example
  - The *length of sequence* is greater than p
  - The sequence must contain a *repeated* state.
  - Then break s according to the *repeated state*

定理: Pumping Lemma for regular languages

If A is a *regular language*, then ... ▸ Pumping Lemma for Regular Languages

证明思路 (续)

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

- $p =$ *number of states* of M
- any string s in A of length *at least* p may be broken into the three pieces xyz
- If s in A has length at least p, consider the *sequence* of states that M goes through when computing with input s. ▸ Example
  - The *length of sequence* is greater than p
  - The sequence must contain a *repeated* state.
  - Then break s according to the *repeated state*

待证问题

Prove that *a language* B is *not regular* ▸ Pumping Lemma for Regular Languages

证明思路 (Proof by contradiction)

1. *Assume* that B is *regular* in order to obtain a *contradiction*
2. Find a string s in B that has length p or greater but that *cannot be pumped*
3. Demonstrate that s cannot be pumped by *considering all ways* of dividing s into x, y, and z

例: 求证 ▸ Pumping Lemma for Regular Languages

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. Prove that B is not regular.

证明

1. Assume to the contrary that B is regular.
   - Let p be the pumping length given by the *pumping lemma*.
2. Choose s to be the string $0^p 1^p$.
   - s can be split into three pieces, $s = xyz$
   - For any $i \geq 0$, the string $xy^i z$ is in B
3. Consider three cases
   - Case 1: The string y consists only of 0s.
   - Case 2: The string y consists only of 1s.
   - Case 3: The string y consists both 0s and 1s.

例: 求证 ▸ Pumping Lemma for Regular Languages

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. Prove that B is not regular.

证明

1. Assume to the contrary that B is regular.
   - Let p be the pumping length given by the *pumping lemma*.
2. Choose s to be the string $0^p 1^p$.
   - s can be split into three pieces, $s = xyz$
   - For any $i \geq 0$, the string $xy^i z$ is in B
3. Consider three cases
   - Case 1: The string y consists only of 0s.
   - Case 2: The string y consists only of 1s.
   - Case 3: The string y consists both 0s and 1s.

例: 求证 ▸ Pumping Lemma for Regular Languages

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. Prove that B is not regular.

证明

1. Assume to the contrary that B is regular.
   - Let p be the pumping length given by the *pumping lemma*.
2. Choose s to be the string $0^p 1^p$.
   - s can be split into three pieces, $s = xyz$
   - For any $i \geq 0$, the string $xy^i z$ is in B
3. Consider three cases
   - Case 1: The string y consists only of 0s.
   - Case 2: The string y consists only of 1s.
   - Case 3: The string y consists both 0s and 1s.

例: 求证 ▸ Pumping Lemma for Regular Languages

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. Prove that B is not regular.

证明

1. Assume to the contrary that B is regular.
   - Let p be the pumping length given by the *pumping lemma*.
2. Choose s to be the string $0^p 1^p$.
   - s can be split into three pieces, $s = xyz$
   - For any $i \geq 0$, the string $xy^i z$ is in B
3. Consider three cases
   - Case 1: The string y consists only of 0s.
   - Case 2: The string y consists only of 1s.
   - Case 3: The string y consists both 0s and 1s.

例: 求证 ▸ Pumping Lemma for Regular Languages

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. Prove that B is not regular.

证明

1. Assume to the contrary that B is regular.
   - Let p be the pumping length given by the *pumping lemma*.
2. Choose s to be the string $0^p 1^p$.
   - s can be split into three pieces, $s = xyz$
   - For any $i \geq 0$, the string $xy^i z$ is in B
3. Consider three cases
   - Case 1: The string y consists only of 0s.
     The string $xyyz$ has *more* 0s than 1s and so is not a member of B
   - Case 2: The string y consists only of 1s.
   - Case 3: The string y consists both 0s and 1s.

例: 求证 ▸ Pumping Lemma for Regular Languages

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. Prove that B is not regular.

证明

1. Assume to the contrary that B is regular.
   - Let p be the pumping length given by the *pumping lemma*.
2. Choose s to be the string $0^p 1^p$.
   - s can be split into three pieces, $s = xyz$
   - For any $i \geq 0$, the string $xy^i z$ is in B
3. Consider three cases
   - Case 1: The string y consists only of 0s.
   - Case 2: The string y consists only of 1s.
     The string $xyyz$ has *more* 1s than 0s and so is not a member of B
   - Case 3: The string y consists both 0s and 1s.

例: 求证 ▸ Pumping Lemma for Regular Languages

Let B be the language $\{0^n 1^n \mid n \geq 0\}$. Prove that B is not regular.

