بسم الله الرحمن الرحیم

نظریه علوم کامپیوتر

نظریه علوم کامپیوتر - بهار ۱۴۰۰-۱۴۰۱ - جلسه دوم: زبان‌های منظم

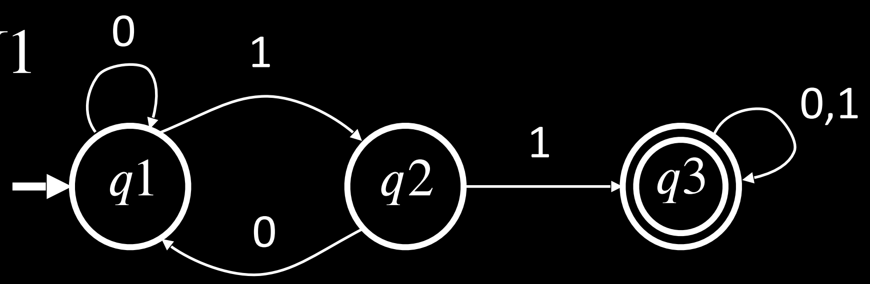
Theory of computation - 002 - S02 - Regular Languanges

نکات صنفی

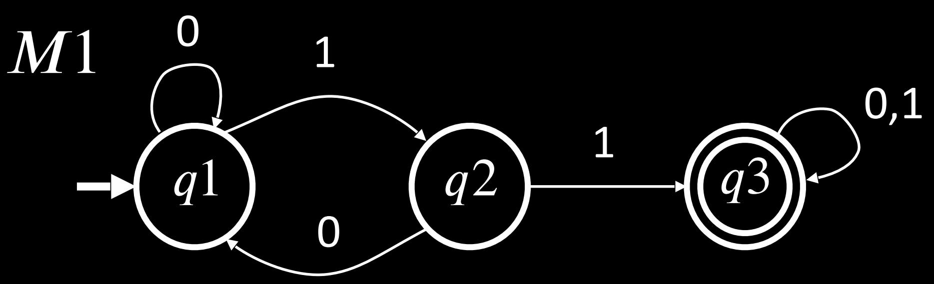
- در CW عضو شوید

Let's begin: Finite Automata

$M1$

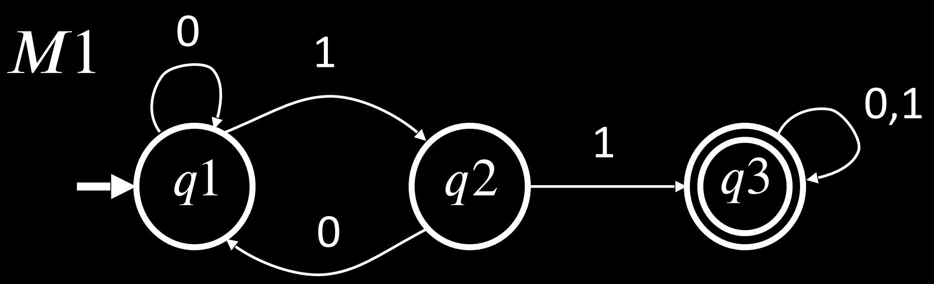


Let's begin: Finite Automata



States: $q1$ $q2$ $q3$

Let's begin: Finite Automata

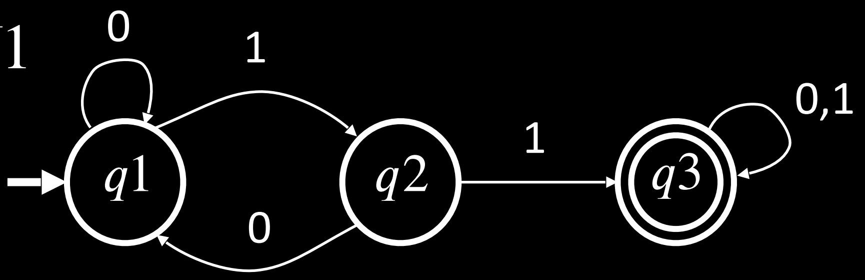


States: $q1$ $q2$ $q3$

Transitions: $\xrightarrow{1}$

Let's begin: Finite Automata

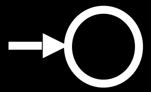
$M1$



States: $q1\ q2\ q3$

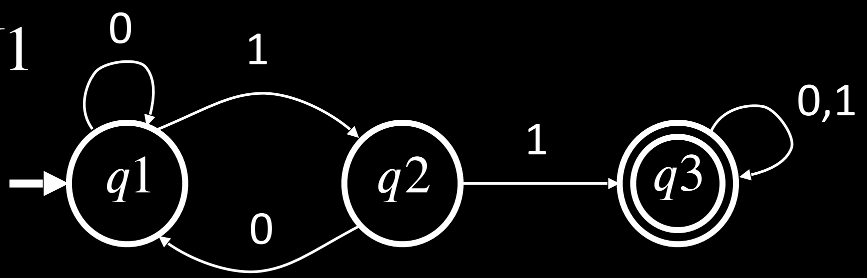
Transitions: $\xrightarrow{\quad 1 \quad}$

Start state:



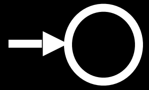
Let's begin: Finite Automata

$M1$



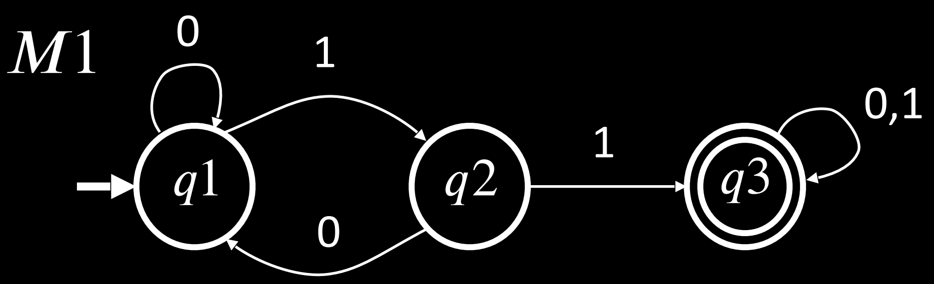
States: $q1$ $q2$ $q3$

Transitions: $\xrightarrow{1}$

Start state: 

Accept states: 

Let's begin: Finite Automata

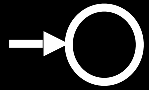


Input: finite string

Output: Accept or Reject

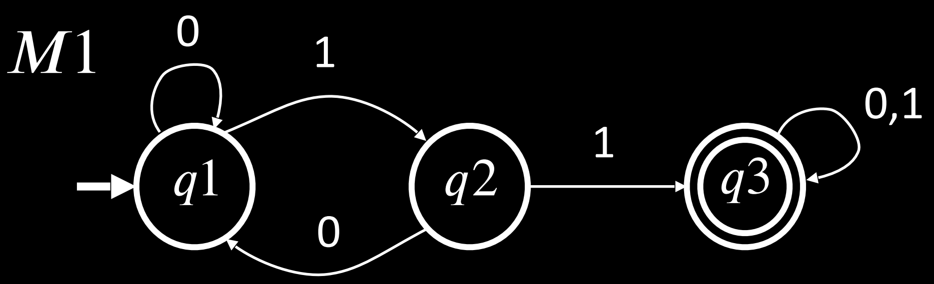
States: $q1$ $q2$ $q3$

Transitions: 

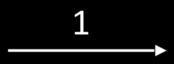
Start state: 

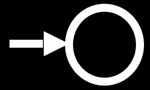
Accept states: 

Let’s begin: Finite Automata



States: $q1\ \ q2\ \ q3$

Transitions: 

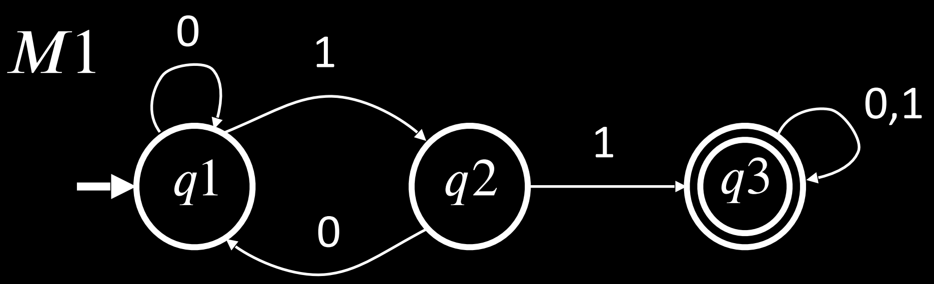
Start state: 

Accept states: 

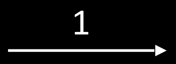
Input: finite string
Output: Accept or Reject

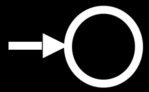
Computation process: Begin at start state,
read input symbols, follow corresponding transitions,
Accept if end with accept state, Reject if not.

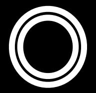
Let’s begin: Finite Automata



States: $q1$ $q2$ $q3$

Transitions: 

Start state: 

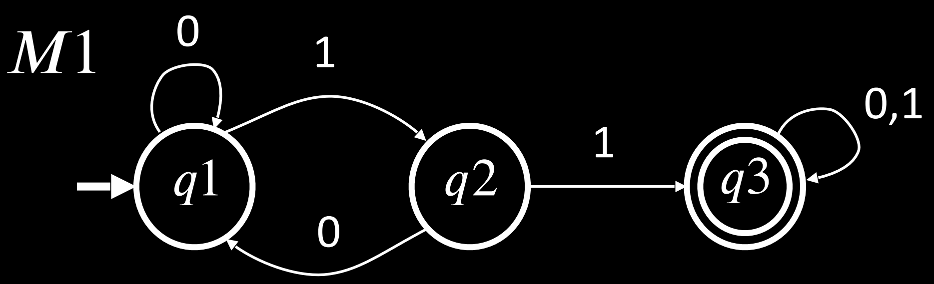
Accept states: 

Input: finite string
Output: <u>Accept</u> or <u>Reject</u>

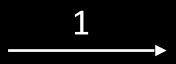
Computation process: Begin at start state,
read input symbols, follow corresponding transitions,
<u>Accept</u> if end with accept state, <u>Reject</u> if not.

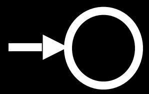
Examples: 01101 → Accept
00101 → Reject

Let's begin: Finite Automata



States: $q1\ q2\ q3$

Transitions: 

Start state: 

Accept states: 

Input: finite string
Output: Accept or Reject

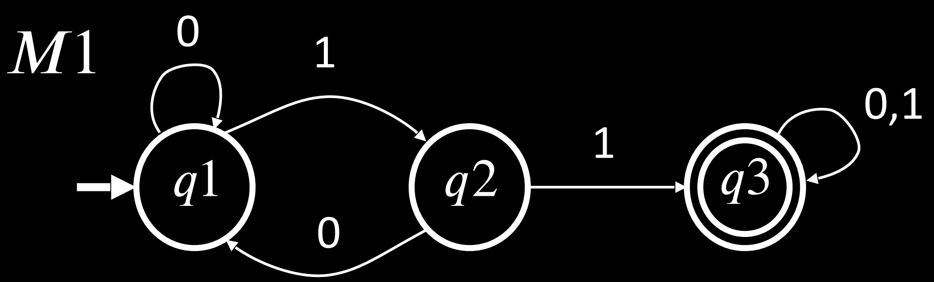
Computation process: Begin at start state, read input symbols, follow corresponding transitions, Accept if end with accept state, Reject if not.

Examples: 01101 → Accept
00101 → Reject

$M1$ accepts exactly those strings in A where

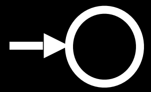
$$A = \{w \mid w \text{ contains substring } 11\}.$$

Let’s begin: Finite Automata



States: $q1\ q2\ q3$

Transitions: $\xrightarrow{1}$

Start state: 

Accept states: 

Input: finite string
Output: <u>Accept</u> or <u>Reject</u>

Computation process: Begin at start state,
read input symbols, follow corresponding transitions,
<u>Accept</u> if end with accept state, <u>Reject</u> if not.

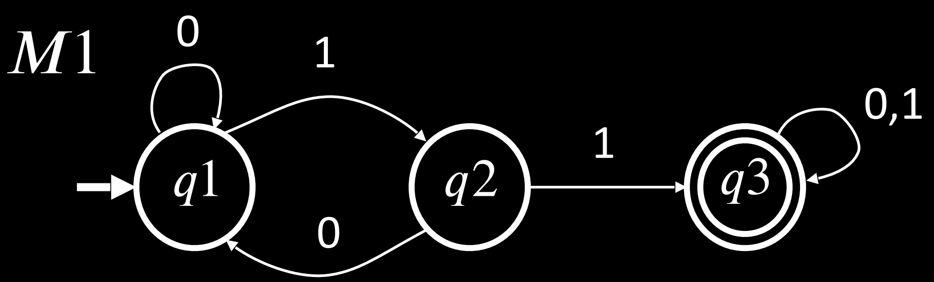
Examples: 01101 → Accept
00101 → Reject

$M1$ accepts exactly those strings in A where

$$A = \{w \mid w \text{ contains substring } 11\}.$$

Say that A is the language of $M1$ and that $M1$ recognizes A and that
$A = L(M1)$.

Let's begin: Finite Automata

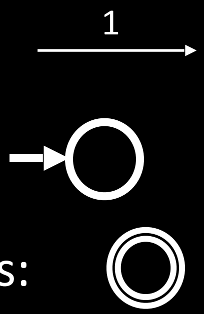


States: $q1$ $q2$ $q3$

Transitions:

Start state:

Accept states:



Input: finite string
Output: <u>Accept</u> or <u>Reject</u>

Computation process: Begin at start state,
read input symbols, follow corresponding transitions,
<u>Accept</u> if end with accept state, <u>Reject</u> if not.

Examples: 01101 → Accept
00101 → Reject

$M1$ accepts exactly those strings in A where
$A = \{w \mid w \text{ contains substring } 11\}$.

Say that A is the language of $M1$ and that $M1$ recognizes A and that
$A = L(M1)$.

Finite Automata – Formal Definition

Defn: A finite automaton M is a 5-tuple
$(Q, \ \Sigma, \ \delta, \ q0, \ F)$

Finite Automata – Formal Definition

Defn: A finite automaton M is a 5-tuple

$(Q,\ \Sigma,\ \delta,\ q0,\ F)$

Q finite set of states

Finite Automata – Formal Definition

Defn: A finite automaton M is a 5-tuple

$(Q,\ \Sigma,\ \delta,\ q0,\ F)$

Q finite set of states

Σ finite set of alphabet symbols

Finite Automata – Formal Definition

Defn: A finite automaton M is a 5-tuple

$(Q,\ \Sigma,\ \delta,\ q0,\ F)$

Q finite set of states

Σ finite set of alphabet symbols

δ transition function $\delta\colon Q \times \Sigma \to Q$

Finite Automata – Formal Definition

Defn: A finite automaton M is a 5-tuple

$(Q,\ \Sigma,\ \delta,\ q0,\ F)$

Q finite set of states

Σ finite set of alphabet symbols

δ transition function $\delta\colon Q \times \Sigma \rightarrow Q$

$\delta(q,\ a) = r$ means $q \xrightarrow{a} r$

Finite Automata – Formal Definition

Defn: A <u>finite automaton</u> M is a 5-tuple

$(Q,\ \Sigma,\ \delta,\ q0,\ F)$

Q finite set of states

Σ finite set of alphabet symbols

δ transition function $\delta\colon Q \times \Sigma \rightarrow Q$

$q0$ start state

$\delta\,(q,\ a) = r$ means $q \xrightarrow{a} r$

Finite Automata – Formal Definition

Defn: A <u>finite automaton</u> M is a 5-tuple

$(Q, \Sigma, \delta, q0, F)$

Q finite set of states

Σ finite set of alphabet symbols

δ transition function $\delta\colon Q \times \Sigma \to Q$

$q0$ start state

$\delta(q, a) = r$ means $q \xrightarrow{a} r$

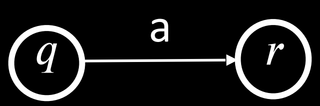
F set of accept states

Finite Automata – Formal Definition

Defn: A <u>finite automaton</u> M is a 5-tuple

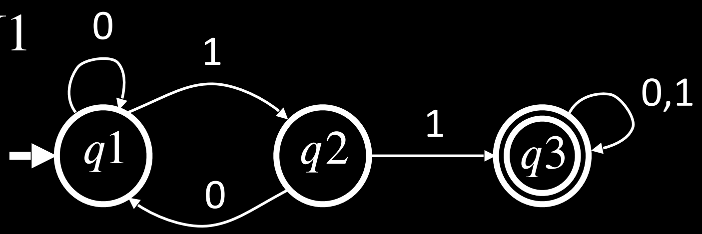
$(Q,\ \Sigma,\ \delta,\ q0,\ F)$

- Q finite set of states
- Σ finite set of alphabet symbols
- δ transition function $\delta\colon\ Q \times \Sigma\ \rightarrow\ Q$
- $q0$ start state
- F set of accept states

$\delta\,(q,\ \ a)\ =\ \ r$ means 

Example:

$M1$



Finite Automata – Formal Definition

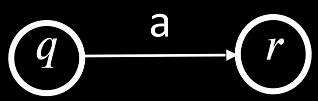
Defn: A <u>finite automaton</u> M is a 5-tuple

$(Q,\ \Sigma,\ \delta,\ q0,\ F)$

Q finite set of states

Σ finite set of alphabet symbols

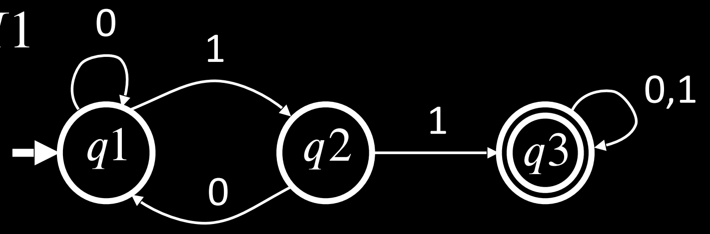
δ transition function $\delta\colon\ Q \times \Sigma\ \rightarrow\ Q$

$q0$ start state $\delta\,(q,\ \ a)\ =\ \ r$ means 

F set of accept states

Example:

$M1$



$M1\ =\ (Q,\ \Sigma,\ \delta,\ q1,\ F)$

$Q\ =\ \{q1,\ q2,\ q3\}$

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

$F\ =\ \{q3\}$

Finite Automata – Formal Definition

Defn: A <u>finite automaton</u> M is a 5-tuple

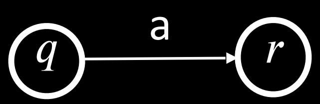
$(Q,\ \Sigma,\ \delta,\ q0,\ F)$

Q finite set of states

Σ finite set of alphabet symbols

δ transition function $\delta\colon Q \times \Sigma \ \to\ Q$

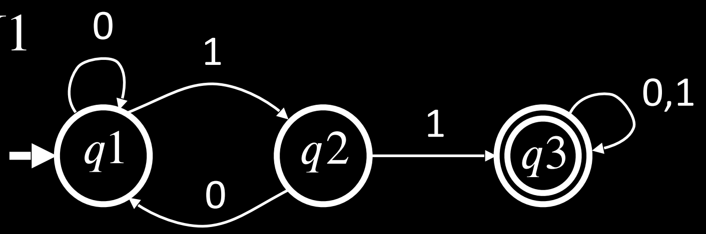
$q0$ start state

$\delta\,(q,\ a) \ =\ r$ means 

F set of accept states

Example:

$M1$



$M1 \ =\ (Q,\ \Sigma,\ \delta,\ q1,\ F)$

$Q \ =\ \{q1,\ q2,\ q3\}$

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

$F \ =\ \{q3\}$

$\delta =$	0	1
q1	q1	q2
q2	q1	q3
q3	q3	q3

Finite Automata – Computation

Strings and languages

- A string is a finite sequence of symbols in Σ
- A language is a set of strings (finite or infinite)

Finite Automata – Computation

Strings and languages

- A <u>string</u> is a finite sequence of symbols in Σ
- A <u>language</u> is a set of strings (finite or infinite)
- The <u>empty string</u> ε is the string of length 0
- The <u>empty language</u> $\emptyset$ is the set with no strings

Finite Automata – Computation

Strings and languages

- A string is a finite sequence of symbols in Σ
- A language is a set of strings (finite or infinite)
- The empty string ε is the string of length 0
- The empty language ϕ is the set with no strings

Defn: M accepts string $w = w1w2 \ldots wn$ each $wi \epsilon \Sigma$
if there is a sequence of states $r0,\ r1,\ r2,\ , \ldots,\ rn\ \epsilon\ Q$
where:

- $r0 = q0$
- $ri = \delta(r_{i-1},\ wi)$ for $1 \leq i \leq n$
- $rn \epsilon F$

Finite Automata – Computation

Strings and languages

- A <u>string</u> is a finite sequence of symbols in Σ
- A <u>language</u> is a set of strings (finite or infinite)
- The <u>empty string</u> ε is the string of length 0
- The <u>empty language</u> ϕ is the set with no strings

Defn: M <u>accepts string</u> $w = w1w2 \ldots wn$ each $wi \in \Sigma$
if there is a sequence of states $r0, r1, r2, , \ldots, rn \in Q$
where:

- $r0 = q0$
- $ri = \delta(r_{i-1}, wi)$ for $1 \le i \le n$
- $rn \in F$

Recognizing languages

- $L(M) = \{w \mid M \text{ accepts } w\}$
- $L(M)$ is <u>the language of</u> M
- M <u>recognizes</u> $L(M)$

Finite Automata – Computation

Strings and languages

- A <u>string</u> is a finite sequence of symbols in Σ
- A <u>language</u> is a set of strings (finite or infinite)
- The <u>empty string</u> ε is the string of length 0
- The <u>empty language</u> ϕ is the set with no strings

Defn: M <u>accepts string</u> $w = w1w2 \ldots wn$ each $wi \in \Sigma$
if there is a sequence of states $r0,\ r1,\ r2,\ , \ldots,\ rn \in Q$ where:

- $r0 = q0$
- $ri = \delta(r_{i-1},\ wi)$ for $1 \leq i \leq n$
- $rn \in F$

Recognizing languages

- $L(M) = \{w \mid M \text{ accepts } w\}$
- $L(M)$ is <u>the language of</u> M
- M <u>recognizes</u> $L(M)$

Defn: A language is <u>regular</u> if some finite automaton recognizes it.

Finite Automata – Computation

Strings and languages

- A <u>string</u> is a finite sequence of symbols in Σ
- A <u>language</u> is a set of strings (finite or infinite)
- The <u>empty string</u> ε is the string of length 0
- The <u>empty language</u> ϕ is the set with no strings

Defn: M <u>accepts string</u> $w = w1w2 \ldots wn$ each $wi \in \Sigma$
if there is a sequence of states $r0,\ r1,\ r2,\ ,\ldots,\ rn \in Q$
where:

- $r0 = q0$
- $ri = \delta(r_{i-1},\ wi)$ for $1 \leq i \leq n$
- $rn \in F$

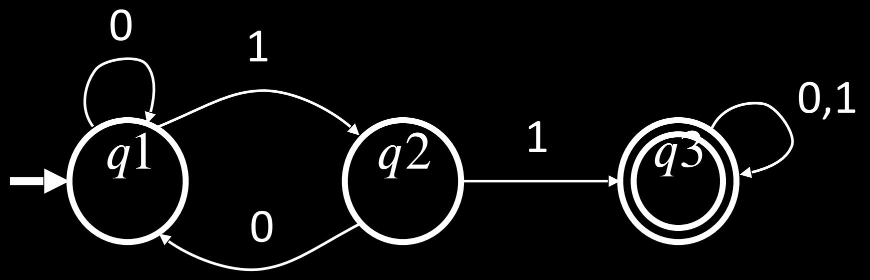
Recognizing languages

- $L(M) = \{w \mid M \text{ accepts } w\}$
- $L(M)$ is <u>the language of</u> M
- M <u>recognizes</u> $L(M)$

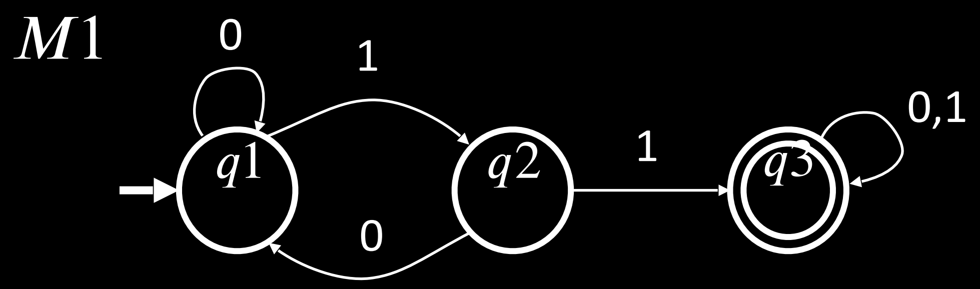
Defn: A language is <u>regular</u> if some finite automaton recognizes it.

Regular Languages – Examples

$M1$

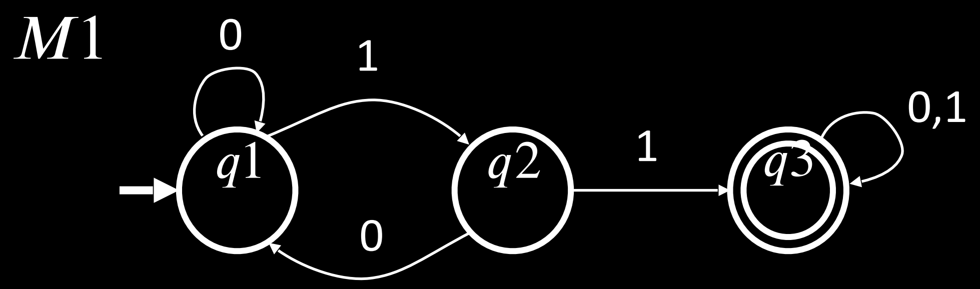


Regular Languages – Examples



$$L(M_1) = \{w \mid w \text{ contains substring } 11\} = A$$

Regular Languages – Examples

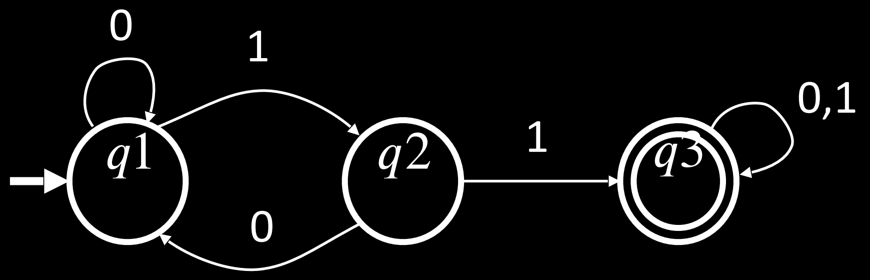


$L(M_1) = \{w \mid w \text{ contains substring } 11\} = A$

<u>Therefore A is regular</u>

Regular Languages – Examples

$M1$



$L(M_1) = \{w \mid w \text{ contains substring } 11\} = A$

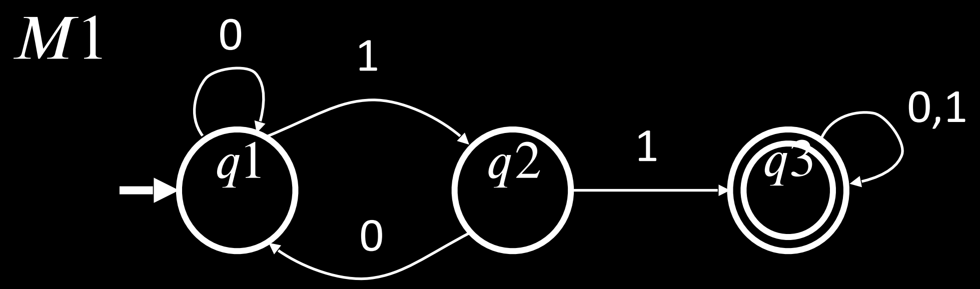
Therefore A is regular

More examples:

Let $B = \{w \mid w \text{ has an even number of 1s}\}$

B is regular (make automaton for practice).

Regular Languages – Examples



$$L(M_1) = \{w \mid w \text{ contains substring } 11\} = A$$

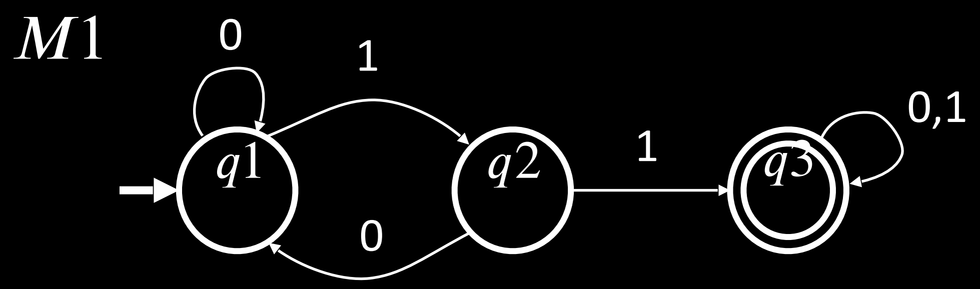
Therefore A is regular

More examples:

Let $B = \{w \mid w \text{ has an even number of 1s}\}$
B is regular (make automaton for practice).

Let $C = \{w \mid w \text{ has equal numbers of 0s and 1s}\}$
C is not regular (we will prove).

Regular Languages – Examples



$$L(M_1) = \{w \mid w \text{ contains substring } 11\} = A$$

<u>Therefore A is regular</u>

More examples:

Let $B = \{w \mid w \text{ has an even number of 1s}\}$
B is regular (make automaton for practice).

Let $C = \{w \mid w \text{ has equal numbers of 0s and 1s}\}$
C is <u>not</u> regular (we will prove).

Goal: Understand the regular languages

Regular Expressions

Regular Expressions

Regular operations. Let A, B be languages:

Regular Expressions

Regular operations. Let A, B be languages:

- Union: $A \cup B = \{w \mid w \in A \text{ or } w \in B\}$

Regular Expressions

Regular operations. Let A, B be languages:

- Union: $A \cup B = \{w \mid w \in A \text{ or } w \in B\}$
- Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\} = AB$

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

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

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}
- $A^* =$ {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, … }

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

Regular expressions

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}
- $A^* =$ {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, … }

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}
- $A^* =$ {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, ... }

Regular expressions

- Built from Σ, members Σ, $\varnothing$, ε [Atomic]

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}
- $A^* =$ {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, … }

Regular expressions

- Built from Σ, members Σ, $\emptyset, \varepsilon$ [Atomic]
- By using $\cup$, $\circ$, $*$ [Composite]

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}
- $A^* =$ {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, … }

Regular expressions

- Built from Σ, members Σ, $\varnothing, \varepsilon$ [Atomic]
- By using $\cup$, $\circ$, $*$ [Composite]

Examples:

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}
- $A^* =$ {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, … }

Regular expressions

- Built from Σ, members Σ, $\emptyset, \varepsilon$ [Atomic]
- By using $\cup$, $\circ$, $*$ [Composite]

Examples:

- $(0 \cup 1)^* = \Sigma^*$ gives all strings over Σ

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}
- $A^* =$ {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, … }

Regular expressions

- Built from Σ, members Σ, $\varnothing, \varepsilon$ [Atomic]
- By using $\cup$, $\circ$, $*$ [Composite]

Examples:

- $(0 \cup 1)^* = \Sigma^*$ gives all strings over Σ
- $\Sigma^* 1$ gives all strings that end with 1

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}
- $A^* =$ {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, ... }

Regular expressions

- Built from Σ, members Σ, $\varnothing, \varepsilon$ [Atomic]
- By using $\cup$, $\circ$, $*$ [Composite]

Examples:

- $(0 \cup 1)^* = \Sigma^*$ gives all strings over Σ
- $\Sigma^* 1$ gives all strings that end with 1
- $\Sigma^* 11 \Sigma^* =$ all strings that contain 11 $= L(M_1)$

Regular Expressions

Regular operations. Let A, B be languages:

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

  Note: $\varepsilon \in A^*$ always

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

- $A \cup B =$ {good, bad, boy, girl}
- $A \circ B = AB =$ {goodboy, goodgirl, badboy, badgirl}
- $A^* =$ {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, … }

Regular expressions

- Built from Σ, members Σ, $\emptyset, \varepsilon$ [Atomic]
- By using $\cup$, $\circ$, $*$ [Composite]

Examples:

- $(0 \cup 1)^* = \Sigma^*$ gives all strings over Σ
- Σ^*1 gives all strings that end with 1
- $\Sigma^*11\Sigma^* =$ all strings that contain 11 $= L(M_1)$

Goal: Show finite automata equivalent to regular expressions

Closure Properties for Regular Languages

Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under $\cup$)

Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under $\cup$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under ∪)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.

Closure Properties for Regular Languages

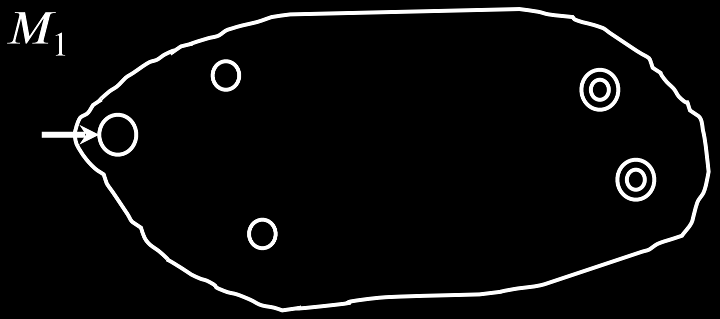
Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under ∪)

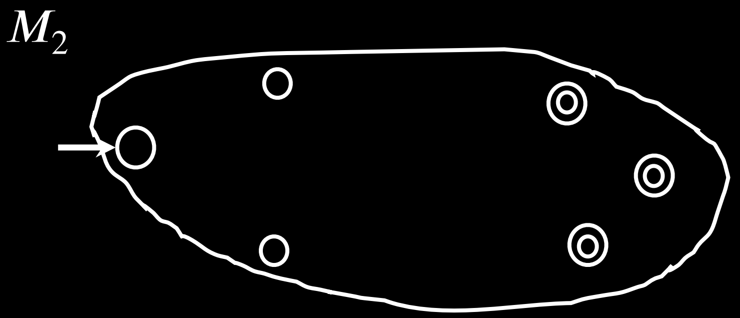
Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.





Closure Properties for Regular Languages

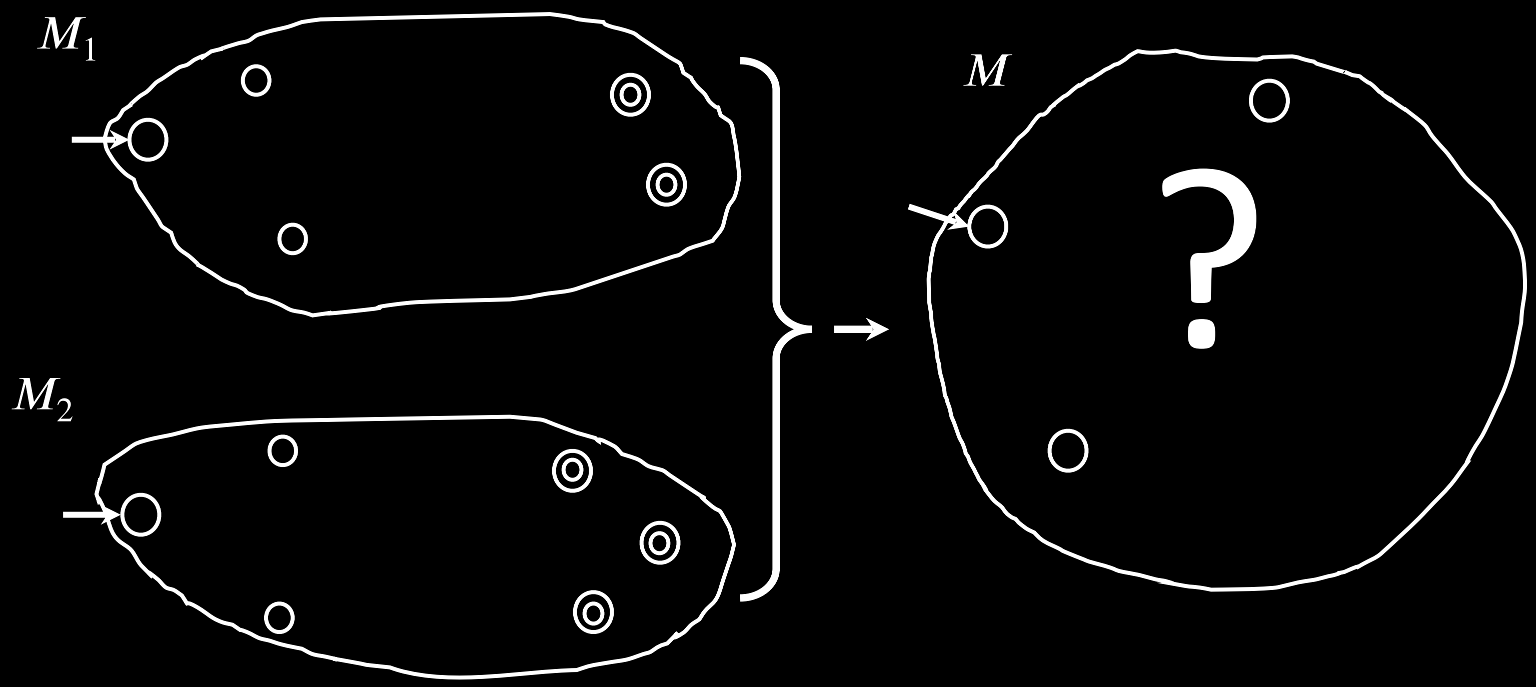
Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under ∪)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.



