01204211 Discrete Mathematics
Lecture 10a: Nondeterministic automata[1]

Jittat Fakcharoenphol

September 5, 2023

[1] 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 Σ,
- a finite set of states Q,
- a transition function $\delta : \boxed{Q \times \Sigma \longrightarrow Q}$
- 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

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

The signature of δ^* 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 $M = (\Sigma, Q, \delta, s, A)$,

$$L(M) = \{w \in \Sigma^* \mid \underbrace{\delta^*(s, w) \in A}_{M \text{ accepts string } w}\}.$$

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

Acceptance

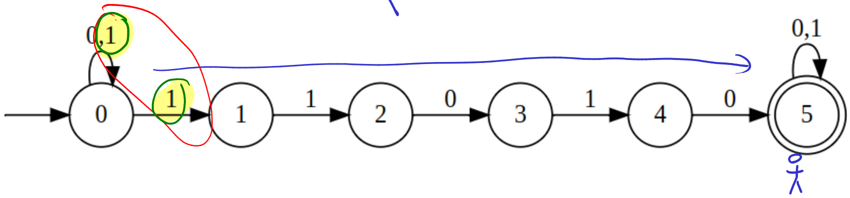
We also says M "**accepts**" $L(M)$.

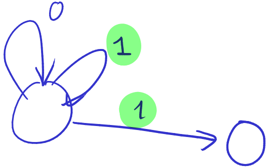
New example 1

input: 01101110101011

hint: XNNXNYXXXXXXXX

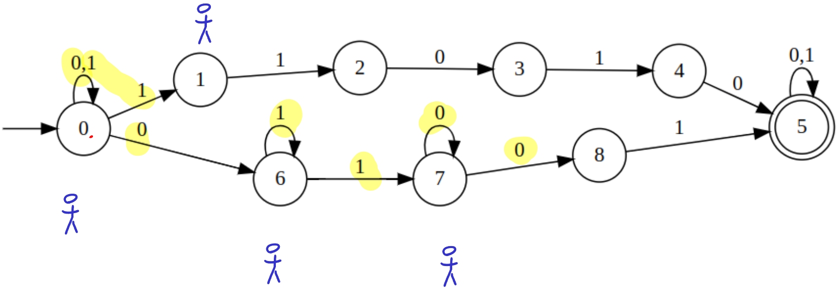
ยาว hint





New example 2

0 1 1 0 1 1

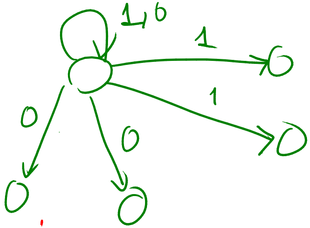


What's going on here?

- nondeterministic

จาก state q บน input a

N สามารถไปได้หลาย ๆ state.



ฉ: accept string w ถ้ามีทางที่

N ซึ่งกิน w และไปจบที่ accepting state.

More relaxed transitions

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

More relaxed transitions

$A = \{1,2,3\}$

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

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

New transition function δ : $Q \times \Sigma \longrightarrow \cancel{\emptyset}\ \boxed{2^Q}$

set של כל subset
של 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 $\delta : Q \times \Sigma \longrightarrow \boxed{2^Q}$. $= \phi$

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 Σ,

NFA (Formal definitions)

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

- the input alphabet Σ,
- a finite set of states Q,

NFA (Formal definitions)

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

- the input alphabet Σ,
- a finite set of states Q,
- a transition function δ

NFA (Formal definitions)

Q

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

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

NFA (Formal definitions)

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

- the input alphabet Σ,
- 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:

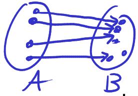
- the input alphabet Σ,
- a finite set of states Q,
- a transition function $\delta : Q \times \Sigma \longrightarrow 2^Q$
- a start state $s \in Q$, and
- a subset $A \subseteq Q$ of accepting states.

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

NFA (Formal definitions)

domain, range

$f: A \to B$



$Q = \{1,2,3\}$

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

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

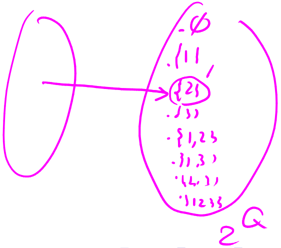
2^Q = powerset של Q

= set של subset

של Q

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

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



Transition

set

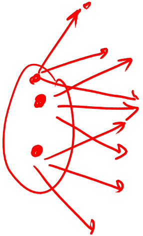
One step move: from state q with input symbol a, the machine changes its state to one of $\delta(q, a)$.

