


01204211 Discrete Mathematics  
Lecture 10a: Nondeterministic automata<sup>1</sup>

Jittat Fakcharoenphol

September 5, 2023

<sup>1</sup>Based on lecture notes of *Models of Computation* course by Jeff Erickson. 

## Review: DFA (Formal definitions)

A **finite-state machine** or a **deterministic finite-state automaton** (DFA) has five components:

- ▶ the input alphabet  $\Sigma$ ,
- ▶ a finite set of states  $Q$ ,
- ▶ a transition function  $\delta : Q \times \Sigma \longrightarrow Q$
- ▶ a start state  $s \in Q$ , and
- ▶ a subset  $A \subseteq Q$  of accepting states.

## Review: Acceptance

**One step move:** from state  $q$  with input symbol  $a$ , the machine changes its state to

$\delta(q, a)$ .

→ **Extension:** from state  $q$  with input string  $w$ , the machine changes its state to  $\delta^*(q, w)$  defined as

$$\underline{\delta^*(q, w)} = \begin{cases} q & \text{if } w = \varepsilon, \\ \delta^*(\delta(q, a), x) & \text{if } w = ax. \end{cases}$$

The signature of  $\delta^*$  is  $Q \times \Sigma^* \longrightarrow Q$ .

→ accepting  $w$

For a finite-state machine with starting state  $s$  and accepting states  $A$ , it accepts string  $w$  iff

$$\delta^*(s, w) \in A.$$

# Language of a DFA

$L(M)$

For a DFA  $M$ , let  $L(M)$  be the set of all strings that  $M$  accepts. More formally, for  $M = (\Sigma, Q, \delta, s, A)$ ,

$$\underline{L(M)} = \{w \in \Sigma^* \mid \delta^*(s, w) \in A\}.$$

We refer to  $L(M)$  as the language of  $M$ .

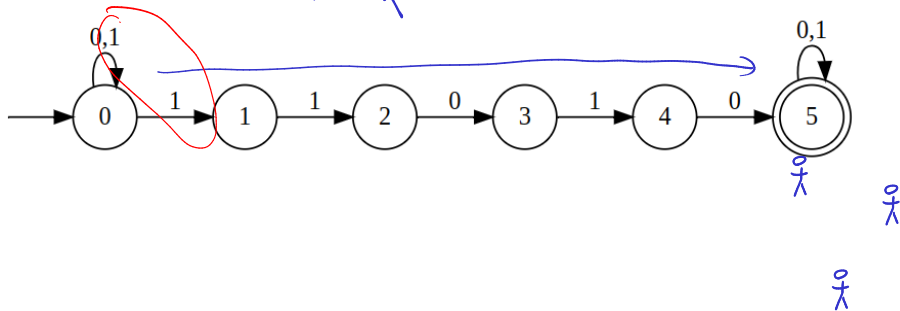
## Acceptance

We also says  $M$  "accepts"  $L(M)$ .

## New example 1

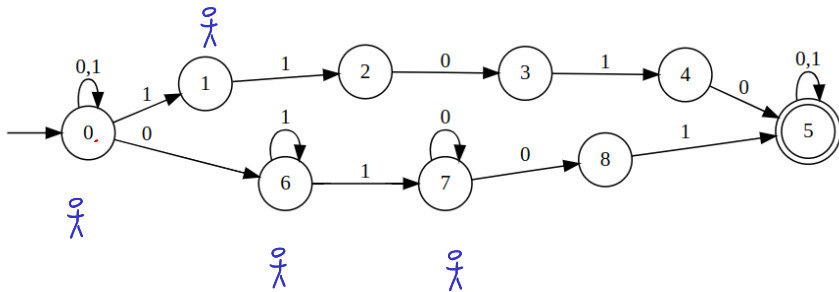
input: 01101110101011

hint: x N N x N x x x x x x x x



## New example 2

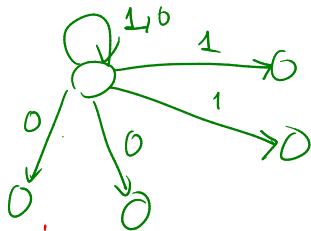
011 011



What's going on here?

- nondeterministic

from state  $q$  on input  $a$   
 $N$  transitions to  $r$  state.