Closure Properties for Regular Languages

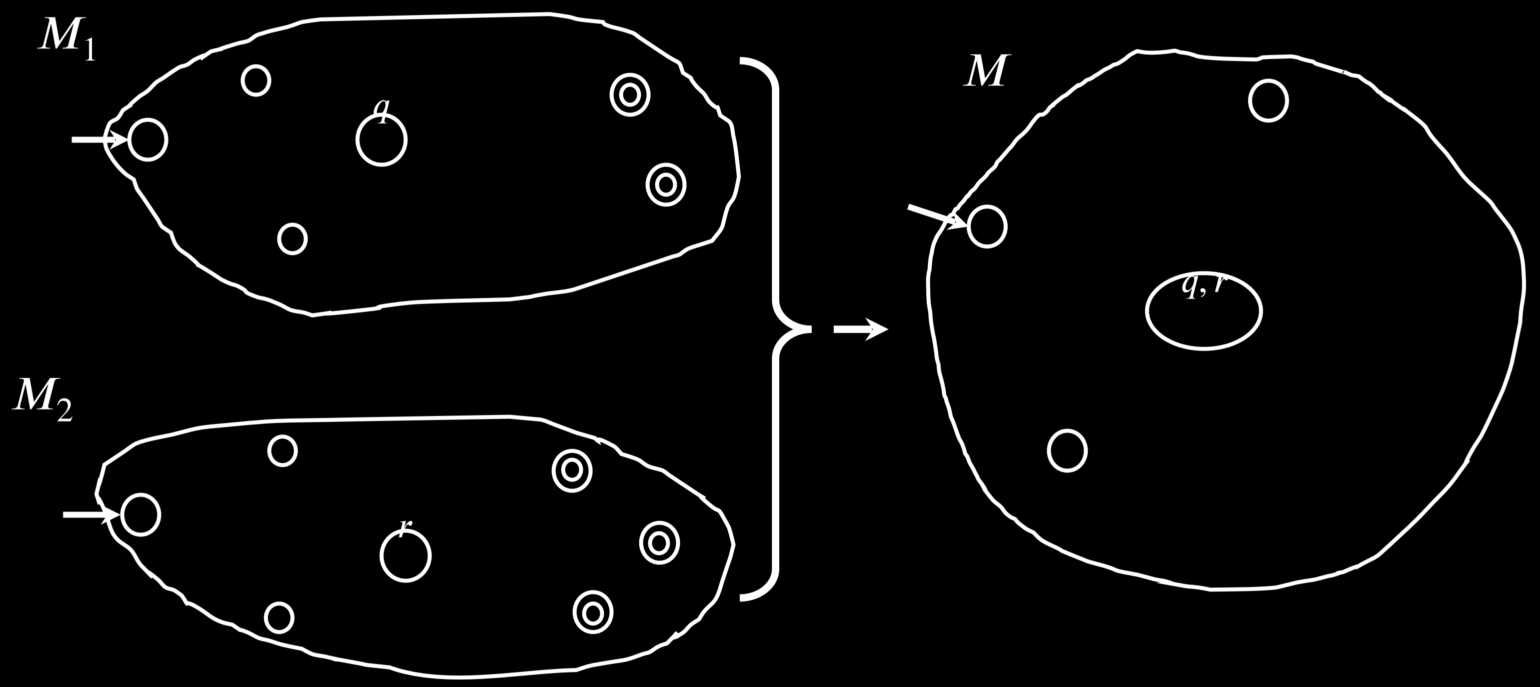
Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under $\cup$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.



Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under ∪)

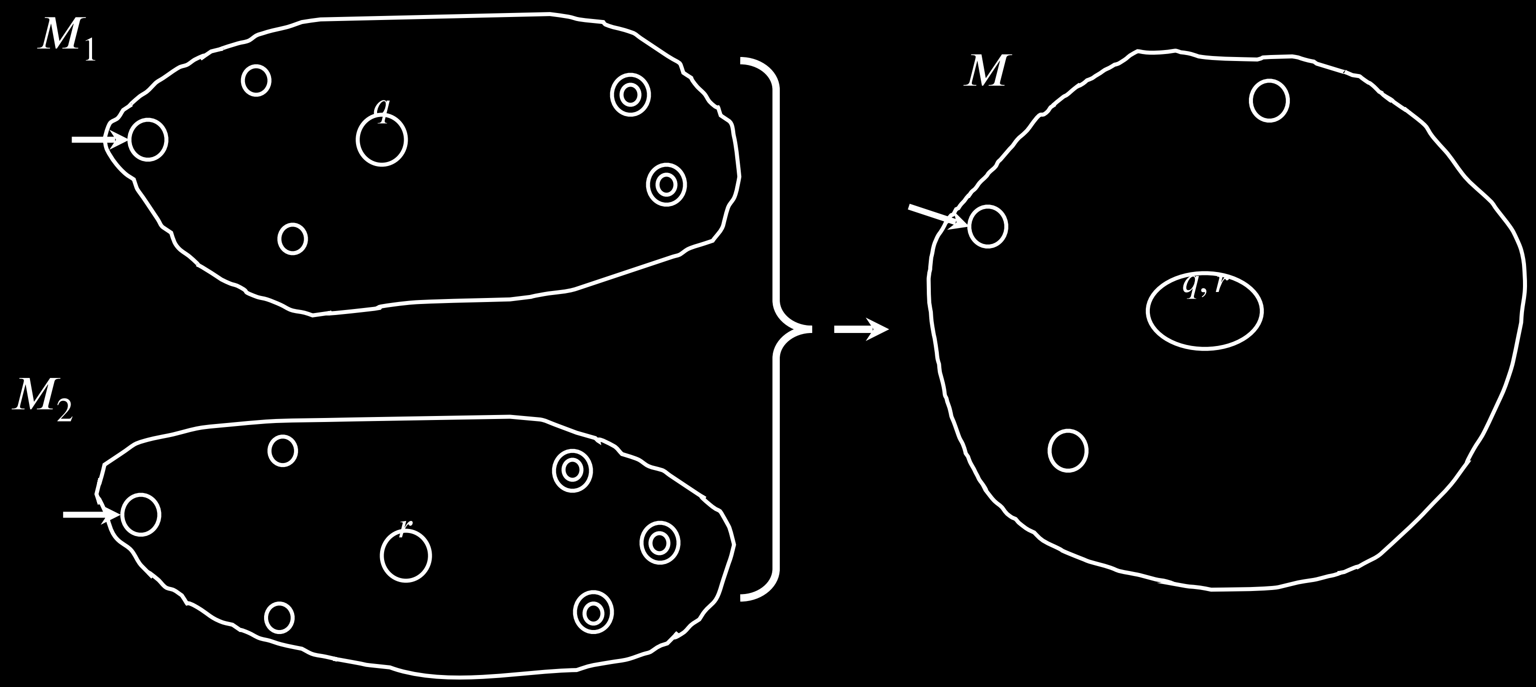
Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.

Components of $\boldsymbol{M}$:



Closure Properties for Regular Languages

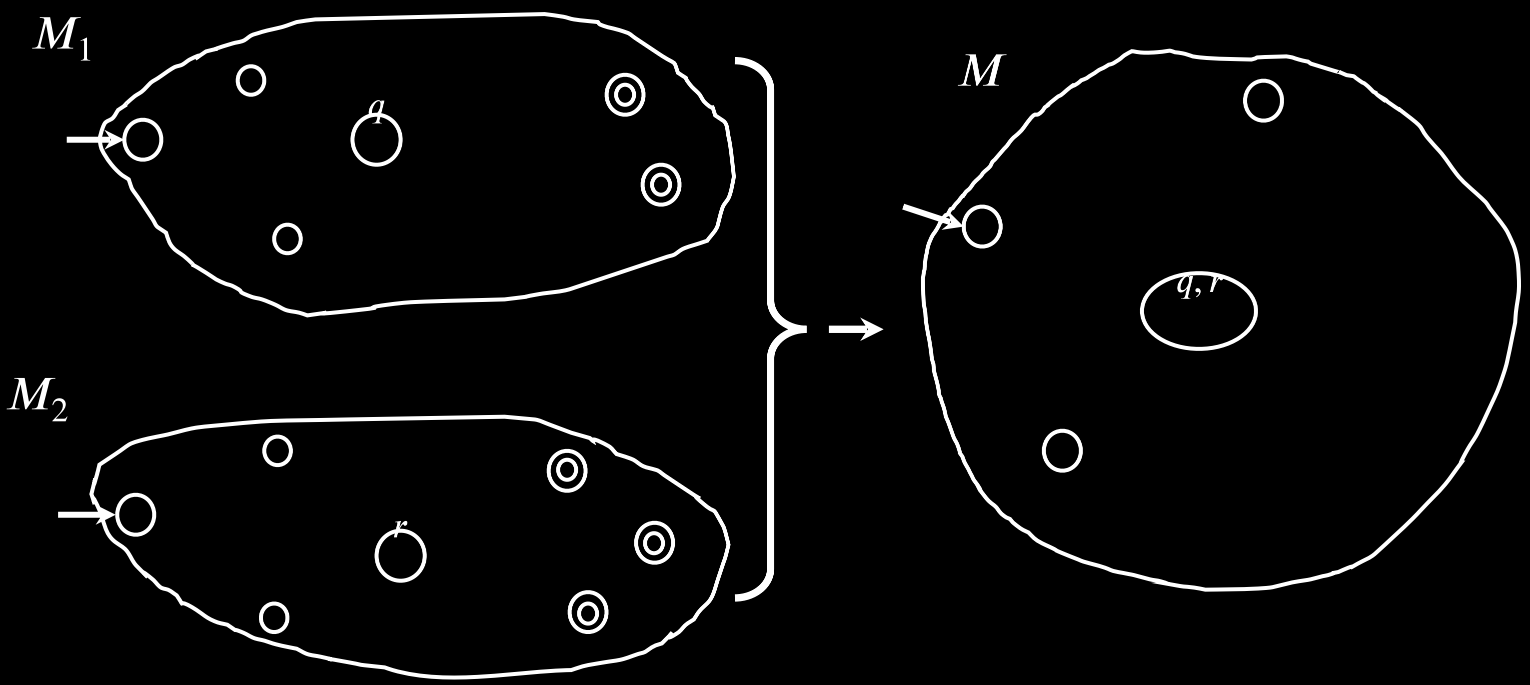
Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under $\cup$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.



Components of $\boldsymbol{M}$:

$$Q = Q_1 \times Q_2$$
$$= \left\{ (q_1, q_2) \,\middle|\, q_1 \in Q_1 \text{ and } q_2 \in Q_2 \right\}$$

Closure Properties for Regular Languages

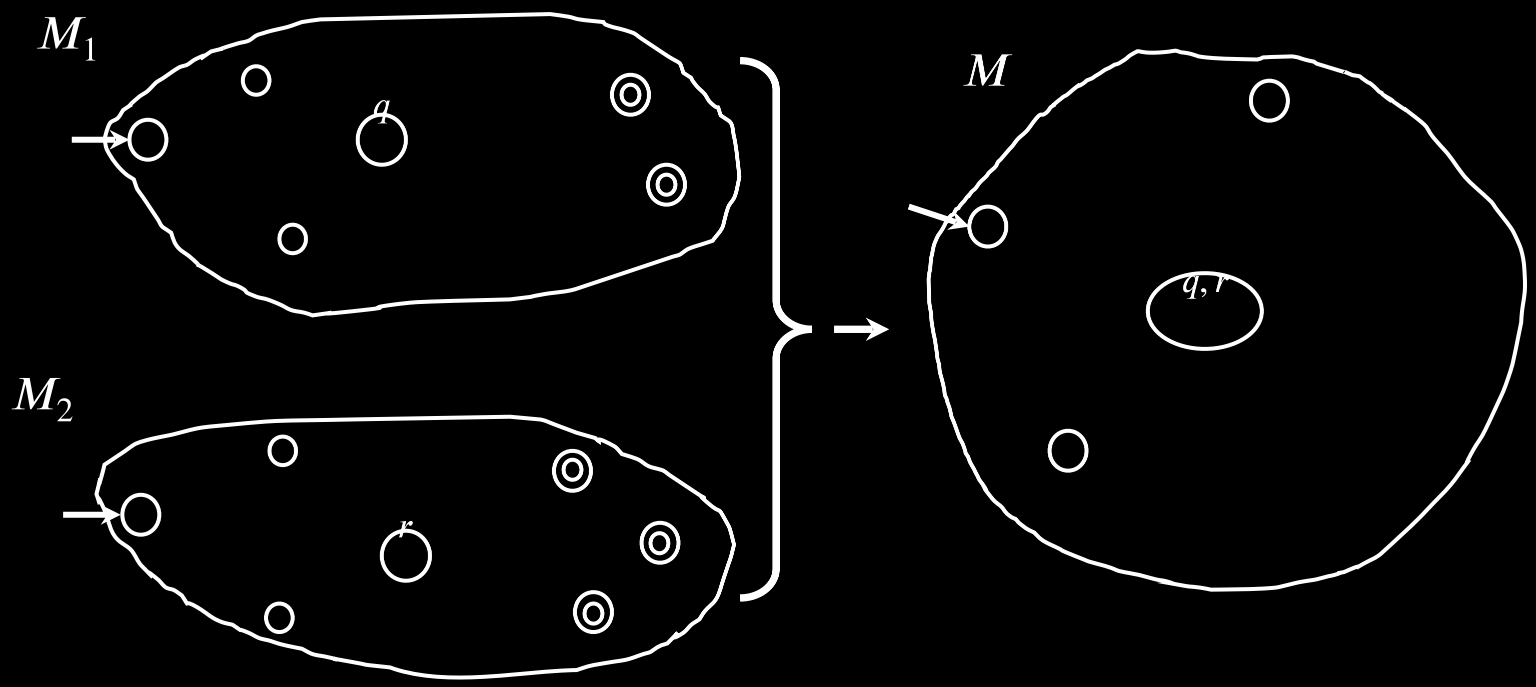
Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under ∪)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.



Components of $\boldsymbol{M}$:

$Q = Q_1 \times Q_2$

$= \left\{ (q_1, q_2) \middle| q_1 \in Q_1 \text{ and } q_2 \in Q_2 \right\}$

$q_0 = (q_1, q_2)$

Closure Properties for Regular Languages

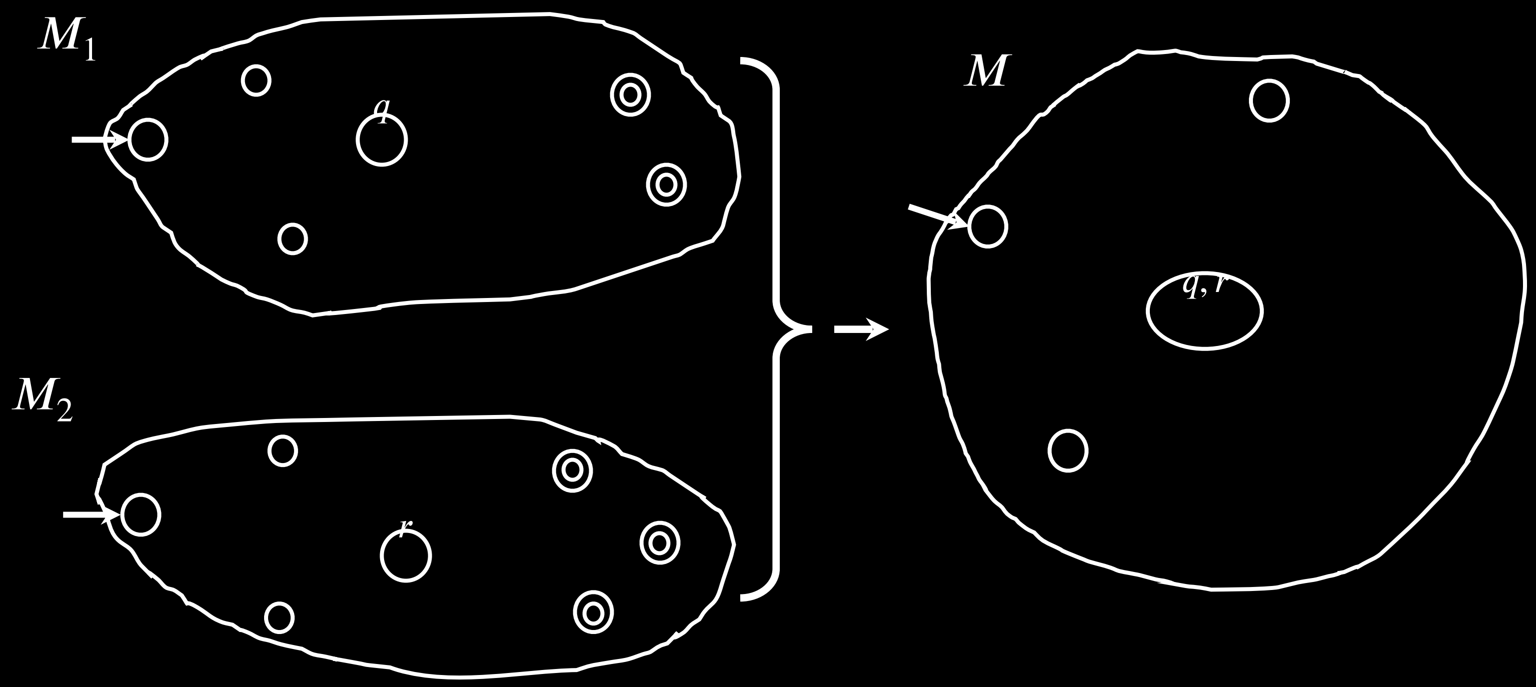
Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under ∪)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.



Components of $\boldsymbol{M}$:

$Q = Q_1 \times Q_2$

$= \left\{ (q_1, q_2) \,\middle|\, q_1 \in Q_1 \text{ and } q_2 \in Q_2 \right\}$

$q_0 = (q_1, q_2)$

$\delta\big((q,r),a\big) = \big(\delta_1(q,a), \delta_2(r,a)\big)$

Closure Properties for Regular Languages

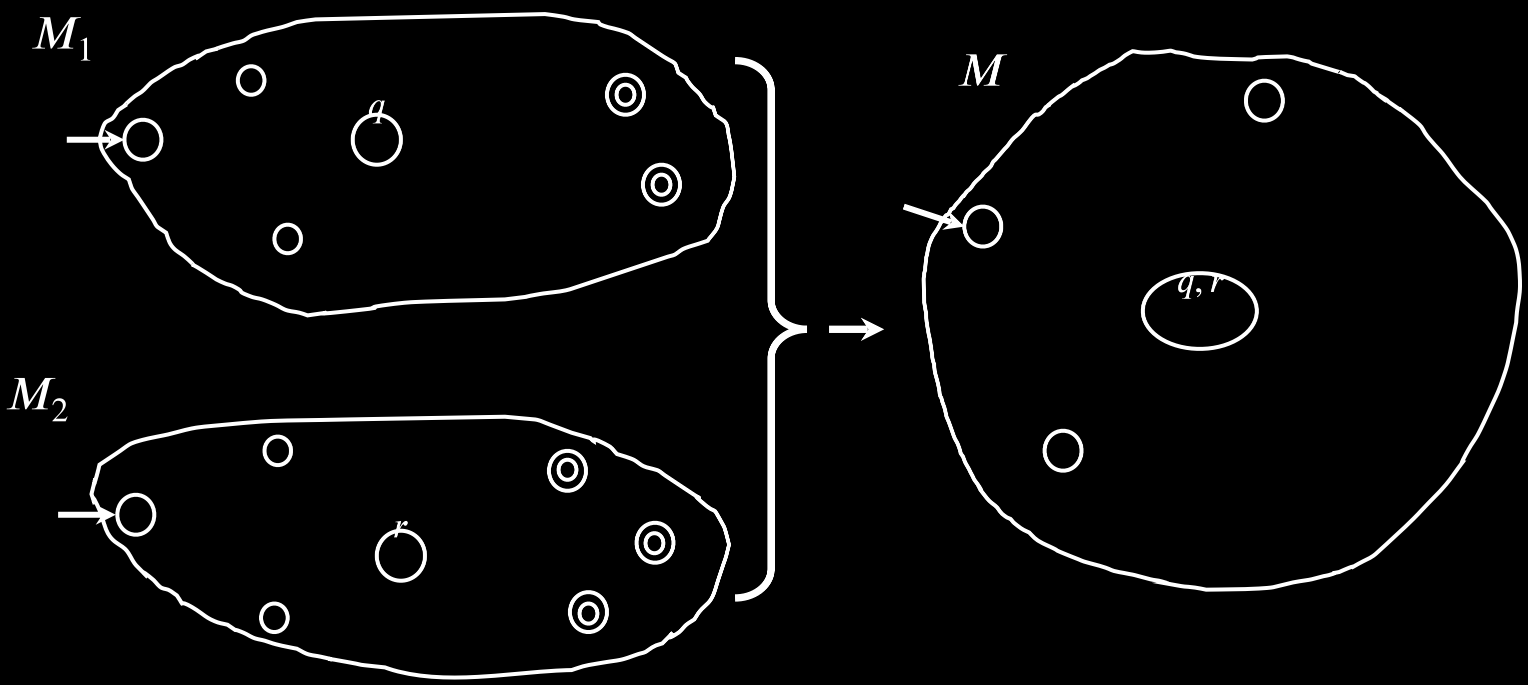
Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under $\cup$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.



Components of $\boldsymbol{M}$:

$Q = Q_1 \times Q_2$

$= \{ (q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2 \}$

$q_0 = (q_1, q_2)$

$\delta\big((q, r), a\big) = \big(\delta_1(q, a), \delta_2(r, a)\big)$

$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Closure Properties for Regular Languages

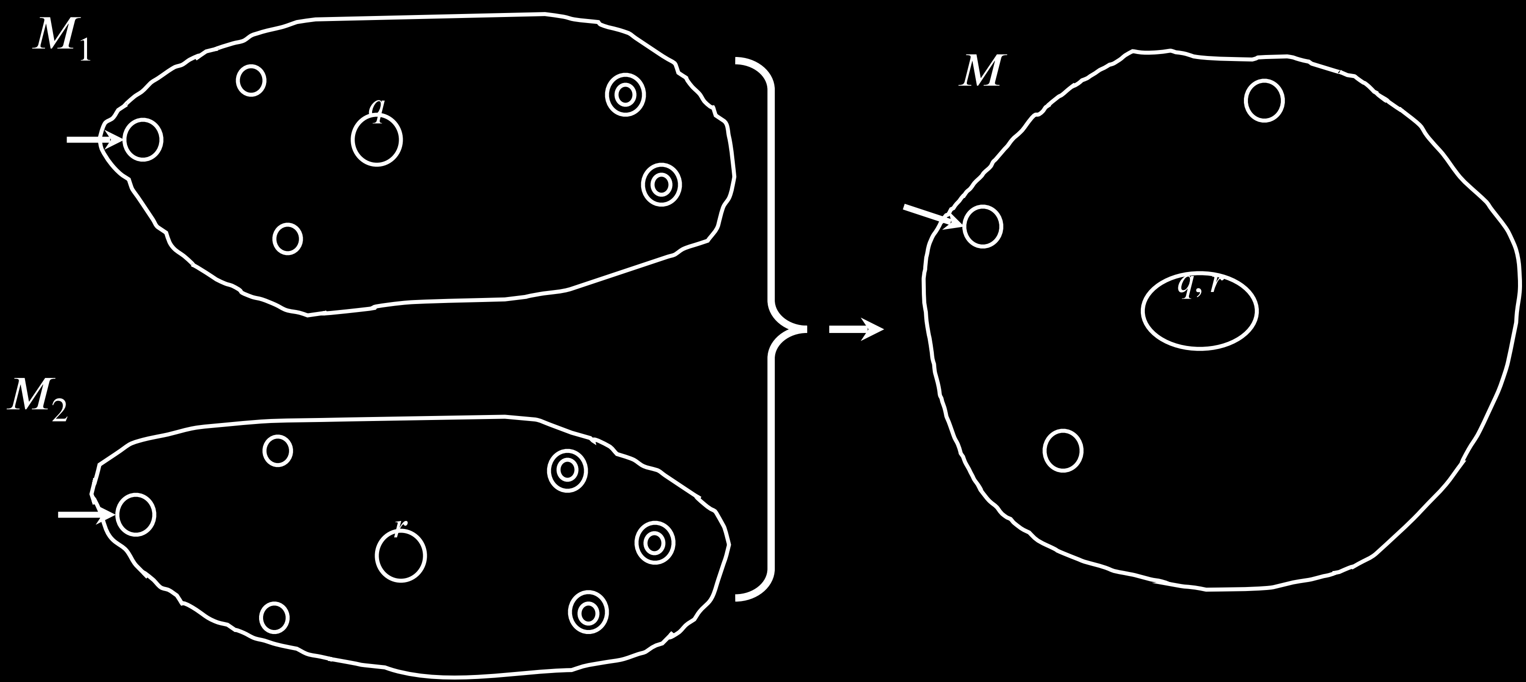
Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under $\cup$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.



Components of $\boldsymbol{M}$:

$$Q = Q_1 \times Q_2$$

$$= \left\{ (q_1, q_2) \,\middle|\, q_1 \in Q_1 \text{ and } q_2 \in Q_2 \right\}$$

$$q_0 = (q_1, q_2)$$

$$\delta\Big((q, r), a\Big) = \Big(\delta_1(q, a), \delta_2(r, a)\Big)$$

$$F = \left(F_1 \times Q_2\right) \cup \left(Q_1 \times F_2\right)$$

Check-in 1.1

Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under $\cup$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.

Check-in 1.1

In the proof, if M_1 and M_2 are finite automata where M_1 has k_1 states and M_2 has k_2 states Then how many states does M have?

(a) $k_1 + k_2$

(b) $(k_1)^2 + (k_2)^2$

(c) $k_1 \times k_2$

Components of $\boldsymbol{M}$:

$$Q = Q_1 \times Q_2$$
$$= \left\{ (q_1, q_2) \,\middle|\, q_1 \in Q_1 \text{ and } q_2 \in Q_2 \right\}$$

$$q_0 = (q_1, q_2)$$

$$\delta\big((q, r), a\big) = \big(\delta_1(q, a), \delta_2(r, a)\big)$$

$$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$$

Check-in 1.1

Closure Properties continued

Closure Properties continued

Theorem: If $A_1,\ A_2$ are regular languages, so is A_1A_2 (closure under $\circ$)

Closure Properties continued

Theorem: If $A_1, \ A_2$ are regular languages, so is $A_1 A_2$ (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \ \Sigma, \ \ \delta_1, \ \ q_1, \ \ F_1)$ recognize A_1

$M_2 = (Q_2, \ \Sigma, \ \ \delta_2, \ \ q_2, \ \ F_2)$ recognize A_2

Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under ∘)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2

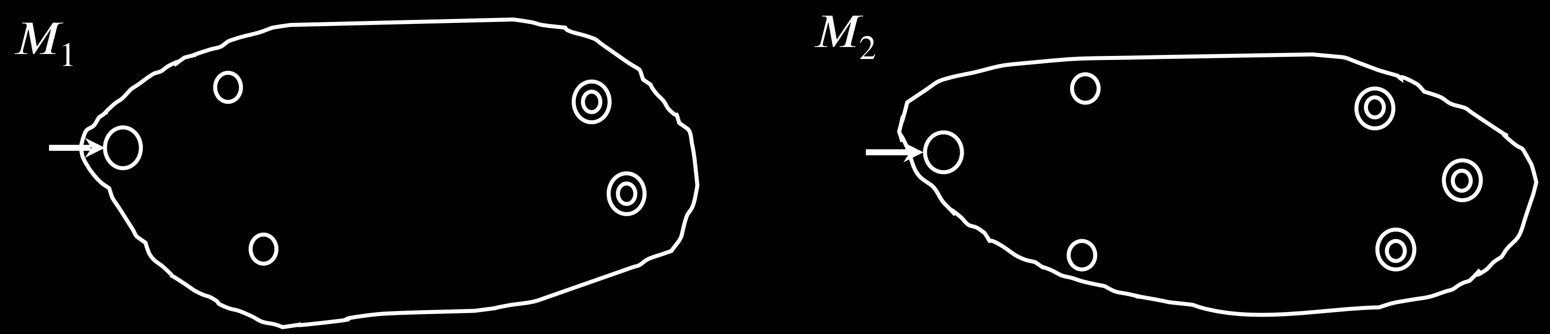
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is $A_1 A_2$ (closure under $\circ$)

Proof: Let $M_1 = (Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1)$ recognize A_1

$M_2 = (Q_2,\ \Sigma,\ \delta_2,\ q_2,\ F_2)$ recognize A_2

Construct $M = (Q,\ \Sigma,\ \delta,\ q0,\ F)$ recognizing $A_1 A_2$



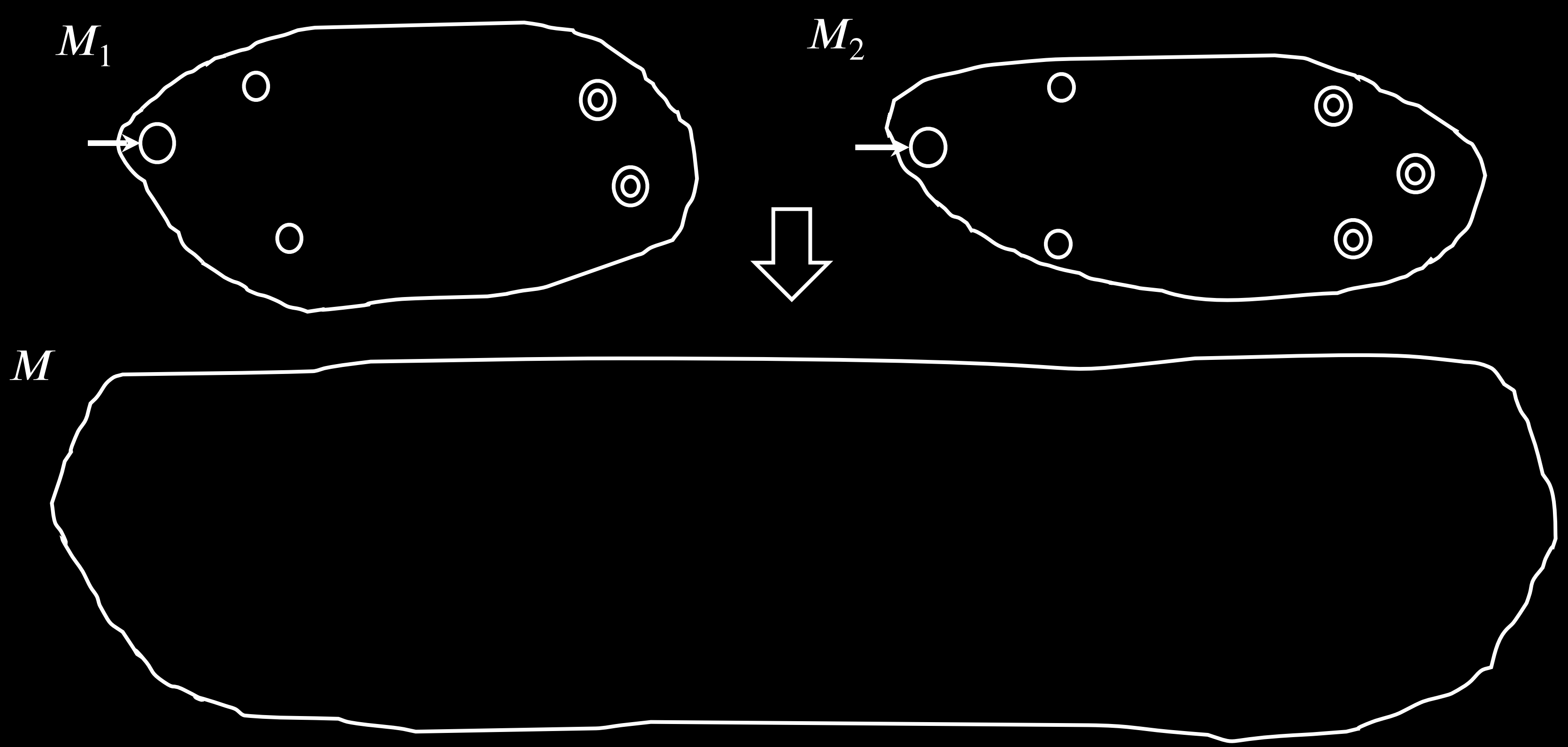
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is $A_1 A_2$ (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 A_2$



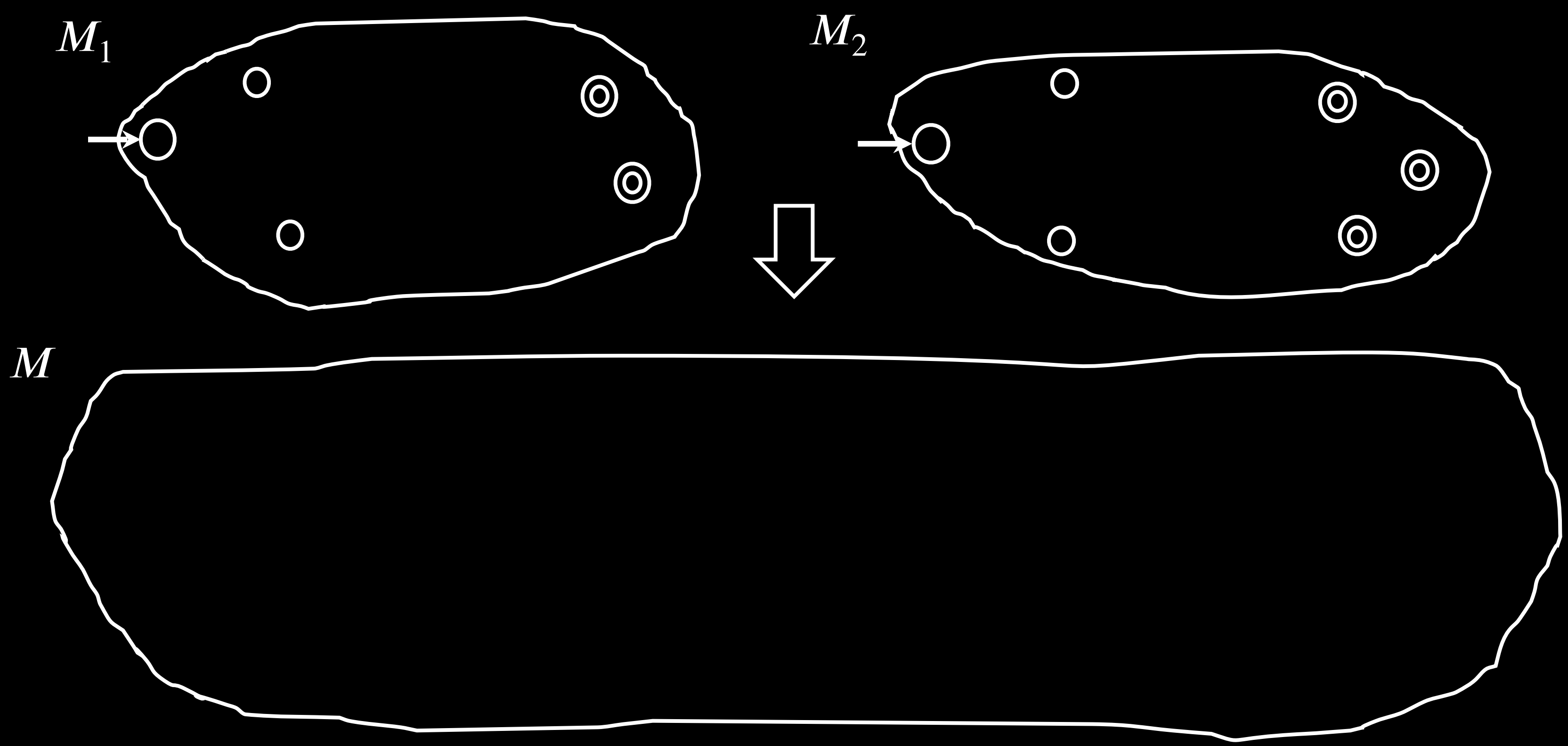
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2



M should accept input w

if $w = xy$ where

M_1 accepts x and M_2 accepts y.