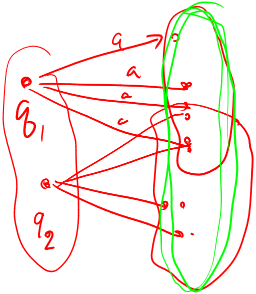
Transition

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 ที่เป็นไปได้.

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?





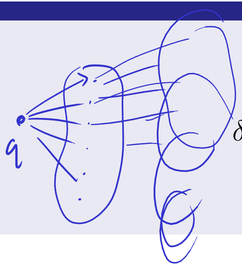
Transition

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



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

$w = aX$

$\delta^*(\delta(q, a), x)$

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

รวมทุกความเป็นไปได้.

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

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

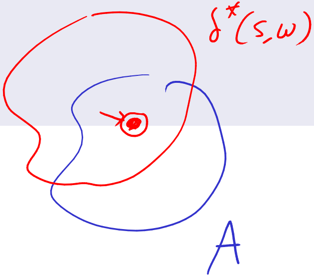
Acceptance

accepting w

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

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

S ทางไปได้



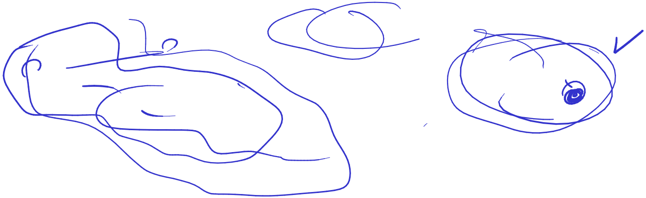
Interpretation

Interpretation

- Clairvoyance. มองเห็นอนาคต → ถ้ามีโอกาสที่จะไปถึง acccpting state ได้ NFA จะเลือกทางนั้น

Interpretation

- Clairvoyance.
- Parallel threads. → หาดูได้: แต่ละเวลา – เก็บ สถานะที่ "เป็นไปได้"



Interpretation

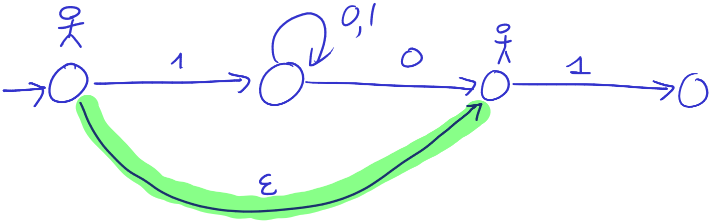
- Clairvoyance.
- Parallel threads.
- Proofs/oracles.

input : $\boxed{x}$

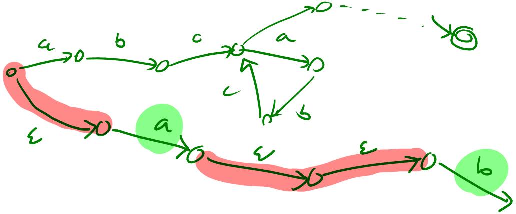
proofs/oracles/hints/ $\boxed{s}$

สมมติว่าไม่โดนบอก

ε-transition



ε-transition



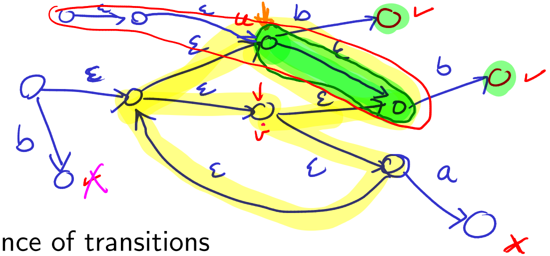
abcabb

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 $1 \leq i \leq k$.

