# Single Final State for NFAs

Any NFA can be converted

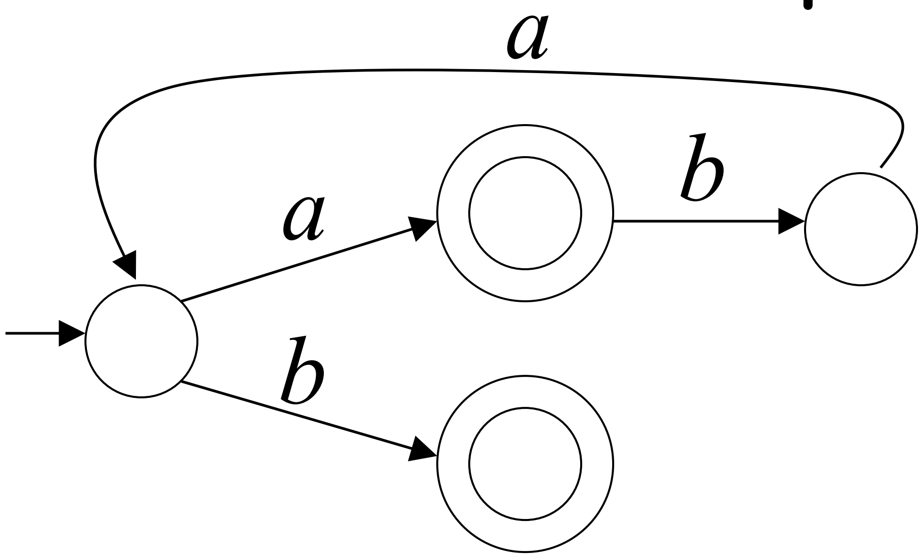
to an equivalent NFA

with a single final state

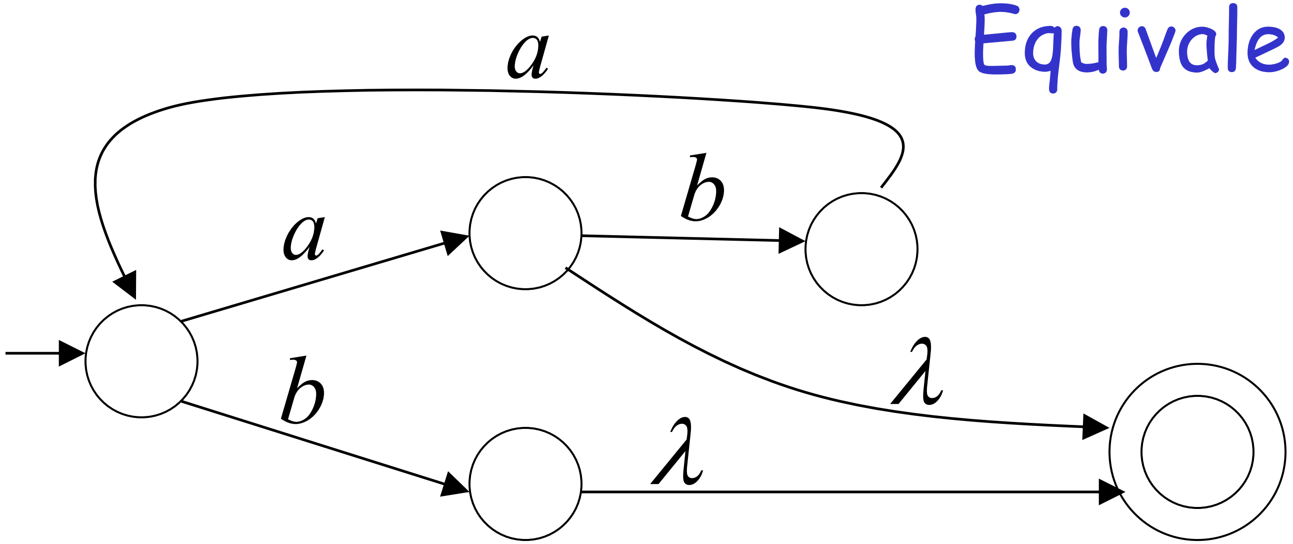
NFA ใดๆ สามารถแทนเป็น NFA ที่มี 1 final state เท่านั้นได้

# Example

NFA

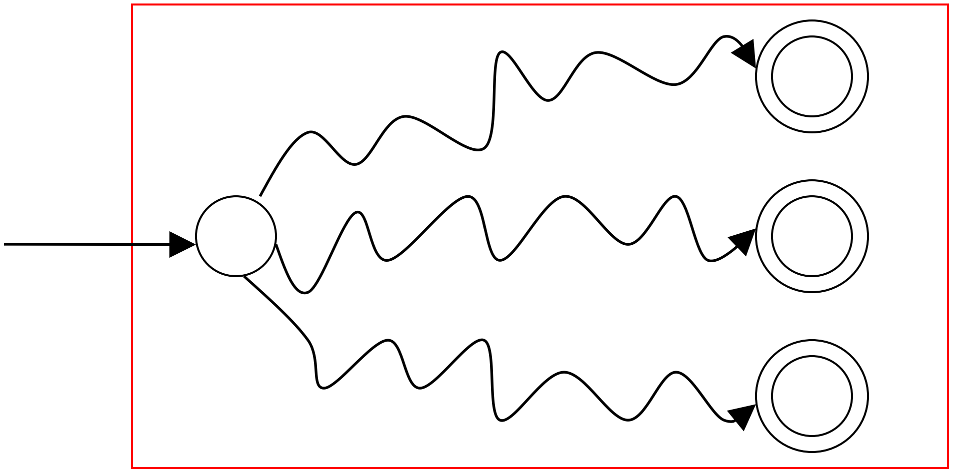



Equivalent NFA

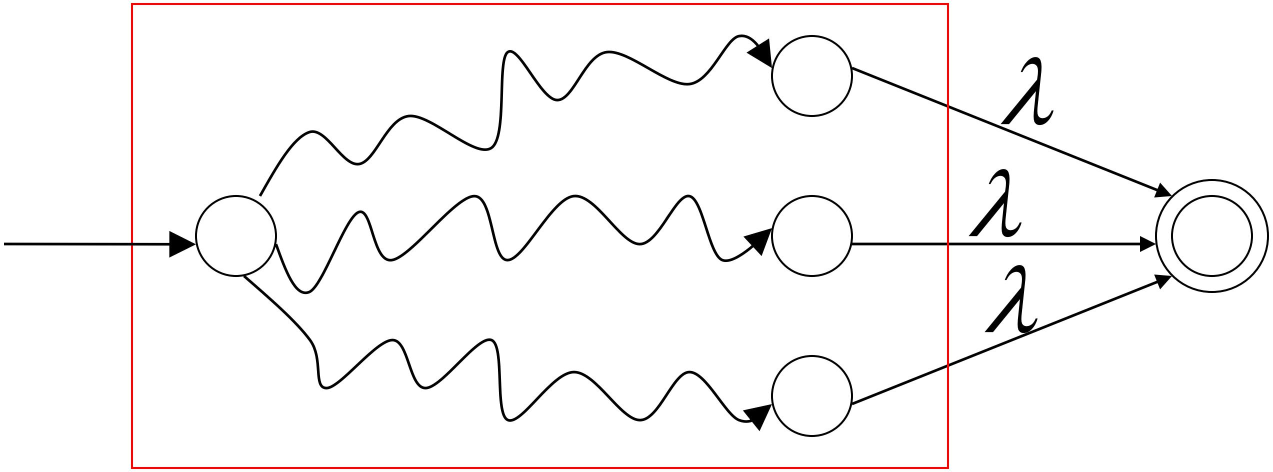


# In General

NFA



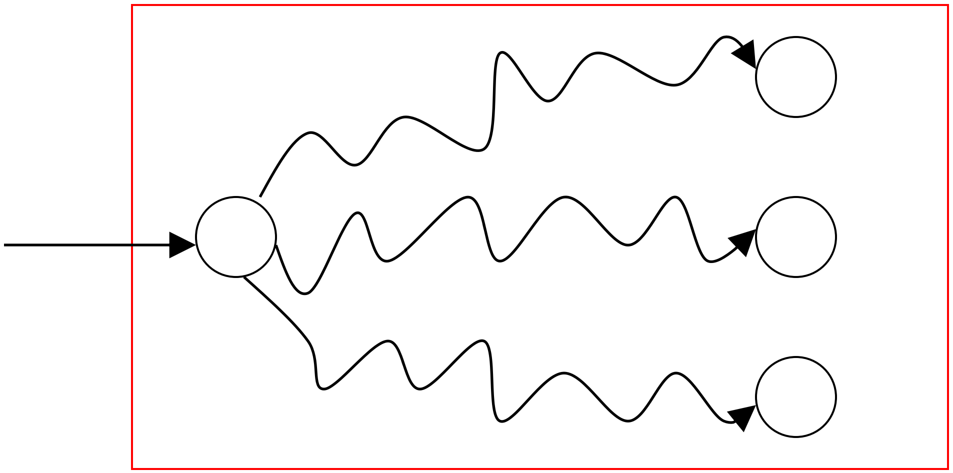
Equivalent NFA




Single final state

# Extreme Case

NFA without final state



Add a final state

Without transitions

# Properties of Regular Languages

คุณสมบัติปิด $I + I = I$

$I - I = I$

For regular languages $L_1$ and $L_2$ we will prove that:

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1 *$

Reversal: $L_1^R$

Complement: $\overline{L_1}$

Intersection: $L_1 \cap L_2$

Are regular Languages

We say: Regular languages are **closed under**

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1{}^*$

Reversal: $L_1{}^R$

Complement: $\overline{L_1}$

Intersection: $L_1 \cap L_2$

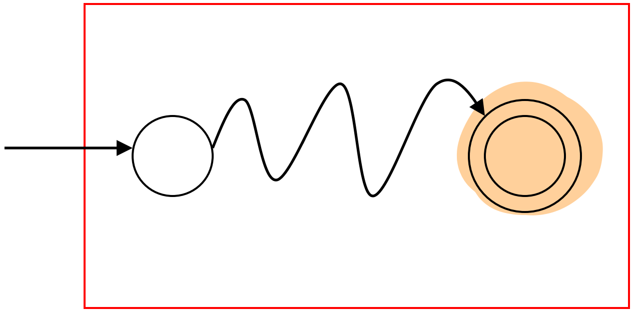
Regular language $L_1$

Regular language $L_2$

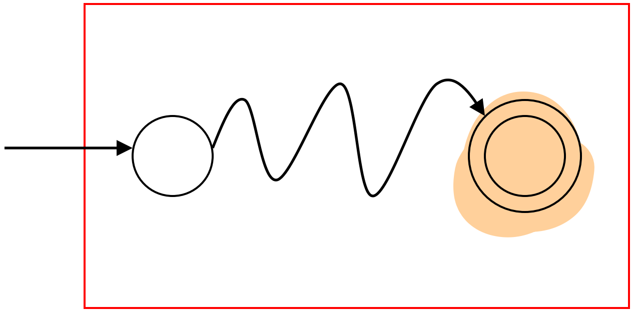
$$L(M_1) = L_1$$

$$L(M_2) = L_2$$

NFA $M_1$



NFA $M_2$



Single final state

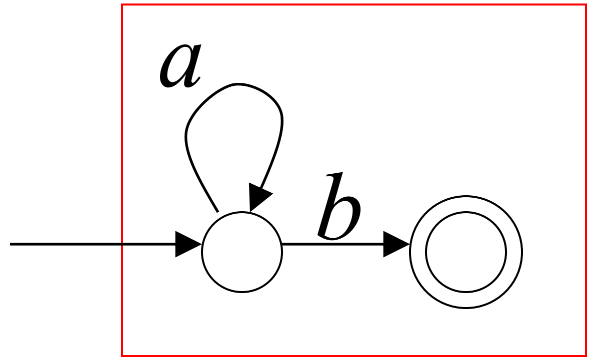
Single final state

# Example

$M_1$

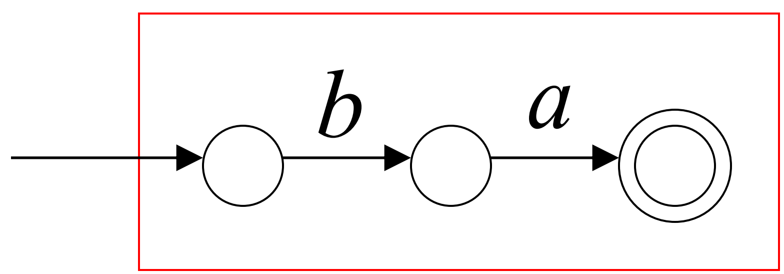
$n \geq 0$

$L_1 = \{a^n b\}$



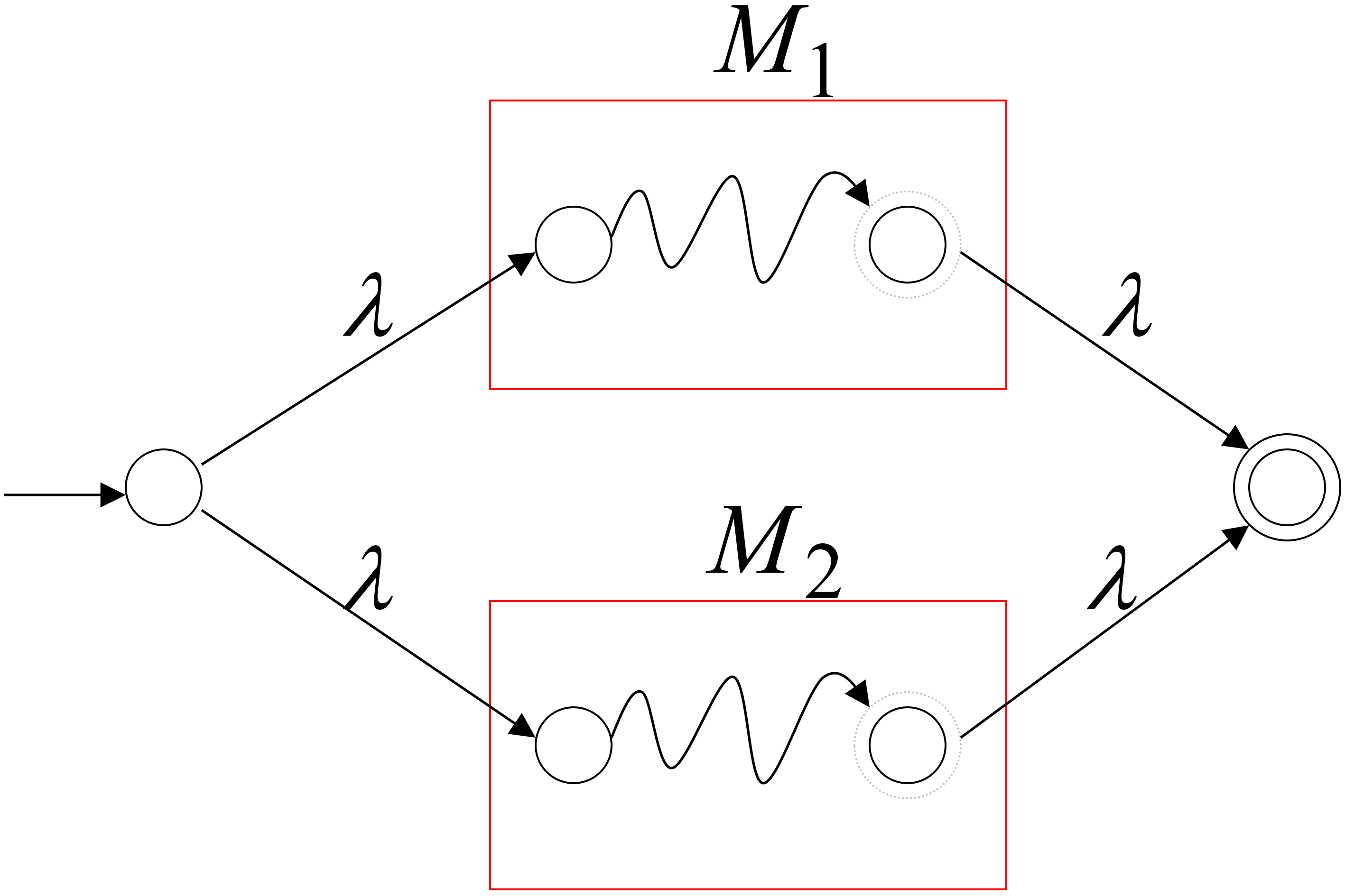

$M_2$

$L_2 = \{ba\}$



# Union

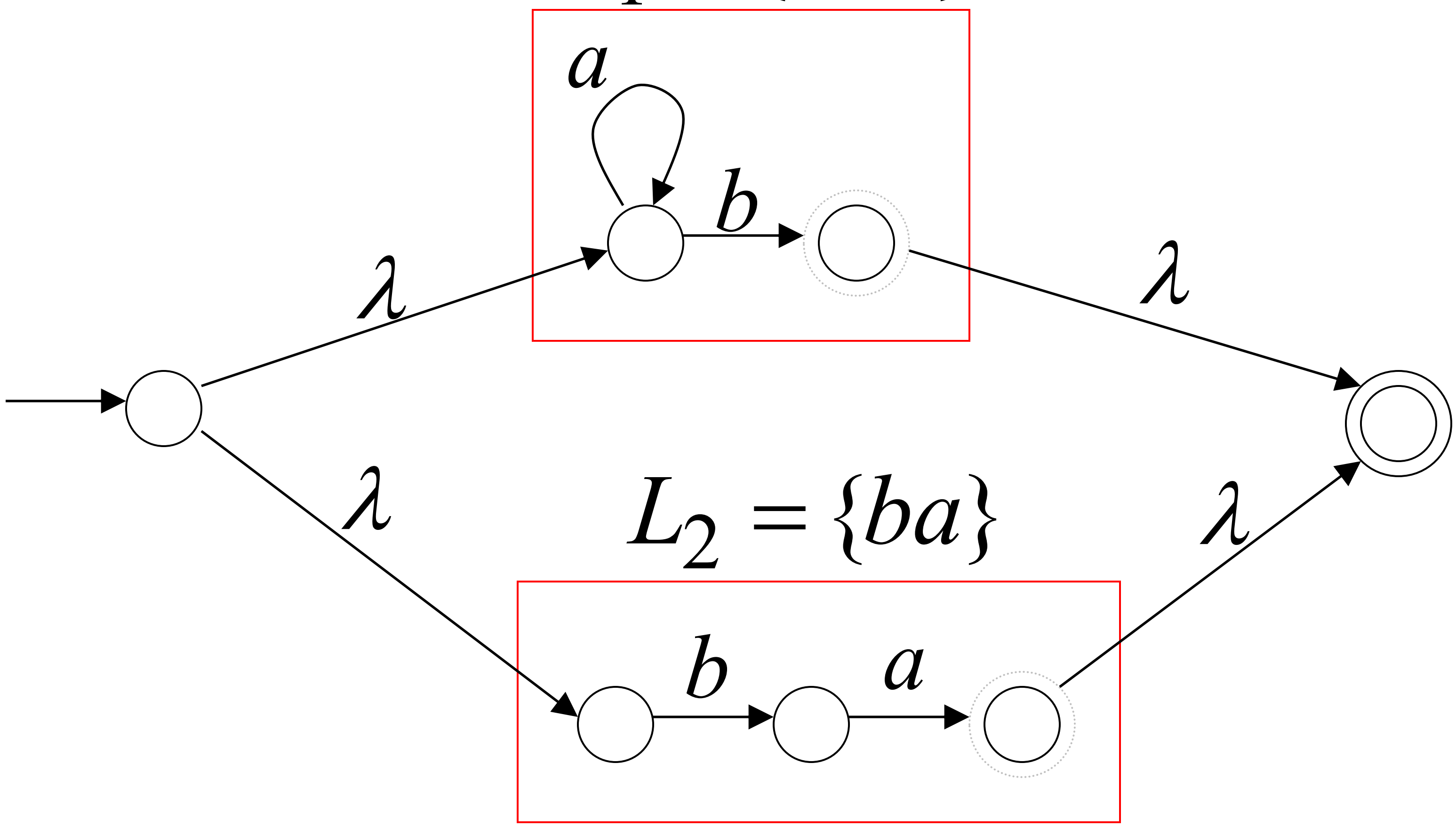
NFA for $L_1 \cup L_2$