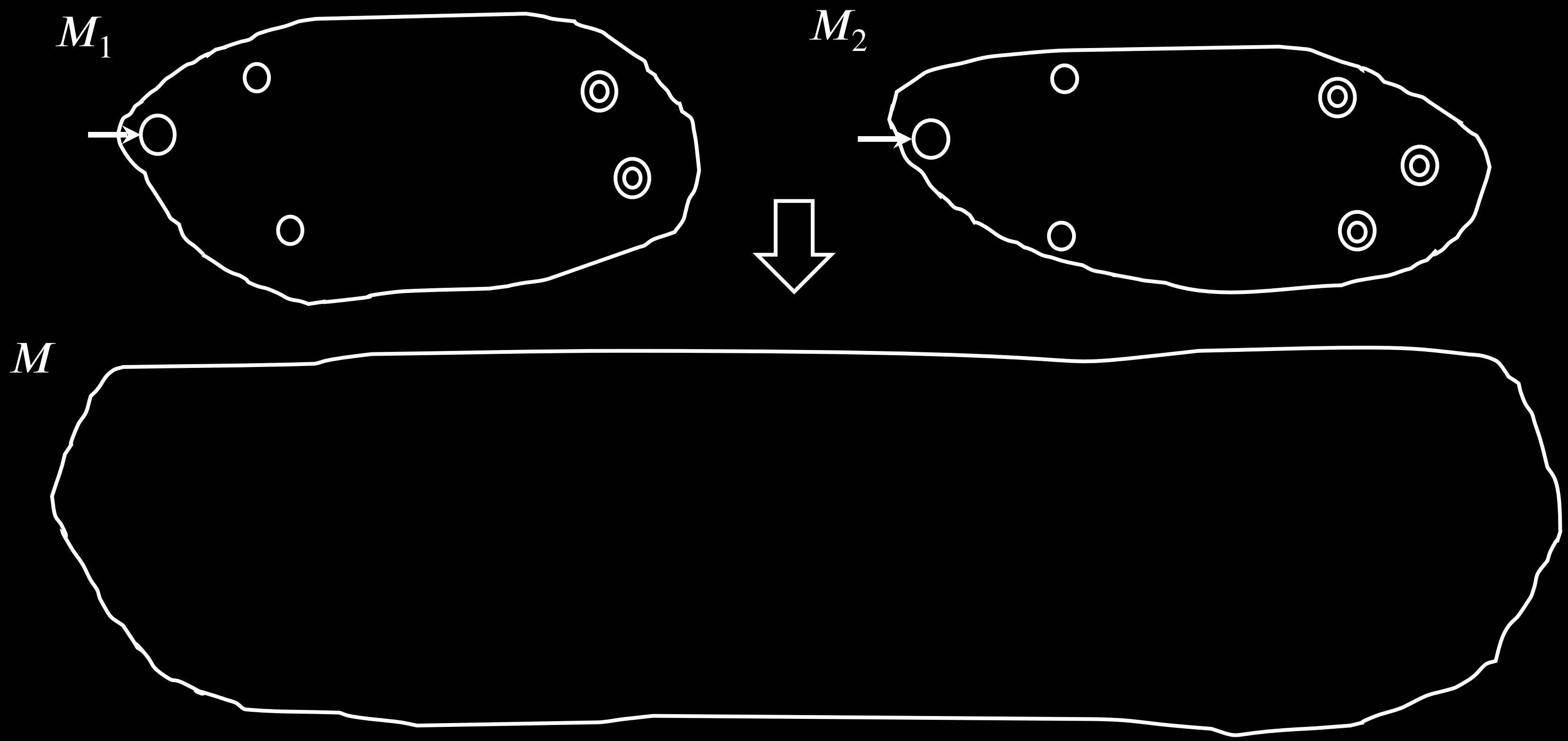
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is $A_1 A_2$ (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 A_2$



M should accept input w

if $w = xy$ where

M_1 accepts x and M_2 accepts y.

w ____________________

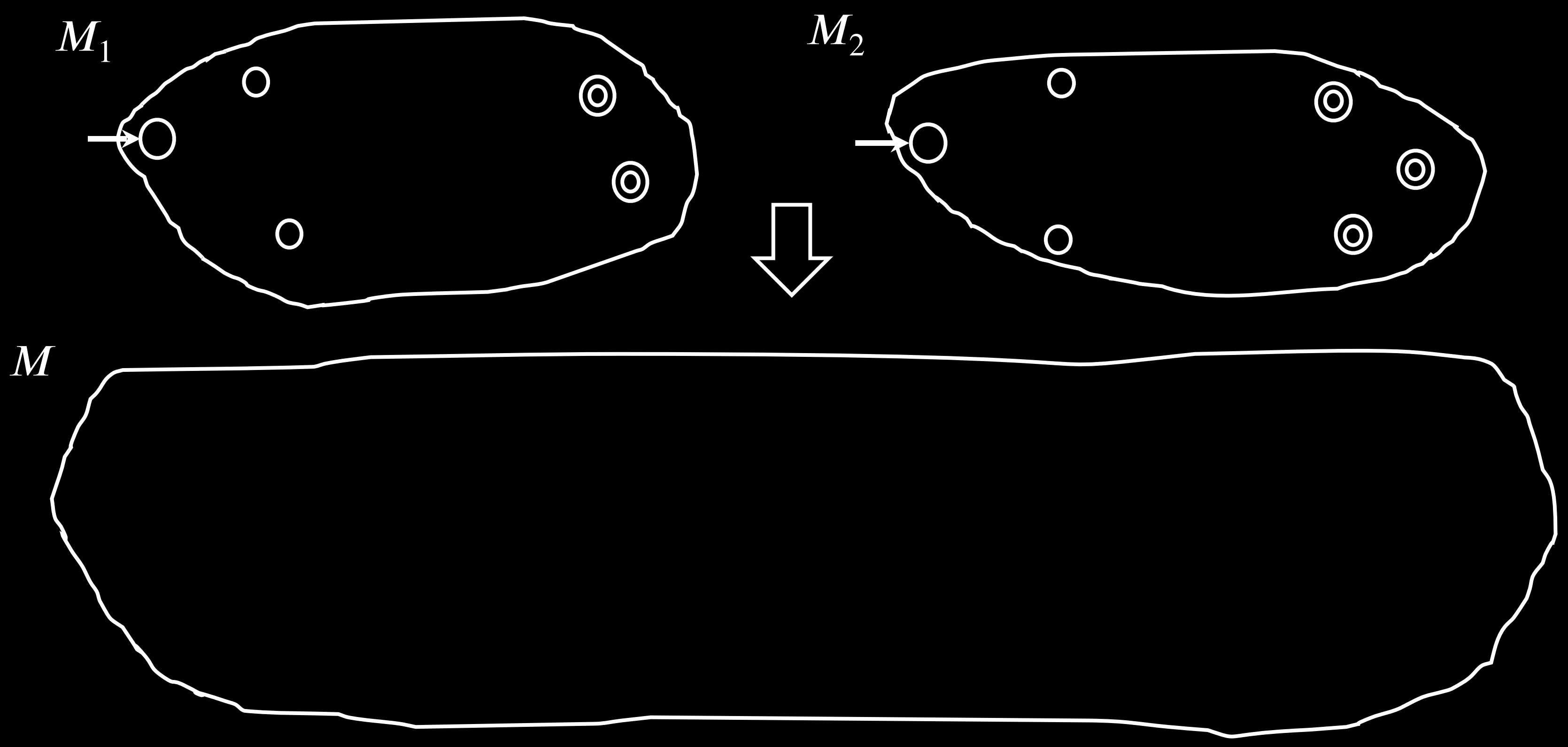
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2



M should accept input w

if $w = xy$ where

M_1 accepts x and M_2 accepts y.

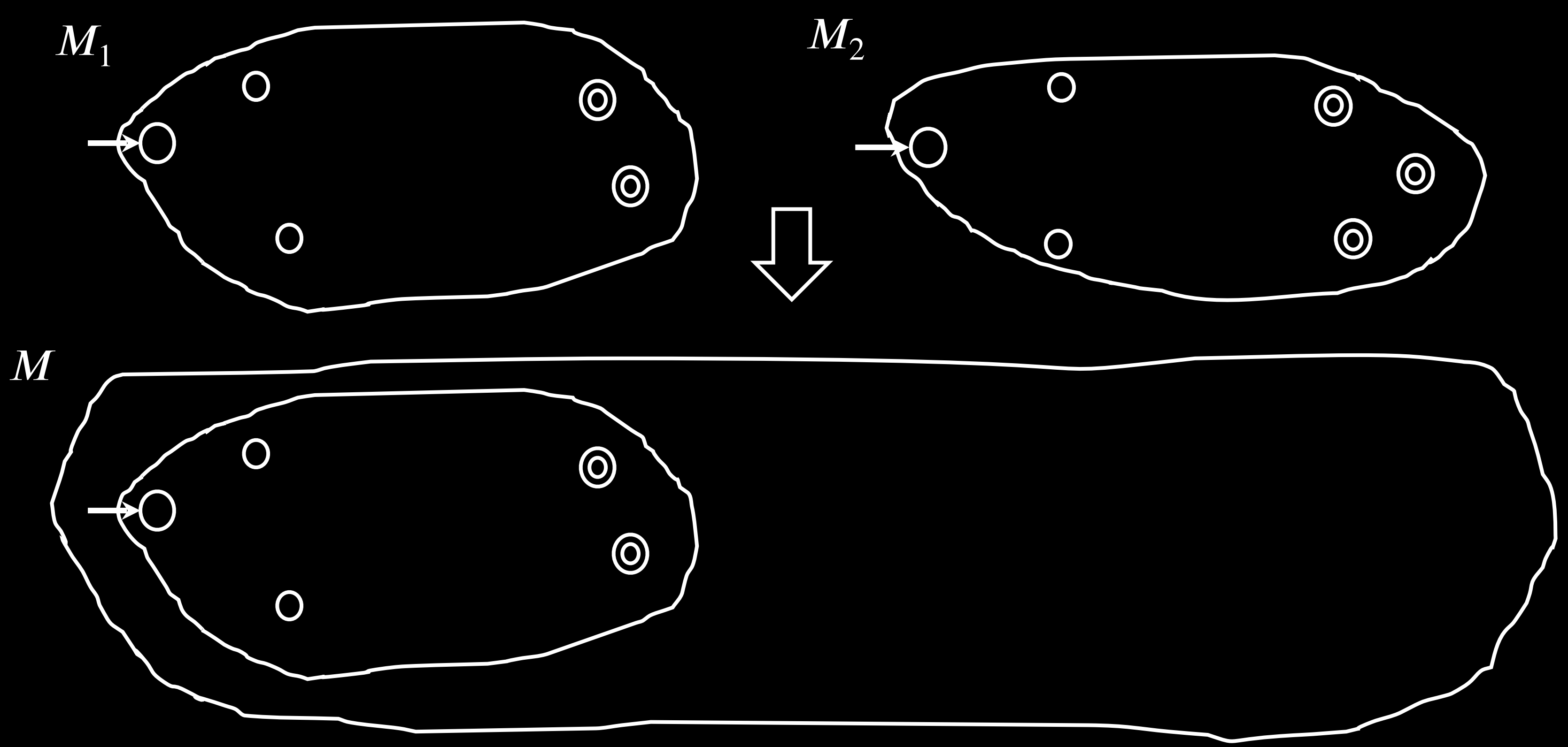
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is $A_1 A_2$ (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 A_2$



M should accept input w
if $w = xy$ where
M_1 accepts x and M_2 accepts y.

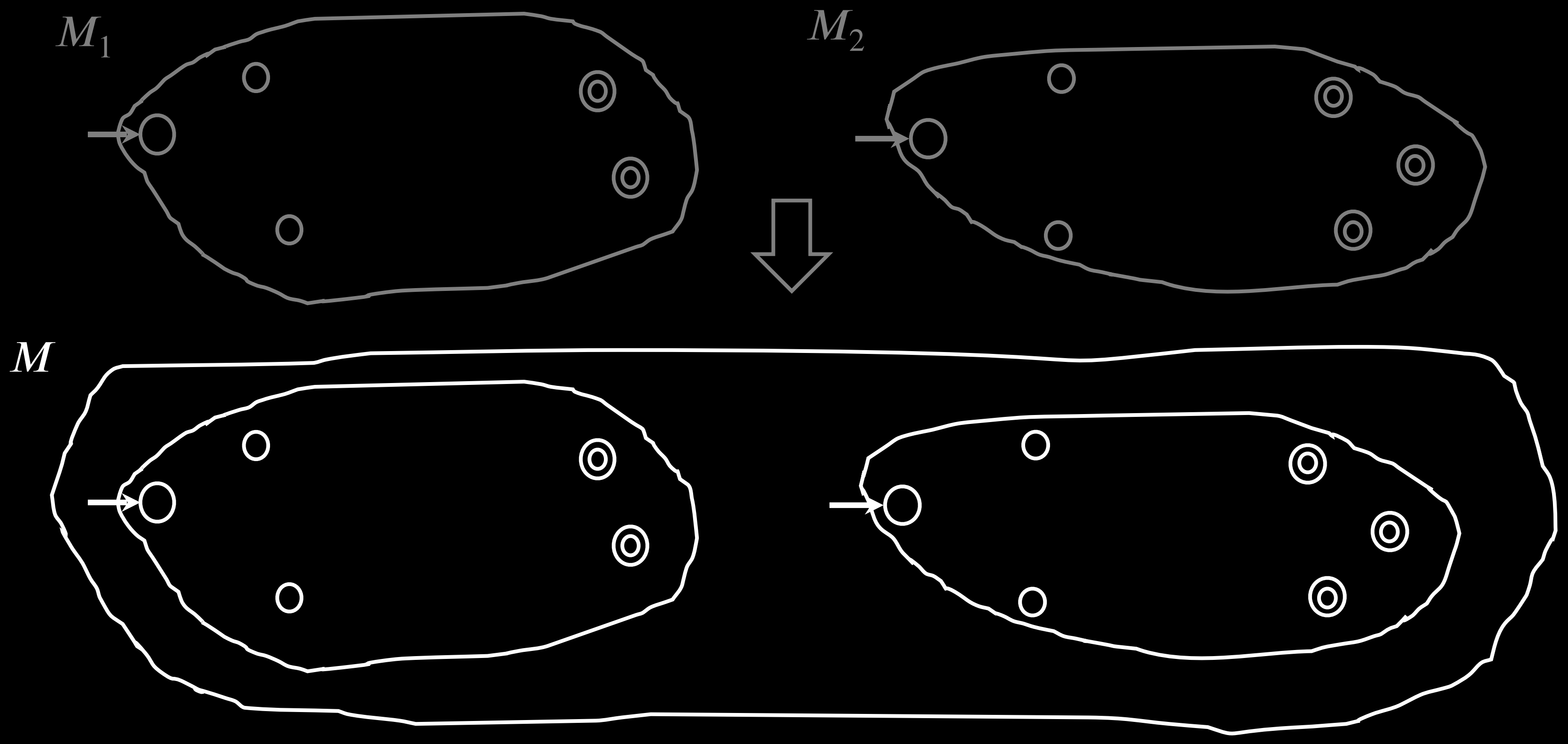
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2



M should accept input w

if $w = xy$ where

M_1 accepts x and M_2 accepts y.

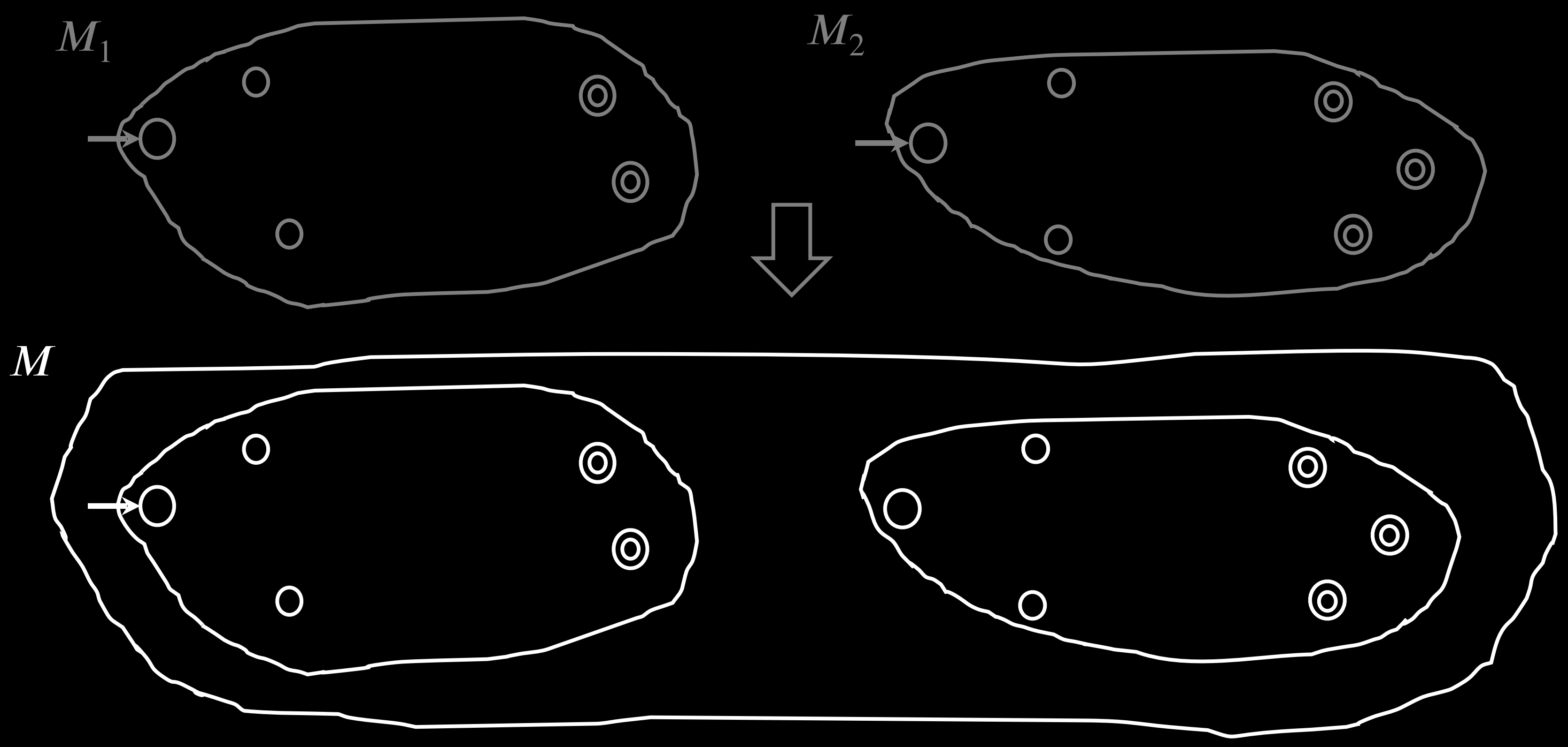
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2



M should accept input w
if $w = xy$ where
M_1 accepts x and M_2 accepts y.

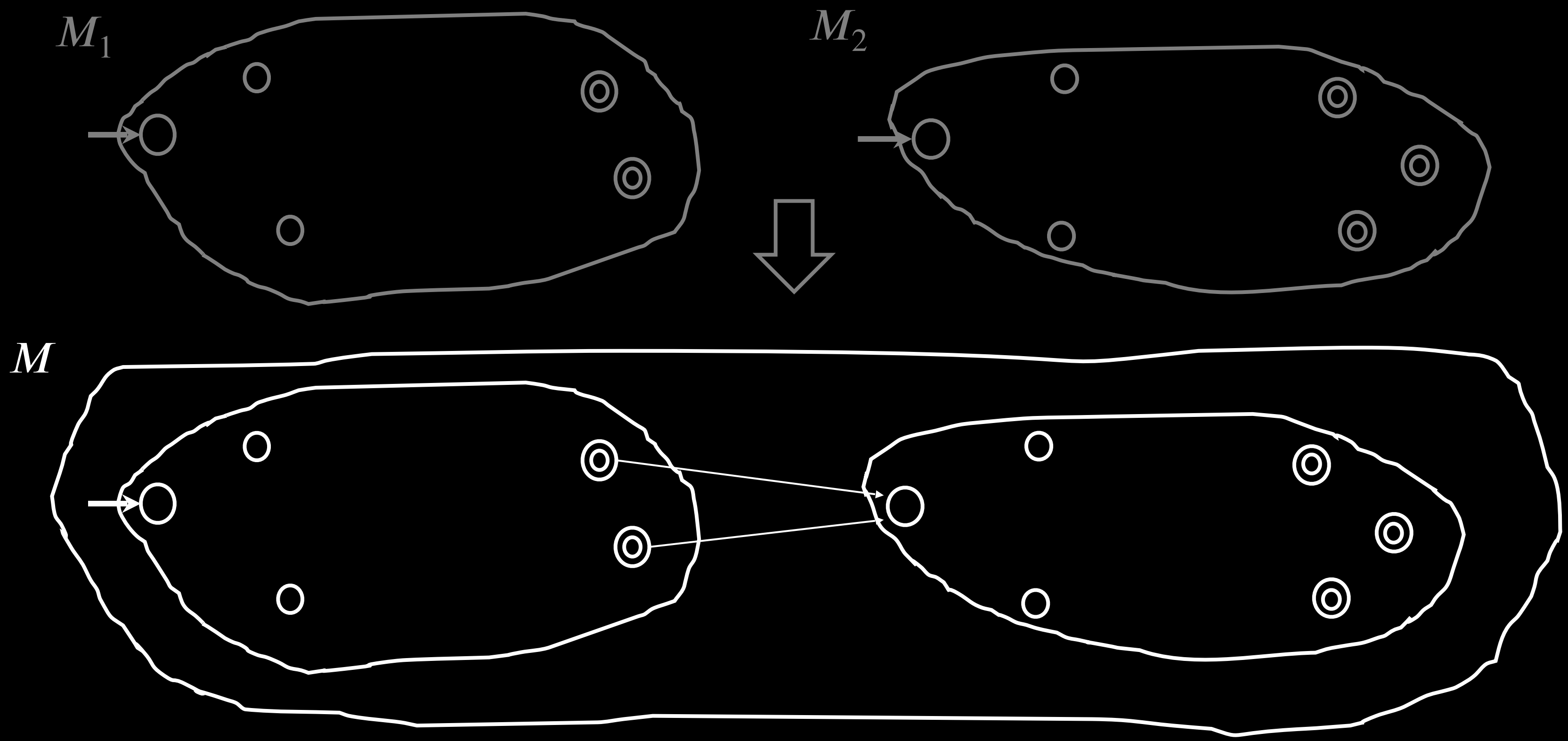
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is $A_1 A_2$ (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing $A_1 A_2$



M should accept input w

if $w = xy$ where

M_1 accepts x and M_2 accepts y.

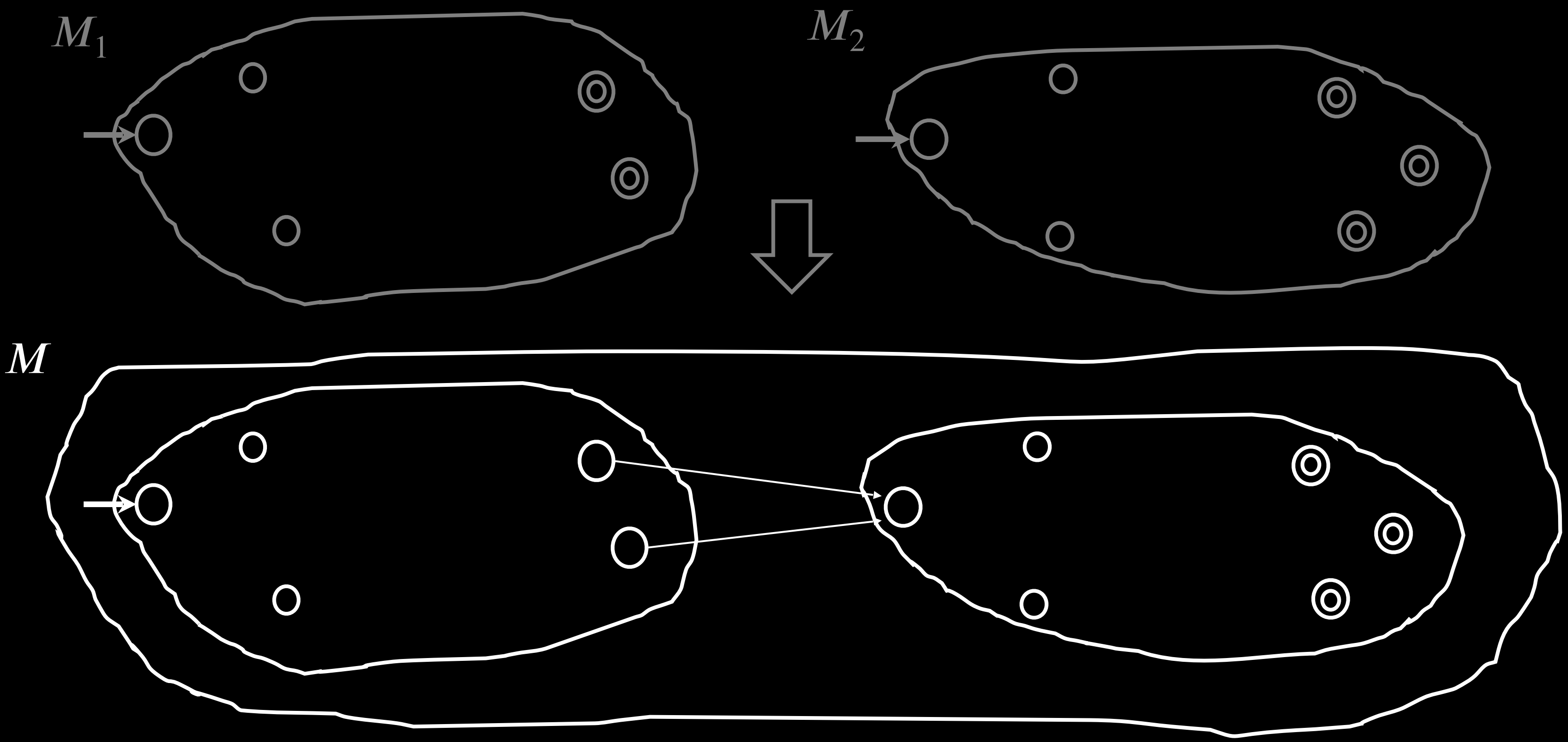
Closure Properties continued

Theorem: If $A_1,\ A_2$ are regular languages, so is A_1A_2 (closure under $\circ$)

Proof: Let $M_1 = (Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1)$ recognize A_1

$M_2 = (Q_2,\ \Sigma,\ \delta_2,\ q_2,\ F_2)$ recognize A_2

Construct $M = (Q,\ \Sigma,\ \delta,\ q0,\ F)$ recognizing A_1A_2



M should accept input w

if $w = xy$ where

M_1 accepts x and M_2 accepts y.

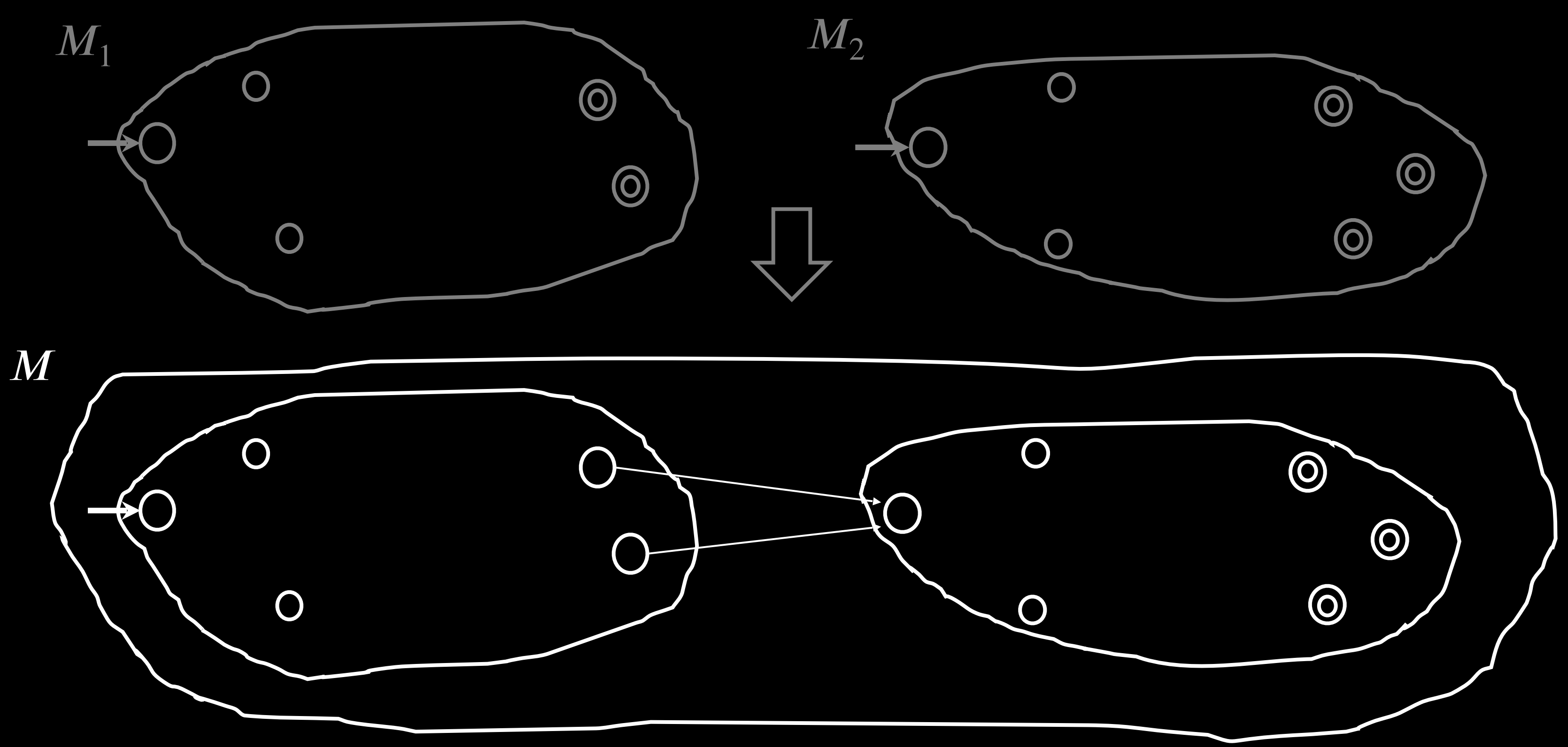
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2



M should accept input w
if $w = xy$ where
M_1 accepts x and M_2 accepts y.

Doesn't work: Where to split w?

Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is $A_1 A_2$ (closure under ∘)

Recall proof attempt: Let $M_1 = (Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1)$ recognize A_1

$M_2 = (Q_2,\ \Sigma,\ \delta_2,\ q_2,\ F_2)$ recognize A_2

Construct $M = (Q,\ \Sigma,\ \delta,\ q0,\ F)$ recognizing $A_1 A_2$

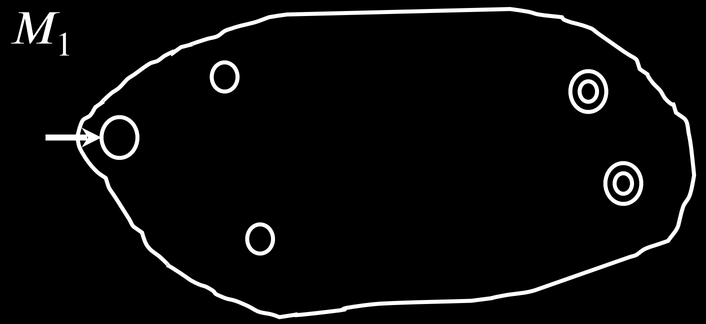
Closure Properties for Regular Languages

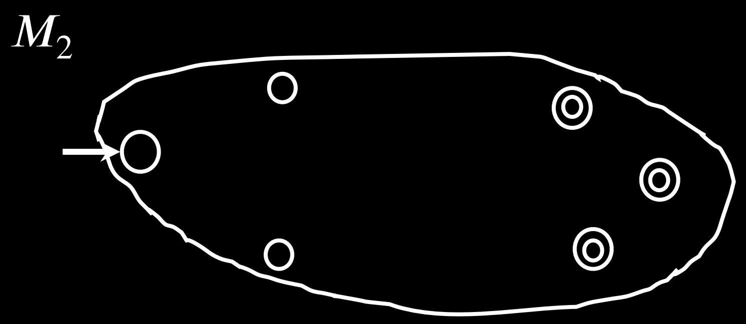
Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Recall proof attempt: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2





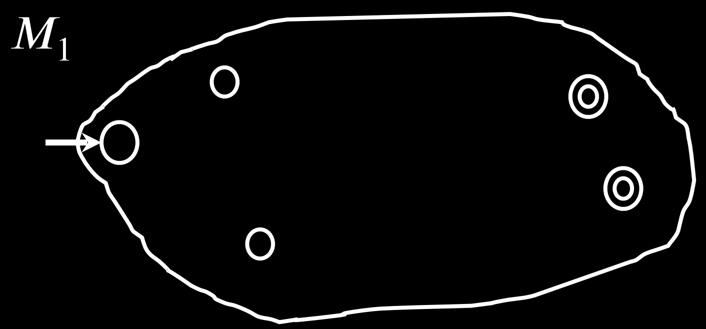
Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Recall proof attempt: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2



M should accept input w if $w = xy$ where M_1 accepts x and M_2 accepts y.

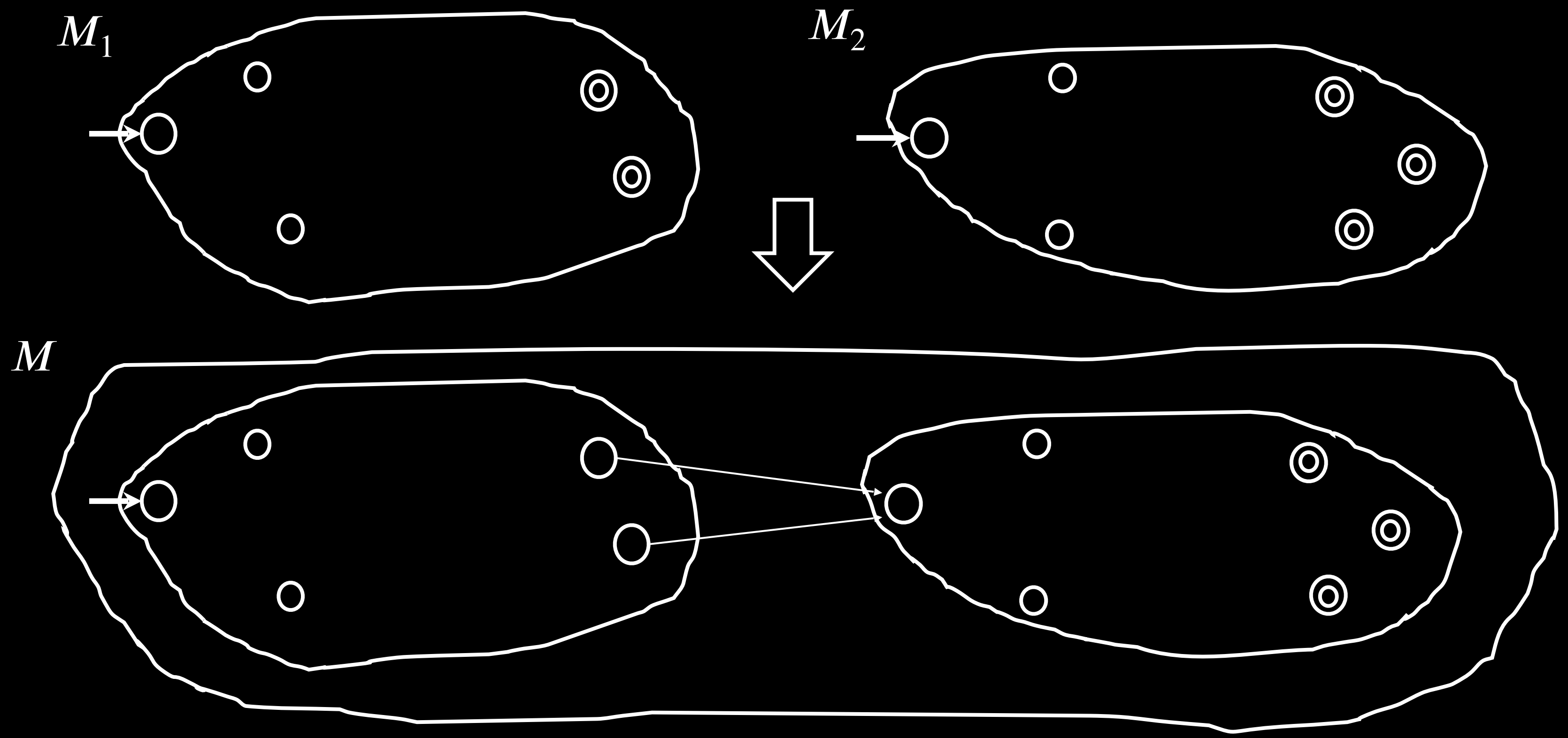
Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Recall proof attempt: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2



M should accept input w
if $w = xy$ where
M_1 accepts x and M_2 accepts y.