$q$ : accept string  $w$  if there exists  $q'$   
 $N$  transitions to  $q'$  accepting state.

## More relaxed transitions

From state  $q \in Q$ , for input  $a$ , the machine can “possibly” change its state to many states.  $= \emptyset$



## More relaxed transitions

$$A = \{1, 2, 3\}$$
$$\boxed{2^A} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

From state  $q \in Q$ , for input  $a$ , the machine can “possibly” change its state to many states.

New transition function  $\delta :$

$$\begin{array}{c} Q \times \Sigma \\ \uparrow \quad \uparrow \end{array} \rightarrow \cancel{\emptyset} \quad \boxed{2^Q}$$

## More relaxed transitions

From state  $q \in Q$ , for input  $a$ , the machine can “possibly” change its state to many states.

New transition function  $\delta : \overset{\downarrow}{Q} \times \overset{\downarrow}{\Sigma} \longrightarrow \boxed{2^Q} = \phi$

We refer to this new kind of automaton as a nondeterministic finite-state automaton or NFA.

# NFA (Formal definitions)

A **nondeterministic finite-state automaton** (NFA) has five components:

- ▶ the input alphabet  $\Sigma$ ,

# NFA (Formal definitions)

A **nondeterministic finite-state automaton** (NFA) has five components:

- ▶ the input alphabet  $\Sigma$ ,
- ▶ a finite set of states  $Q$ ,

# NFA (Formal definitions)

A **nondeterministic finite-state automaton** (NFA) has five components:

- ▶ the input alphabet  $\Sigma$ ,
- ▶ a finite set of states  $Q$ ,
- ▶ a transition function  $\delta$

# NFA (Formal definitions)

$Q$

A **nondeterministic finite-state automaton** (NFA) has five components:

- ▶ the input alphabet  $\Sigma$ ,
- ▶ a finite set of states  $Q$ ,
- ▶ a transition function  $\delta : Q \times \Sigma \longrightarrow 2^Q$

# NFA (Formal definitions)

A **nondeterministic finite-state automaton** (NFA) has five components:

- ▶ the input alphabet  $\Sigma$ ,
- ▶ a finite set of states  $Q$ ,
- ▶ a transition function  $\delta : Q \times \Sigma \longrightarrow 2^Q$
- ▶ a start state  $s \in Q$ , and
- ▶ a subset  $A \subseteq Q$  of accepting states.



$$\delta^*(q, w)$$

# NFA (Formal definitions)

A **nondeterministic finite-state automaton** (NFA) has five components:

- ▶ the input alphabet  $\Sigma$ ,
- ▶ a finite set of states  $Q$ ,
- ▶ a transition function  $\delta : Q \times \Sigma \longrightarrow 2^Q$
- ▶ a start state  $s \in Q$ , and
- ▶ a subset  $A \subseteq Q$  of accepting states.

Remark:  $\delta$  can return the empty set  $\emptyset$ .



# NFA (Formal definitions)

A **nondeterministic finite-state automaton** (NFA) has five components:

- ▶ the input alphabet  $\Sigma$ ,
- ▶ a finite set of states  $Q$ ,
- ▶ a transition function  $\delta : Q \times \Sigma \longrightarrow 2^Q$
- ▶ a start state  $s \in Q$ , and
- ▶ a subset  $A \subseteq Q$  of accepting states.

Remark:  $\delta$  can return the empty set  $\emptyset$ .

What else do we need to define to “properly” talk about NFAs?

## Transition

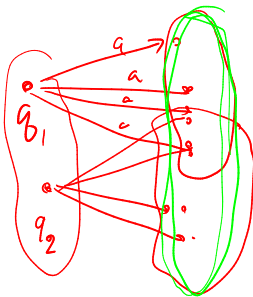
**One step move:** from state  $q$  with input symbol  $a$ , the machine changes its state to one of  $\delta(q, a)$ .

## Transition

One step move: from state  $q$  with input symbol  $a$ , the machine changes its state to one of  $\delta(q, a)$ .

Thus, instead of thinking of a machine that maintains **one** state, we can think of an NFA as a machine that maintains a **set** of states. *state is 1007104.*

If the current set of states is  $C \subseteq Q$  and the input is  $\underline{a} \in \Sigma$  what would the new set of states be?