# Example

NFA for $L_1 \cup L_2 = \{a^n b\} \cup \{ba\}$

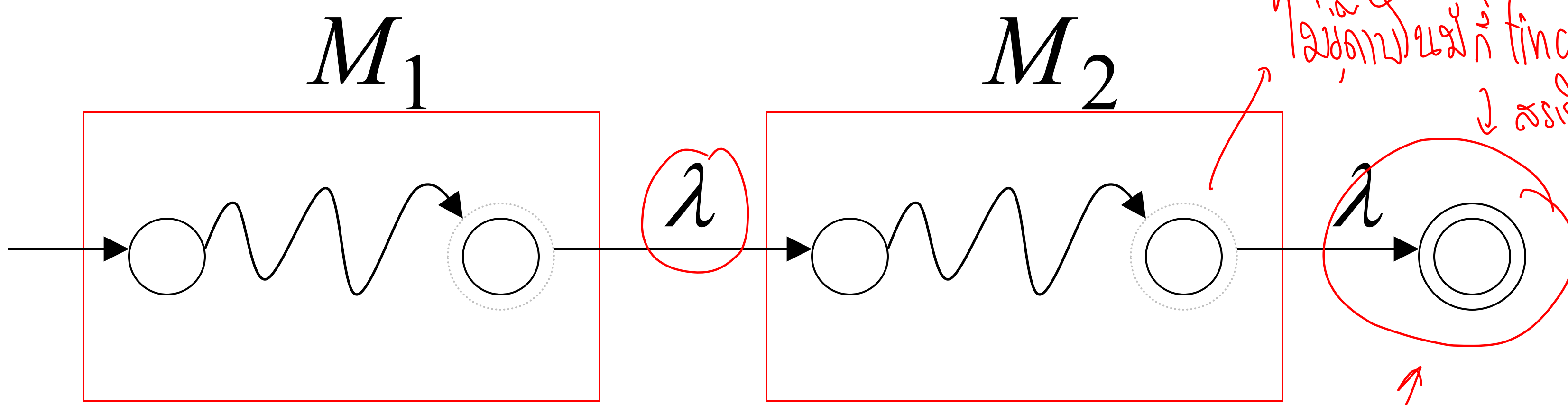
$L_1 = \{a^n b\}$




$L_2 = \{ba\}$

# Concatenation

NFA for $L_1L_2$




ไม่ได้เป็นแล้วที่ final

สร้างใหม่ช่วงสุด

จุดเริ่มและเชื่อมการสร้าง NFA

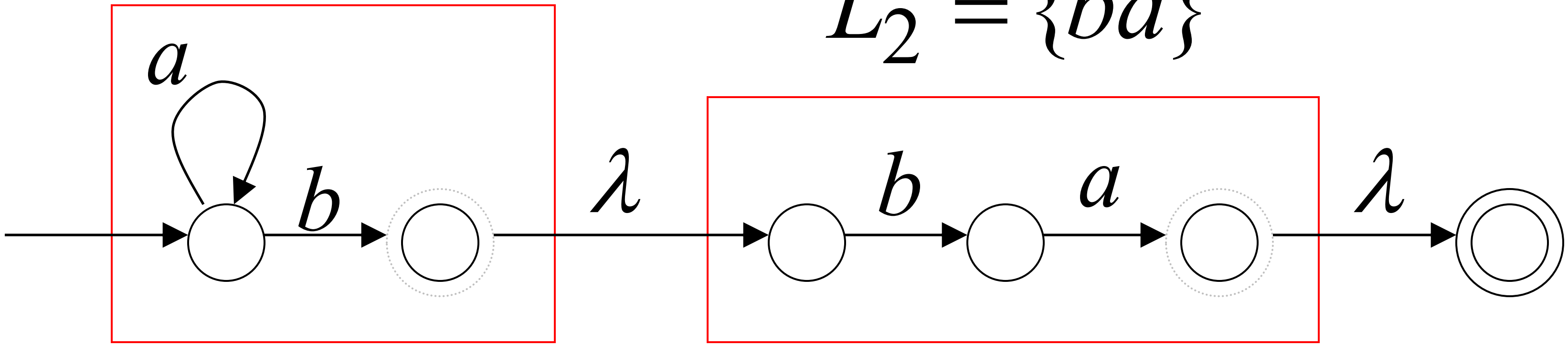
ที่มี final state อันเดียว

# Example

NFA for $L_1 L_2 = \{a^n b\}\{ba\} = \{a^n bba\}$

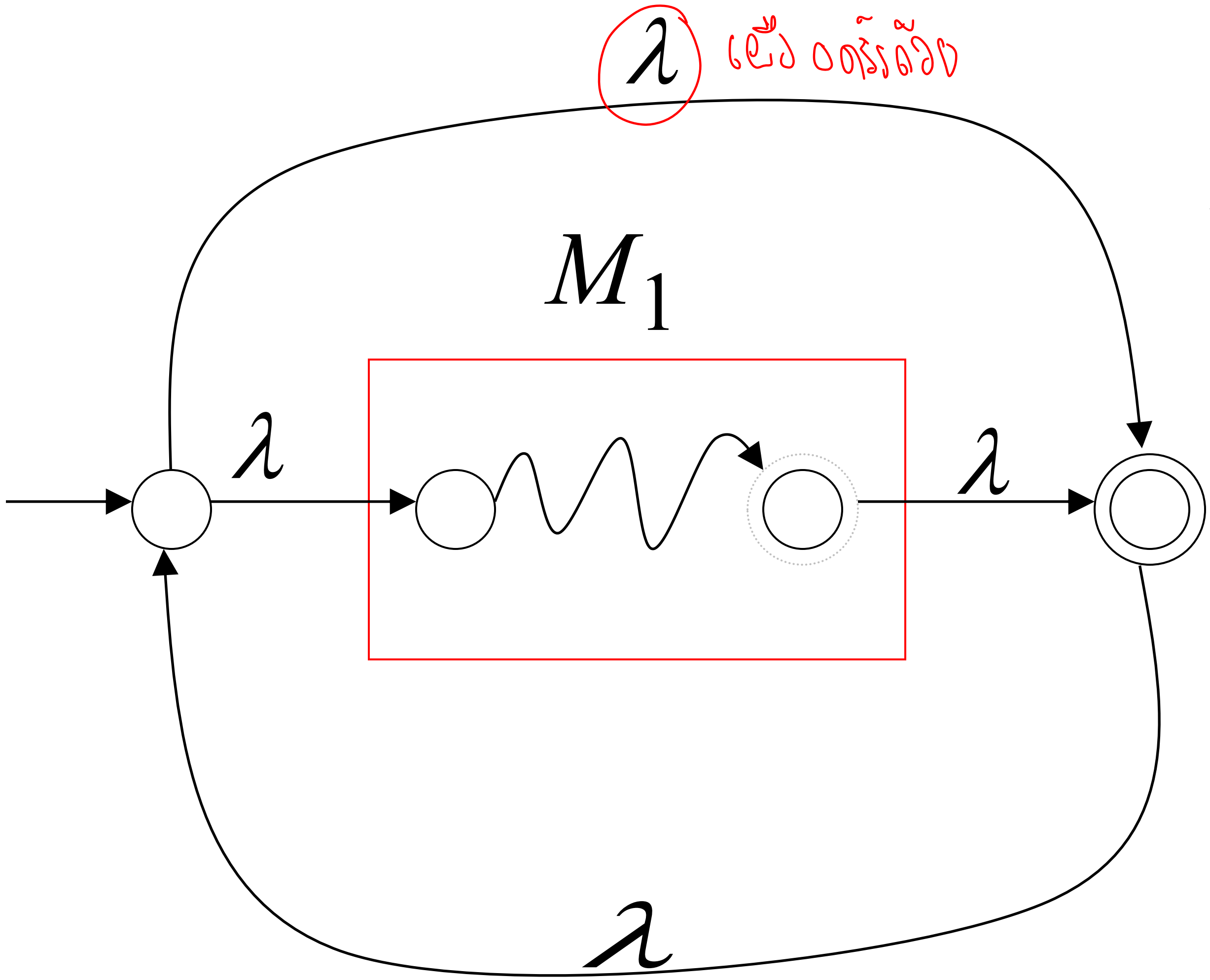
$L_1 = \{a^n b\}$

$L_2 = \{ba\}$



# Star Operation

NFA for $L_1{}^*$ การทำซ้ำ $n=0, \dots, \infty$




เผื่อ 0 ครั้งด้วย

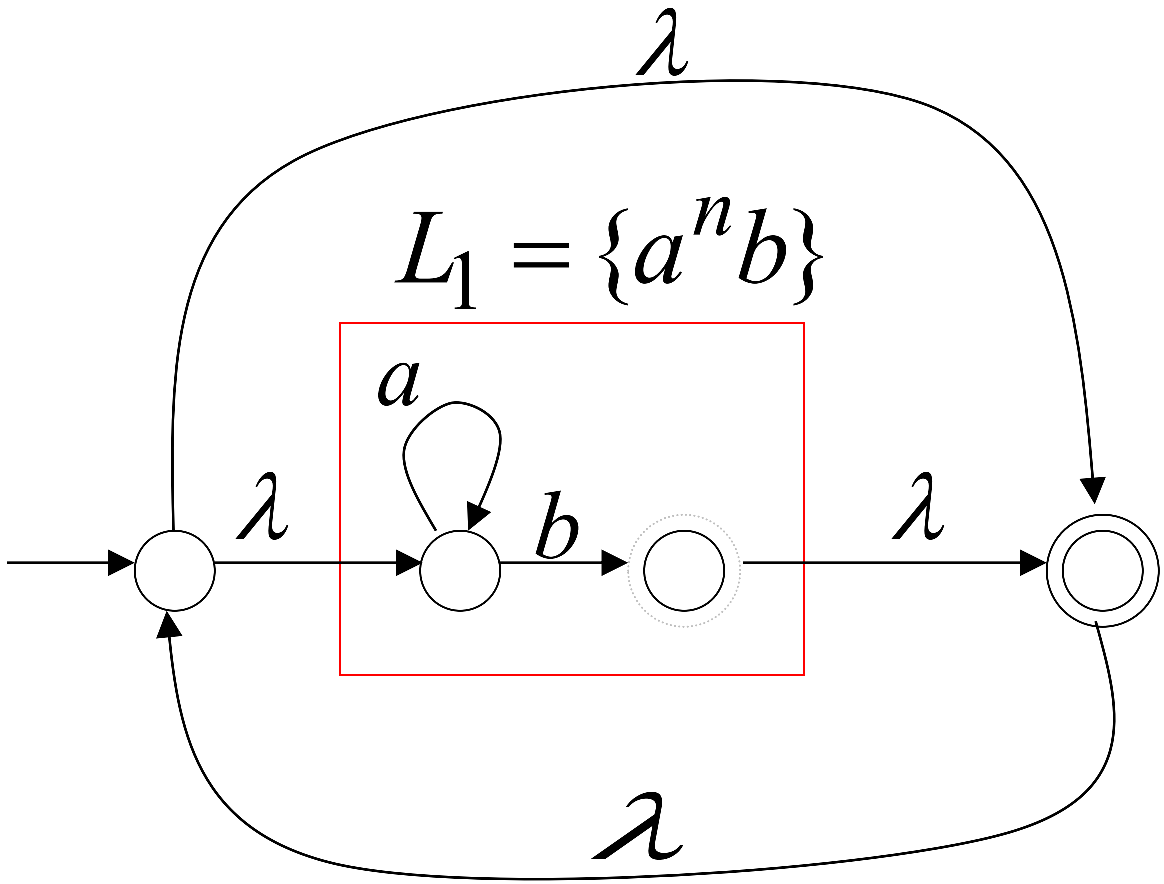
$\lambda \in L_1{}^*$

# Example

NFA for $L_1{}^* = \{a^n b\}^*$

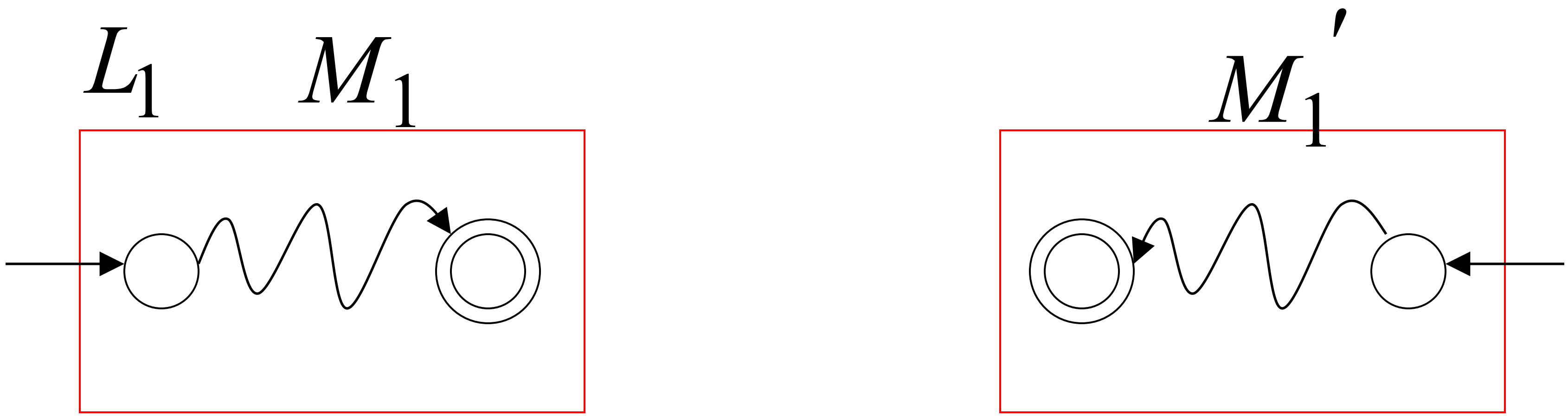
$w = w_1 w_2 \cdots w_k$

$w_i \in L_1$

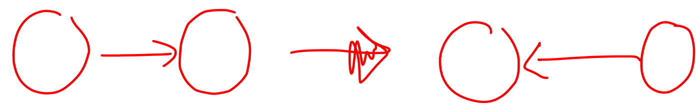


# Reverse

NFA for $L_1^R$







ขั้นตอน

1. Reverse all transitions

ถ้ามีหลาย final? → ต้องทำให้เหลือ 1 final ก่อนดี