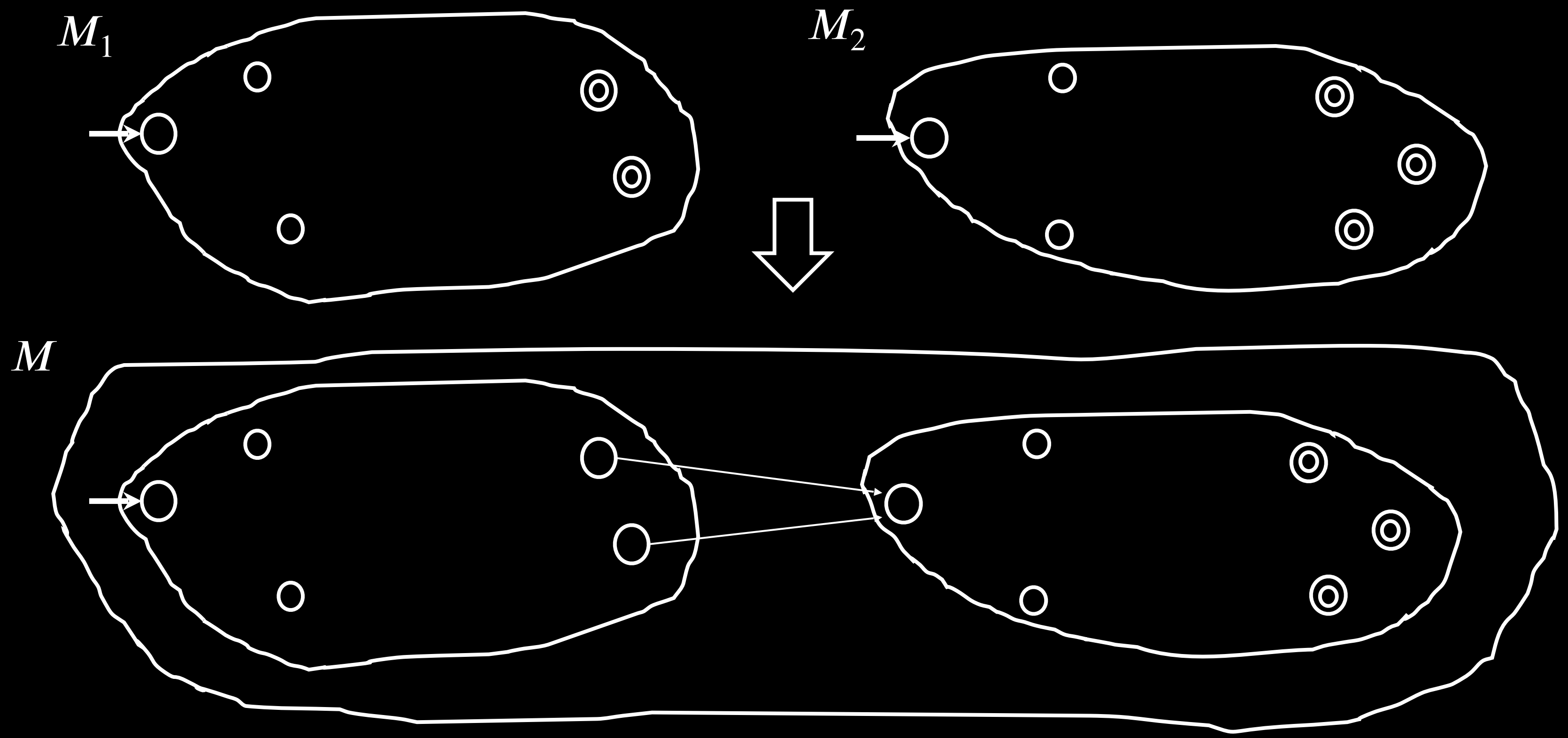
Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Recall proof attempt: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2



M should accept input w
if $w = xy$ where
M_1 accepts x and M_2 accepts y.

Doesn't work: Where to split w?

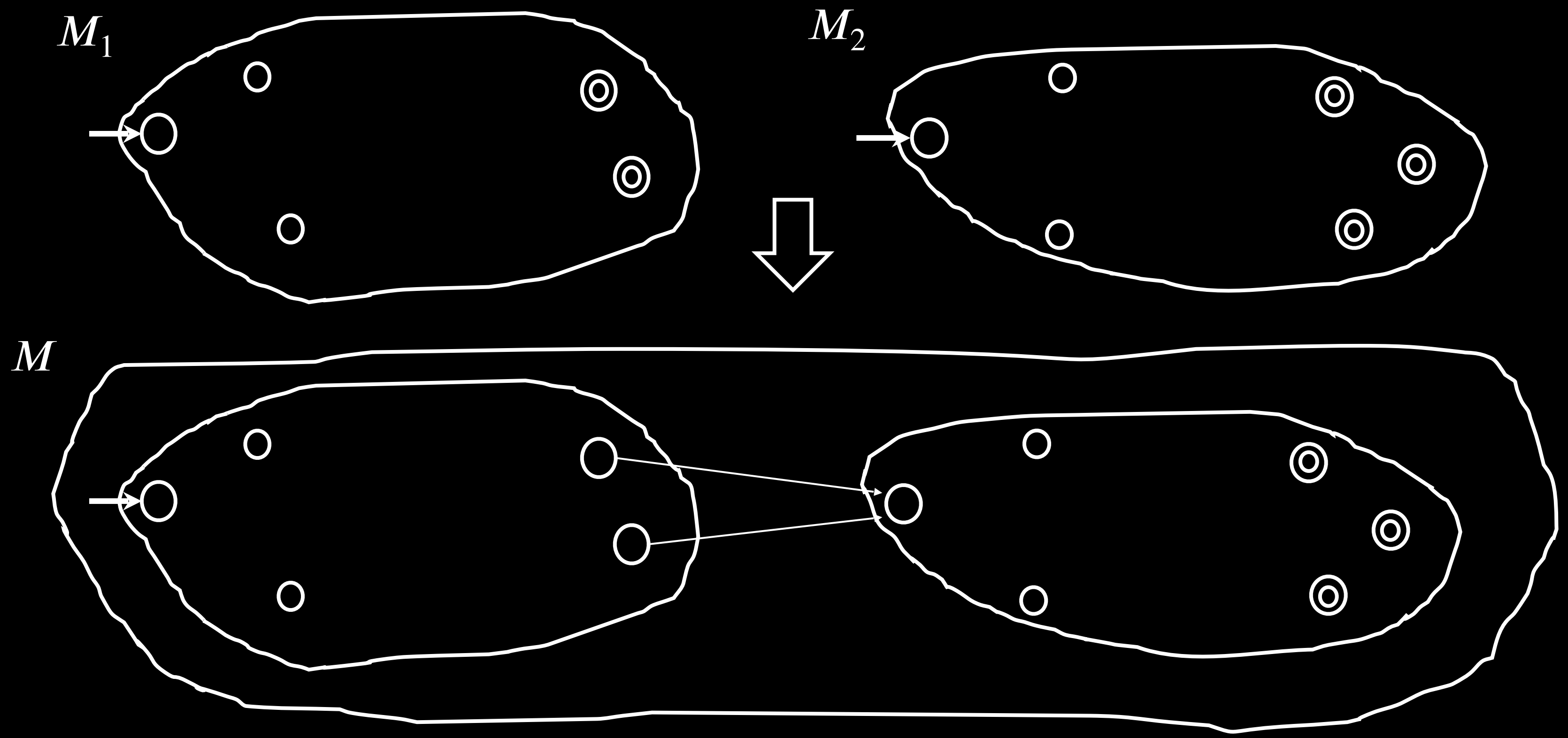
Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Recall proof attempt: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q0, F)$ recognizing A_1A_2



M should accept input w
if $w = xy$ where
M_1 accepts x and M_2 accepts y.

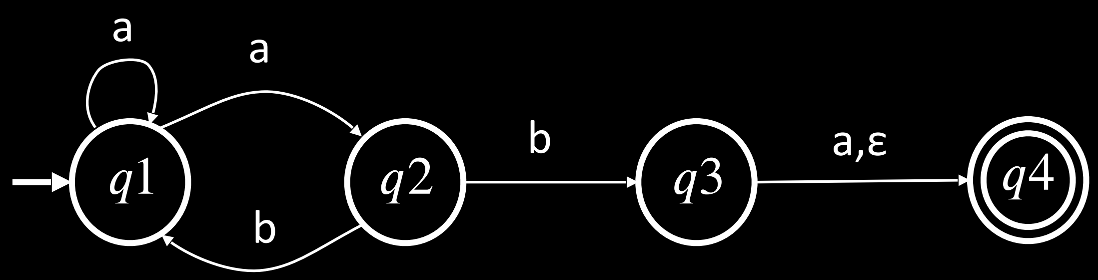
Doesn't work: Where to split w?

Hold off. Need new concept.

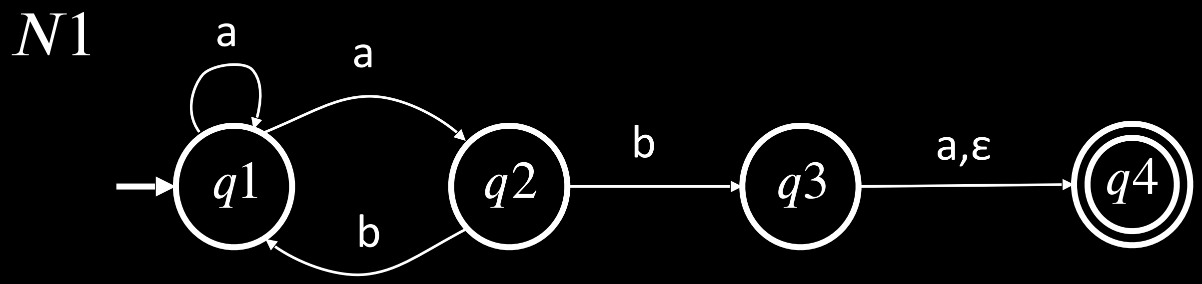
Nondeterministic Finite Automata

Nondeterministic Finite Automata

$N1$



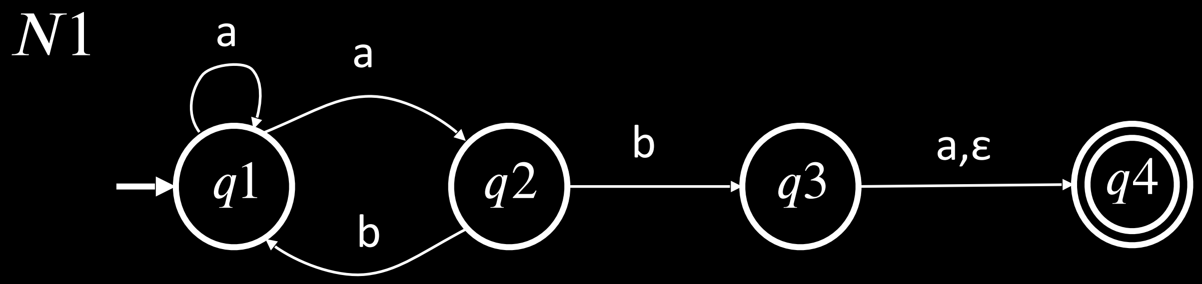
Nondeterministic Finite Automata



New features of nondeterminism:

- multiple paths possible (0, 1 or many at each step)

Nondeterministic Finite Automata

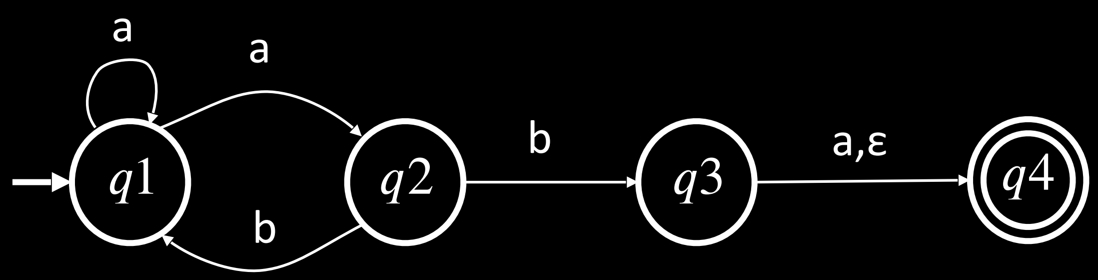


New features of nondeterminism:

- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input

Nondeterministic Finite Automata

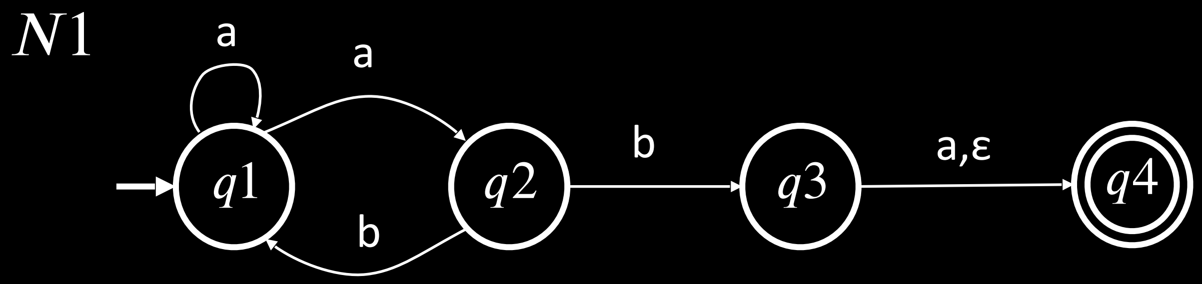
$N1$



New features of nondeterminism:

- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input
- Accept input if <u>some</u> path leads to ◎ accept

Nondeterministic Finite Automata



New features of nondeterminism:

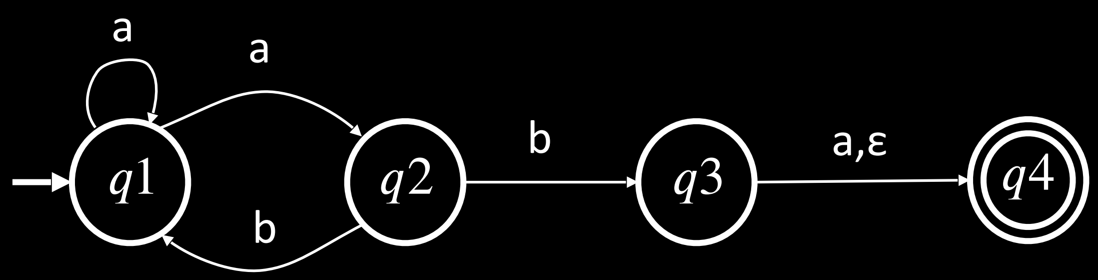
- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input
- Accept input if <u>some</u> path leads to ◎ accept

Example inputs:

- ab

Nondeterministic Finite Automata

$N1$



New features of nondeterminism:

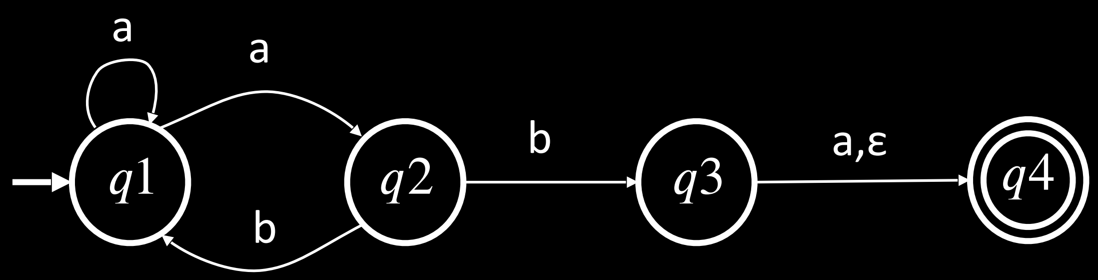
- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input
- Accept input if <u>some</u> path leads to accept

Example inputs:

- ab <u>accept</u>

Nondeterministic Finite Automata

$N1$



New features of nondeterminism:

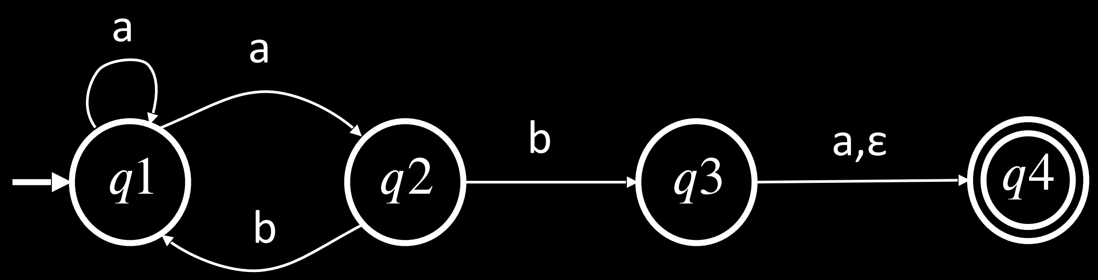
- multiple paths possible (0, 1 or many at each step)
- ε-transition is a “free” move without reading input
- Accept input if <u>some</u> path leads to  accept

Example inputs:

- ab <u>accept</u>
- aa

Nondeterministic Finite Automata

$N1$



New features of nondeterminism:

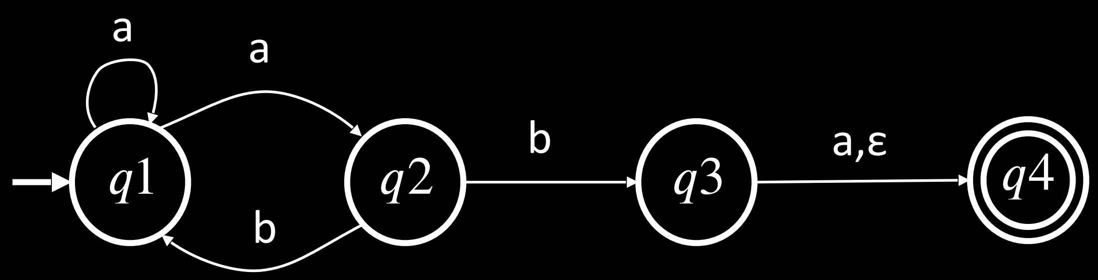
- multiple paths possible (0, 1 or many at each step)
- ε-transition is a “free” move without reading input
- Accept input if <u>some</u> path leads to accept

Example inputs:

- ab <u>accept</u>
- aa <u>reject</u>

Nondeterministic Finite Automata

$N1$



New features of nondeterminism:

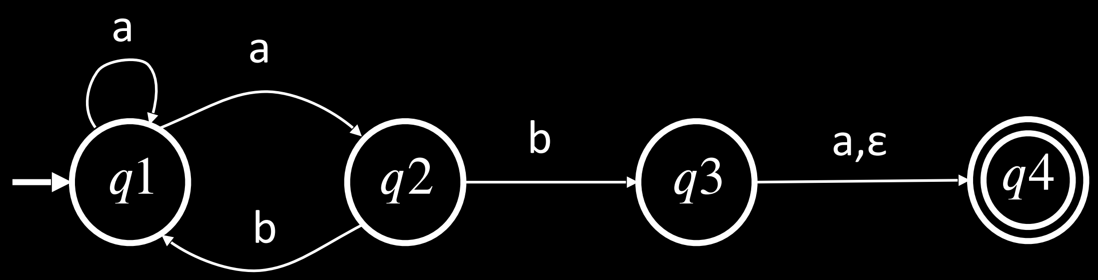
- multiple paths possible (0, 1 or many at each step)
- ε-transition is a “free” move without reading input
- Accept input if <u>some</u> path leads to ◎ accept

Example inputs:

- ab <u>accept</u>
- aa <u>reject</u>
- aba

Nondeterministic Finite Automata

$N1$



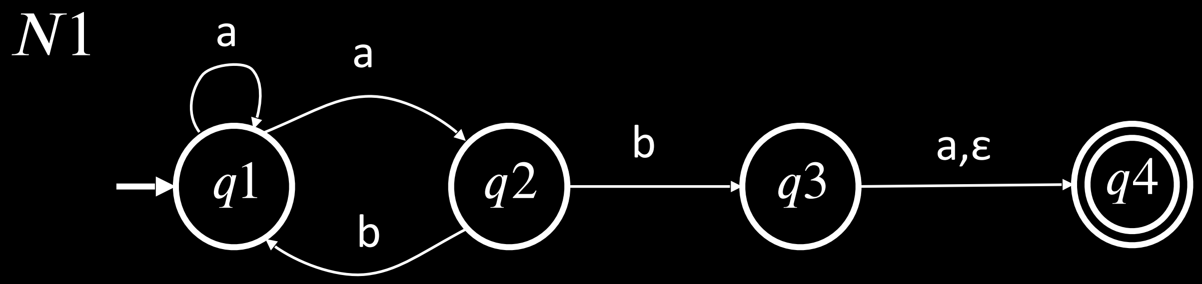
New features of nondeterminism:

- multiple paths possible (0, 1 or many at each step)
- ε-transition is a “free” move without reading input
- Accept input if <u>some</u> path leads to ◎ accept

Example inputs:

- ab <u>accept</u>
- aa <u>reject</u>
- aba <u>accept</u>

Nondeterministic Finite Automata



New features of nondeterminism:

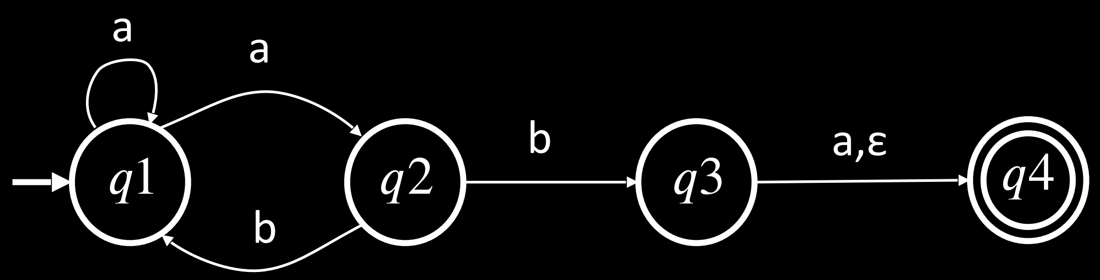
- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input
- Accept input if <u>some</u> path leads to ◎ accept

Example inputs:

- ab <u>accept</u>
- aa <u>reject</u>
- aba <u>accept</u>
- abb

Nondeterministic Finite Automata

$N1$



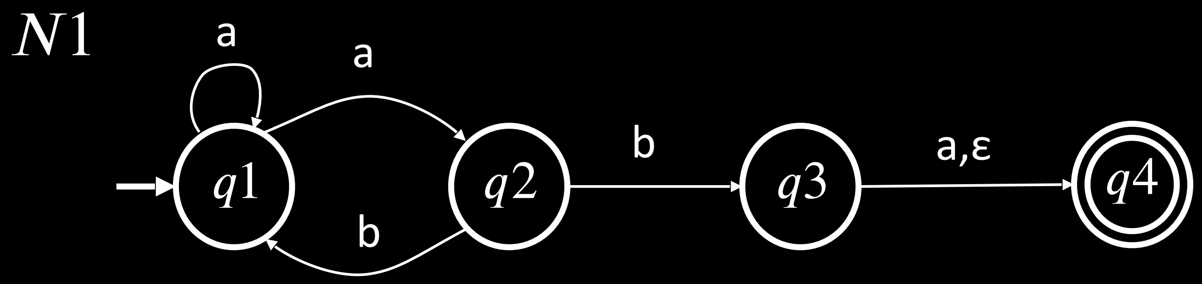
New features of nondeterminism:

- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input
- Accept input if <u>some</u> path leads to ◎ accept

Example inputs:

- ab <u>accept</u>
- aa <u>reject</u>
- aba <u>accept</u>
- abb <u>reject</u>

Nondeterministic Finite Automata



Nondeterminism doesn't correspond to a physical machine we can build. However, it is useful mathematically.

New features of nondeterminism:

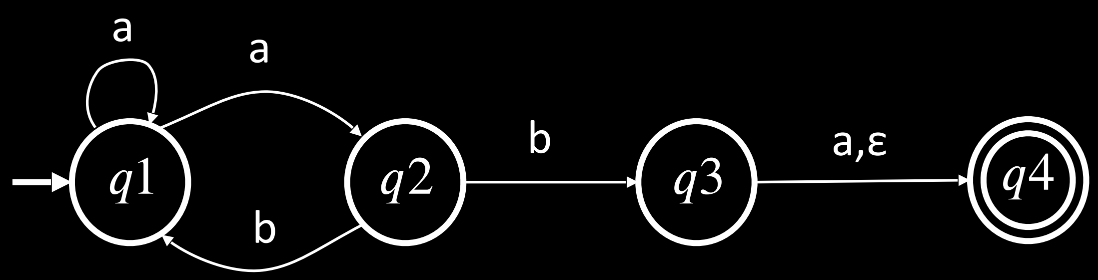
- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input
- Accept input if <u>some</u> path leads to accept

Example inputs:

- ab <u>accept</u>
- aa <u>reject</u>
- aba <u>accept</u>
- abb <u>reject</u>

Nondeterministic Finite Automata

$N1$



Nondeterminism doesn't correspond to a physical machine we can build. However, it is useful mathematically.

New features of nondeterminism:

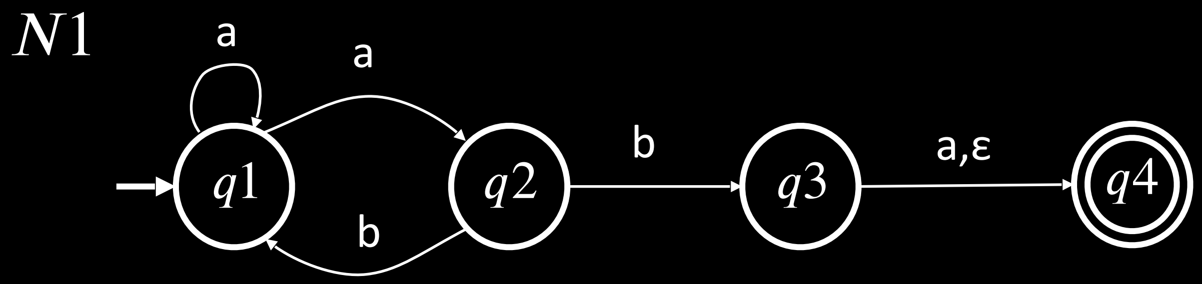
- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input
- Accept input if <u>some</u> path leads to accept

Example inputs:

- ab <u>accept</u>
- aa <u>reject</u>
- aba <u>accept</u>
- abb <u>reject</u>

Check-in 2.1

Nondeterministic Finite Automata



Nondeterminism doesn't correspond to a physical machine we can build. However, it is useful mathematically.

New features of nondeterminism:

- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input
- Accept input if <u>some</u> path leads to accept

Example inputs:

- ab <u>accept</u>
- aa <u>reject</u>
- aba <u>accept</u>
- abb <u>reject</u>

Check-in 2.1

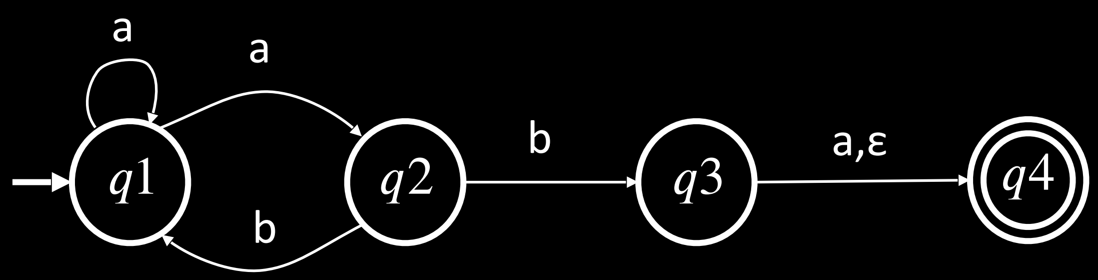
What does N_1 do on input aab ?

(a) Accept
(b) Reject
(c) Both Accept and Reject

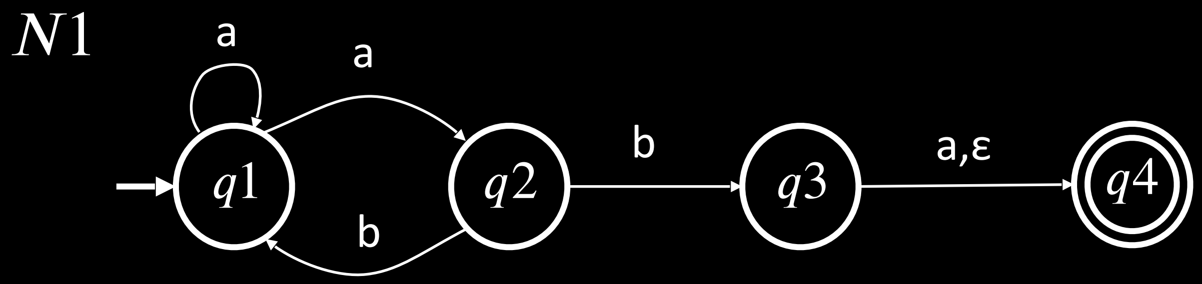
Check-in 2.1

NFA – Formal Definition

$N1$



NFA – Formal Definition

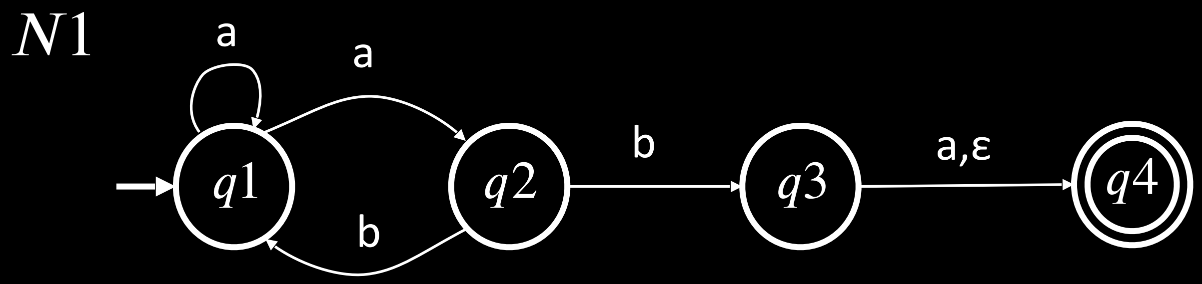


Defn: A <u>nondeterministic finite automaton (NFA)</u>

N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states alphabet transition function start state accept states

NFA – Formal Definition



Defn: A <u>nondeterministic finite automaton (NFA)</u>

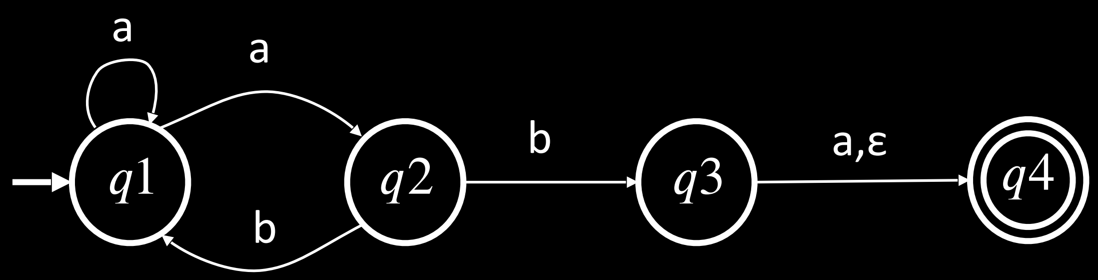
N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states
alphabet
transition function
start state
accept states

- all same as before except δ

NFA – Formal Definition

$N1$



Defn: A <u>nondeterministic finite automaton (NFA)</u>

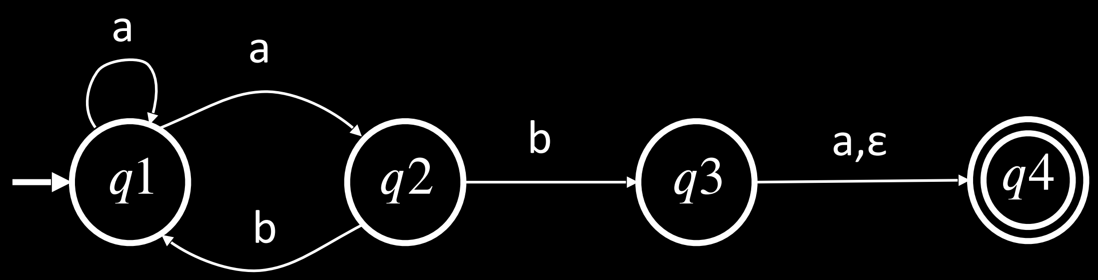
N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states, alphabet, transition function, start state, accept states

- all same as before except δ
- $\delta : Q \times \Sigma_{\varepsilon} \to \mathscr{P}(Q) = \{R \mid R \subseteq Q\}$

NFA – Formal Definition

$N1$



Defn: A <u>nondeterministic finite automaton (NFA)</u>

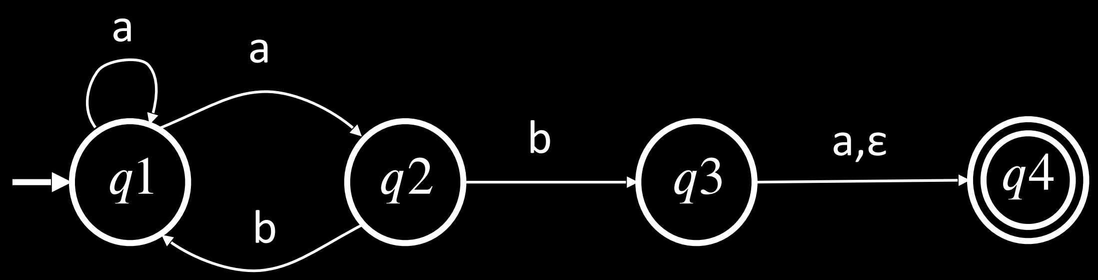
N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states
alphabet
transition function
start state
accept states

- all same as before except δ
- $\delta : Q \times \underbrace{\Sigma_\varepsilon}_{\Sigma \cup \{\varepsilon\}} \to \mathscr{P}(Q) = \{R \mid R \subseteq Q\}$

NFA – Formal Definition

$N1$



Defn: A <u>nondeterministic finite automaton (NFA)</u>

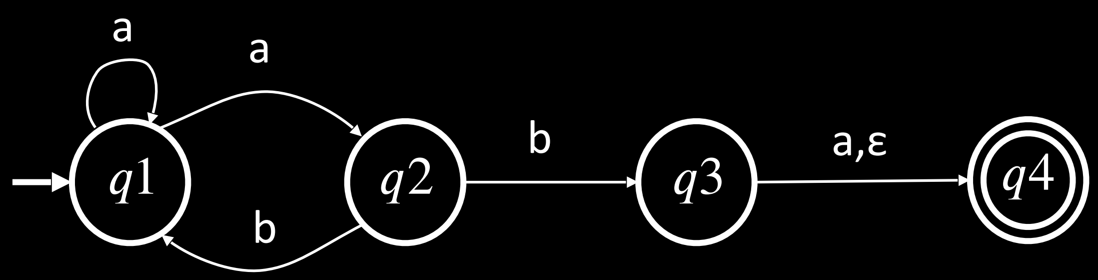
N is a 5-tuple $(Q, \ \Sigma, \ \delta, \ q0, \ F)$

states, alphabet, transition function, start state, accept states

- all same as before except δ
- $\delta : Q \times \underbrace{\Sigma_{\varepsilon}}_{\Sigma \cup \{\varepsilon\}} \rightarrow \underbrace{\mathscr{P}(Q)}_{\text{power set}} = \{R \mid R \subseteq Q\}$