证明

1. Assume to the contrary that B is regular.
   - Let p be the pumping length given by the *pumping lemma*.
2. Choose s to be the string $0^p 1^p$.
   - s can be split into three pieces, $s = xyz$
   - For any $i \geq 0$, the string $xy^i z$ is in B
3. Consider three cases
   - Case 1: The string y consists only of 0s.
   - Case 2: The string y consists only of 1s.
   - Case 3: The string y consists both 0s and 1s. They will be *out of order* with some 1s before 0s

Outline

总结

- 定义
  - finite automaton, DFA
  - language
  - deterministic, non-deterministic
  - NFA
  - regular operations
  - recognize
  - regular language
  - closed
  - regular expression
- 定理
  - Closure, under union, star, concatenation
  - NFAs $\equiv$ DFAs
  - Regular Expression "$\equiv$" Regular Language
  - Pumping Lemma for Regular Languages
- 推论
  - NFA "$\equiv$" Regular language

作业

1.3 The formal description of a DFA M is $\big(\{q_1, q_2, q_3, q_4, q_5\}, \{\mathtt{u}, \mathtt{d}\}, \delta, q_3, \{q_3\}\big)$, where δ is given by the following table. Give the state diagram of this machine.

	u	d
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_5
q_5	q_4	q_5

1.6 Give state diagrams of DFAs recognizing the following languages. In all parts, the alphabet is {0,1}.

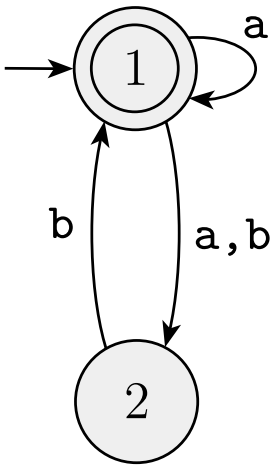
a. $\{w |\ w$ begins with a 1 and ends with a 0$\}$

b. $\{w |\ w$ contains at least three 1s$\}$

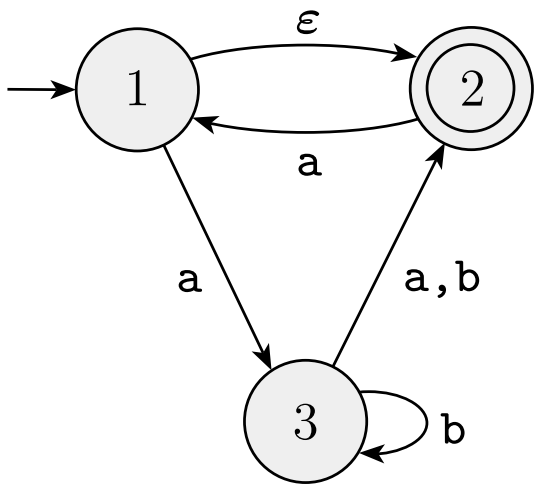
c. $\{w |\ w$ contains the substring 0101 (i.e., $w = x0101y$ for some x and $y)\}$

作业

1.16 Use the construction given in Theorem 1.39 to convert the following two nondeterministic finite automata to equivalent deterministic finite automata.



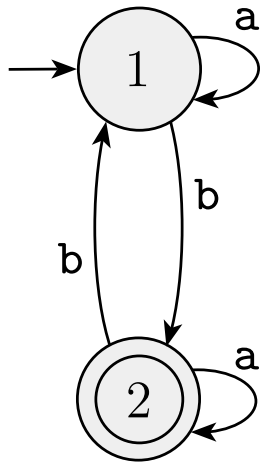
(a)



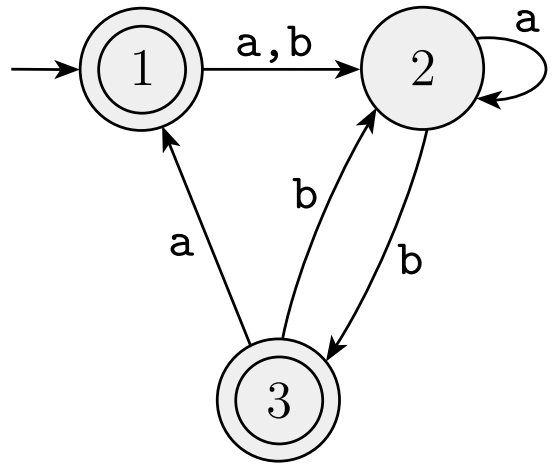
(b)

作业

1.21 Use the procedure described in Lemma 1.60 to convert the following finite automata to regular expressions.



(a)



(b)

1.29 Use the pumping lemma to show that the following languages are not regular.

[A]a. $A_1 = \{0^n 1^n 2^n \mid n \geq 0\}$