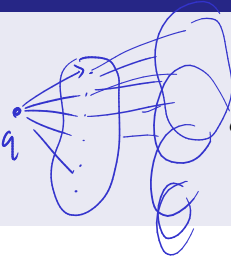
## Transition

**One step move:** from state  $q$  with input symbol  $a$ , the machine changes its state to one of  $\delta(q, a)$ .

Thus, instead of thinking of a machine that maintains **one** state, we can think of an NFA as a machine that maintains a **set** of states.

If the current set of states is  $C \subseteq Q$  and the input is  $a \in \Sigma$  what would the new set of states be?

**Extension:** from state  $q$  with input string  $w$ , the machine changes its set of states  $\delta^*(q, w)$  defined as


$$\delta^*(q, w) = \begin{cases} \{q\} & \text{if } w = \epsilon, \\ \delta^*(\delta(q, a), x) & \text{if } w = ax \end{cases}$$

$w = a(x)$

## Transition

**One step move:** from state  $q$  with input symbol  $a$ , the machine changes its state to one of  $\delta(q, a)$ .

Thus, instead of thinking of a machine that maintains **one** state, we can think of an NFA as a machine that maintains a **set** of states.

If the current set of states is  $C \subseteq Q$  and the input is  $a \in \Sigma$  what would the new set of states be?

**Extension:** from state  $q$  with input string  $w$ , the machine changes its set of states  $\delta^*(q, w)$  defined as

$$\delta^*(q, w) = \begin{cases} \{q\} & \text{if } w = \varepsilon, \\ \bigcup_{r \in \delta(q, a)} \delta^*(r, x) & \text{if } w = ax. \end{cases}$$

The signature of  $\delta^*$  is  $Q \times \Sigma^* \longrightarrow 2^Q$ .

# Acceptance

## accepting $w$

For a nondeterministic finite-state machine with starting state  $s$  and accepting states  $A$ , it accepts string  $w$  iff

$$\delta^*(s, w) \cap A \neq \emptyset.$$

# Interpretation

## Interpretation

- Clairvoyance.

มองเห็นอนาคต  $\rightarrow$  ถ้ารู้สถานะที่จะไปถึง accepting state ของ NFA จะรู้ผลการทำงานนั้น



# Interpretation

► Clairvoyance.

► Parallel threads. → หมายเหตุ: บบต๋อ: 1027 - 1 กั๋น ๕๓๓๓: ๗ "๕๓๓๓๓๓"

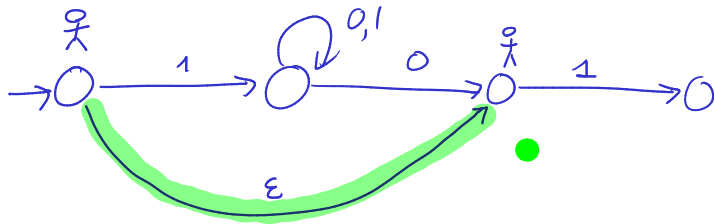


# Interpretation

- ▶ Clairvoyance.
- ▶ Parallel threads.
- ▶ Proofs/oracles.

input :  $\boxed{X}$   
proofs/oracles/hints /  $\boxed{5}$

# $\epsilon$ -transition



## $\varepsilon$ -transition

An NFA accepts string  $w$  iff there is a sequence of transitions

$$s \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} q_3 \xrightarrow{a_4} \cdots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} q_k,$$

where  $q_k \in A$  and  $w = a_1 a_2 \cdots a_k$  where  $a_i \in \Sigma \cup \{\varepsilon\}$  for  $1 \leq i \leq k$ .