NFA – Formal Definition

$N1$



Defn: A <u>nondeterministic finite automaton (NFA)</u>

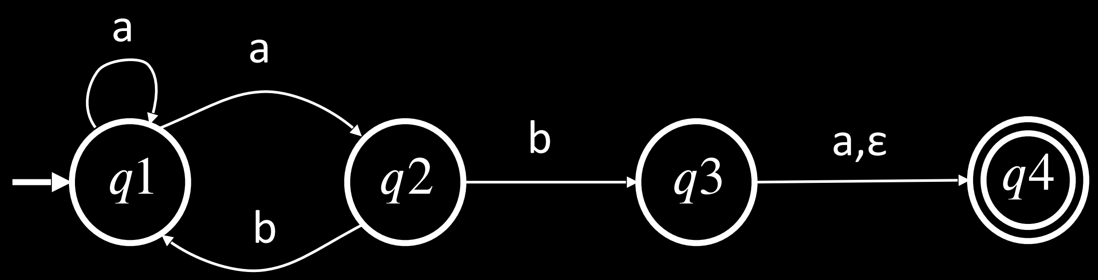
N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states, alphabet, transition function, start state, accept states

- all same as before except δ
- $\delta : Q \times \underbrace{\Sigma_\varepsilon}_{\Sigma \cup \{\varepsilon\}} \rightarrow \underbrace{\mathcal{P}(Q)}_{\text{power set}} = \{R \mid R \subseteq Q\}$
- In the N_1 example: $\delta(q_1, \mathrm{a}) = \{q_1, q_2\}$

NFA – Formal Definition

$N1$



Defn: A <u>nondeterministic finite automaton (NFA)</u>

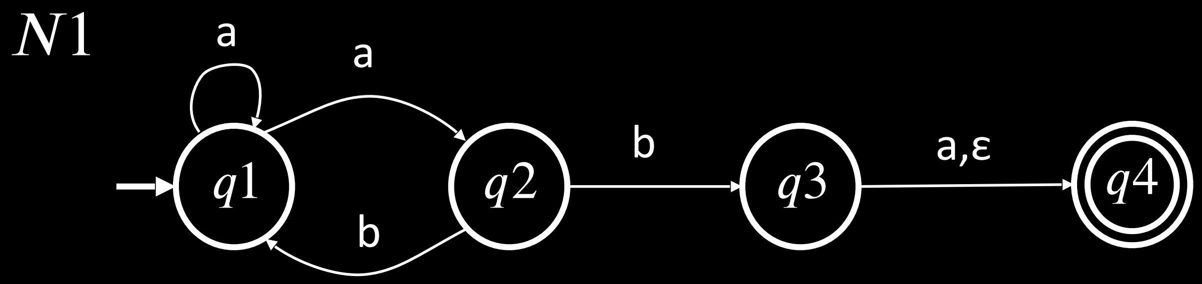
N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states, alphabet, transition function, start state, accept states

- all same as before except δ
- $\delta : Q \times \underbrace{\Sigma_{\varepsilon}}_{\Sigma \cup \{\varepsilon\}} \rightarrow \underbrace{\mathscr{P}(Q)}_{\text{power set}} = \{R \mid R \subseteq Q\}$
- In the N_1 example: $\delta(q_1, \mathrm{a}) = \{q_1, q_2\}$

  $\delta(q_1, \mathrm{b}) = \varnothing$

NFA – Formal Definition



Defn: A <u>nondeterministic finite automaton (NFA)</u>

N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states, alphabet, transition function, start state, accept states

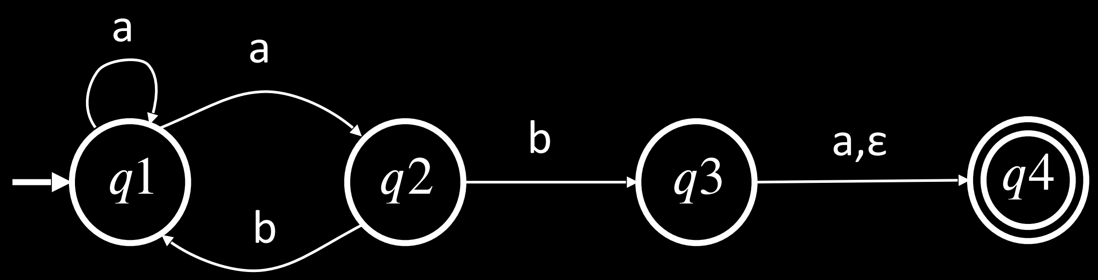
Ways to think about nondeterminism:

- all same as before except δ
- $\delta : Q \times \underbrace{\Sigma_\varepsilon}_{\Sigma \cup \{\varepsilon\}} \to \underbrace{\mathscr{P}(Q)}_{\text{power set}} = \{R \mid R \subseteq Q\}$
- In the N_1 example: $\delta(q_1, \mathrm{a}) = \{q_1, q_2\}$

  $\delta(q_1, \mathrm{b}) = \varnothing$

NFA – Formal Definition

$N1$



Defn: A <u>nondeterministic finite automaton (NFA)</u>

N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states, alphabet, transition function, start state, accept states

- all same as before except δ
- $\delta : Q \times \underbrace{\Sigma_{\varepsilon}}_{\Sigma \cup \{\varepsilon\}} \to \mathscr{P}(Q) = \{R \mid R \subseteq Q\}$

  power set

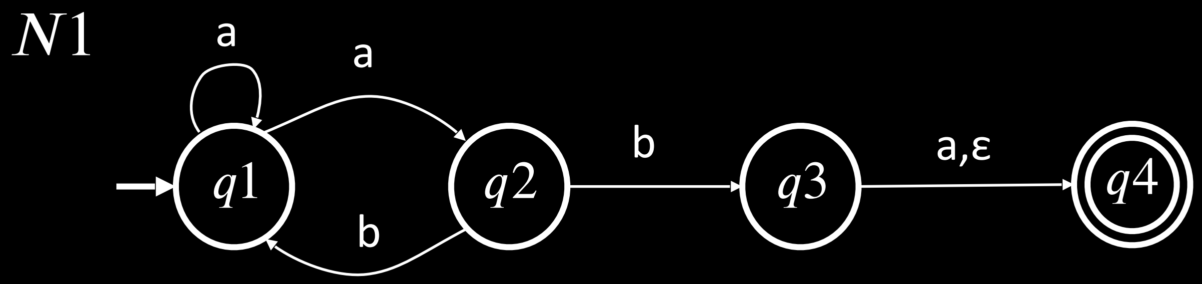
- In the N_1 example: $\delta(q_1, \mathrm{a}) = \{q_1, q_2\}$

  $\delta(q_1, \mathrm{b}) = \varnothing$

Ways to think about nondeterminism:

<u>Computational:</u> Fork new parallel thread and accept if any thread leads to an accept state.

NFA – Formal Definition



Defn: A <u>nondeterministic finite automaton (NFA)</u>

N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states, alphabet, transition function, start state, accept states

- all same as before except δ
- $\delta : Q \times \underbrace{\Sigma_\varepsilon}_{\Sigma \cup \{\varepsilon\}} \to \mathscr{P}(Q) = \{R \mid R \subseteq Q\}$ power set
- In the N_1 example: $\delta(q_1, \mathrm{a}) = \{q_1, q_2\}$

  $\delta(q_1, \mathrm{b}) = \varnothing$

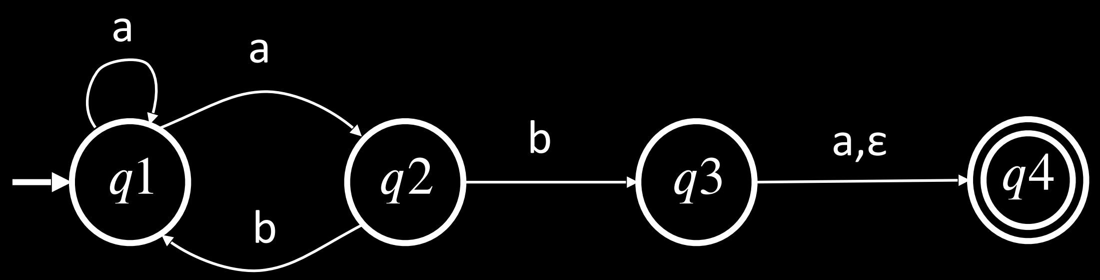
Ways to think about nondeterminism:

<u>Computational:</u> Fork new parallel thread and accept if any thread leads to an accept state.

<u>Mathematical:</u> Tree with branches. Accept if any branch leads to an accept state.

NFA – Formal Definition

$N1$



Defn: A <u>nondeterministic finite automaton (NFA)</u>

N is a 5-tuple $(Q,\ \Sigma,\ \delta,\ q0,\ F)$

states, alphabet, transition function, start state, accept states

- all same as before except δ
- $\delta : Q \times \underbrace{\Sigma_\varepsilon}_{\Sigma \cup \{\varepsilon\}} \rightarrow \underbrace{\mathscr{P}(Q)}_{\text{power set}} = \{R \mid R \subseteq Q\}$
- In the N_1 example: $\delta(q_1, \text{a}) = \{q_1, q_2\}$

  $\delta(q_1, \text{b}) = \varnothing$

Ways to think about nondeterminism:

<u>Computational:</u> Fork new parallel thread and accept if any thread leads to an accept state.

<u>Mathematical:</u> Tree with branches. Accept if any branch leads to an accept state.

<u>Magical:</u> Guess at each nondeterministic step which way to go. Machine always makes the right guess that leads to accepting, if possible.

Converting NFAs to DFAs

Converting NFAs to DFAs

Theorem: If an NFA recognizes A then A is regular

Converting NFAs to DFAs

Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

Converting NFAs to DFAs

Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q,\ \Sigma, \delta,\ \ q_0,\ F)$ recognize A

Construct DFA $M' = (Q',\ \Sigma, \delta',\ \ q_0',\ F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

Converting NFAs to DFAs

Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the subset of possible states in NFA M.

Converting NFAs to DFAs

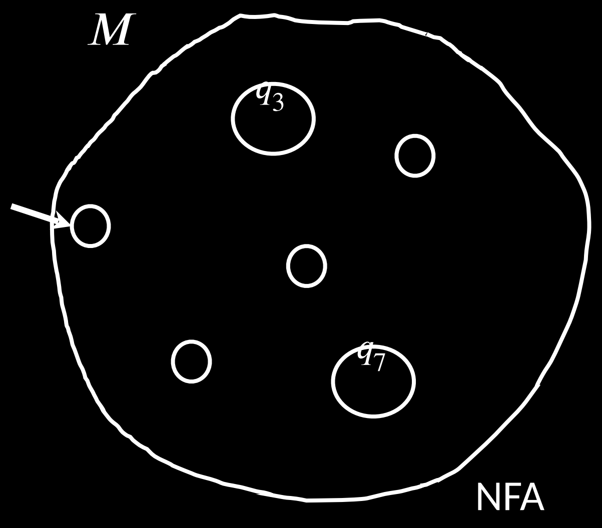
Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q,\ \Sigma, \delta,\ \ q_0,\ F)$ recognize A

Construct DFA $M' = (Q',\ \Sigma, \delta',\ \ q_0',\ F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the <u>subset of possible states</u> in NFA M.



Converting NFAs to DFAs

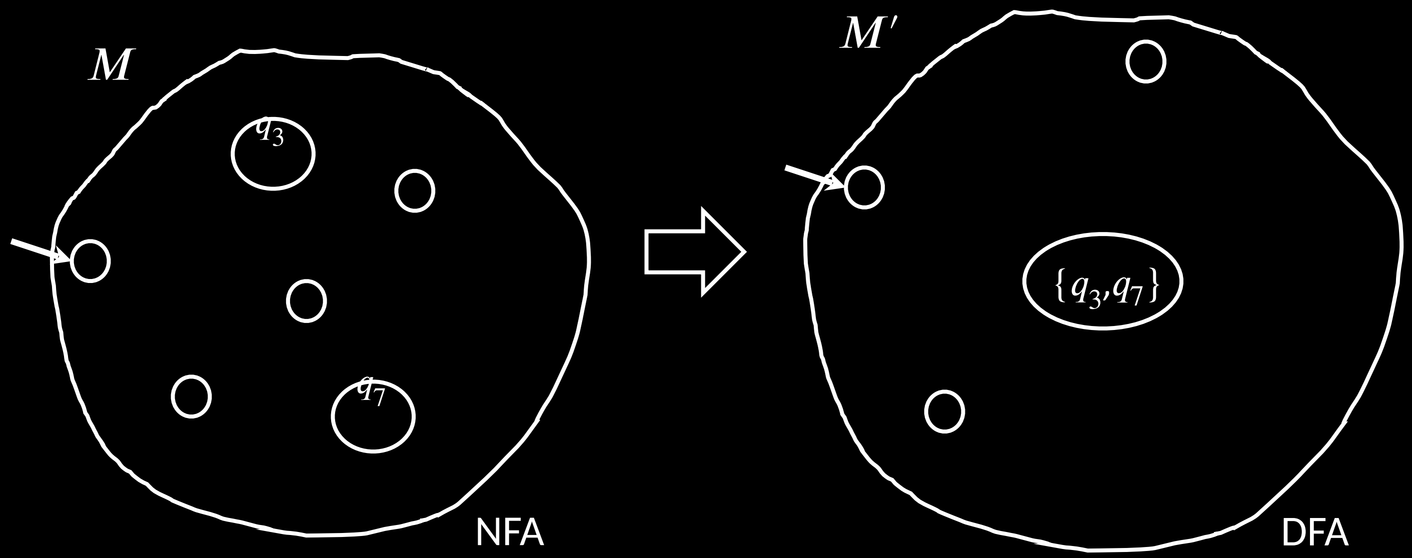
Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the subset of possible states in NFA M.



Converting NFAs to DFAs

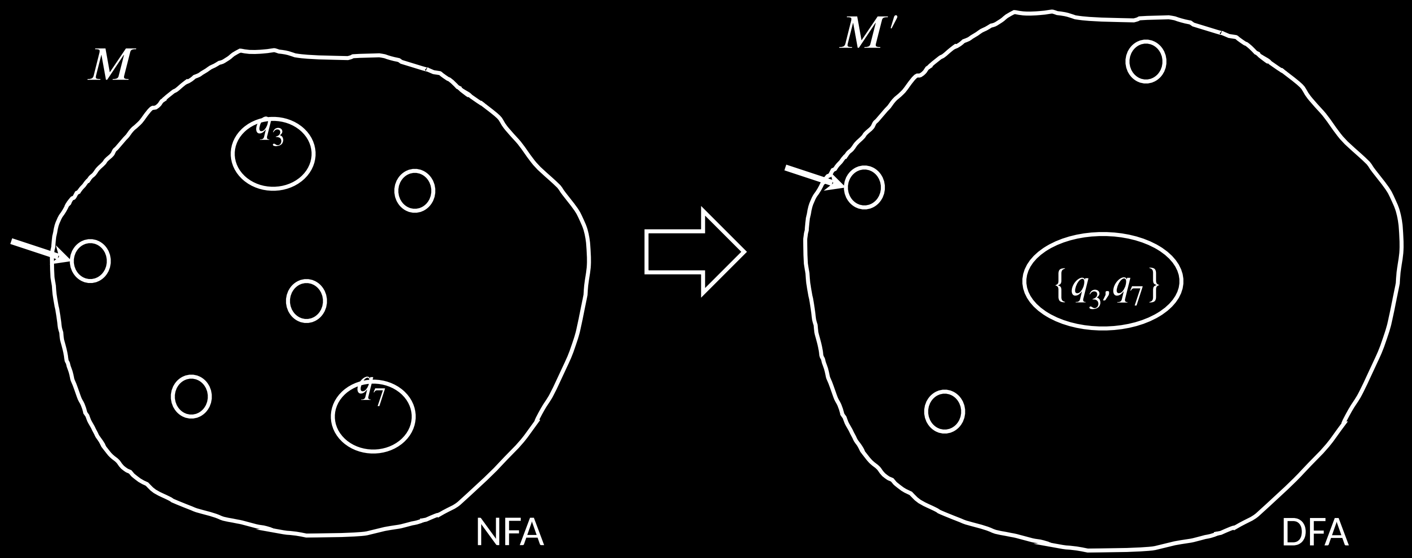
Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the subset of possible states in NFA M.



Construction of M':

Converting NFAs to DFAs

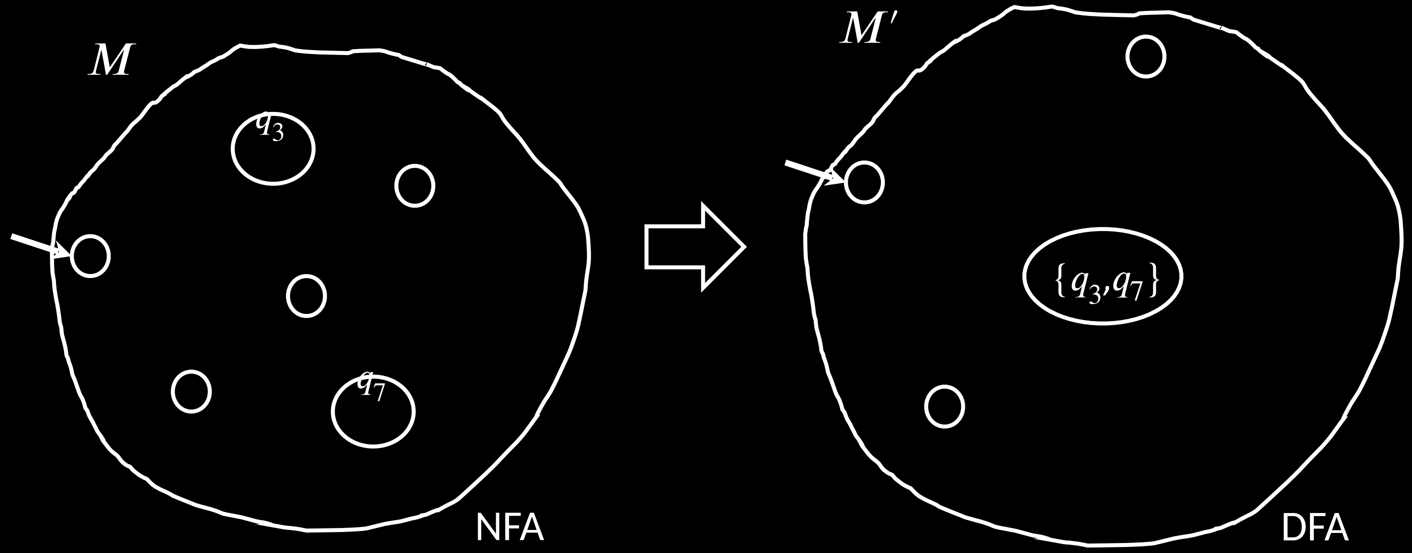
Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the <u>subset of possible states</u> in NFA M.



Construction of $\boldsymbol{M'}$:

$Q' = \mathscr{P}(Q)$

Converting NFAs to DFAs

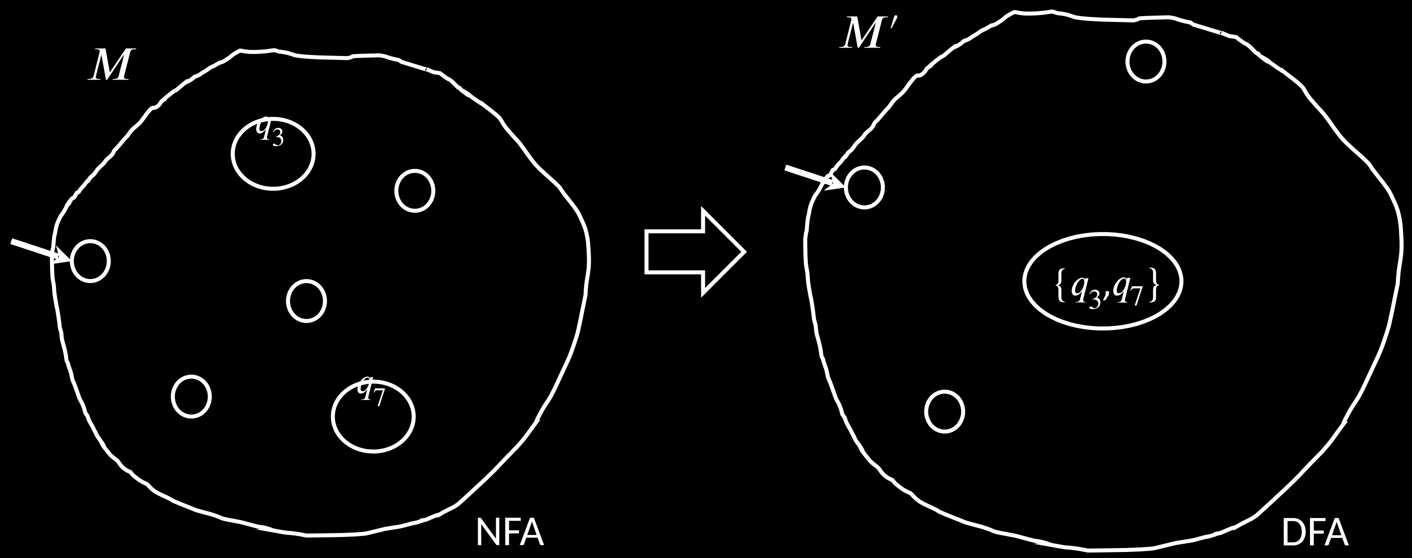
Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the <u>subset of possible states</u> in NFA M.



Construction of $\boldsymbol{M'}$:

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

$\delta'(R, a) =$

$R \in Q'$

Converting NFAs to DFAs

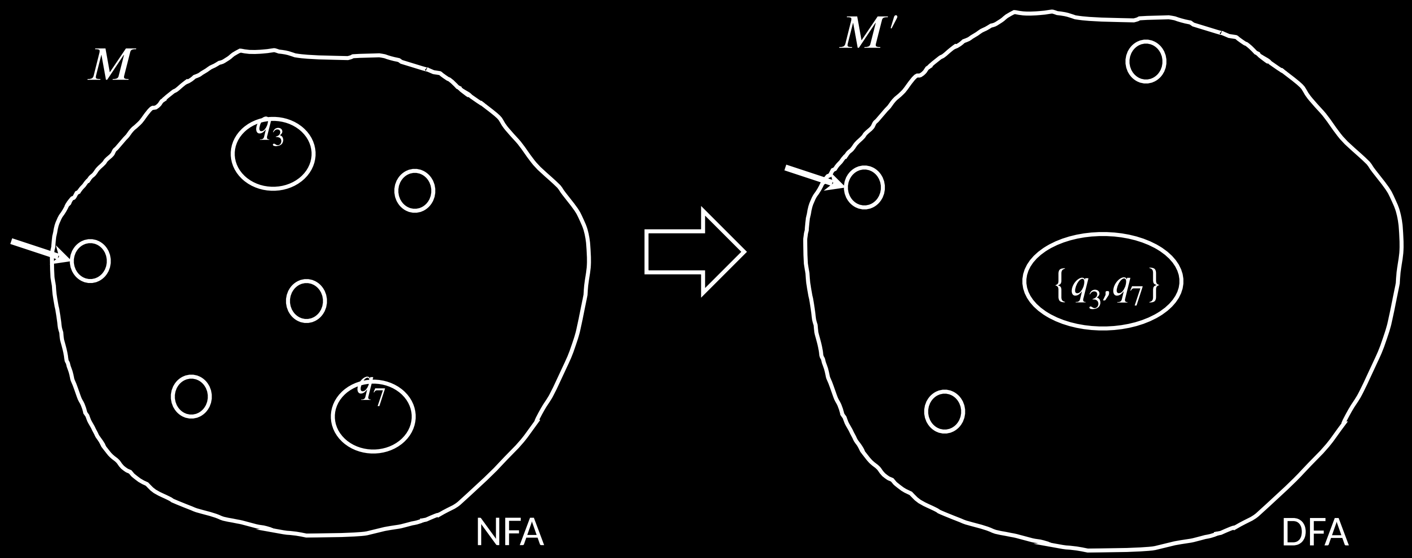
Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the <u>subset of possible states</u> in NFA M.



Construction of $\boldsymbol{M'}$:

$Q' = \mathscr{P}(Q)$

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

$R \in Q'$

Converting NFAs to DFAs

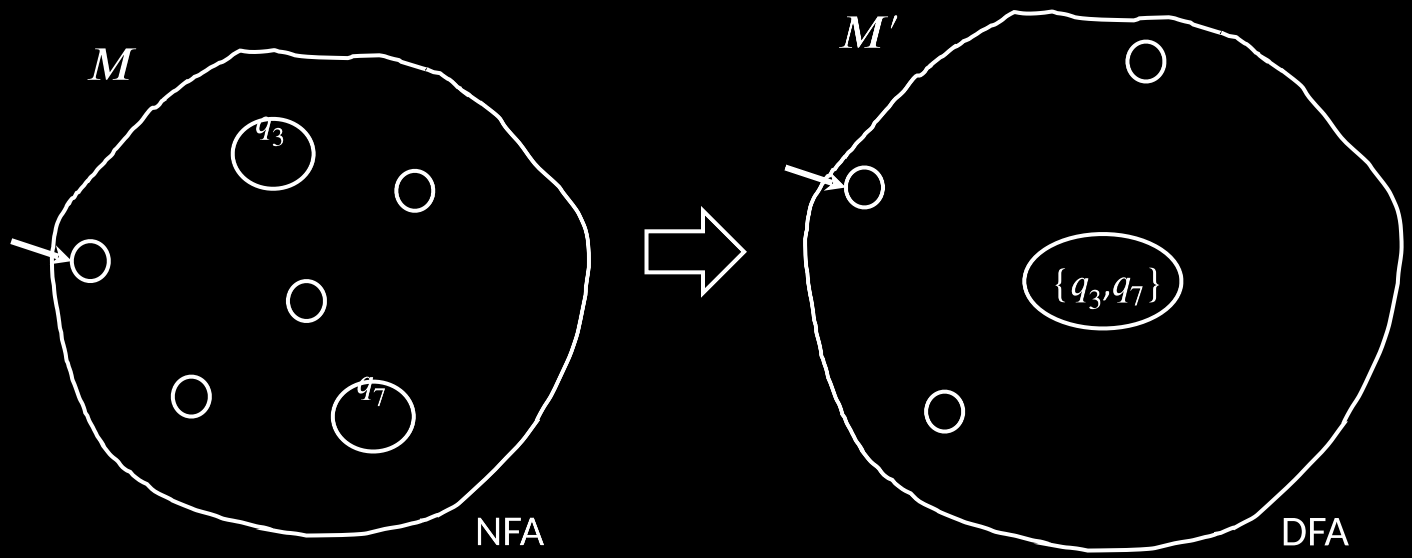
Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q,\ \Sigma, \delta,\ \ q_0,\ \ F)$ recognize A

Construct DFA $M' = (Q',\ \Sigma, \delta',\ \ q_0',\ \ F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the <u>subset of possible states</u> in NFA M.



Construction of $\boldsymbol{M'}$:

$Q' = \mathscr{P}(Q)$

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

$q_0' = \{\mathrm{q}_0\}$

Converting NFAs to DFAs

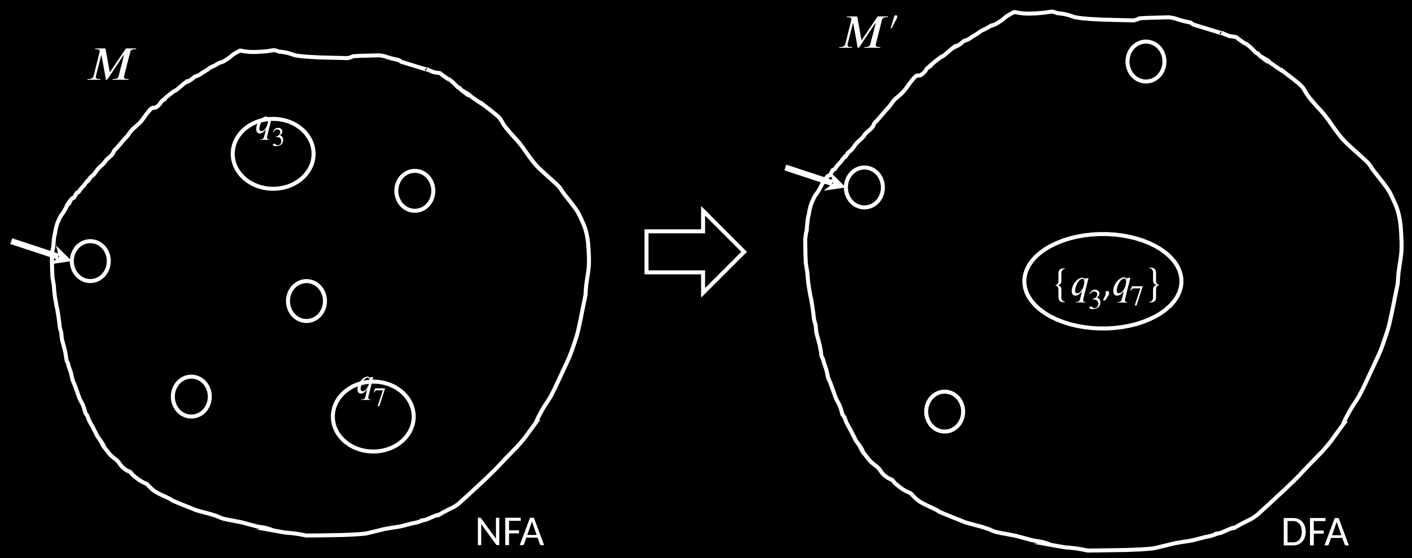
Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the <u>subset of possible states</u> in NFA M.



Construction of $\boldsymbol{M'}$:

$Q' = \mathscr{P}(Q)$

$\underbrace{\delta'(R, a)}_{R \in Q'} = \{ q \mid q \in \delta(r, a)$ for some $r \in R \}$

$q_0' = \{q_0\}$

$F' = \{ R \in Q' \mid R$ intersects $F \}$

Converting NFAs to DFAs

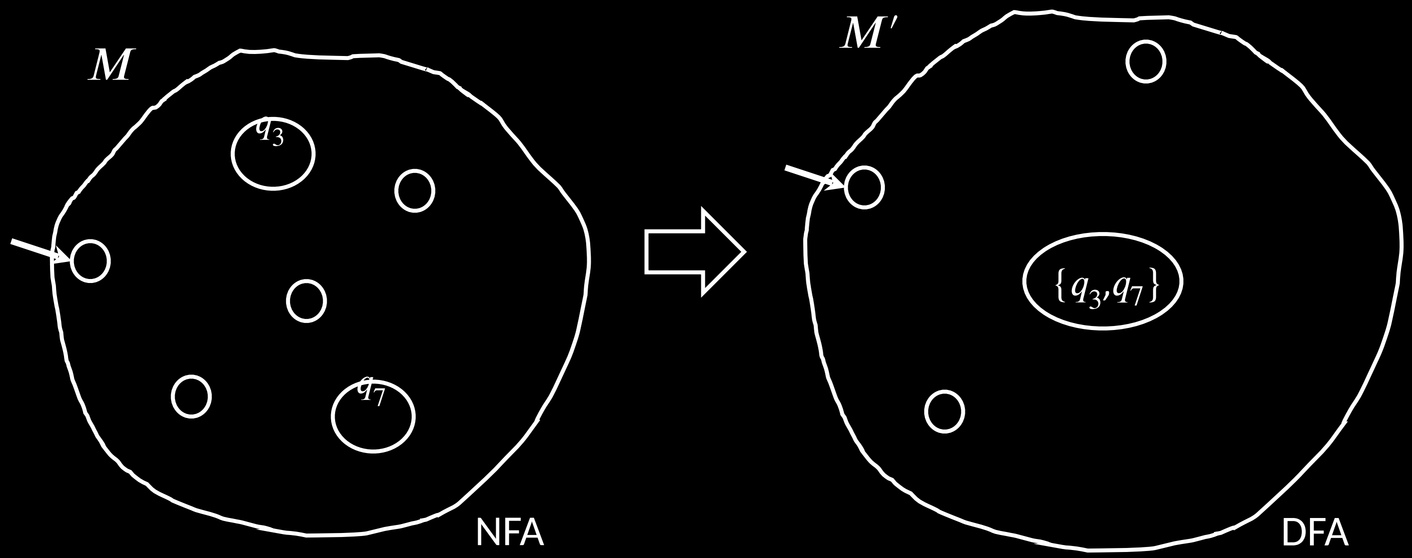
Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q,\ \Sigma, \delta,\ q_0,\ F)$ recognize A

Construct DFA $M' = (Q',\ \Sigma, \delta',\ q_0',\ F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the <u>subset of possible states</u> in NFA M.



Construction of $\boldsymbol{M'}$:

$Q' = \mathscr{P}(Q)$

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

$q_0' = \{q_0\}$

$F' = \{ R \in Q' \mid R \text{ intersects } F \}$

Check-in 2.2

Converting NFAs to DFAs

Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

(Ignore the ε-transitions, can easily modify to handle them)

IDEA: DFA M' keeps track of the subset of possible states in NFA M.

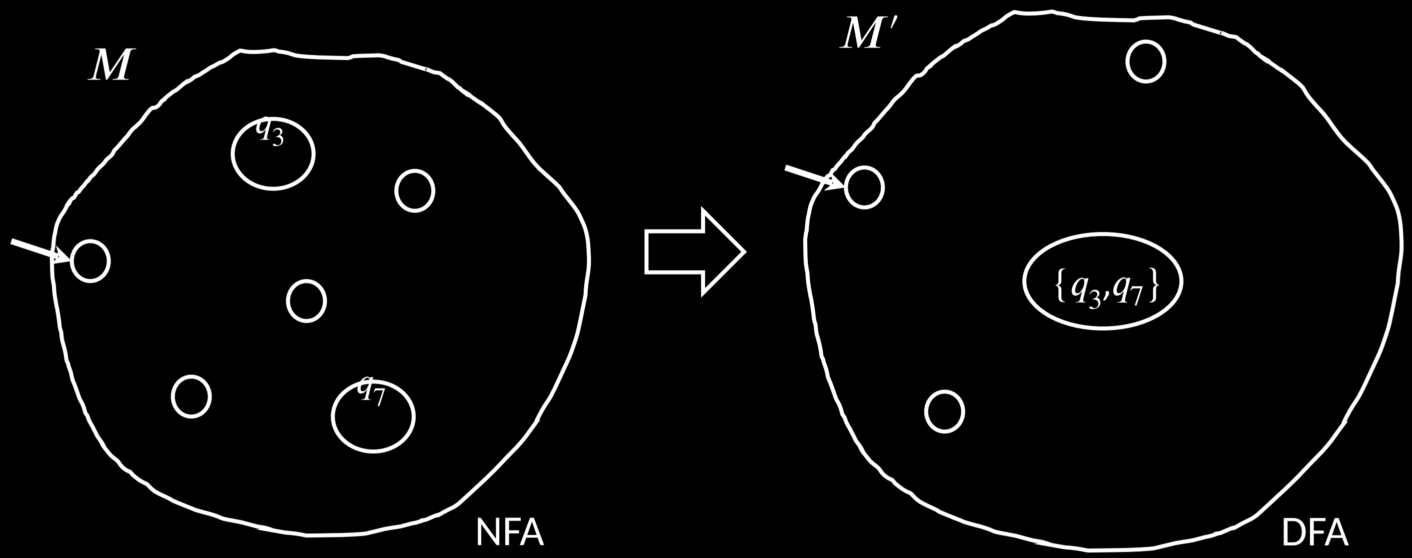
Check-in 2.2

If M has n states, how many states does M' have by this construction?