ε-transition



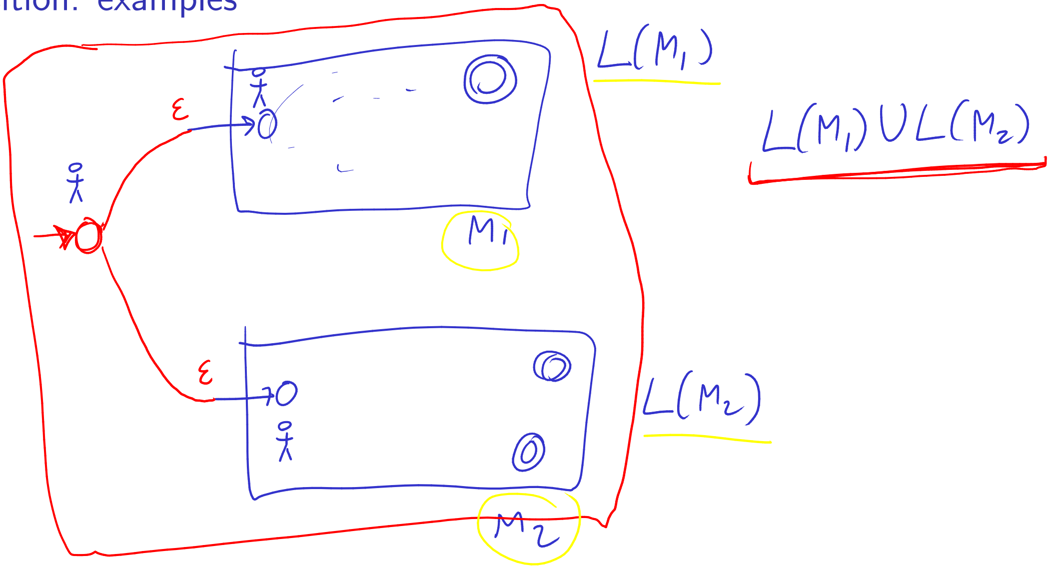
An NFA accepts string w iff there is a sequence of transitions

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

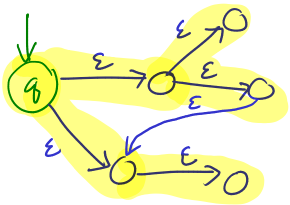
where $q_k \in A$ and $w = a_1 a_2 \cdots a_k$ where $a_i \in \Sigma \cup \{\varepsilon\}$ for $1 \leq i \leq k$.
The transition function also changes its domain to $Q \times (\Sigma \cup \{\varepsilon\})$.

ε-reach(u)

ε-transition: examples



ε-reach



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

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

Extended transition function: δ^*

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

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

$\delta(r, a)$

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

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

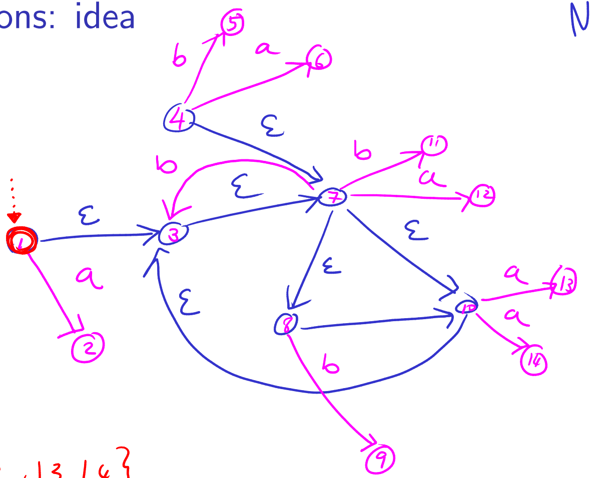
Extended transition function: δ^* (with shorter notation)

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

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

Removing ε-transitions: idea

NFA หัว



$\delta^*(1, a) = \{2, 12, 13, 14\}$

$\delta^*(1, b) = \{3, 11, 9\}$

Lemma 1

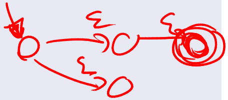
For any NFA $M = (\Sigma, Q, \delta, s, A)$ *with* ε*-transitions, there is an NFA* $M' = (\Sigma, Q', \delta', s', A')$ *without* ε*-transitions such that* $L(M) = L(M')$.

Proof.