## $\varepsilon$ -transition

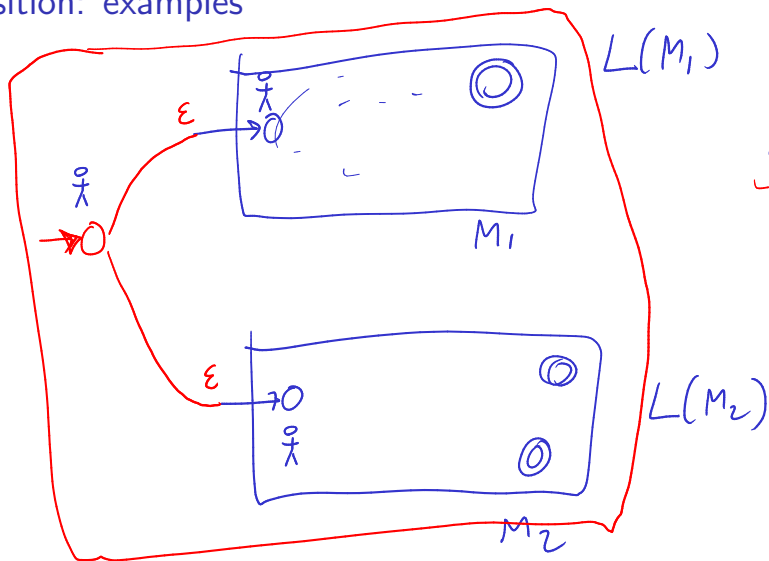
An NFA accepts string  $w$  iff there is a sequence of transitions

$$s \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} q_3 \xrightarrow{a_4} \cdots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} q_k,$$

where  $q_k \in A$  and  $w = a_1 a_2 \cdots a_k$  where  $a_i \in \Sigma \cup \{\varepsilon\}$  for  $1 \leq i \leq k$ .

The transition function also changes its domain to  $Q \times (\Sigma \cup \{\varepsilon\})$ .

## $\epsilon$ -transition: examples



$$\underline{L(M_1) \cup L(M_2)}$$

## $\varepsilon$ -reach

The  $\varepsilon$ -reach of state  $q \in Q$  (denoted by  $\varepsilon\text{-reach}(q)$ ) consists of all states  $r$  that satisfy one of the following conditions:

- ▶  $r = q$ , or
- ▶  $r \in \delta(q', \varepsilon)$  for some state  $q'$  in the  $\varepsilon$ -reach of  $q$ .

## Extended transition function: $\delta^*$

We define  $\delta^* : Q \times \Sigma^* \longrightarrow 2^Q$  as follows:

$$\delta^*(q, w) = \begin{cases} \varepsilon\text{-reach}(p) & \text{if } w = \varepsilon \\ \bigcup_{r \in \varepsilon\text{-reach}(p)} \bigcup_{q \in \delta(r, q)} \delta^*(q, x) & \text{if } w = ax. \end{cases}$$



## Notation abuse

We sometimes also write, for subset  $S \subseteq Q$ ,

$$\delta(S, a) = \bigcup_{q \in S} \delta(q, a),$$

## Notation abuse

We sometimes also write, for subset  $S \subseteq Q$ ,

$$\delta(S, a) = \bigcup_{q \in S} \delta(q, a),$$

$$\delta^*(S, a) = \bigcup_{q \in S} \delta^*(q, a),$$

## Notation abuse

We sometimes also write, for subset  $S \subseteq Q$ ,

$$\delta(S, a) = \bigcup_{q \in S} \delta(q, a),$$

$$\delta^*(S, a) = \bigcup_{q \in S} \delta^*(q, a),$$

and

$$\varepsilon\text{-reach}(S) = \bigcup_{q \in S} \varepsilon\text{-reach}(q).$$

## Extended transition function: $\delta^*$ (with shorter notation)

We define  $\delta^* : Q \times \Sigma^* \longrightarrow 2^Q$  as follows:

$$\delta^*(q, w) = \begin{cases} \varepsilon\text{-reach}(p) & \text{if } w = \varepsilon \\ \delta^*(\delta(\varepsilon\text{-reach}(p), a), x) & \text{if } w = ax. \end{cases}$$

## Removing $\varepsilon$ -transitions: idea

## Lemma 1

*For any NFA  $M = (\Sigma, Q, \delta, s, A)$  with  $\varepsilon$ -transitions, there is an NFA  $M' = (\Sigma, Q', \delta', s', A')$  without  $\varepsilon$ -transitions such that  $L(M) = L(M')$ .*

Proof.



## Main question

- ▶ We see that  $\varepsilon$ -transitions does not add any “power” to the machine.
- ▶ Does nondeterminism add any power to NFA (over typical DFA)?