(a) $2n$

(b) n^2

(c) 2^n



Construction of $\boldsymbol{M'}$:

$Q' = \mathscr{P}(Q)$

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

$R \in Q'$

$q_0' = \{q_0\}$

$F' = \{ R \in Q' \mid R \text{ intersects } F \}$

Check-in 2.2

Return to Closure Properties

Recall Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$

(The class of regular languages is closed under union)

New Proof (sketch): Given DFAs M_1 and M_2 recognizing A_1 and A_2

Construct NFA M recognizing $A_1 \cup A_2$

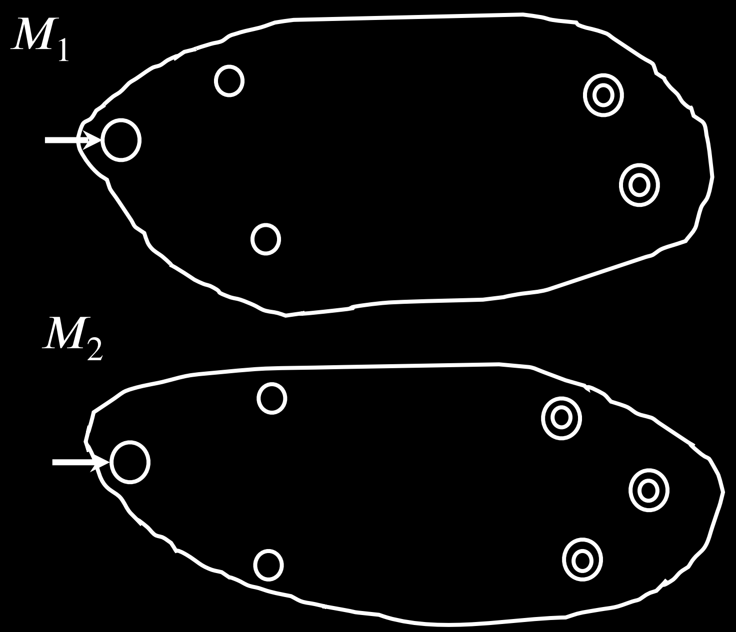
Return to Closure Properties

Recall Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$

(The class of regular languages is closed under union)

New Proof (sketch): Given DFAs M_1 and M_2 recognizing A_1 and A_2

Construct NFA M recognizing $A_1 \cup A_2$



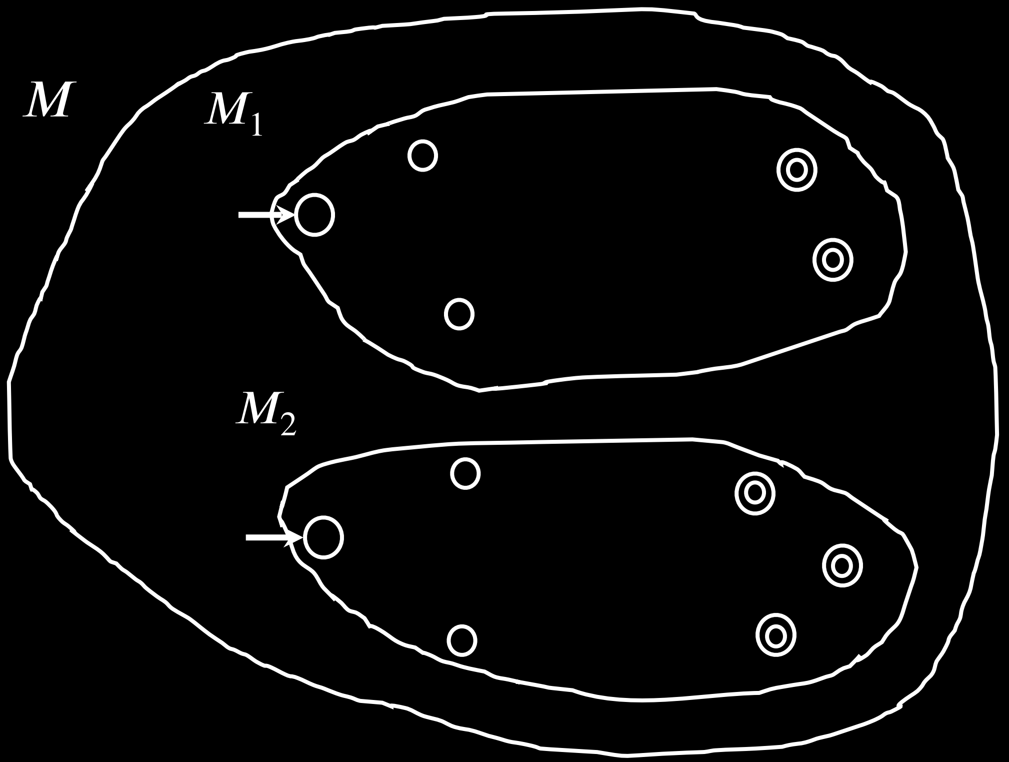
Return to Closure Properties

Recall Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$

(The class of regular languages is closed under union)

New Proof (sketch): Given DFAs M_1 and M_2 recognizing A_1 and A_2

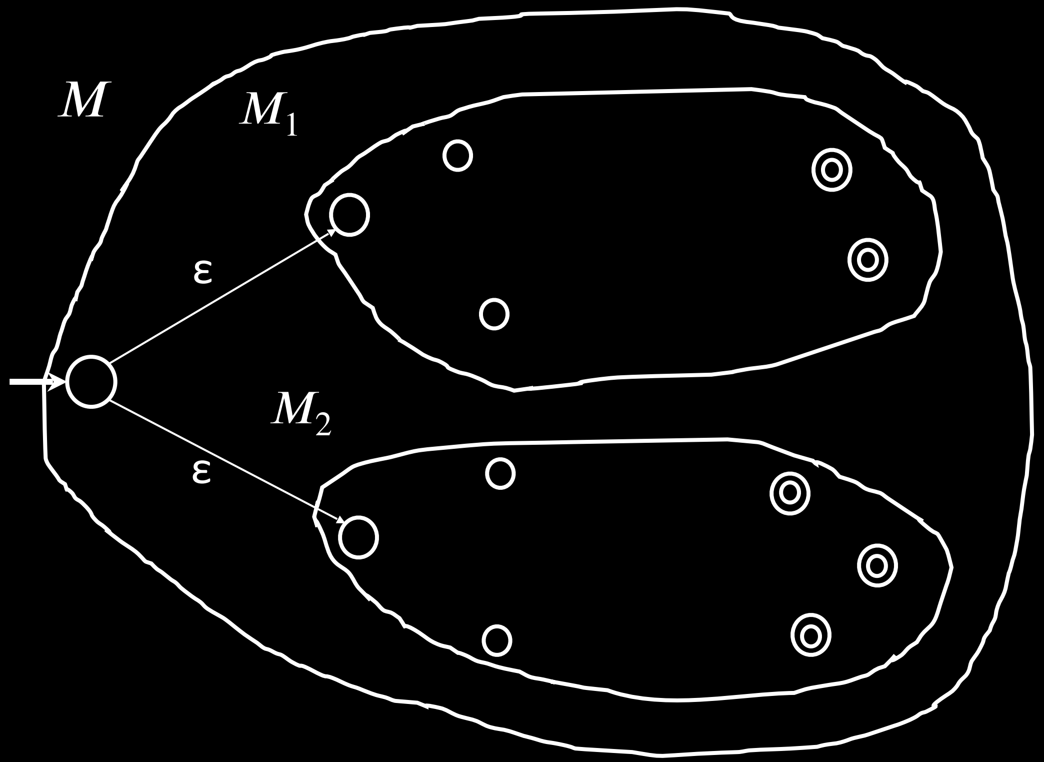
Construct NFA M recognizing $A_1 \cup A_2$



Return to Closure Properties

Recall Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$
(The class of regular languages is closed under union)

New Proof (sketch): Given DFAs M_1 and M_2 recognizing A_1 and A_2
Construct NFA M recognizing $A_1 \cup A_2$



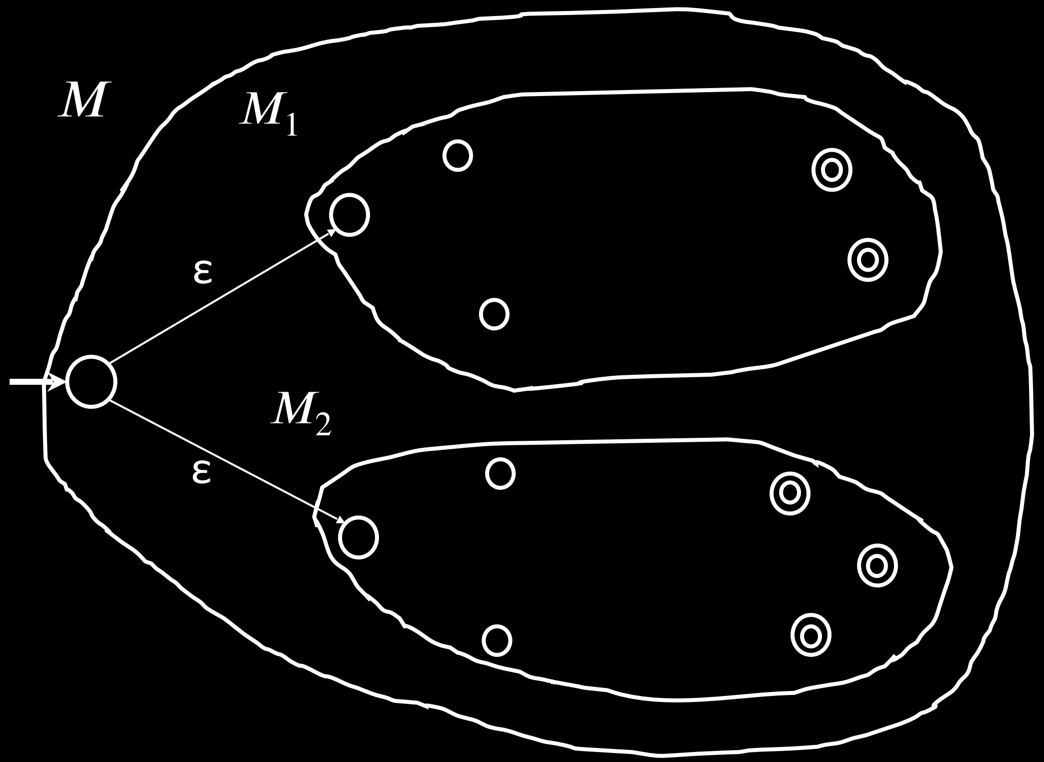
Return to Closure Properties

Recall Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$

(The class of regular languages is closed under union)

New Proof (sketch): Given DFAs M_1 and M_2 recognizing A_1 and A_2

Construct NFA M recognizing $A_1 \cup A_2$



<u>Nondeterminism</u>

parallelism

vs

guessing

Closure under $\circ$ (concatenation)

Theorem: If A_1, A_2 are regular languages, so is A_1A_2

Closure under ∘ (concatenation)

Theorem: If A_1, A_2 are regular languages, so is $A_1 A_2$

Proof sketch: Given DFAs M_1 and M_1 recognizing A_1 and A_2

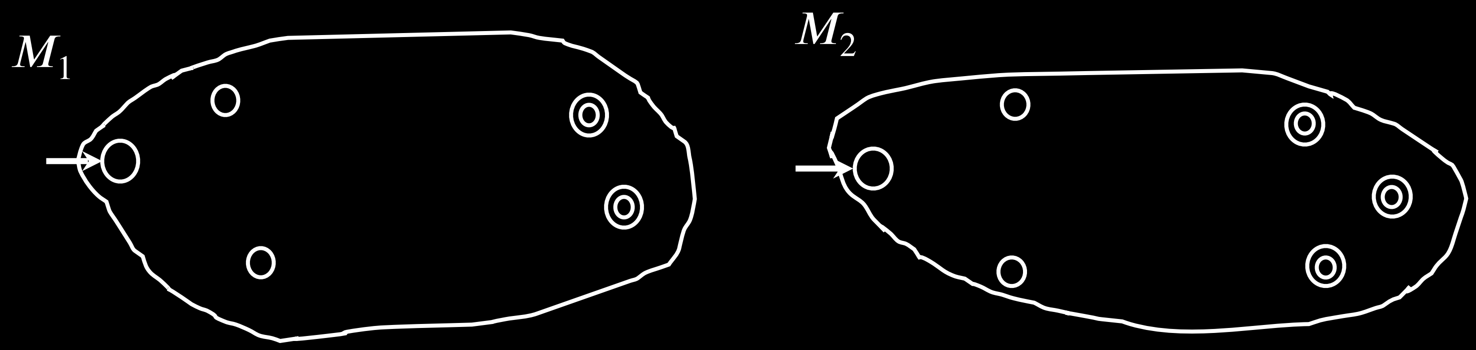
Construct NFA M recognizing $A_1 A_2$

Closure under ∘ (concatenation)

Theorem: If A_1, A_2 are regular languages, so is $A_1 A_2$

Proof sketch: Given DFAs M_1 and M_1 recognizing A_1 and A_2

Construct NFA M recognizing $A_1 A_2$

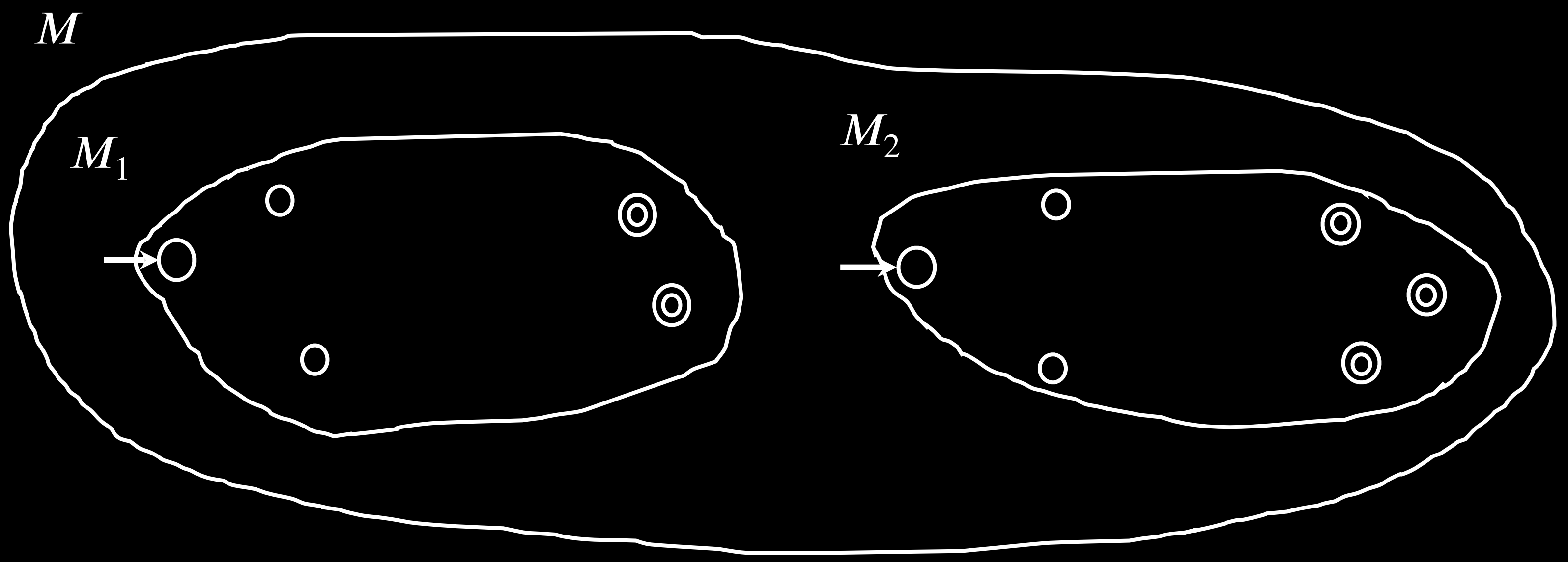


Closure under ∘ (concatenation)

Theorem: If A_1, A_2 are regular languages, so is A_1A_2

Proof sketch: Given DFAs M_1 and M_1 recognizing A_1 and A_2

Construct NFA M recognizing A_1A_2

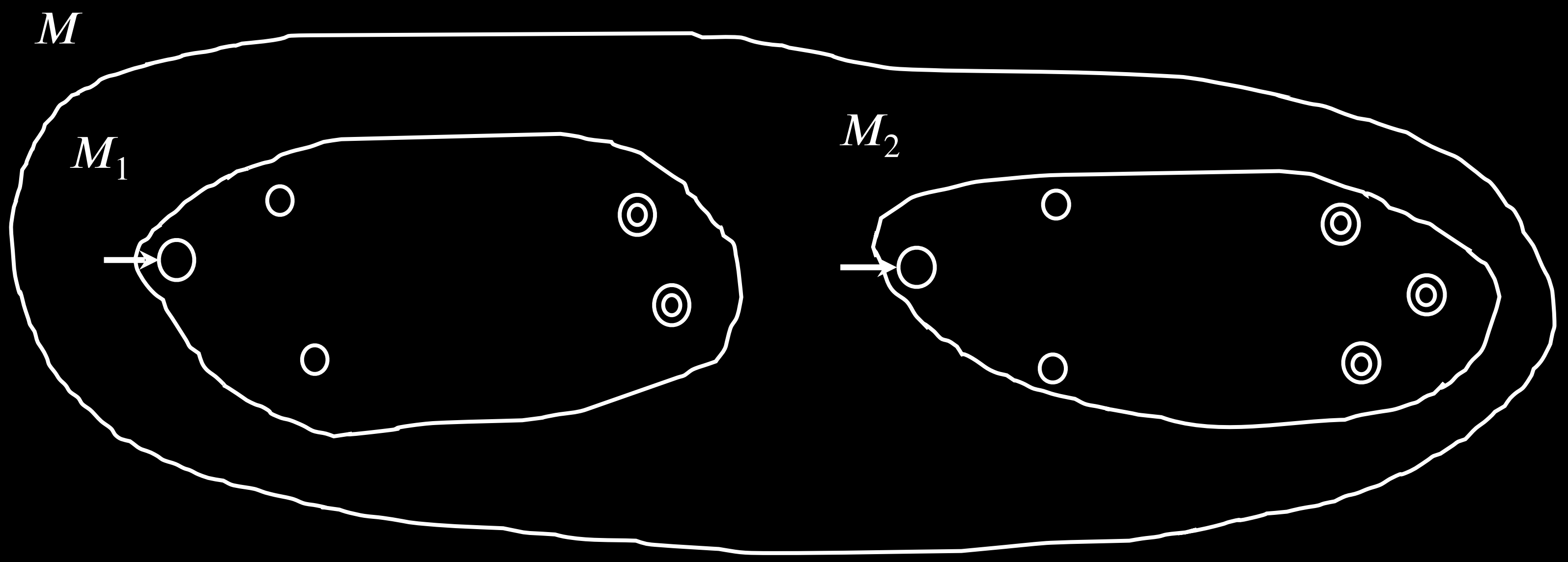


Closure under ∘ (concatenation)

Theorem: If A_1, A_2 are regular languages, so is A_1A_2

Proof sketch: Given DFAs M_1 and M_1 recognizing A_1 and A_2

Construct NFA M recognizing A_1A_2



M should accept input w

if $w = xy$ where

M_1 accepts x and M_2 accepts y.

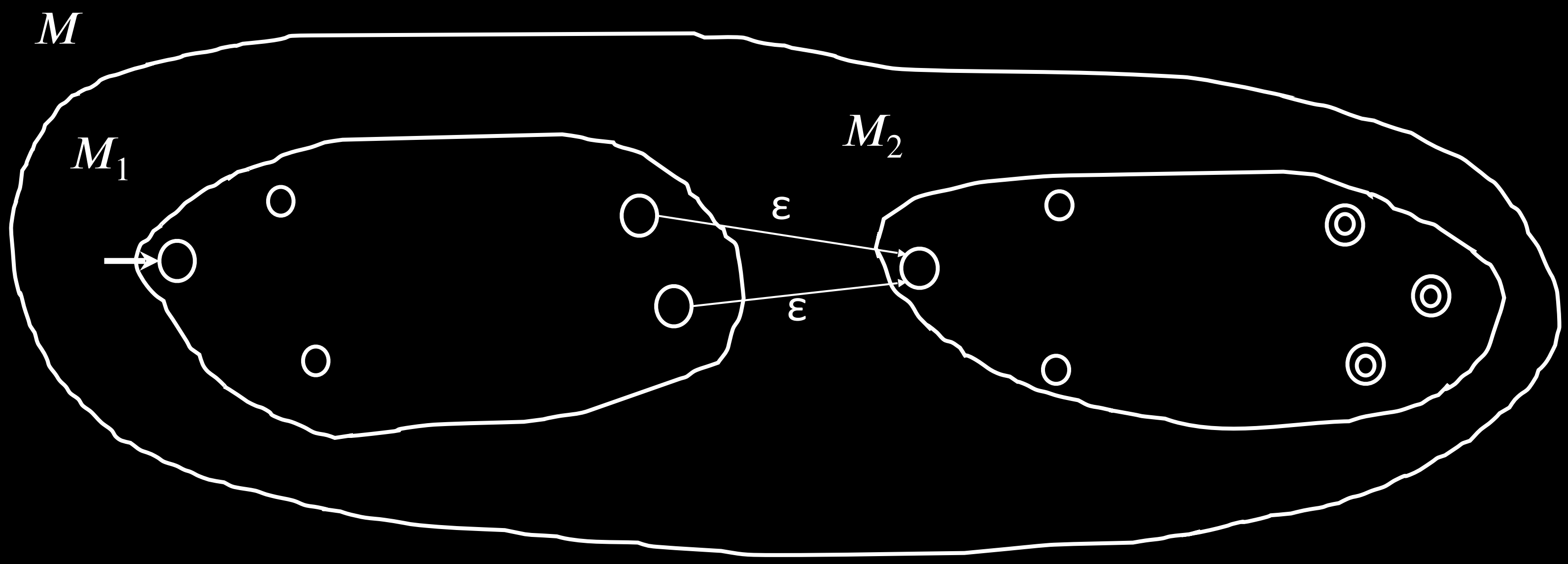
$w = x \mid y$

Closure under ∘ (concatenation)

Theorem: If A_1, A_2 are regular languages, so is A_1A_2

Proof sketch: Given DFAs M_1 and M_1 recognizing A_1 and A_2

Construct NFA M recognizing A_1A_2



M should accept input w

if $w = xy$ where

M_1 accepts x and M_2 accepts y.

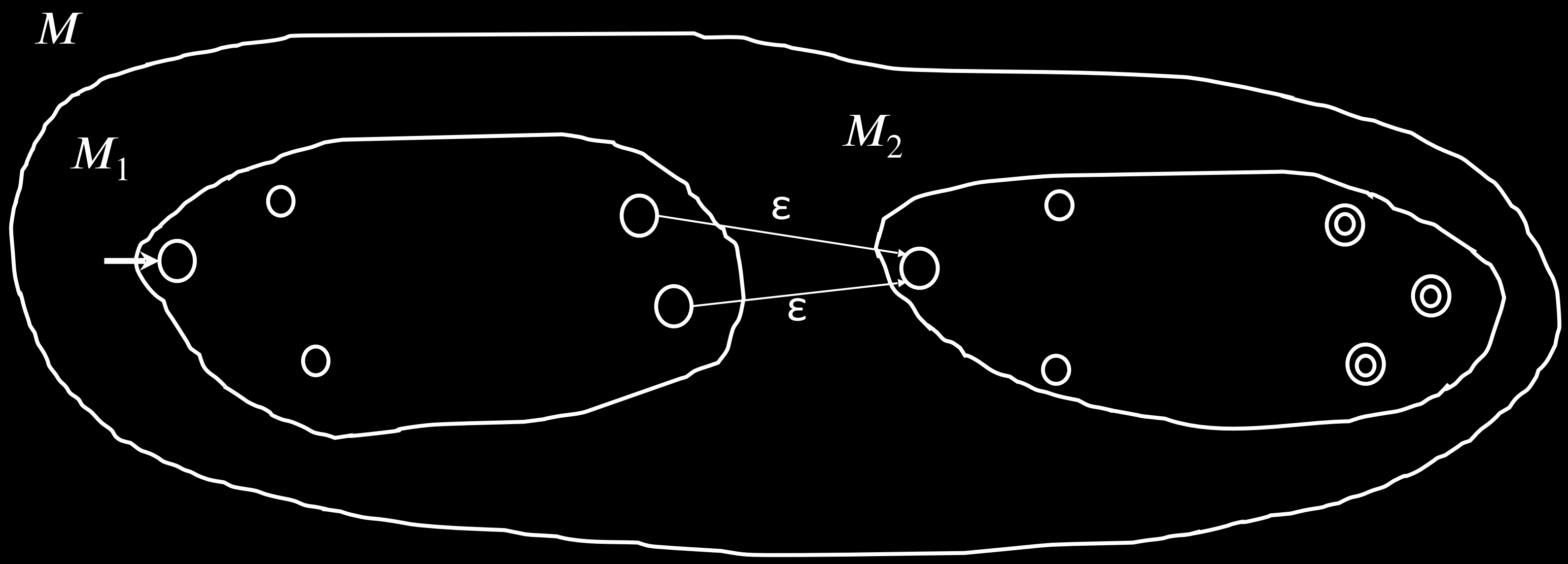
$w =$ x | y

Closure under ∘ (concatenation)

Theorem: If A_1, A_2 are regular languages, so is A_1A_2

Proof sketch: Given DFAs M_1 and M_1 recognizing A_1 and A_2

Construct NFA M recognizing A_1A_2



M should accept input w

if $w = xy$ where

M_1 accepts x and M_2 accepts y.

$w =$

Nondeterministic M' has the option to jump to M_2 when M_1 accepts.

Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

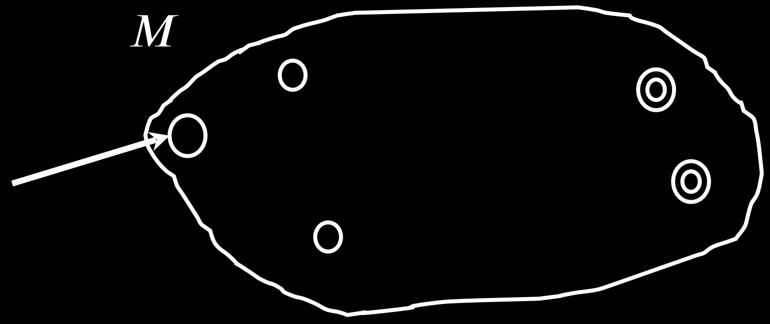
Construct NFA M' recognizing A^*

Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

Construct NFA M' recognizing A^*

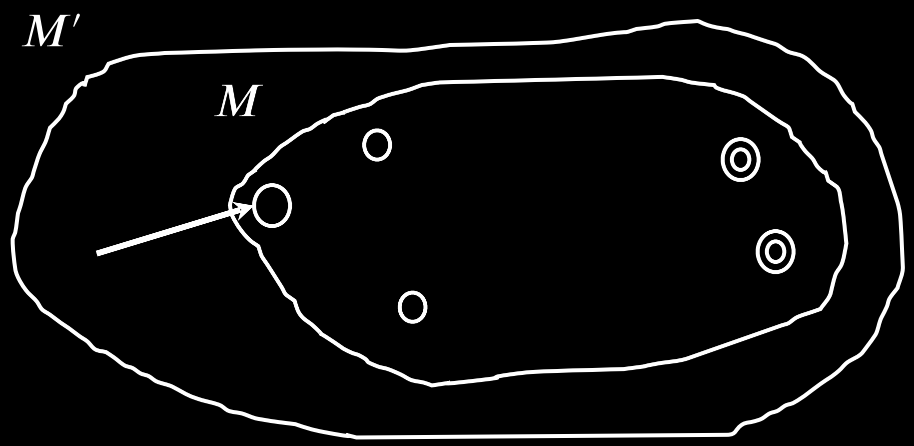


Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

Construct NFA M' recognizing A^*

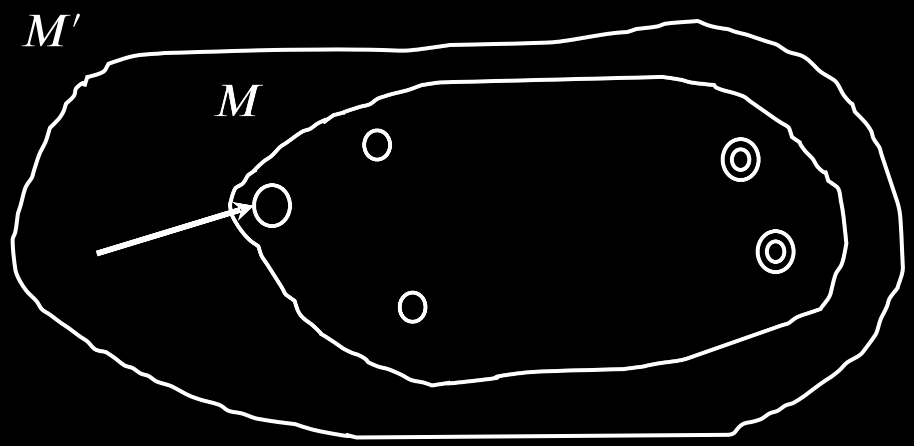


Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

Construct NFA M' recognizing A^*



M' should accept input w

if $w = x_1 x_2 \ldots\ x_k$

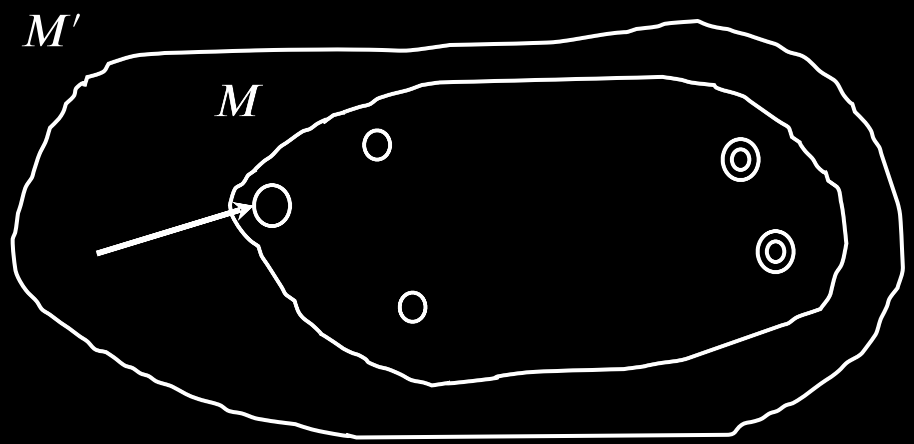
where $k \geq 0$ and M accepts each x_i

Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

Construct NFA M' recognizing A^*



M' should accept input w

if $w = x_1 x_2 \ldots\ x_k$

where $k \geq 0$ and M accepts each x_i

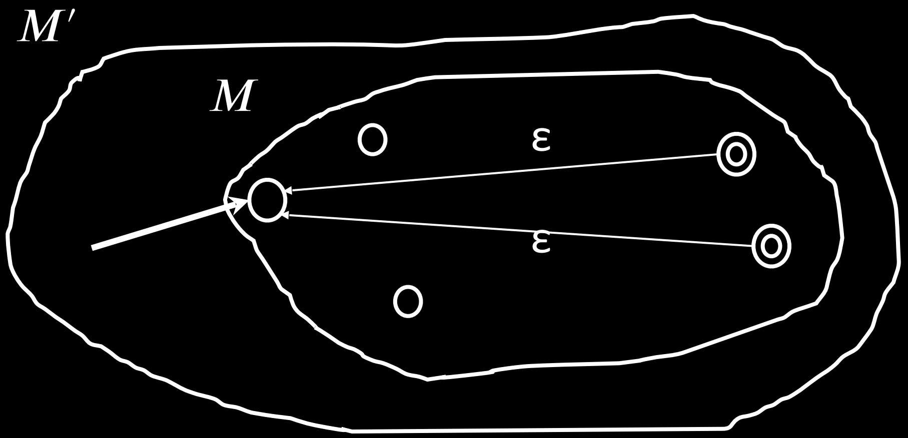
$w = \underline{\ x_1\ |\ x_2\ |\ x_3\ |\ x_4\ }$

Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

Construct NFA M' recognizing A^*



M' should accept input w

if $w = x_1 x_2 \ldots\ x_k$

where $k \geq 0$ and M accepts each x_i

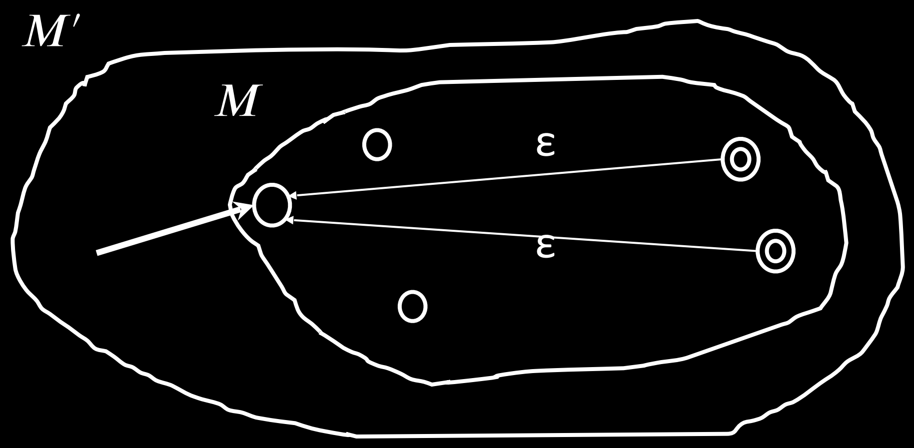
$w = \; x_1 \,|\, x_2 \,|\, x_3 \,|\, x_4$

Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

Construct NFA M' recognizing A^*



Make sure M' accepts ε

M' should accept input w

if $w = x_1 x_2 \ldots x_k$

where $k \geq 0$ and M accepts each x_i

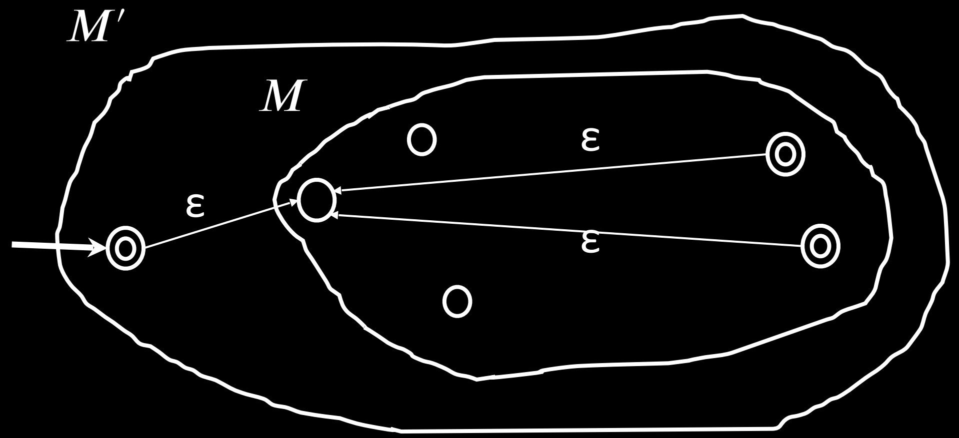
$w = x_1 \mid x_2 \mid x_3 \mid x_4$

Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

Construct NFA M' recognizing A^*

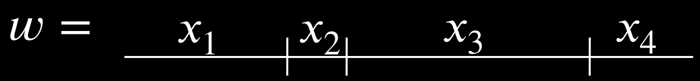


Make sure M' accepts ε

M' should accept input w

if $w = x_1 x_2 \ldots\ x_k$

where $k \geq 0$ and M accepts each x_i

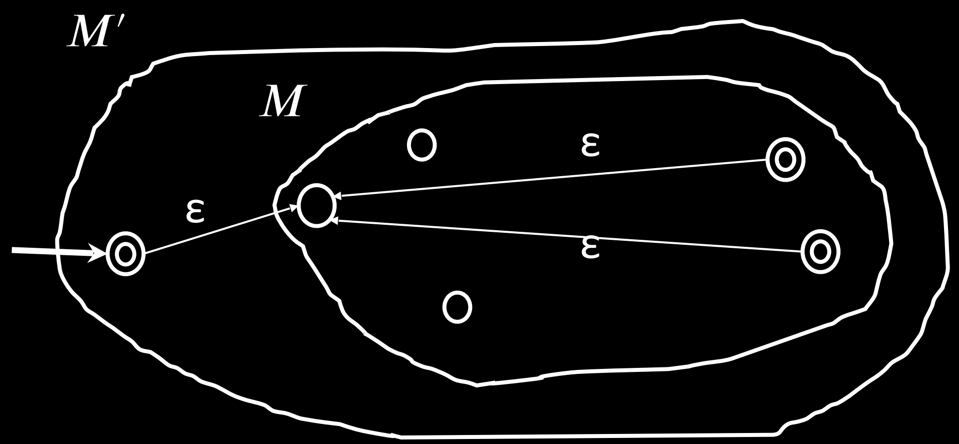


Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

Construct NFA M' recognizing A^*



Make sure M' accepts ε

M' should accept input w

if $w = x_1 x_2 \ldots\ x_k$

where $k \geq 0$ and M accepts each x_i

$w = \underbrace{}_{x_1}\,|\,\underbrace{}_{x_2}\,|\,\underbrace{}_{x_3}\,|\,\underbrace{}_{x_4}$

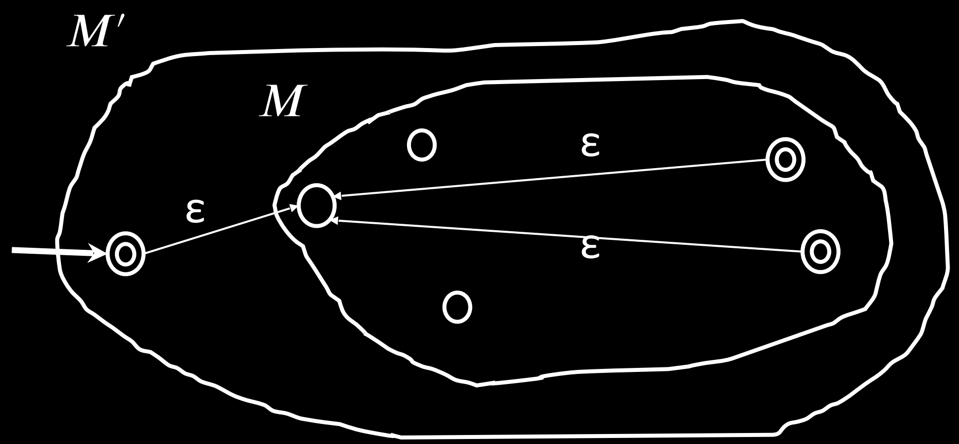
Check-in 2.3

Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A

Construct NFA M' recognizing A^*



Make sure M' accepts ε

Check-in 2.3

If M has n states, how many states does M' have by this construction?