นิยาม M' ดังนี้

$\delta : Q \times \Sigma \cup \{\varepsilon\} \to 2^Q$

$$Q' = Q$$



นิยาม $\delta' : Q \times \Sigma \to 2^Q$ ดังนี้

$$\delta'(q, a) = \bigcup_{r \in \varepsilon\text{-reach}(q)} \delta(r, a)$$

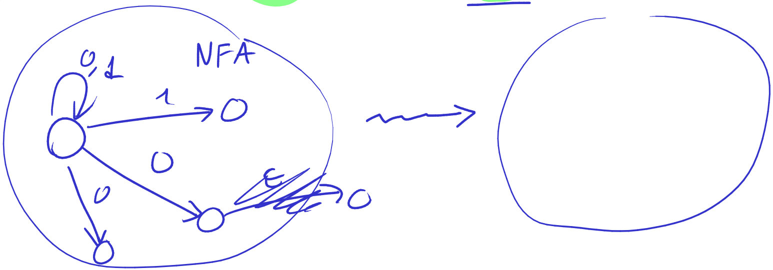
$$s' = s$$

$$A' = \{ q \in Q \mid \varepsilon\text{-reach}(q) \cap A \neq \emptyset \}$$

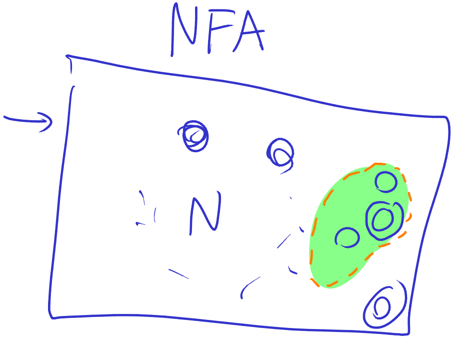
$\square$

Main question

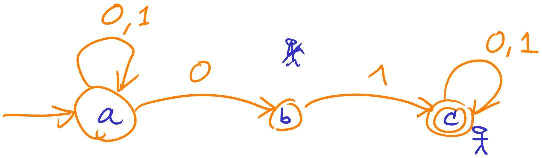
- We see that ε-transitions does not add any “power” to the machine.
- Does nondeterminism add any power to NFA (over typical DFA)?

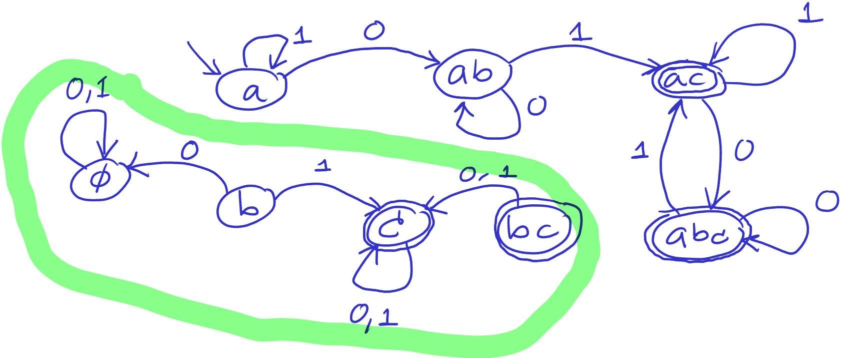


Simulating parallel machines



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

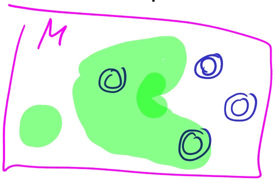
NFA to DFA: subset construction

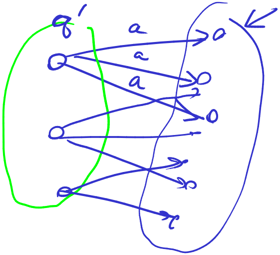
Given an NFA $M = (\Sigma, Q, \delta, s, A)$, we can construct an equivalent DFA $M' = (\Sigma, Q', \delta', s', \boxed{A'})$ as follows:

- Let $Q' = 2^Q$,
- $s' = \{s\}$,
- $\boxed{A'} = \{S \subseteq Q \mid \underline{S \cap A} \neq \emptyset\}$,

  Has accepting state in M too

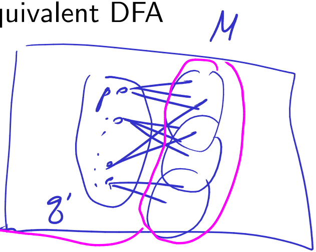
- and let $\delta' : \underbrace{Q'}_{} \times \Sigma \longrightarrow \underbrace{Q'}_{}$ be such that





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:

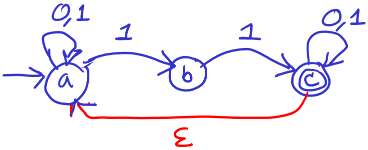


- Let $Q' = 2^Q$,
- $s' = \{s\}$,
- $A' = \{S \subseteq Q \mid S \cap A \neq \emptyset\}$,
- and let $\delta' : Q' \times \Sigma \longrightarrow Q'$ be such that