# Simulating parallel machines



# Subset construction: idea

## NFA to DFA: subset construction

Given an NFA  $M = (\Sigma, Q, \delta, s, A)$ , we can construct an equivalent DFA  $M' = (\Sigma, Q', \delta', s', A')$  as follows:

- ▶ Let  $Q' = 2^Q$ ,
- ▶  $s' = \{s\}$ ,

## NFA to DFA: subset construction

Given an NFA  $M = (\Sigma, Q, \delta, s, A)$ , we can construct an equivalent DFA  $M' = (\Sigma, Q', \delta', s', A')$  as follows:

- ▶ Let  $Q' = 2^Q$ ,
- ▶  $s' = \{s\}$ ,
- ▶  $A' = \{S \subseteq Q \mid S \cap A \neq \emptyset\}$ ,
- ▶ and let  $\delta' : Q' \times \Sigma \longrightarrow Q'$  be such that

## NFA to DFA: subset construction

Given an NFA  $M = (\Sigma, Q, \delta, s, A)$ , we can construct an equivalent DFA  $M' = (\Sigma, Q', \delta', s', A')$  as follows:

- ▶ Let  $Q' = 2^Q$ ,
- ▶  $s' = \{s\}$ ,
- ▶  $A' = \{S \subseteq Q \mid S \cap A \neq \emptyset\}$ ,
- ▶ and let  $\delta' : Q' \times \Sigma \longrightarrow Q'$  be such that

$$\delta'(q', a) = \bigcup_{p \in q'} \delta(p, a),$$

for all  $q' \subseteq Q$  and  $a \in \Sigma$ .

# Example

## Kleene's Theorem

Every language  $L$  can be described by a regular expression if and only if  $L$  is the language accepted by a DFA.

## Kleene's Theorem

Every language  $L$  can be described by a regular expression if and only if  $L$  is the language accepted by a DFA.

Steps:

- ▶ Every DFA can be transformed into an equivalent NFA.

## Kleene's Theorem

Every language  $L$  can be described by a regular expression if and only if  $L$  is the language accepted by a DFA.

Steps:

- ▶ Every DFA can be transformed into an equivalent NFA. (trivial)



## Kleene's Theorem

Every language  $L$  can be described by a regular expression if and only if  $L$  is the language accepted by a DFA.

Steps:

- ▶ Every DFA can be transformed into an equivalent NFA. (trivial)
- ▶ Every NFA can be transformed into an equivalent DFA.

## Kleene's Theorem

Every language  $L$  can be described by a regular expression if and only if  $L$  is the language accepted by a DFA.

Steps:

- ▶ Every DFA can be transformed into an equivalent NFA. (trivial)
- ▶ Every NFA can be transformed into an equivalent DFA. (done)

## Kleene's Theorem

Every language  $L$  can be described by a regular expression if and only if  $L$  is the language accepted by a DFA.

Steps:

- ▶ Every DFA can be transformed into an equivalent NFA. (trivial)
- ▶ Every NFA can be transformed into an equivalent DFA. (done)
- ▶ Every regular expression can be transformed into an equivalent NFA.

## Kleene's Theorem

Every language  $L$  can be described by a regular expression if and only if  $L$  is the language accepted by a DFA.

Steps:

- ▶ Every DFA can be transformed into an equivalent NFA. (trivial)
- ▶ Every NFA can be transformed into an equivalent DFA. (done)
- ▶ Every regular expression can be transformed into an equivalent NFA. (TODO)

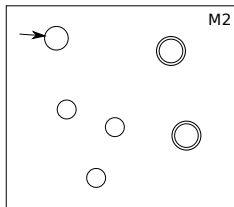
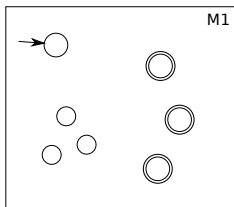
## Kleene's Theorem

Every language  $L$  can be described by a regular expression if and only if  $L$  is the language accepted by a DFA.