(a) n

(b) $n+1$

(c) $2n$

Check-in 2.3

Regular Expression

Regular Expression

- Is a string

Regular Expression

- Is a string
- “a” (for a in Σ) matches to “a”

Regular Expression

- Is a string
- “a” (for a in Σ) matches to “a”
- “R1 $\cup$ R2” matches to strings that match to R1 or R2

Regular Expression

- Is a string
- “a” (for a in Σ) matches to “a”
- “R1 $\cup$ R2” matches to strings that match to R1 or R2
- “R1R2” matches to every string w if w could be written as w=xy where x matches to R1 and y matches to R2

Regular Expression

- Is a string
- “a” (for a in Σ) matches to “a”
- “R1 ∪ R2” matches to strings that match to R1 or R2
- “R1R2” matches to every string w if w could be written as w=xy where x matches to R1 and y matches to R2
- “(R)” matches to strings that match to R

Regular Expression

- Is a string
- "a" (for a in Σ) matches to "a"
- "R1 $\cup$ R2" matches to strings that match to R1 or R2
- "R1R2" matches to every string w if w could be written as w=xy where x matches to R1 and y matches to R2
- "(R)" matches to strings that match to R
- "R*" matches to string s if s could be written as $s=x_1x_2\ldots x_k$ where each x_i matches to R.

Regular Expression

- Is a string
- "a" (for a in Σ) matches to "a"
- "R1 ∪ R2" matches to strings that match to R1 or R2
- "R1R2" matches to every string w if w could be written as w=xy where x matches to R1 and y matches to R2
- "(R)" matches to strings that match to R
- "R*" matches to string s if s could be written as $s=x_1x_2\ldots x_k$ where each x_i matches to R.

Example:

$$(a \cup ab)^*$$

Regular Expressions $\rightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Regular Expressions $\rightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

Regular Expressions $\to$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u>

Regular Expressions $\rightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

If R is atomic:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$

Regular Expressions $\rightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u>

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \varnothing$

If R is <u>composite:</u>

Regular Expressions $\longrightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

If R is atomic:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$

If R is composite:

$R = R_1 \cup R_2$

$R = R_1 \circ R_2$

$R = R_1^*$

Regular Expressions $\rightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u>

Equivalent M is:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$

If R is <u>composite:</u>

$R = R_1 \cup R_2$

$R = R_1 \circ R_2$

$R = R_1^*$

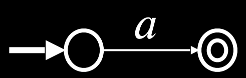
Regular Expressions $\longrightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

$R = a$ for $a \in \Sigma$



$R = \varepsilon$

$R = \varnothing$

If R is <u>composite:</u>

$R = R_1 \cup R_2$

$R = R_1 \circ R_2$

$R = R_1^*$

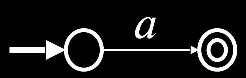
Regular Expressions $\longrightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

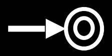
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \emptyset$

If R is <u>composite:</u>

$R = R_1 \cup R_2$

$R = R_1 \circ R_2$

$R = R_1^*$

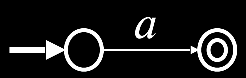
Regular Expressions $\longrightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

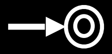
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

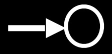
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \varnothing$



If R is <u>composite:</u>

$R = R_1 \cup R_2$

$R = R_1 \circ R_2$

$R = R_1^*$

Regular Expressions $\to$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

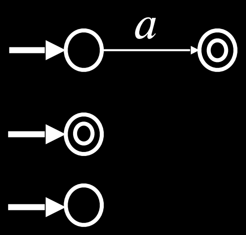
If R is <u>atomic:</u>

Equivalent M is:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$



If R is <u>composite:</u>

$R = R_1 \cup R_2$

$R = R_1 \circ R_2$

$R = R_1^*$

Use closure constructions

Regular Expressions $\rightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

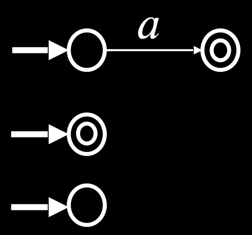
If R is <u>atomic:</u>

Equivalent M is:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$



If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

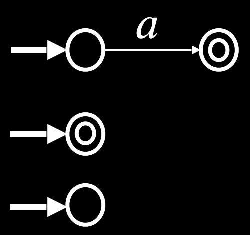
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$



If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

a:

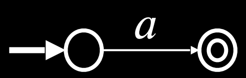
Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

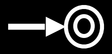
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

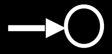
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \varnothing$



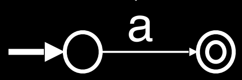
If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

a:



Regular Expressions $\to$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

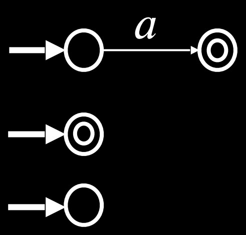
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$



If R is <u>composite:</u>

$R = R_1 \cup R_2$

$R = R_1 \circ R_2$

$R = R_1^*$

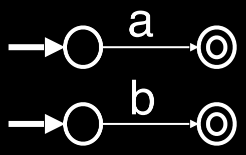
} Use closure constructions

Example:

Convert $(a \cup ab)^*$ to equivalent NFA

a:

b:

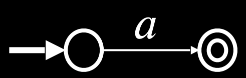


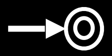
Regular Expressions → NFA

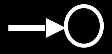
Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

$R = a$ for $a \in \Sigma$ 

$R = \varepsilon$ 

$R = \varnothing$ 

If R is <u>composite:</u>

$R = R_1 \cup R_2$

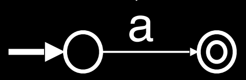
$R = R_1 \circ R_2$

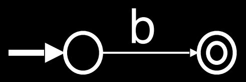
$R = R_1^*$

} Use closure constructions

Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

a: 

b: 

ab:

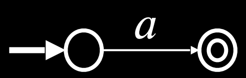
Regular Expressions $\rightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

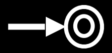
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

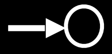
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \varnothing$



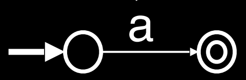
If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\}\text{ Use closure constructions}$$

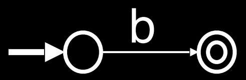
Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

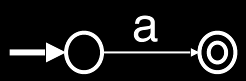
a:



b:



ab:



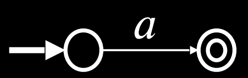
Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

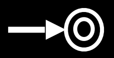
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

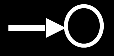
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \emptyset$



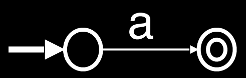
If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

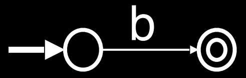
Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

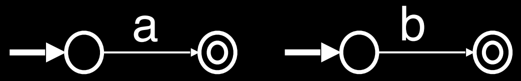
a:



b:



ab:



Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

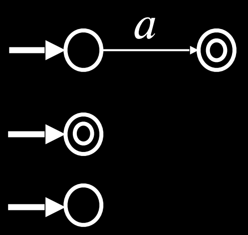
If R is <u>atomic:</u>

Equivalent M is:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$



If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

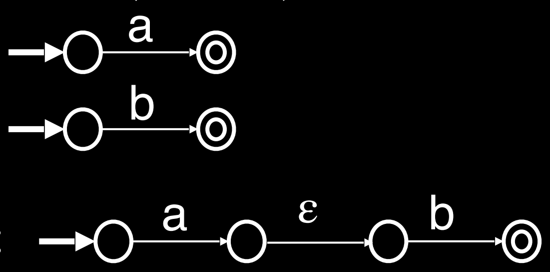
Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

a:

b:

ab:



Regular Expressions $\rightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

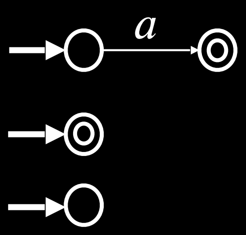
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$



If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

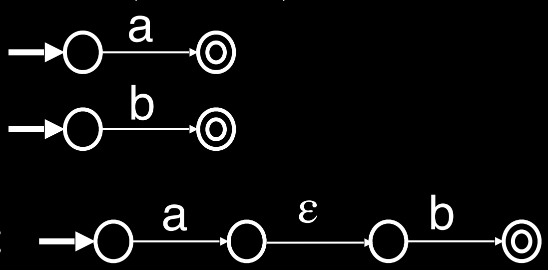
Example:

Convert $(\text{a} \cup \text{ab})^*$ to equivalent NFA

a:

b:

ab:



a ∪ ab:

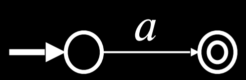
Regular Expressions $\rightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

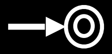
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

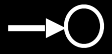
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \emptyset$



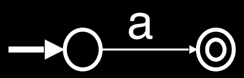
If R is <u>composite:</u>

$$\left.\begin{aligned} R &= R_1 \cup R_2 \\ R &= R_1 \circ R_2 \\ R &= R_1^* \end{aligned}\right\} \text{ Use closure constructions}$$

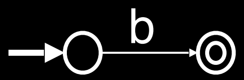
Example:

Convert $(\text{a} \cup \text{ab})^*$ to equivalent NFA

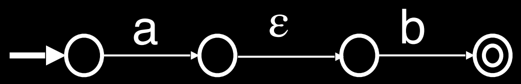
a:



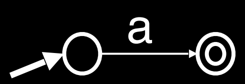
b:



ab:



a ∪ ab:



Regular Expressions $\to$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

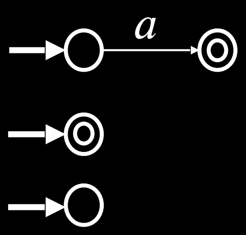
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \varnothing$



If R is <u>composite:</u>

$R = R_1 \cup R_2$

$R = R_1 \circ R_2$

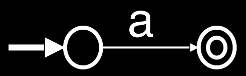
$R = R_1^*$

$\Big\}$ Use closure constructions

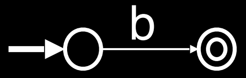
Example:

Convert $(\text{a} \cup \text{ab})^*$ to equivalent NFA

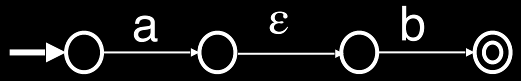
a:



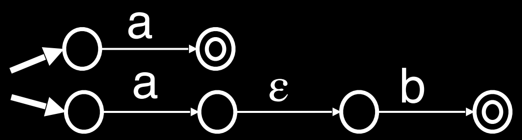
b:



ab:



a ∪ ab:



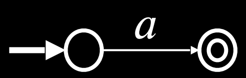
Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

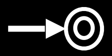
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

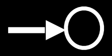
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \varnothing$



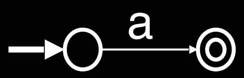
If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

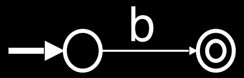
Example:

Convert $(\text{a} \cup \text{ab})^*$ to equivalent NFA

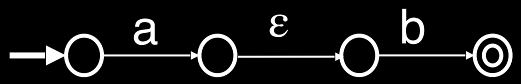
a:



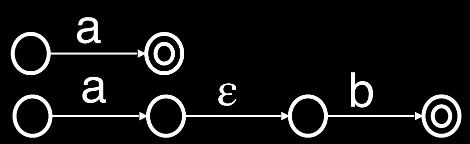
b:



ab:



a ∪ ab:



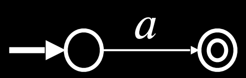
Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

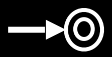
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

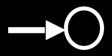
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \varnothing$



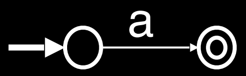
If R is <u>composite:</u>

$$\left.\begin{aligned} R &= R_1 \cup R_2 \\ R &= R_1 \circ R_2 \\ R &= R_1^* \end{aligned}\right\}$$ Use closure constructions

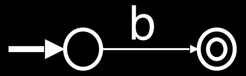
Example:

Convert $(\text{a} \cup \text{ab})^*$ to equivalent NFA

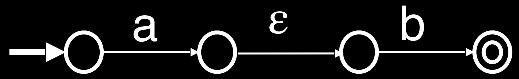
a:



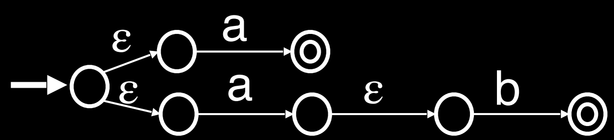
b:



ab:



a ∪ ab:



Regular Expressions $\longrightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

Proof: Convert R to equivalent NFA M:

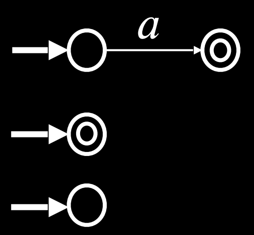
If R is <u>atomic:</u>

Equivalent M is:

$R = a$ for $a \in \Sigma$

$R = \varepsilon$

$R = \emptyset$



If R is <u>composite:</u>

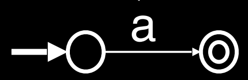
$$R = R_1 \cup R_2$$
$$R = R_1 \circ R_2$$
$$R = R_1^*$$

Use closure constructions

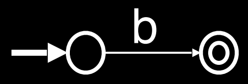
Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

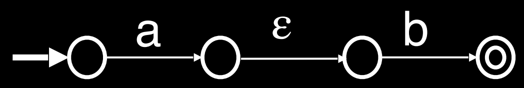
a:



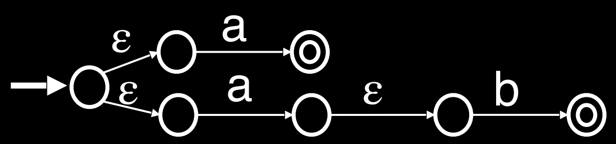
b:



ab:



a ∪ ab:



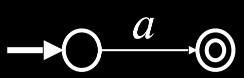
Regular Expressions $\longrightarrow$ NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

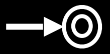
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

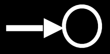
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \varnothing$



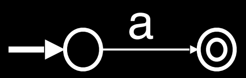
If R is <u>composite:</u>

$$\left.\begin{aligned} R &= R_1 \cup R_2 \\ R &= R_1 \circ R_2 \\ R &= R_1^* \end{aligned}\right\} \text{Use closure constructions}$$

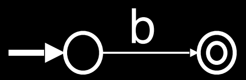
Example:

Convert $(a \cup ab)^*$ to equivalent NFA

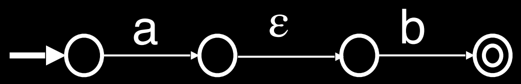
a:



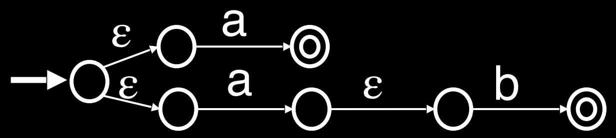
b:



ab:



a ∪ ab:



$(a \cup ab)^*$:

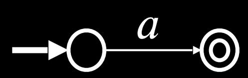
Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

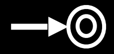
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

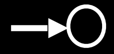
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \emptyset$



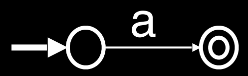
If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

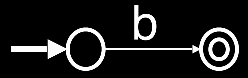
Example:

Convert $(a \cup ab)^*$ to equivalent NFA

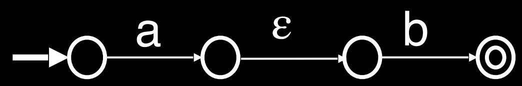
a:



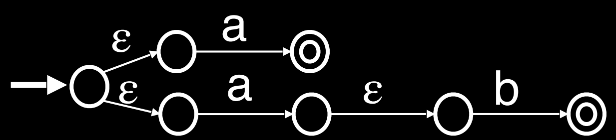
b:



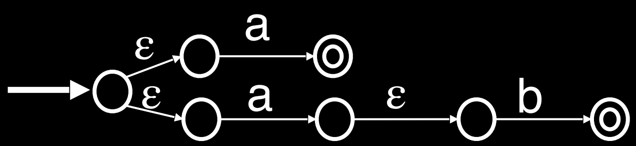
ab:



a ∪ ab:



$(a \cup ab)^*$:



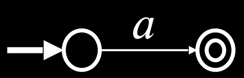
Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

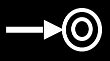
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

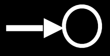
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \varnothing$



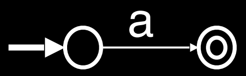
If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

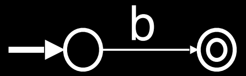
Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

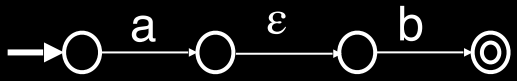
a:



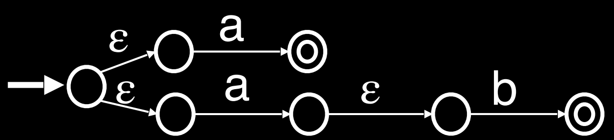
b:



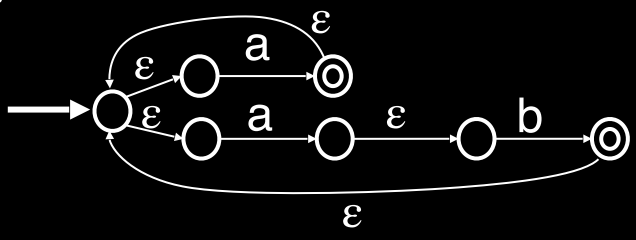
ab:



a ∪ ab:



$(\mathrm{a} \cup \mathrm{ab})^*$:



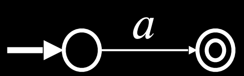
Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

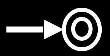
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

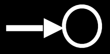
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \varnothing$



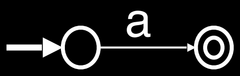
If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

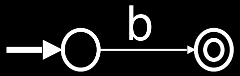
Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

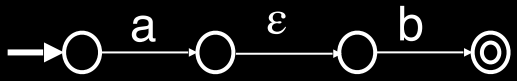
a:



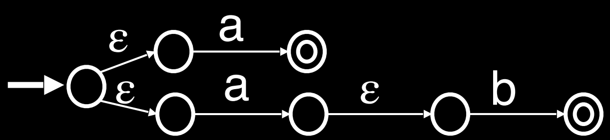
b:



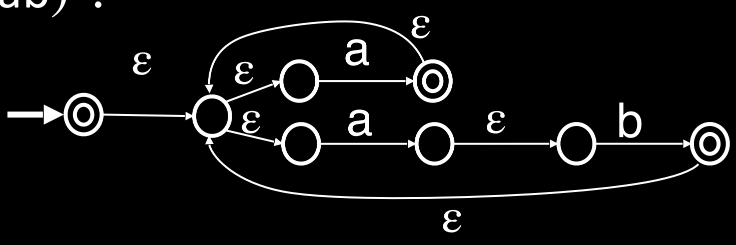
ab:



a ∪ ab:



$(\mathrm{a} \cup \mathrm{ab})^*$:



DFAs $\rightarrow$ Regular Expressions

DFAs $\rightarrow$ Regular Expressions

Recall Theorem: If R is a regular expressipn and $A = L(R)$ then A is regular

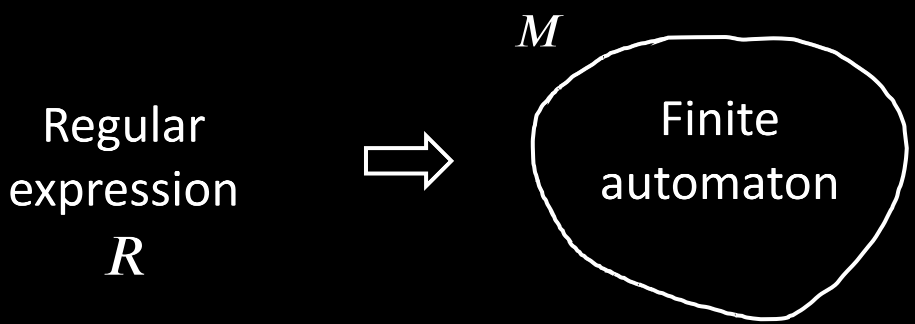
Proof: Conversion $R \rightarrow$ NFA $M \rightarrow$ DFA M'

Regular expression R $\Rightarrow$ M Finite automaton

DFAs $\rightarrow$ Regular Expressions

Recall Theorem: If R is a regular expressipn and $A = L(R)$ then A is regular

Proof: Conversion $R \rightarrow$ NFA $M \rightarrow$ DFA M'



Recall: we did $(\mathrm{a} \cup \mathrm{ab})^*$ as an example

DFAs $\longrightarrow$ Regular Expressions

Recall Theorem: If R is a regular expressipn and $A = L(R)$ then A is regular

Proof: Conversion $R \;\rightarrow$ NFA $M \rightarrow$ DFA M'

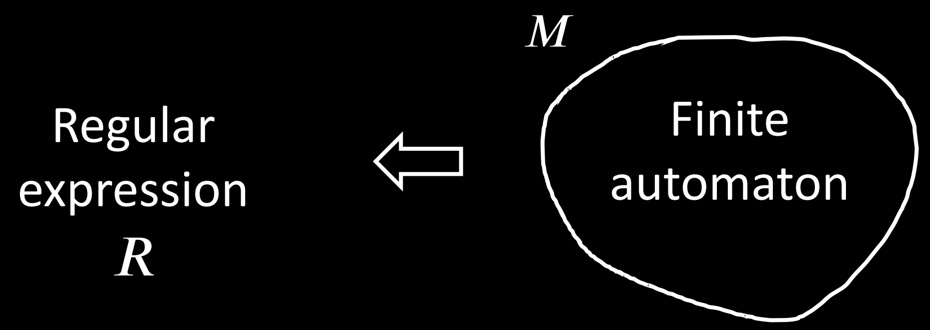
Regular expression R $\Longleftarrow$ M Finite automaton

Recall: we did $(\text{a} \cup \text{ab})^*$ as an example

DFAs $\rightarrow$ Regular Expressions

Recall Theorem: If R is a regular expressipn and $A = L(R)$ then A is regular

Proof: Conversion $R \rightarrow$ NFA $M \rightarrow$ DFA M'



Recall: we did $(\text{a} \cup \text{ab})^*$ as an example

Today's Theorem: If A is regular then $A = L(R)$ for some regular expr R

Proof: Give conversion DFA $M \rightarrow R$

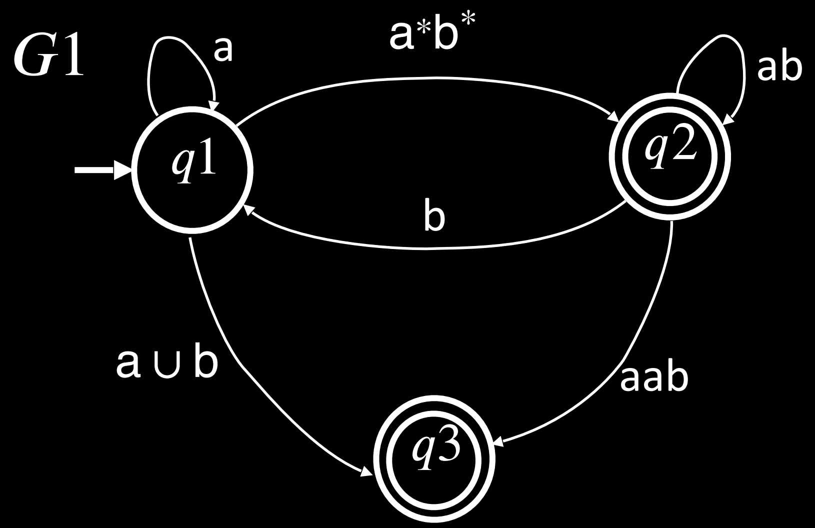
WAIT! Need new concept first.

Generalized NFA

Defn: A Generalized Nondeterministic Finite Automaton (GNFA) is similar to an NFA, but allows regular expressions as transition labels

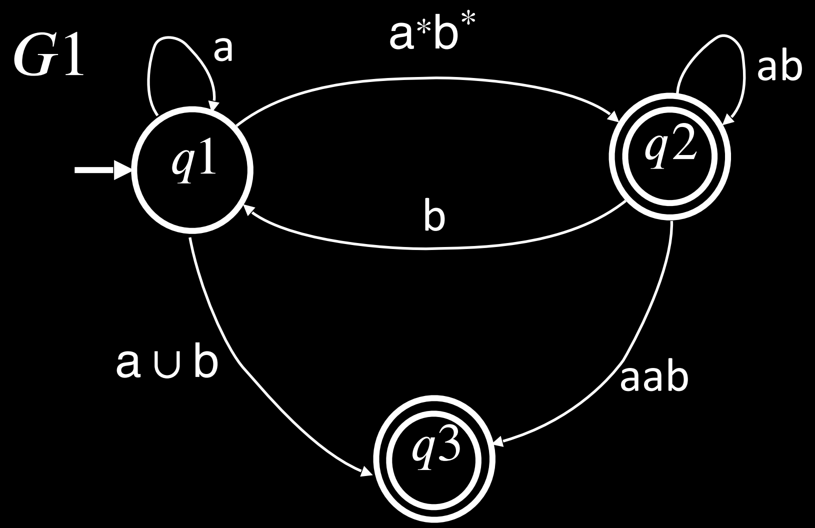
Generalized NFA

Defn: A <u>Generalized Nondeterministic Finite Automaton</u> (GNFA) is similar to an NFA, but allows regular expressions as transition labels



Generalized NFA

Defn: A <u>Generalized Nondeterministic Finite Automaton</u> (GNFA) is similar to an NFA, but allows regular expressions as transition labels



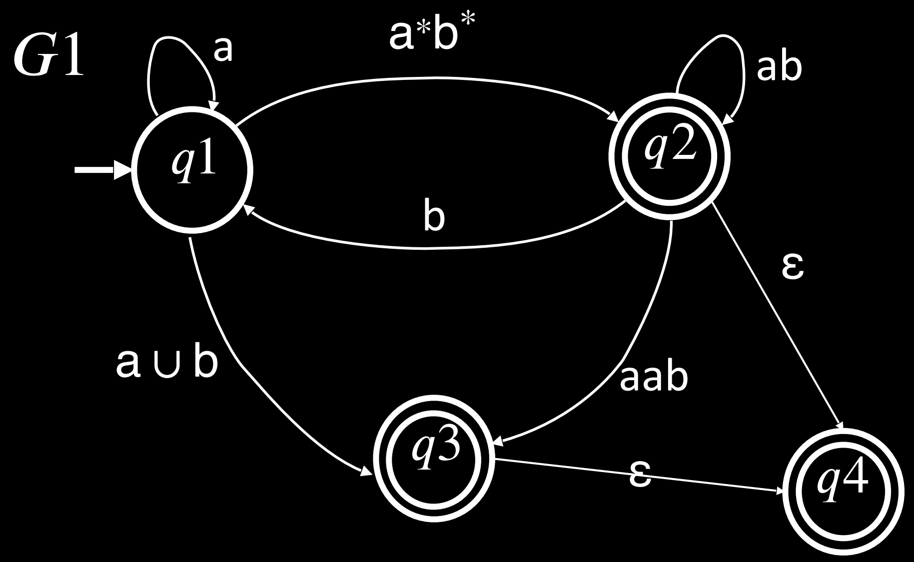
For convenience we will assume:

- One accept state, separate from the start state
- One arrow from each state to each state, except
  a) only exiting the start state
  b) only entering the accept state

We can easily modify a GNFA to have this <u>special form</u>.

Generalized NFA

Defn: A <u>Generalized Nondeterministic Finite Automaton</u> (GNFA) is similar to an NFA, but allows regular expressions as transition labels



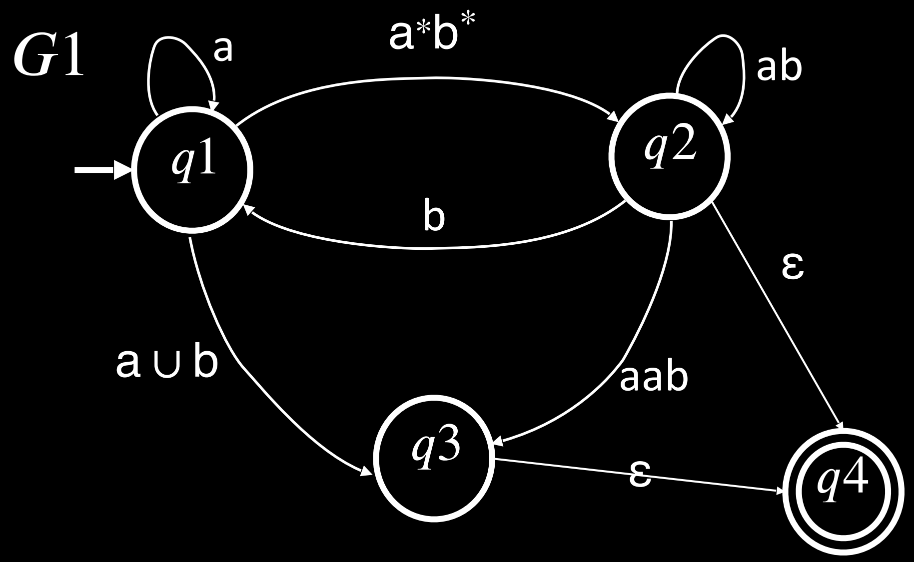
For convenience we will assume:

- One accept state, separate from the start state
- One arrow from each state to each state, except
  a) only exiting the start state
  b) only entering the accept state

We can easily modify a GNFA to have this <u>special form</u>.

Generalized NFA

Defn: A <u>Generalized Nondeterministic Finite Automaton</u> (GNFA) is similar to an NFA, but allows regular expressions as transition labels



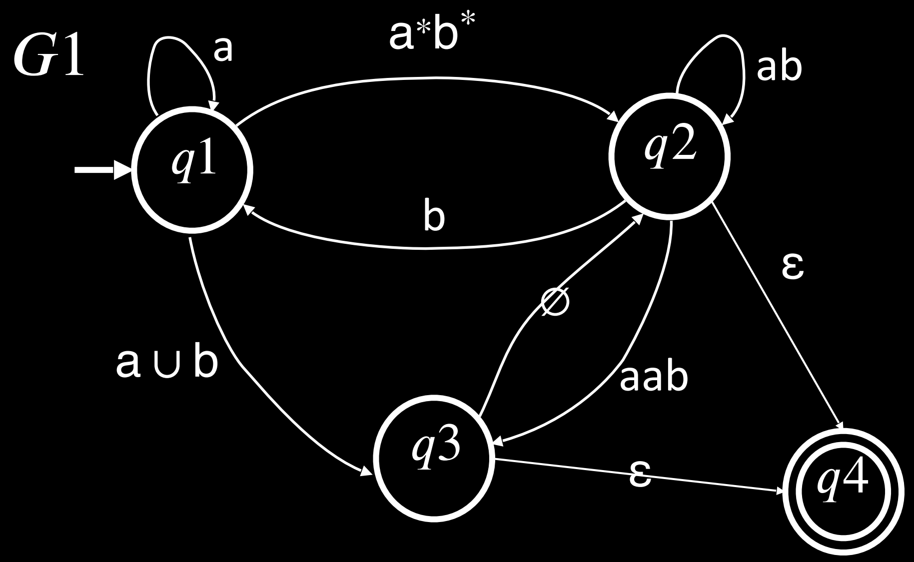
For convenience we will assume:

- One accept state, separate from the start state
- One arrow from each state to each state, except
  a) only exiting the start state
  b) only entering the accept state

We can easily modify a GNFA to have this <u>special form</u>.

Generalized NFA

Defn: A <u>Generalized Nondeterministic Finite Automaton</u> (GNFA) is similar to an NFA, but allows regular expressions as transition labels



For convenience we will assume:

- One accept state, separate from the start state
- One arrow from each state to each state, except
  a) only exiting the start state
  b) only entering the accept state

We can easily modify a GNFA to have this <u>special form</u>.