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

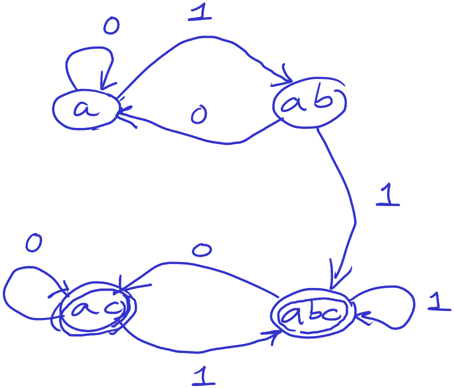
for all $q' \subseteq Q$ and $a \in \Sigma$.

Example



states	ε-reach	0	1
a	a	a	ab
b	b	$\emptyset$	c
c	ac	ac	abc

ε-reach ε-reach



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.

DFA $\longleftrightarrow$ 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)

DFA $\longleftrightarrow$ 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.

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)

$\underbrace{0^n 1^n}$

DFA ⟷ NFA

regex

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.

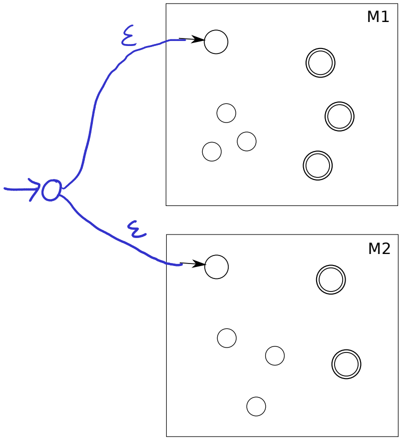
Steps:

- 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~~ DFA can be transformed into an equivalent regular expression. (only idea) *

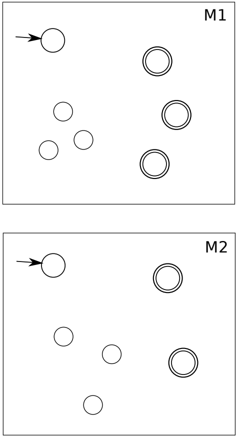
recursive def: $\emptyset$, $\boxed{s, \varepsilon}$ ε + symbol, concat, union, star

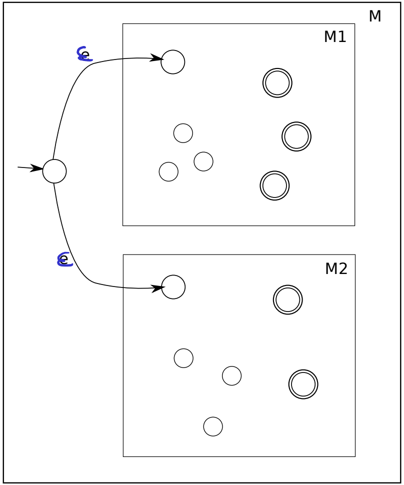
Warm-up: $\underbrace{\text{union of DFA}} \Longrightarrow \underbrace{\text{NFA}}$

หา N ที่ $L(N) = L(M_1) \cup L(M_2)$

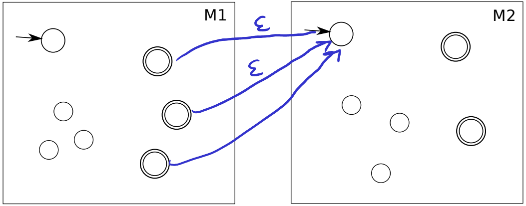


Warm-up: union of DFA $\Longrightarrow$ NFA



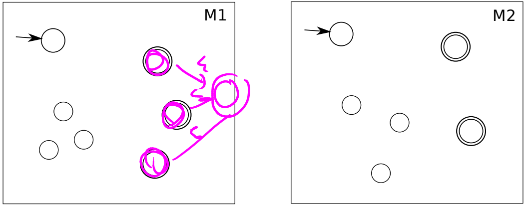


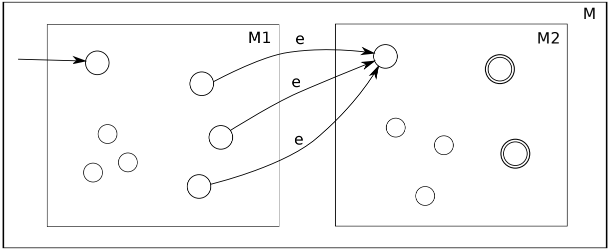
Concatenation: idea



$\varepsilon \cup \; L(M_1) \cdot L(M_2)$

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.