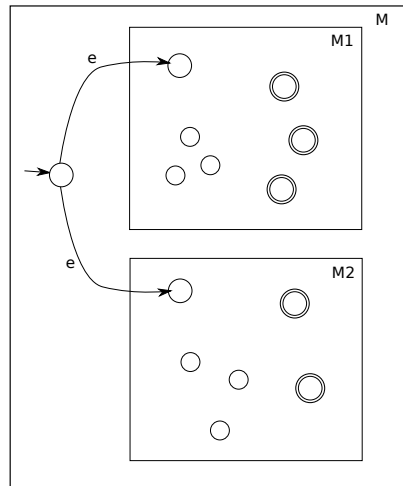
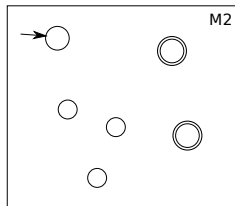
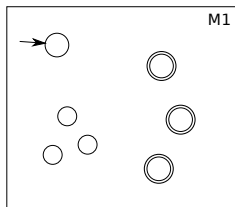
Steps:

- ▶ Every DFA can be transformed into an equivalent NFA. (trivial)
- ▶ Every NFA can be transformed into an equivalent DFA. (done)
- ▶ Every regular expression can be transformed into an equivalent NFA. (TODO)
- ▶ Every NFA can be transformed into an equivalent regular expression. (only idea)

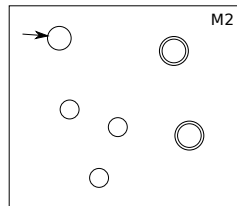
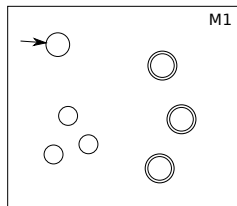
## Warm-up: union of DFA $\Rightarrow$ NFA



## Warm-up: union of DFA $\Rightarrow$ NFA

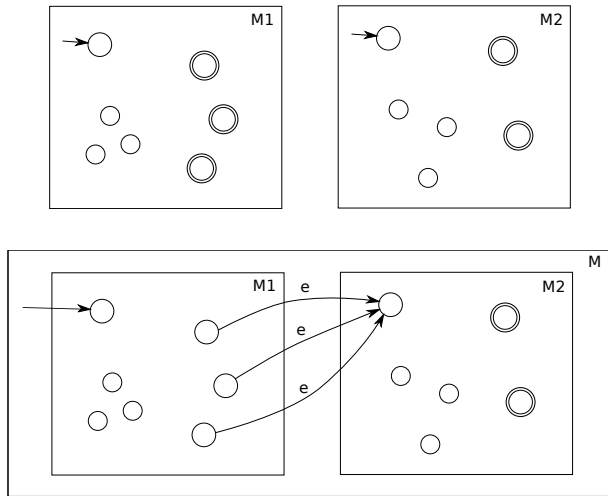


## Concatenation: idea





## Concatenation: idea



## Stronger claim

Our goal is to prove:

### Lemma 2

*Every regular language is accepted by a nondeterministic finite-state automaton.*

# Stronger claim

Our goal is to prove:

## Lemma 2

*Every regular language is accepted by a nondeterministic finite-state automaton.*

But we will prove a “stronger” claim.

## Lemma 3 (Thompson’s algorithm)

*Every regular language is accepted by a nondeterministic finite-state automaton with exactly one accepting state, which is different from its start state.*

## Proof (Thompson's algorithm).

Consider any regular expression  $R$  over alphabet  $\Sigma$ . We prove that there is an NFA  $N$  that accepts the language described by  $R$  by induction.

**Induction hypothesis:** for any subexpression  $S$  of  $R$ , there is an NFA that accepts the language described by  $S$ .

We denote an NFA with this notation:

## Proof (Thompson's algorithm).

Consider any regular expression  $R$  over alphabet  $\Sigma$ . We prove that there is an NFA  $N$  that accepts the language described by  $R$  by induction.

**Induction hypothesis:** for any subexpression  $S$  of  $R$ , there is an NFA that accepts the language described by  $S$ .

We denote an NFA with this notation:

There are 6 cases:

## Proof (Thompson's algorithm).

Consider any regular expression  $R$  over alphabet  $\Sigma$ . We prove that there is an NFA  $N$  that accepts the language described by  $R$  by induction.

**Induction hypothesis:** for any subexpression  $S$  of  $R$ , there is an NFA that accepts the language described by  $S$ .

We denote an NFA with this notation:

There are 6 cases:

►  $R = \emptyset$ :

## Proof (Thompson's algorithm).

Consider any regular expression  $R$  over alphabet  $\Sigma$ . We prove that there is an NFA  $N$  that accepts the language described by  $R$  by induction.

**Induction hypothesis:** for any subexpression  $S$  of  $R$ , there is an NFA that accepts the language described by  $S$ .