GNFA $\rightarrow$ Regular Expressions

Lemma: Every GNFA G has an equivalent regular expression R

GNFA $\rightarrow$ Regular Expressions

Lemma: Every GNFA G has an equivalent regular expression R

Proof: By induction on the number of states k of G

GNFA $\rightarrow$ Regular Expressions

Lemma: Every GNFA G has an equivalent regular expression R

Proof: By induction on the number of states k of G

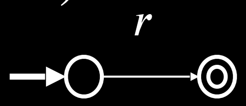
Basis ($k = 2$):

GNFA $\rightarrow$ Regular Expressions

Lemma: Every GNFA G has an equivalent regular expression R

Proof: By induction on the number of states k of G

<u>Basis</u> $(k = 2)$:

$G =$ 

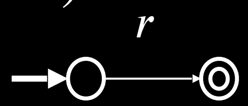
Remember: G is in special form

GNFA $\rightarrow$ Regular Expressions

Lemma: Every GNFA G has an equivalent regular expression R

Proof: By induction on the number of states k of G

<u>Basis</u> $(k = 2)$:

$G =$ 

Remember: G is in special form

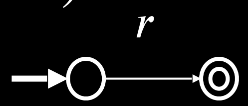
Let $R = r$

GNFA $\rightarrow$ Regular Expressions

Lemma: Every GNFA G has an equivalent regular expression R

Proof: By induction on the number of states k of G

<u>Basis</u> $(k = 2)$:

$G =$ 

Remember: G is in special form

Let $R = r$

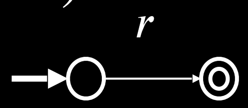
Induction step $(k > 2)$: Assume Lemma true for $k-1$ states and prove for k states

GNFA $\rightarrow$ Regular Expressions

Lemma: Every GNFA G has an equivalent regular expression R

Proof: By induction on the number of states k of G

<u>Basis</u> $(k = 2)$:

$G =$ 

Remember: G is in special form

Let $R = r$

Induction step $(k > 2)$: Assume Lemma true for $k - 1$ states and prove for k states

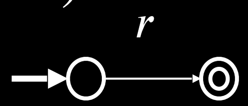
IDEA: Convert k-state GNFA to equivalent $(k - 1)$ -state GNFA

GNFA $\rightarrow$ Regular Expressions

Lemma: Every GNFA G has an equivalent regular expression R

Proof: By induction on the number of states k of G

<u>*Basis*</u> $(k = 2)$:

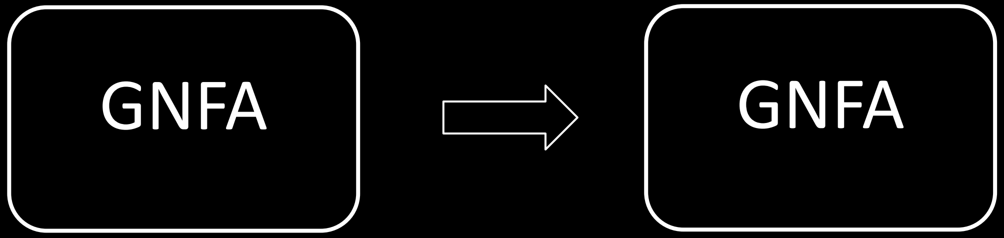
$G =$ 

Remember: G is in special form

Let $R = r$

Induction step $(k > 2)$: Assume Lemma true for $k - 1$ states and prove for k states

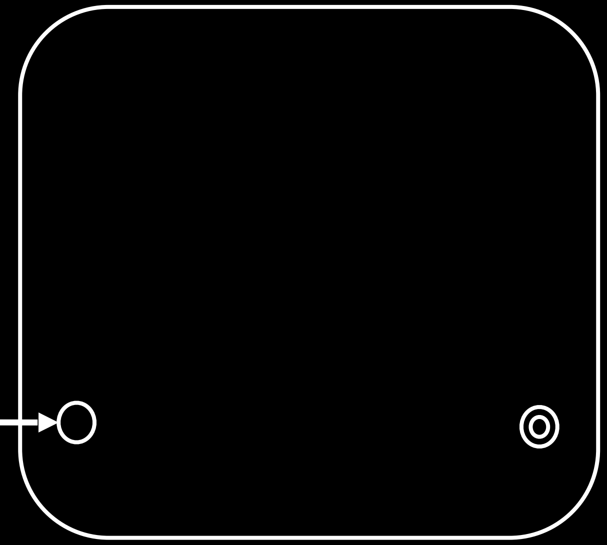
IDEA: Convert k-state GNFA to equivalent $(k - 1)$ -state GNFA



k states $\qquad$ $k - 1$ states

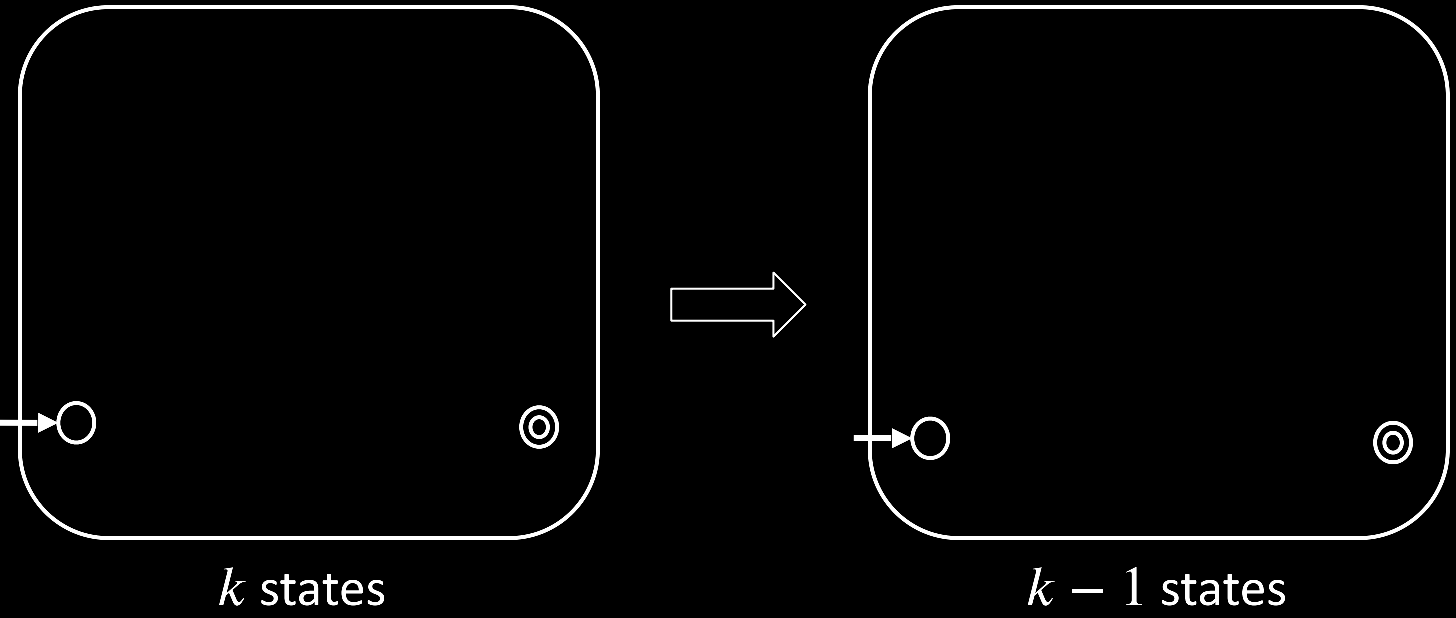
k-state GNFA $\rightarrow$ $(k-1)$-state GNFA

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA

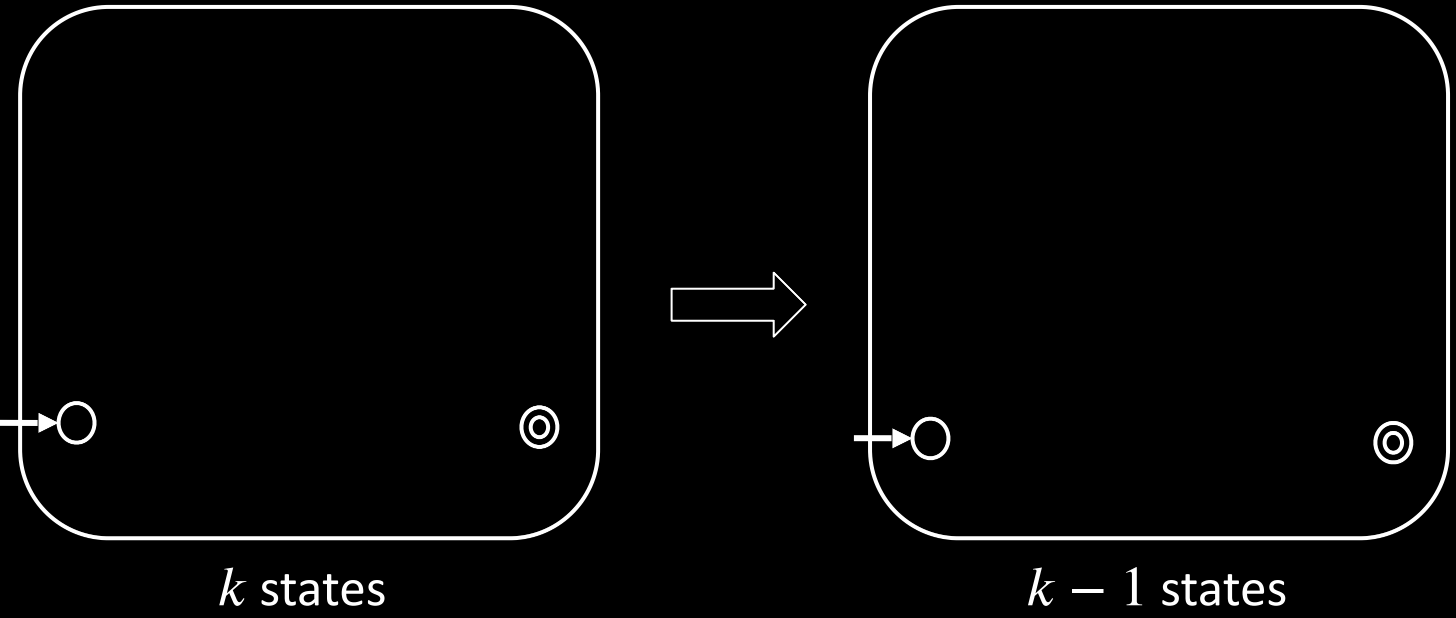


k states

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA

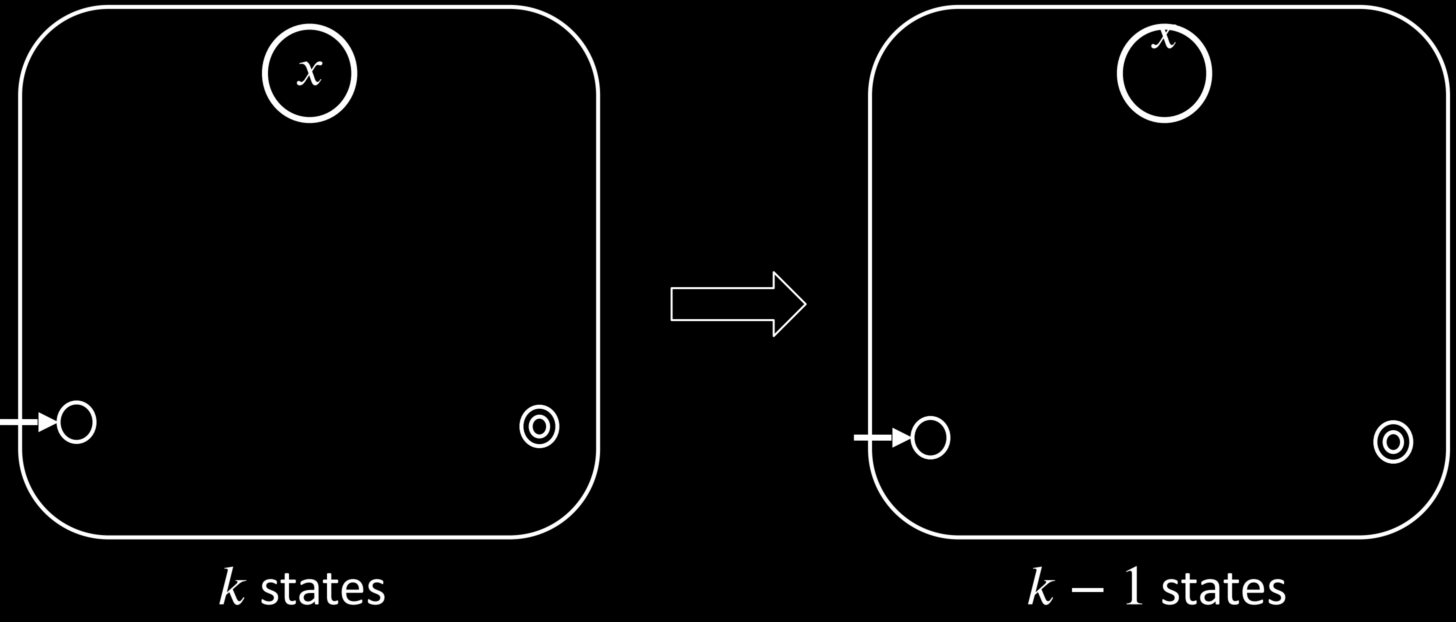


k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



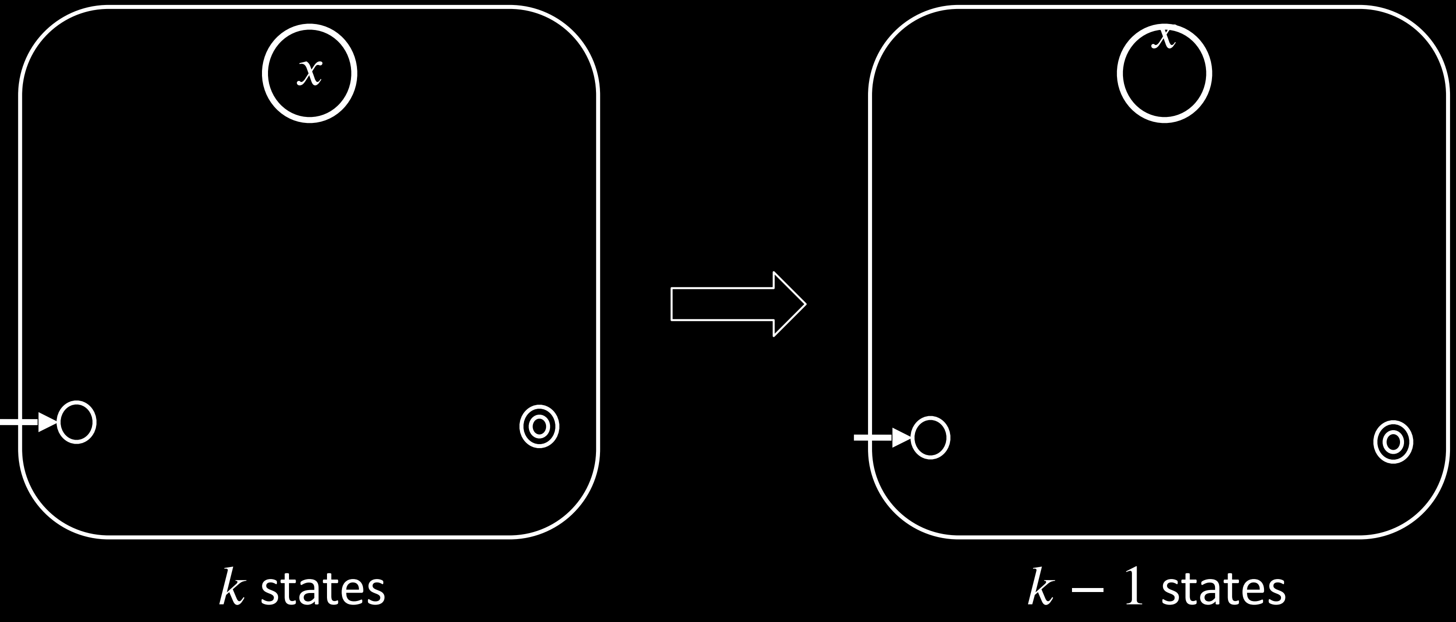
1. Pick any state x except the start and accept states.

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



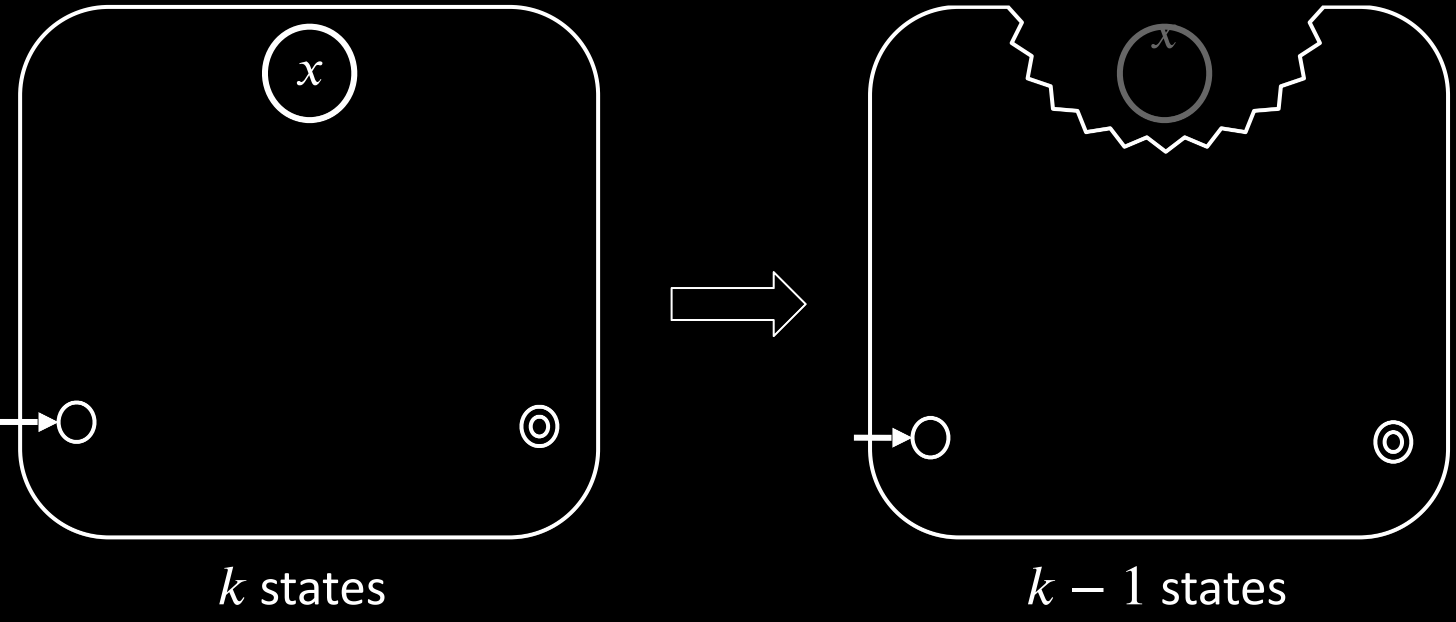
1. Pick any state x except the start and accept states.

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



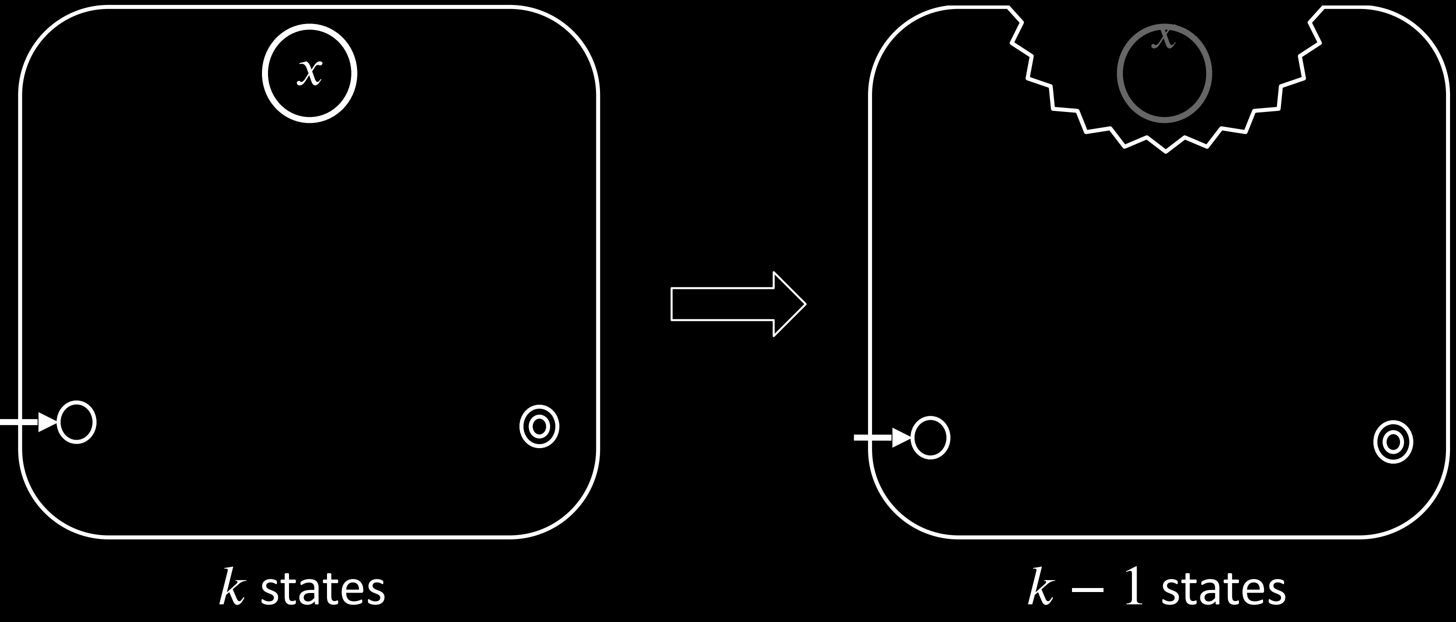
1. Pick any state x except the start and accept states.
2. Remove x.

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



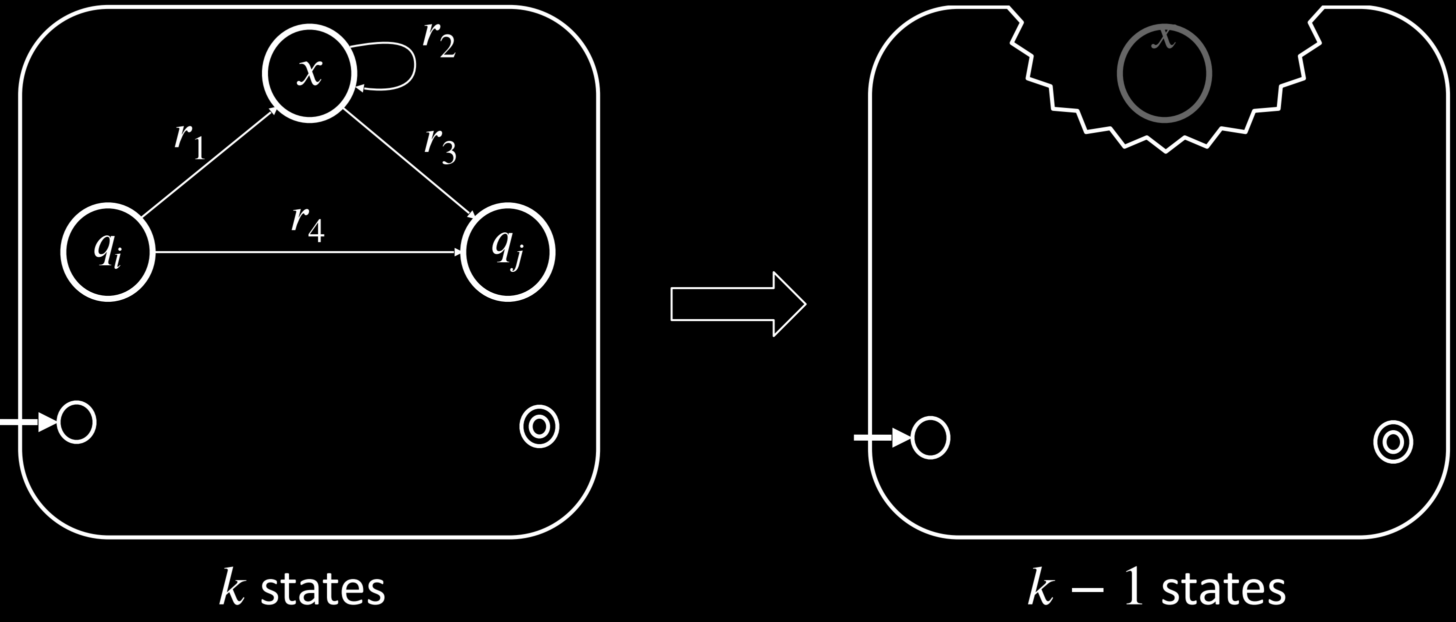
1. Pick any state x except the start and accept states.
2. Remove x.

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



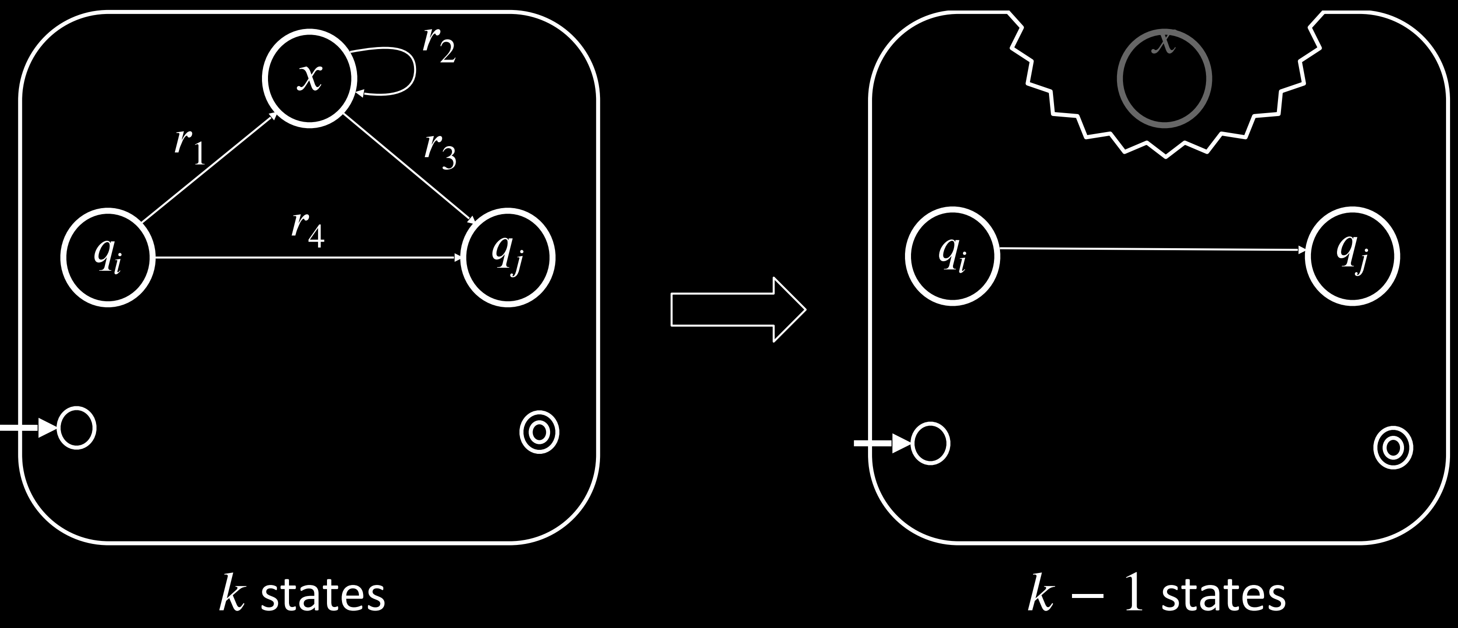
1. Pick any state x except the start and accept states.
2. Remove x.
3. Repair the damage by recovering all paths that went through x.

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



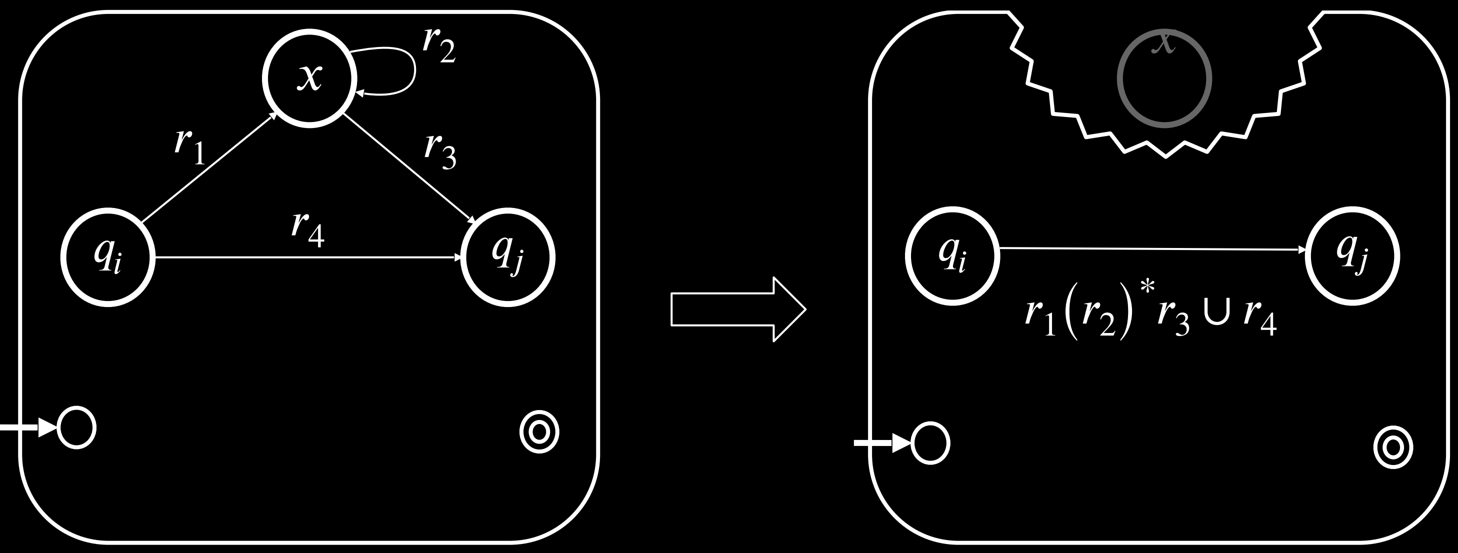
1. Pick any state x except the start and accept states.
2. Remove x.
3. Repair the damage by recovering all paths that went through x.

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



1. Pick any state x except the start and accept states.
2. Remove x.
3. Repair the damage by recovering all paths that went through x.

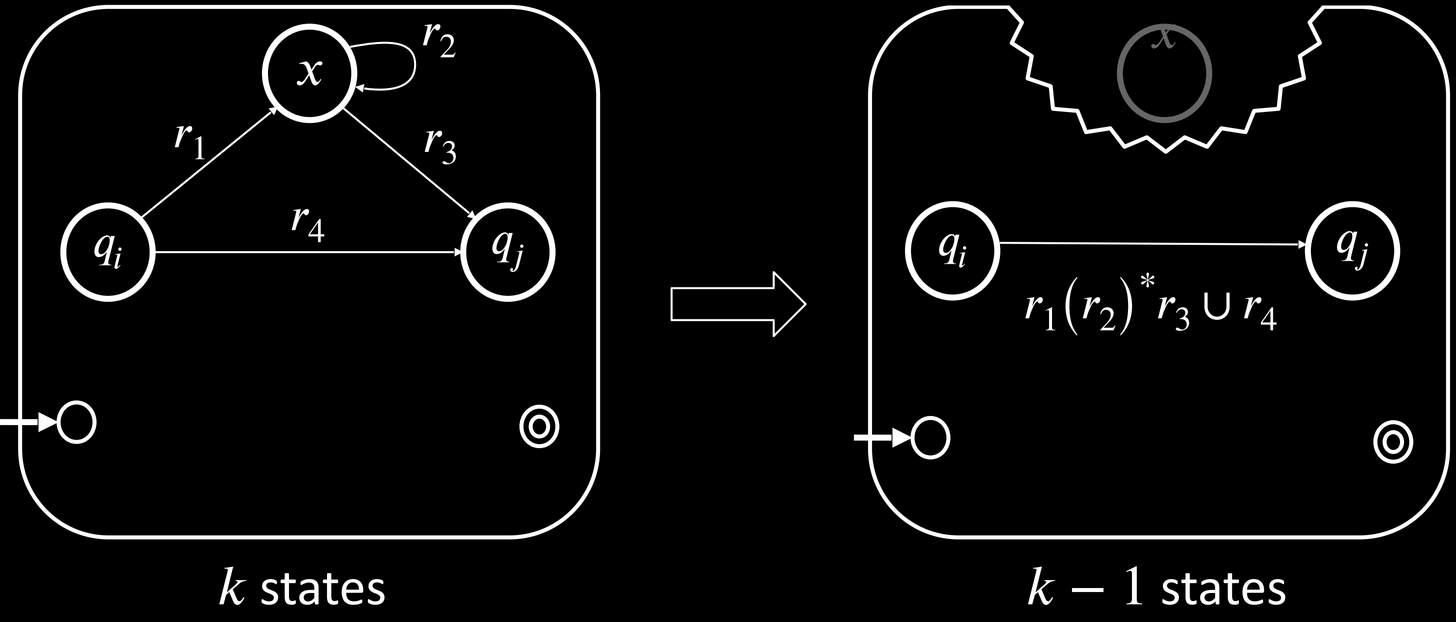
k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



k states $k-1$ states

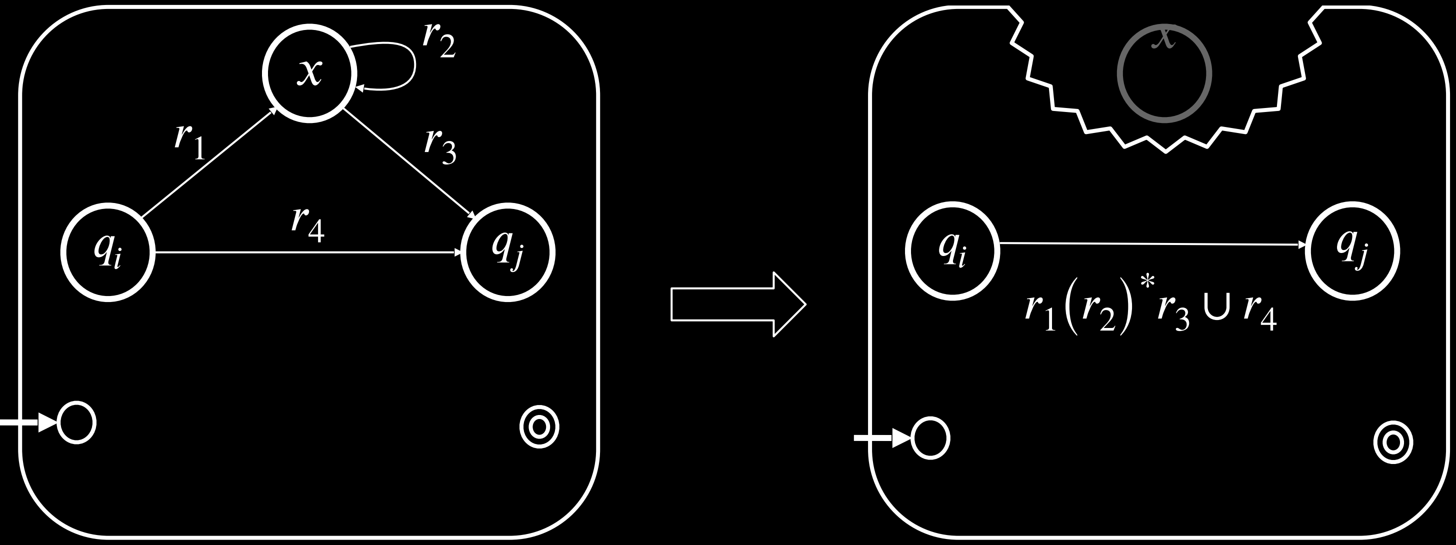
1. Pick any state x except the start and accept states.
2. Remove x.
3. Repair the damage by recovering all paths that went through x.

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



1. Pick any state x except the start and accept states.
2. Remove x.
3. Repair the damage by recovering all paths that went through x.
4. Make the indicated change for each pair of states q_i, q_j.

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



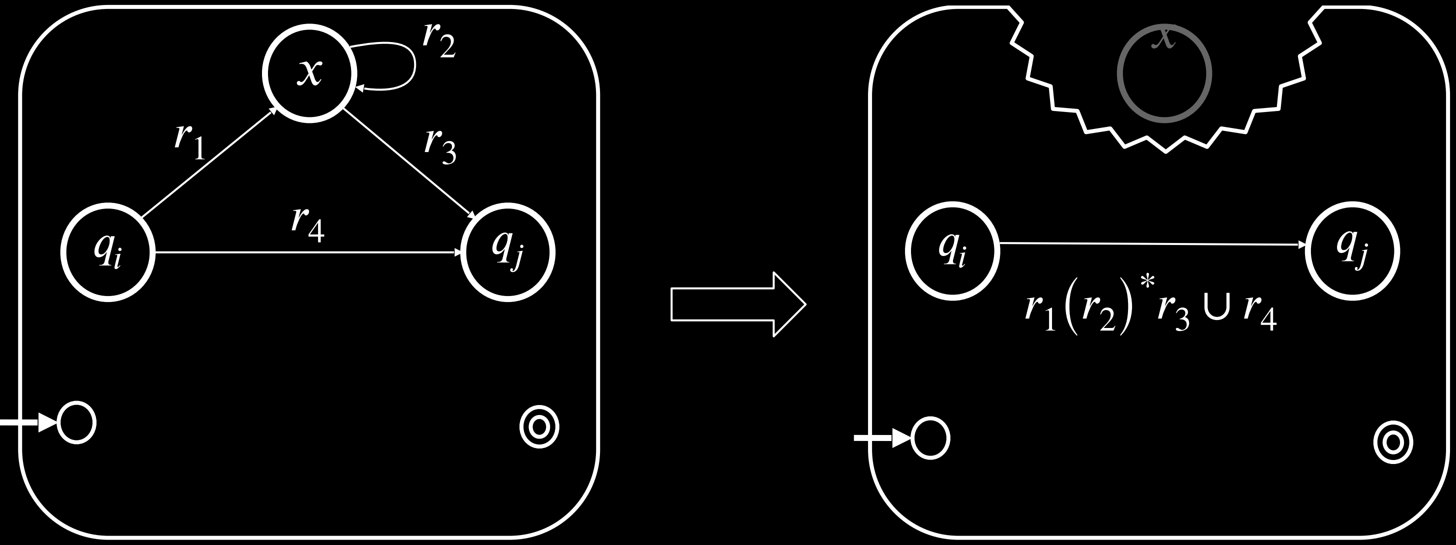
k states

$k-1$ states

1. Pick any state x except the start and accept states.
2. Remove x.
3. Repair the damage by recovering all paths that went through x.
4. Make the indicated change for each pair of states q_i, q_j.

Thus DFAs and regular expressions are equivalent.

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA



k states

$k-1$ states

1. Pick any state x except the start and accept states.
2. Remove x.
3. Repair the damage by recovering all paths that went through x.
4. Make the indicated change for each pair of states q_i, q_j.

Thus DFAs and regular expressions are equivalent.

Check-in 3.1

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA

Check-in 3.1

We just showed how to convert GNFAs to regular expressions but our goal was to show that how to convert DFAs to regular expressions. How do we finish our goal?

(a) Show how to convert DFAs to GNFAs

(b) Show how to convert GNFAs to DFAs

(c) We are already done. DFAs are a type of GNFAs.

1. Pick any state x except the start and accept states.
2. Remove x.
3. Repair the damage by recovering all paths that went through x.
4. Make the indicated change for each pair of states q_i, q_j.

Thus DFAs and regular expressions are equivalent.

Check-in 3.1