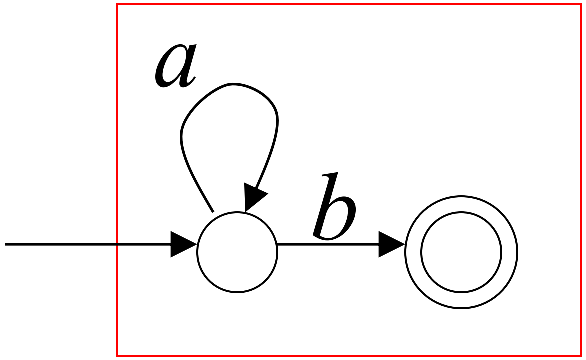
2. Make initial state final state and vice versa

สลับ init กับ final

# Example

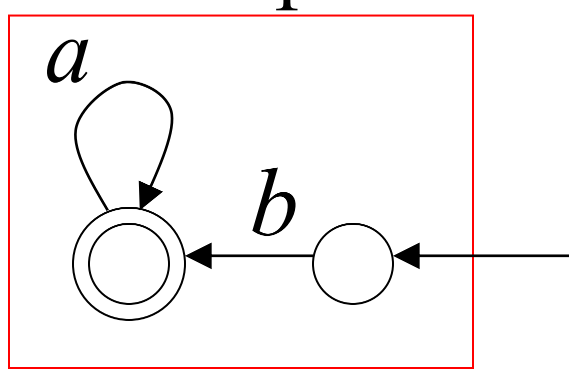
$$L_1 = \{a^n b\}$$

$M_1$

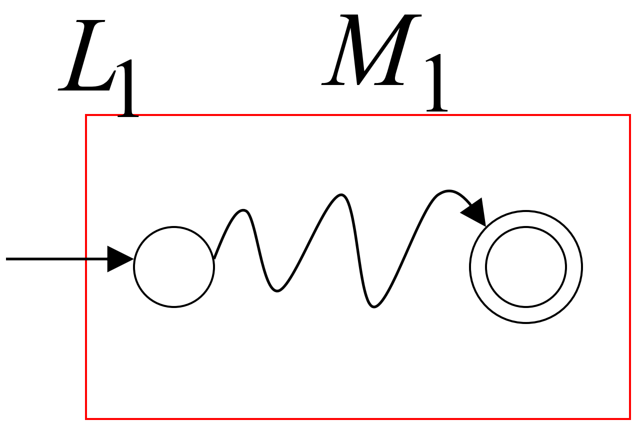


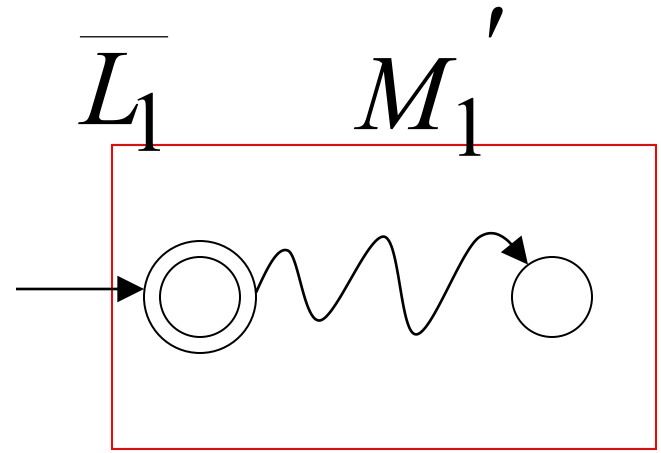

$$L_1^R = \{ba^n\}$$

$M_1'$



# Complement




แปลงเป็น DFA

1. Take the **DFA** that accepts $L_1$

สาเหตุที่ต้องทำเฉพาะ NFA จะทำการละทิ้ง state ไป

2. Make final states non-final, and vice-versa

สลับ state ทั้งคู่

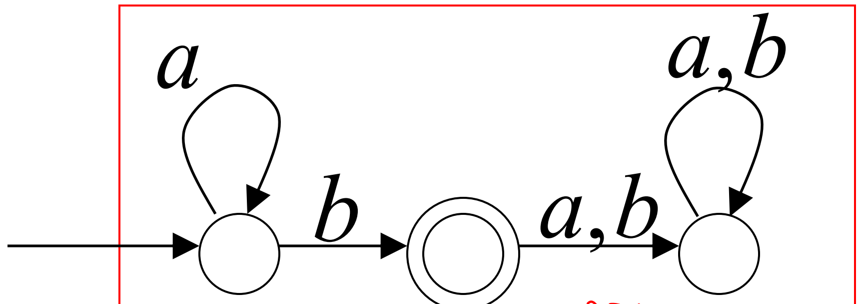
complement คือทุกตัว ยกเว้นตัวมันเอง

หาทุกกรณี → สลับ final

# Example

$M_1$

$$L_1 = \{a^n b\}$$

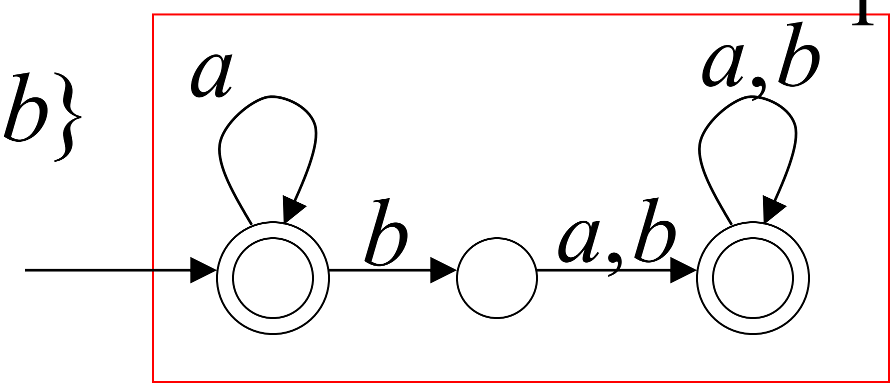



(1) หา DFA

(2) สลับที่ดิน

$M_1'$

$$\overline{L_1} = \{a,b\}^* - \{a^n b\}$$



# Intersection

DeMorgan's Law: $$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

Reg $\cup$ Reg = Reg  $\overline{\text{Reg}}$ = Reg

$L_1$ , $L_2$ regular

$\Rightarrow$ $\overline{L_1}$ , $\overline{L_2}$ regular

$\Rightarrow$ $\overline{L_1} \cup \overline{L_2}$ regular

$\Rightarrow$ $\overline{\overline{L_1} \cup \overline{L_2}}$ regular

$\Rightarrow$ $L_1 \cap L_2$ regular

# Example

$L_1 = \{a^n b\}$ $\cap$ regular

$L_2 = \{ab, ba\}$ regular

$\Rightarrow$ $L_1 \cap L_2 = \{ab\}$

regular

# Regular Expressions

# Regular Expressions

Regular expressions → ใช้ในการอธิบายภาษา regular

describe regular languages

Example: $(a+b\cdot c)^*$

แปลงเป็นรูปของ set

describes the language

$$\{a,bc\}^* = \{\lambda, a, bc, aa, abc, bca, \ldots\}$$

set notation

ซ้ำไปเรื่อย

Why do we need Regular Expressions ? <<Click>>

# Recursive Definition

Primitive regular expressions: $\emptyset, \quad \lambda, \quad \alpha$

↳ base case = ไม่สามารถแบ่งแยกย่อยๆได้อีก

alphabet 1ตัว

แยกไร?