We denote an NFA with this notation:

There are 6 cases:

- ▶  $R = \emptyset$ :
- ▶  $R = \varepsilon$ :

## Proof (Thompson's algorithm).

Consider any regular expression  $R$  over alphabet  $\Sigma$ . We prove that there is an NFA  $N$  that accepts the language described by  $R$  by induction.

**Induction hypothesis:** for any subexpression  $S$  of  $R$ , there is an NFA that accepts the language described by  $S$ .

We denote an NFA with this notation:

There are 6 cases:

- ▶  $R = \emptyset$ :
- ▶  $R = \varepsilon$ :
- ▶  $R = a$  for some  $a \in \Sigma$ :



## Proof (Thompson's algorithm).

Consider any regular expression  $R$  over alphabet  $\Sigma$ . We prove that there is an NFA  $N$  that accepts the language described by  $R$  by induction.

**Induction hypothesis:** for any subexpression  $S$  of  $R$ , there is an NFA that accepts the language described by  $S$ .

We denote an NFA with this notation:

There are 6 cases:

- ▶  $R = \emptyset$ :
- ▶  $R = \varepsilon$ :
- ▶  $R = a$  for some  $a \in \Sigma$ :
- ▶  $R = ST$  for some regular expression  $S$  and  $T$ :

## Proof (Thompson's algorithm).

Consider any regular expression  $R$  over alphabet  $\Sigma$ . We prove that there is an NFA  $N$  that accepts the language described by  $R$  by induction.

**Induction hypothesis:** for any subexpression  $S$  of  $R$ , there is an NFA that accepts the language described by  $S$ .

We denote an NFA with this notation:

There are 6 cases:

- ▶  $R = \emptyset$ :
- ▶  $R = \varepsilon$ :
- ▶  $R = a$  for some  $a \in \Sigma$ :
- ▶  $R = ST$  for some regular expression  $S$  and  $T$ :
- ▶  $R = S + T$  for some regular expression  $S$  and  $T$ :

## Proof (Thompson's algorithm).

Consider any regular expression  $R$  over alphabet  $\Sigma$ . We prove that there is an NFA  $N$  that accepts the language described by  $R$  by induction.

**Induction hypothesis:** for any subexpression  $S$  of  $R$ , there is an NFA that accepts the language described by  $S$ .

We denote an NFA with this notation:

There are 6 cases:

- ▶  $R = \emptyset$ :
- ▶  $R = \varepsilon$ :
- ▶  $R = a$  for some  $a \in \Sigma$ :
- ▶  $R = ST$  for some regular expression  $S$  and  $T$ :
- ▶  $R = S + T$  for some regular expression  $S$  and  $T$ :
- ▶  $R = S^*$  for some regular expression  $S$ :

## Proof (Thompson's algorithm).

Consider any regular expression  $R$  over alphabet  $\Sigma$ . We prove that there is an NFA  $N$  that accepts the language described by  $R$  by induction.

**Induction hypothesis:** for any subexpression  $S$  of  $R$ , there is an NFA that accepts the language described by  $S$ .

We denote an NFA with this notation:

There are 6 cases:

- ▶  $R = \emptyset$ :
- ▶  $R = \varepsilon$ :
- ▶  $R = a$  for some  $a \in \Sigma$ :
- ▶  $R = ST$  for some regular expression  $S$  and  $T$ :
- ▶  $R = S + T$  for some regular expression  $S$  and  $T$ :
- ▶  $R = S^*$  for some regular expression  $S$ :

In all cases, the language  $L(R)$  is accepted by an NFA with exactly one accepting state which is different from its start state, as required. □

Example:  $1 + 00$

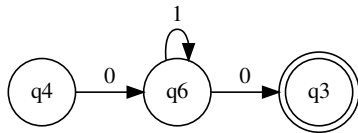
Example:  $(1 + 00)^*$

Example:  $(1 + 00)^* + 1^*0$

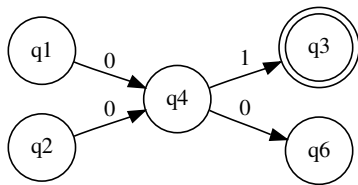
# NFA to Regular expressions



## State elimination: example 1



## State elimination: example 2



## State elimination: example 3