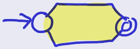
Lemma 3 (Thompson's algorithm)

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 alphaget Σ. We prove that there is an NFA N that accepts the language described by R by induction. +[s.t. N has 1 accepting st. diff. from s']

Induction hypothesis: for any subexpression S of R, there is an NFA that accepts the language described by S. * s.t. it has one accepting state different from starting state. *

We denote an NFA with this notation: 

Proof (Thompson's algorithm).

Consider any regular expression R over alphaget Σ. 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 alphaget Σ. 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).

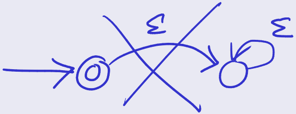
Consider any regular expression R over alphaget Σ. We prove that there is an NFA N that accepts the language described by R by induction.

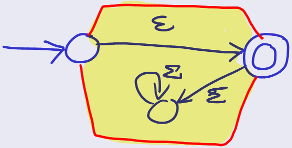
Induction hypothesis: for any subexpression S of R, there is an NFA that accepts the language described by S.

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 alphaget Σ. 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 alphaget Σ. We prove that there is an NFA N that accepts the language described by R by induction.

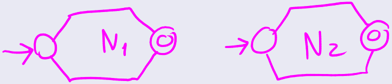
Induction hypothesis: for any subexpression S of R, there is an NFA that accepts the language described by S.

We denote an NFA with this notation:

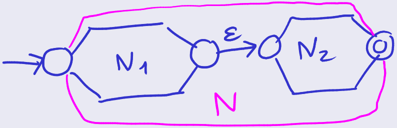
There are 6 cases:

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

จาก I.H. มี NFA N_1 ที่ accept $L(S)$
และ N_2 ที่ accept $L(T)$



จะสร้าง N ดังนี้



Proof (Thompson's algorithm).

Consider any regular expression R over alphaget Σ. We prove that there is an NFA N that accepts the language described by R by induction.

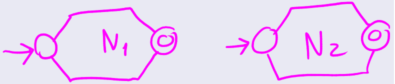
Induction hypothesis: for any subexpression S of R, there is an NFA that accepts the language described by S.

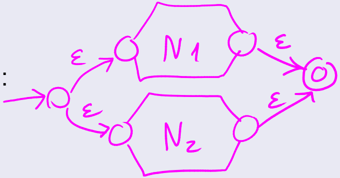
We denote an NFA with this notation:

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:

จาก I.H. มี NFA N_1 ที่ accept $L(S)$ และ N_2 ที่ accept $L(T)$





Proof (Thompson's algorithm).

Consider any regular expression R over alphaget Σ. We prove that there is an NFA N that accepts the language described by R by induction.

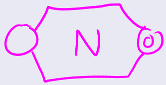
Induction hypothesis: for any subexpression S of R, there is an NFA that accepts the language described by S.

We denote an NFA with this notation:

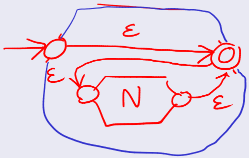
There are 6 cases:

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

จาก I.H ได้ NFA N
ที่ accepts L(S)
ที่มี 1 a



จะสร้าง N' ที่ accepts L(S*)



Proof (Thompson's algorithm).

Consider any regular expression R over alphaget Σ. 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.

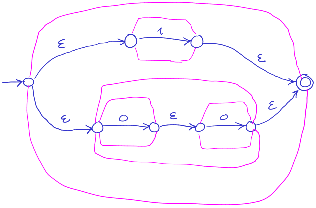
We denote an NFA with this notation:

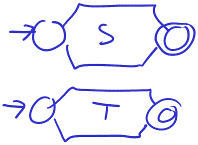
There are 6 cases:

- $R = \emptyset$:
- $R = \varepsilon$:
- $R = a$ 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. $\square$

Example: 1 + 00



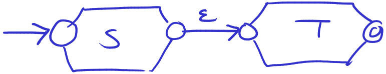


∅:

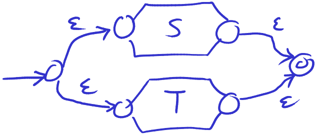
ε:

a:

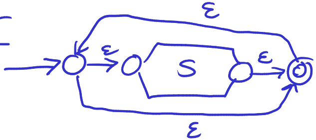
Concat



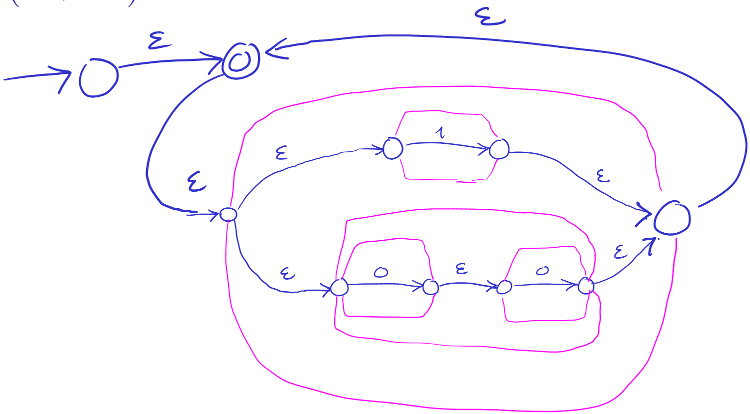
Union



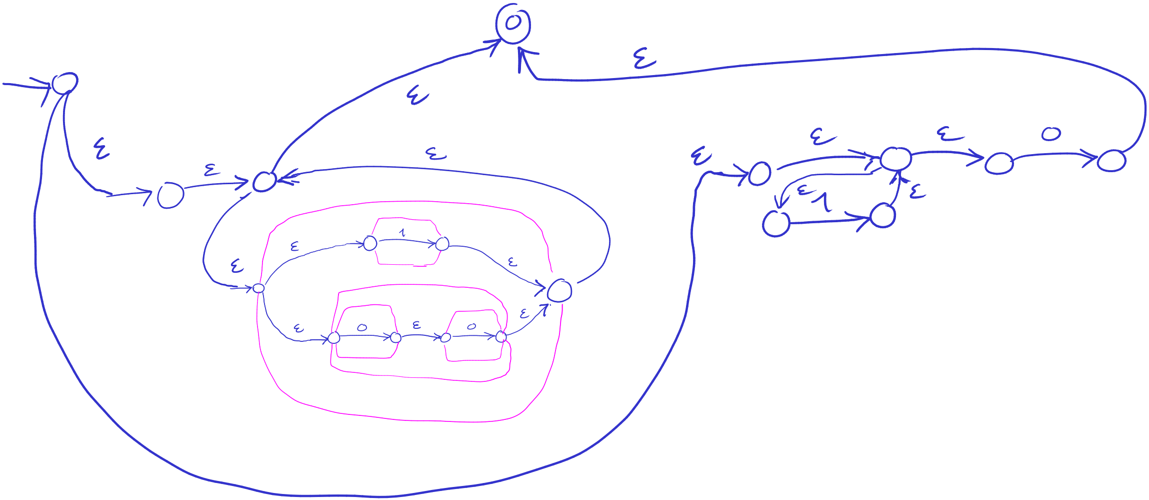
Star



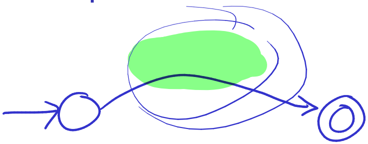
Example: $(1 + 00)^*$



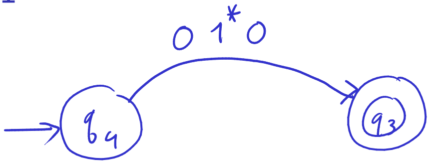
Example: $(1 + 00)^* + 1^*0$

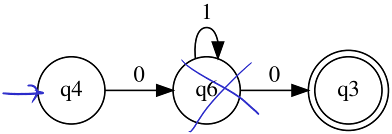


NFA to Regular expressions

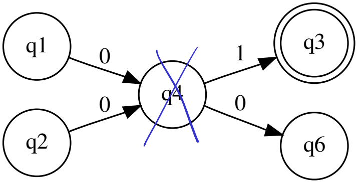


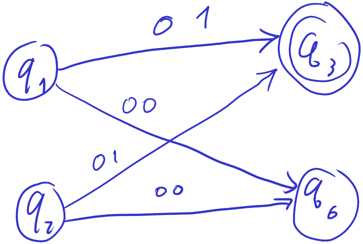
State elimination: example 1





State elimination: example 2





State elimination: example 3