Given regular expressions $r_1$ and $r_2$

กำหนด

union

concat

rep

( )

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \cdot r_2 \\ r_1{}^{*} \\ (r_1) \end{array}\right\} \text{Are regular expressions}$$

# Examples

A regular expression: $(a+b\cdot c)^{*}\cdot(c+\varnothing)$

? wtf

Not a regular expression: $(a+b+)$

# Languages of Regular Expressions

$L(r)$ : language of regular expression $r$

Example

ภาษาของregular expression

$$L((a+b\cdot c)^*)=\{\lambda,a,bc,aa,abc,bca,...\}$$

$\{a, bc\}^*$

# Definition

For primitive regular expressions:

$$L(\varnothing) = \varnothing$$

$$L(\lambda) = \{\lambda\}$$

ได้ string ที่ไม่มีอยู่จริง เหมือนเดิม

คล้ายกัน

$$L(a) = \{a\}$$

# Definition (continued)

เหมือนเอา $r_1$ มา operation กับ $r_2$

For regular expressions $r_1$ and $r_2$

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$ เหมือนการแตก L เข้าไป

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$

$$L(r_1 *) = (L(r_1))^*$$

$$L((r_1)) = L(r_1)$$

# Example

Regular expression: $(a+b)\cdot a^*$

อธิบายภาษาของมัน

$$
\begin{aligned}
L((a+b)\cdot a^*) &= L((a+b))\,L(a^*) \\
&= L(a+b)\,L(a^*) \\
&= (L(a)\cup L(b))\,(L(a))^* \\
&= (\{a\}\cup\{b\})\,(\{a\})^* \\
&= \{a,b\}\,\{\lambda,a,aa,aaa,\ldots\} \\
&= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}
\end{aligned}
$$

# Example

Regular expression $r = (a+b)^*(a+bb)$

รูปการแจกแจง

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}$$

# Example

Regular expression $r = (aa)^*(bb)^*b$

รูปทั่วไป

$$L(r) = \{a^{2n}b^{2m}b : \quad n, m \geq 0\}$$

# Example

ขอให้มีเลข 0 อยู่ติดกัน 1 คู่

Regular expression $r = (0+1)^*00(0+1)^*$

เป็นอะไรก็ได้ any string ด้วย a , b

$L(r)$ = { all strings with at least two consecutive 0 }

ติดกัน

110011 1100011

# Example

Regular expression $r = (1 + 01)^*(0 + \lambda)$

ถ้าอยากรู้ว่าเป็นคือ L อาจใช้ brute force เขียนออกมา

$\{\lambda, 1, 0,1, 11, 101, 0101, \ldots\} \cdot \{0, \lambda\}$

เรียงลำดับดู $\{\lambda, 0, 1, 10, 01, 010$

$L(r)$ = { all strings without two consecutive 0 }

ไม่มีทางจะได้เลข 0 ติดกัน

# Equivalent Regular Expressions

Definition:

ถ้ามันจะเท่ากัน L ต้องเท่ากัน → เหมือนใน LFA & DFA

Regular expressions $r_1$ and $r_2$ ที่จะเท่ากันก็ต่อเมื่อภาษามันเท่ากัน

are **equivalent** if $L(r_1) = L(r_2)$

# Example

$L$ = { all strings without two consecutive 0 }

$$r_1 = (1 + 01)^*(0 + \lambda)$$ ที่ด้านบน

ยากกว่า

$$r_2 = (1^* 0 1 1^*)^*(0 + \lambda) + 1^*(0 + \lambda)$$

$r_1$ เหมือน $r_2$ → $r_1$ version simplify

$$L(r_1) = L(r_2) = L \Rightarrow$$

$r_1$ and $r_2$ are equivalent regular expr.

# Regular Expressions
# and ความสัมพันธ์
# Regular Languages

# Theorem

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} = \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

# Theorem - Part 1

พิสูจน์ว่า $A \subseteq B$

$$\left\{\begin{array}{l}\text{Languages} \\ \text{Generated by} \\ \text{Regular Expressions}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular} \\ \text{Languages}\end{array}\right\}$$

1. For any regular expression $r$ the language $L(r)$ is regular

# Theorem - Part 2

$$\left\{\begin{array}{l}\text{Languages} \\ \text{Generated by} \\ \text{Regular Expressions}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular} \\ \text{Languages}\end{array}\right\}$$

2. For any regular language $L$ there is a regular expression $r$ with $L(r) = L$

# Proof - Part 1

1. For any regular expression $r$
   the language $L(r)$ is regular

Proof by induction on the size of $r$

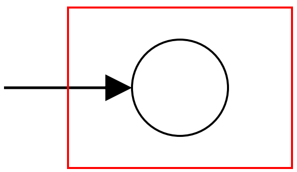
# Induction Basis

Primitive Regular Expressions: $\varnothing, \quad \lambda, \quad \alpha$

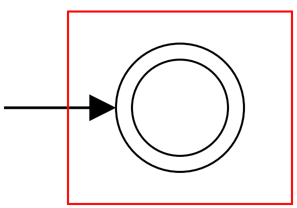
regular expression ที่สั้นที่สุด "primitive"

NFAs

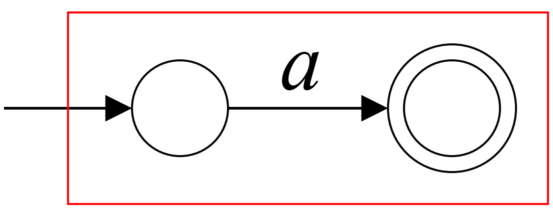
สามารถสร้าง NFA ได้ → regular language



$$L(M_1) = \varnothing = L(\varnothing)$$



$$L(M_2) = \{\lambda\} = L(\lambda)$$




$$L(M_3) = \{a\} = L(a)$$

regular languages

# Inductive Hypothesis

Assume     เดิมสมมุติแล้ว

for regular expressions $r_1$ and $r_2$

that

$L(r_1)$ and $L(r_2)$ are regular languages

# Inductive Step

We will prove:

เอามาทำoperationกัน

$$\left.\begin{array}{l} L(r_1 + r_2) \\ \\ L(r_1 \cdot r_2) \\ \\ L(r_1 *) \\ \\ L((r_1)) \end{array}\right\} \text{Are regular Languages}$$

ทั้งหมดของมันเป็น yes!

# By definition of regular expressions:

$$L(r_1 + r_2) = \overbrace{L(r_1)}^{\text{reg}} \cup \overbrace{L(r_2)}^{\text{reg}}$$ ตามสมมุติฐาน

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$ เป็น reg ทั้งหมด

$$L(r_1 *) = (L(r_1))*$$

$$L((r_1)) = L(r_1)$$

By inductive hypothesis we know:

$L(r_1)$ and $L(r_2)$ are regular languages

We also know: สรุป

Regular languages are closed under:

| | |
|---|---|
| *Union* | $L(r_1) \cup L(r_2)$ |
| *Concatenation* | $L(r_1)\,L(r_2)$ |
| *Star* | $(L(r_1))^*$ |

Therefore:

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)\,L(r_2)$$

$$L(r_1 *) = (L(r_1))*$$

Are regular languages

And trivially:

$L((r_1))$ is a regular language

# Proof - Part 2

2. For any regular language $L$ there is a regular expression $r$ with $L(r) = L$

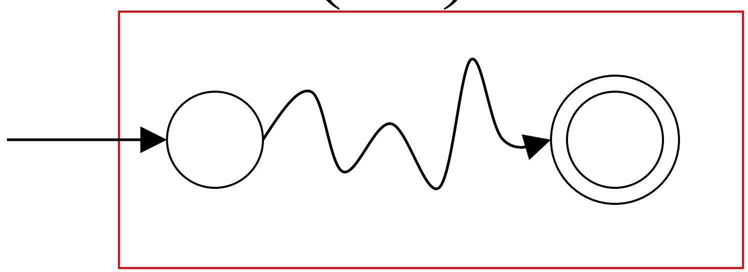
การทำให้ดูเป็นตัวอย่าง → สร้าง algo ขึ้นมาเพื่อบอกว่ามันทำได้

Proof by construction of regular expression

ต้องสร้าง NFA ได้ 1 เป็น single final state

Since $L$ is regular take the NFA $M$ that accepts it

$$L(M) = L$$



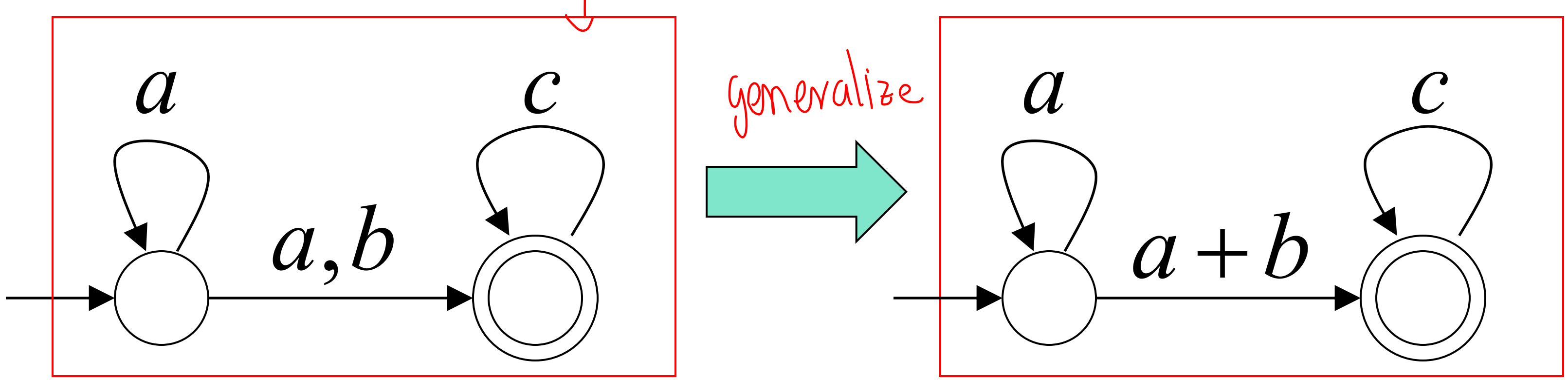
Single final state

From $M$ construct the equivalent **Generalized Transition Graph** in which transition labels are regular expressions

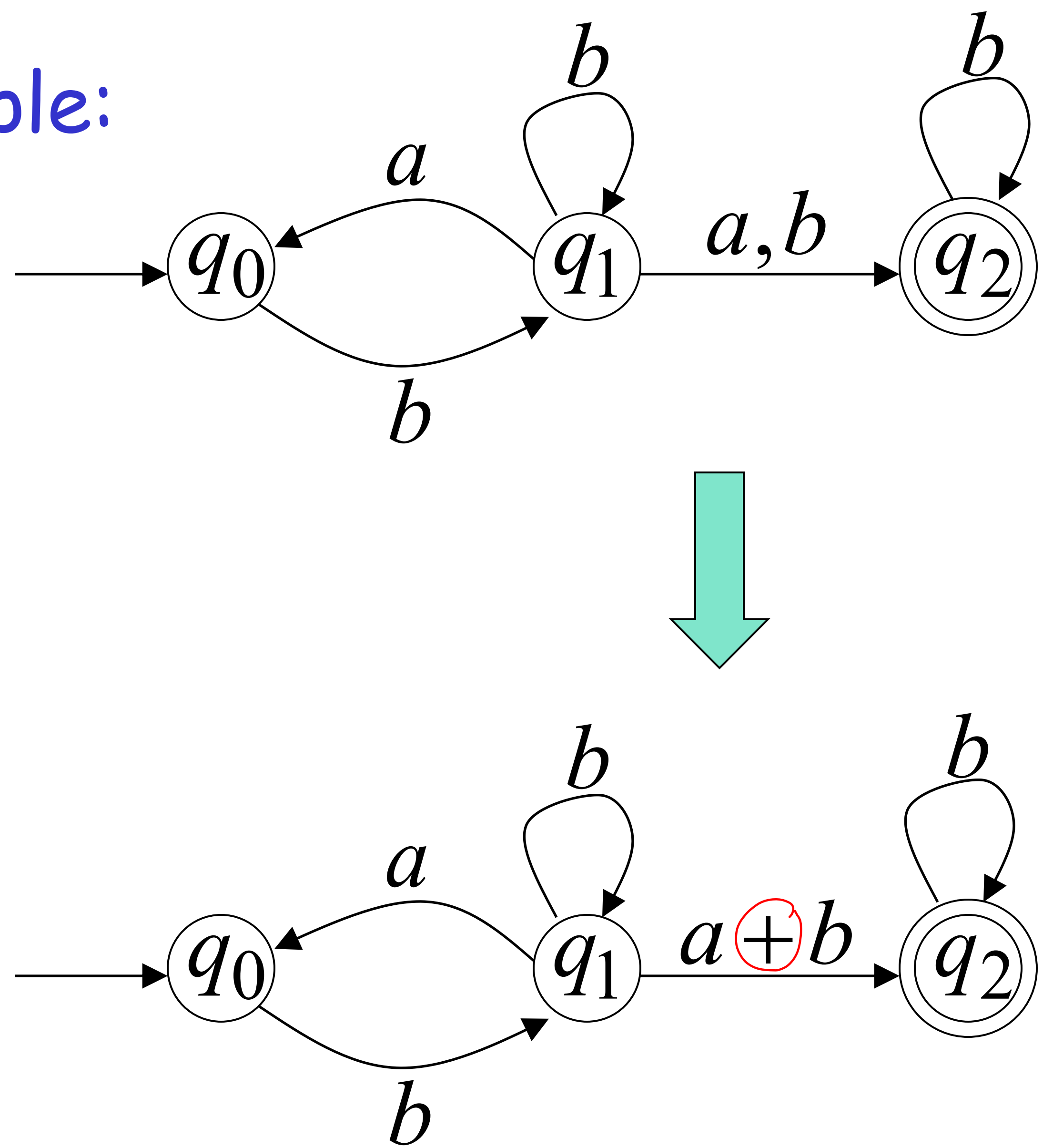
Example:

ยัง interface จาก NFA

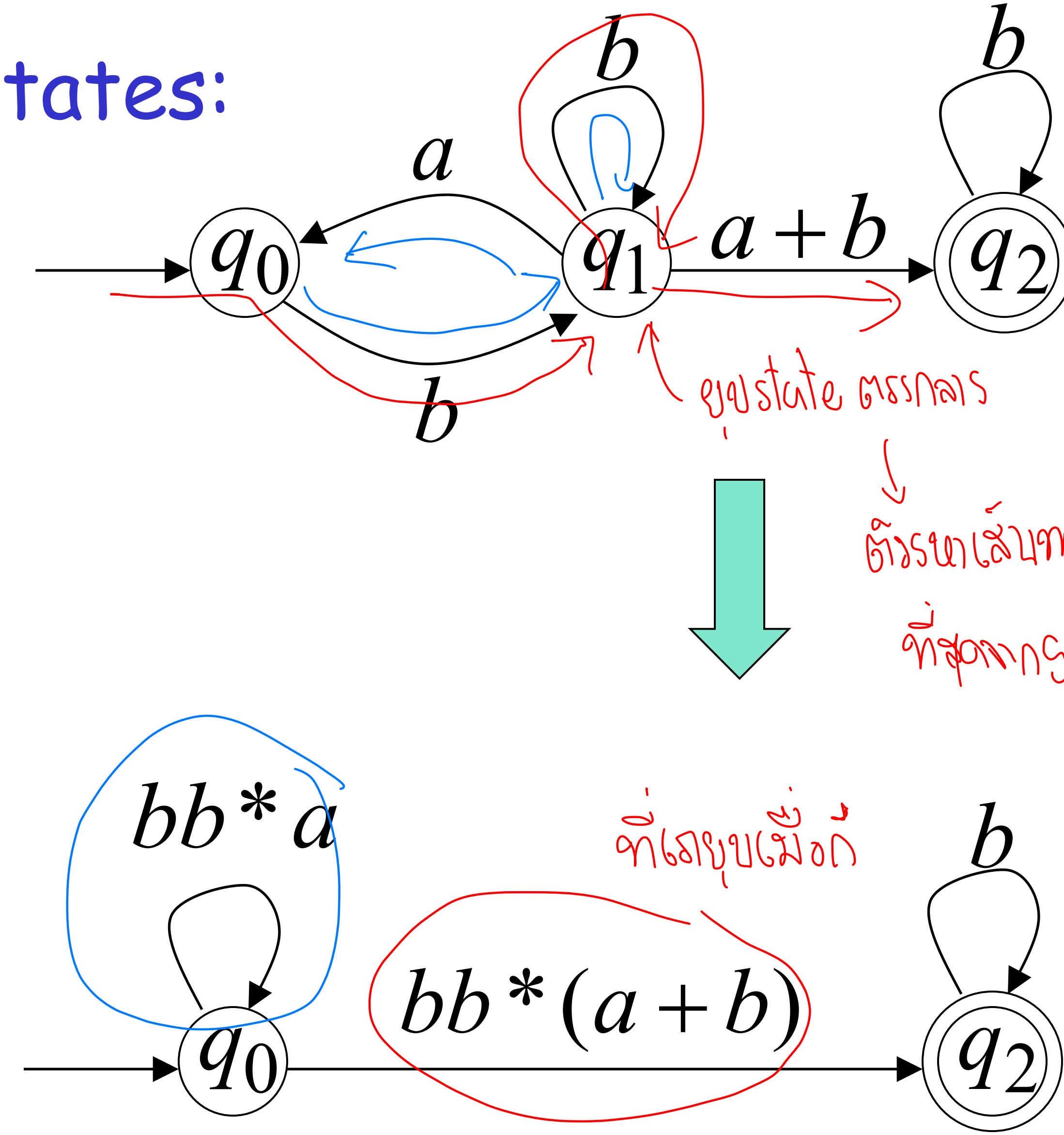
$M$



# Another Example:



# Reducing the states:



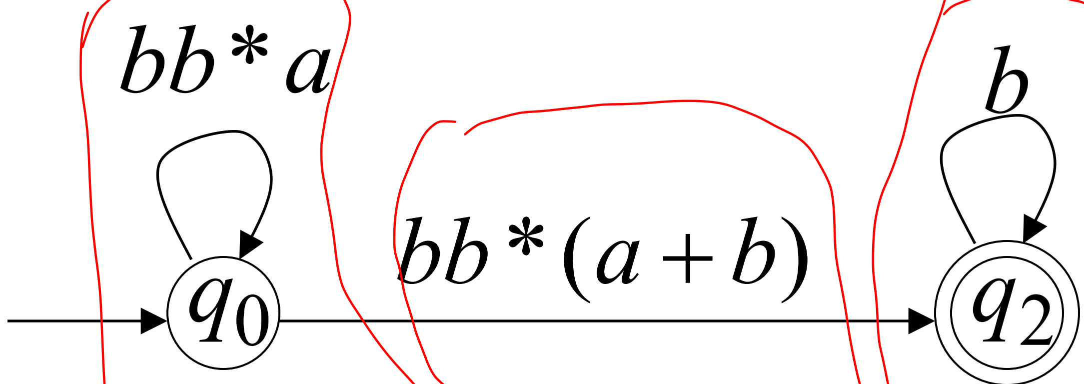

ยุบ state ตรงกลาง

ต้องหาเส้นทางที่ยาวที่สุดจาก $q_0 \rightarrow q_2$

ที่เหลือยุบเมื่อกี้

# Resulting Regular Expression:

ถ้าเป็น regular ต้องแทนด้วย NFA ได้ และจะหลาย gen regexp ได้

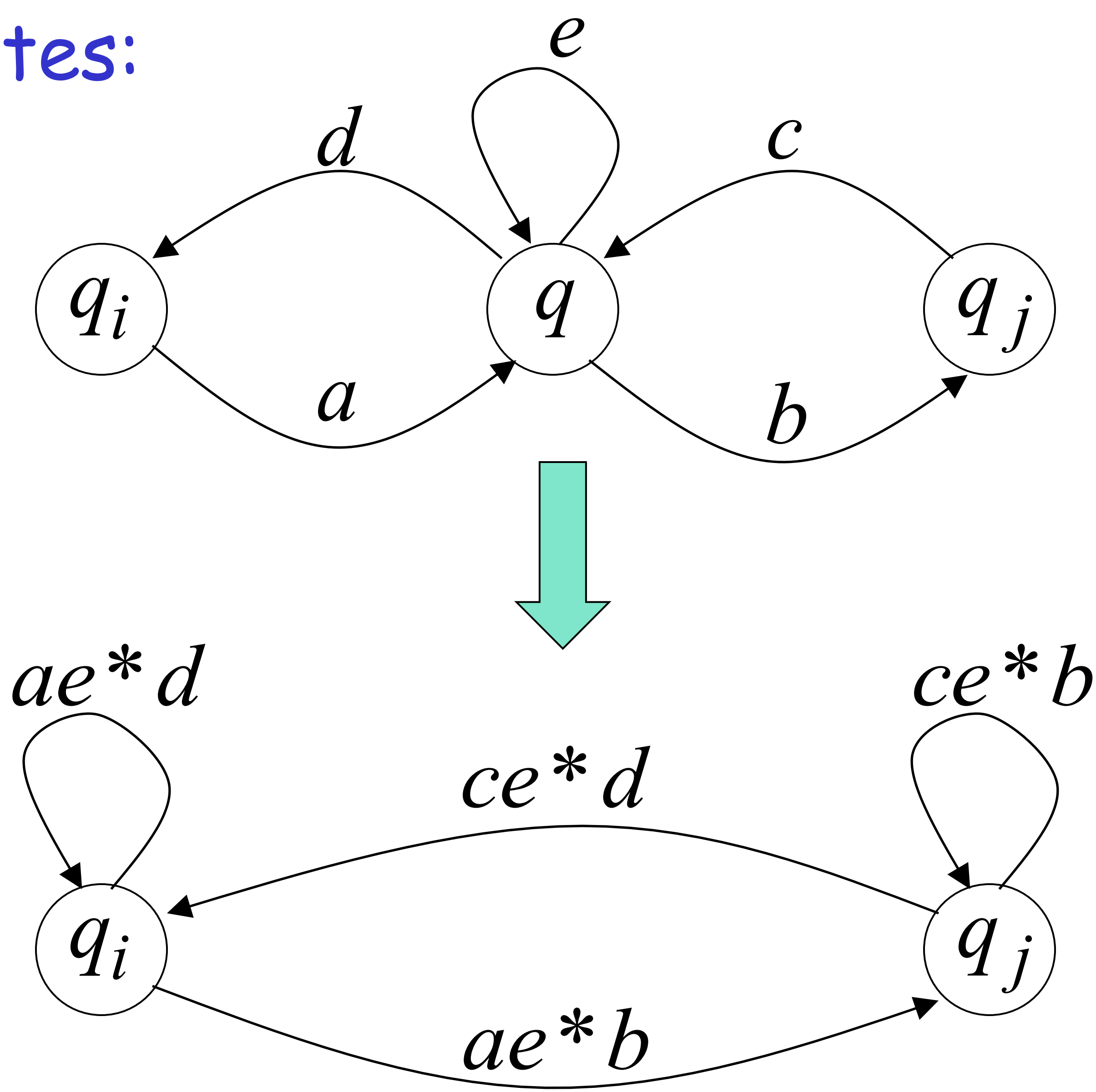



$$r = (bb^*a)^* bb^*(a+b)b^*$$

$$L(r) = L(M) = L$$

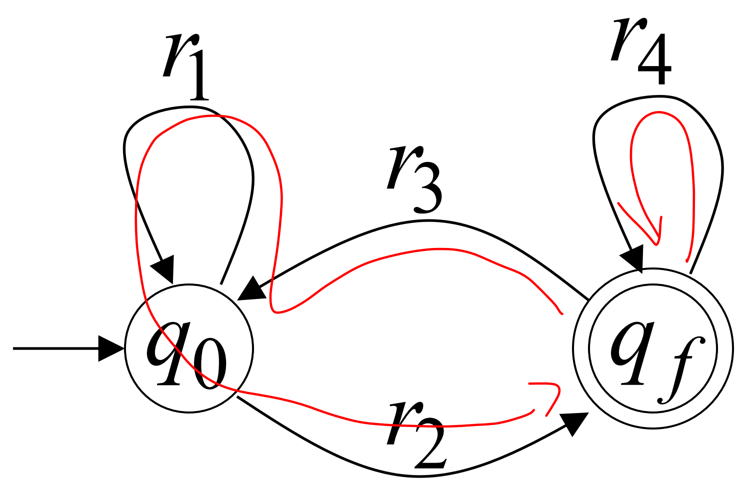
# In General

Removing states:



The final transition graph:

การแทนให้ดูง่าย




ค่อยๆแปลง

The resulting regular expression:

$$r = r_1{}^* r_2 (r_4 + r_3 r_1{}^* r_2)^*$$

$$L(r) = L(M) = L$$

# Why do we need Regular Expressions ?

มี text ที่ยาวมากๆ → อยากหาของที่เป็นตัวอักษร

Let's say you want to find a phone number in a string. You know the pattern: three numbers, a hyphen, three numbers, a hyphen, and four numbers.

Here's an example: 415-555-4242.

**Without RE, your Python code may look lengthy like this.**

sub string ไม่เจาะจง

เขียนแบบนี้ ยาวๆ

```
def isPhoneNumber(text):
    if len(text) != 12:
        return False
    for i in range(0, 3):
        if not text[i].isdecimal():
            return False
    if text[3] != '-':
        return False
    for i in range(4, 7):
        if not text[i].isdecimal():
            return False
    if text[7] != '-':
        return False
    for i in range(8, 12):
        if not text[i].isdecimal():
            return False
    return True
```

# Why do we need Regular Expressions ?

**With Regular Expression, your Python code will be compact.**

ถ้าใช้regex

```
import re
phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d’)
mo = phoneNumRegex.search('My number is 415-555-4242.’)
print('Phone number found: ' + mo.group())

Output:
Phone number found: 415-555-4242
```

<<Back>>