Single Final State for NFAs

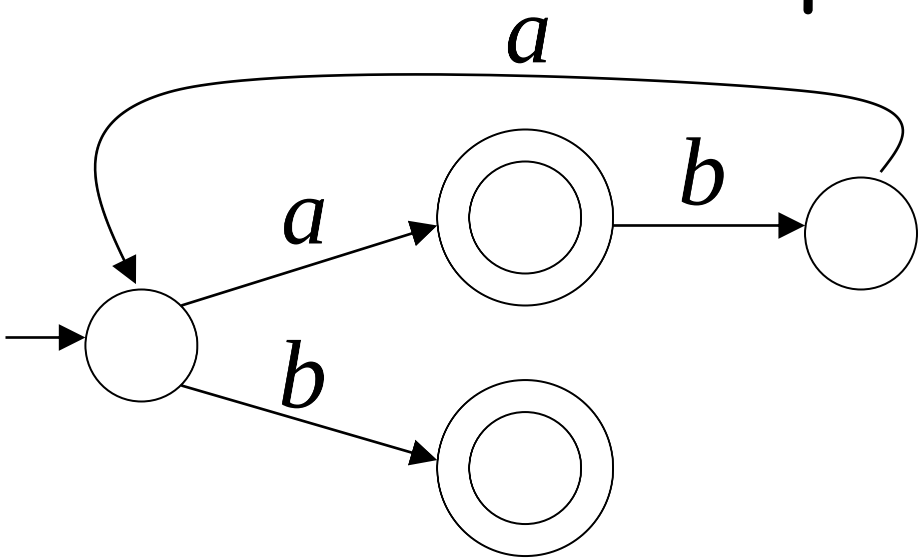
Any NFA can be converted

to an equivalent NFA

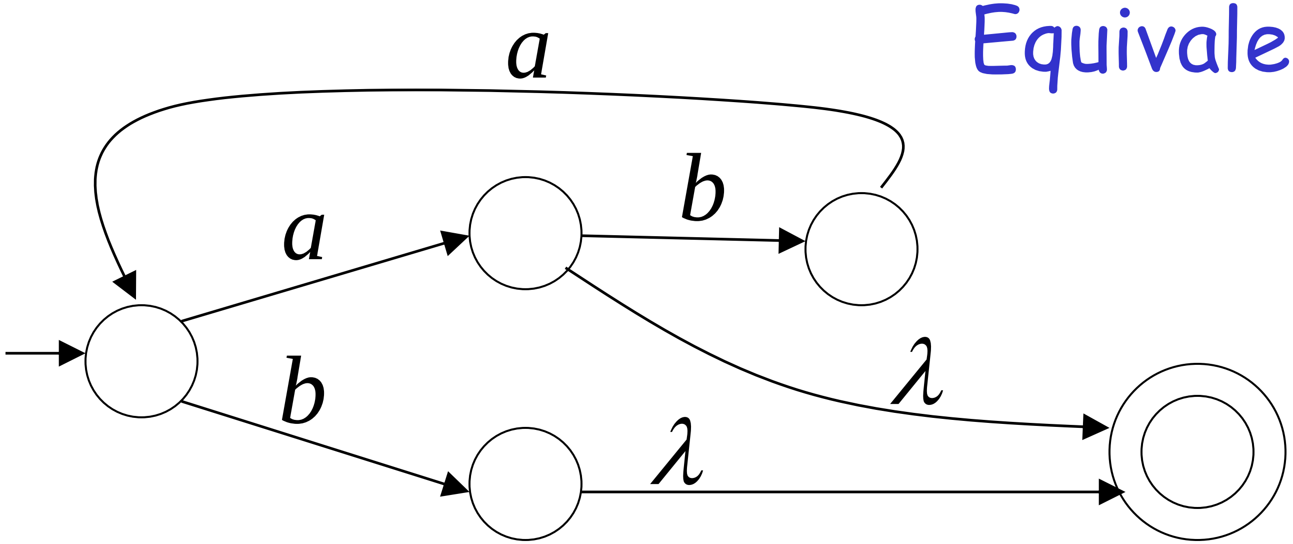
with a single final state

Example

NFA

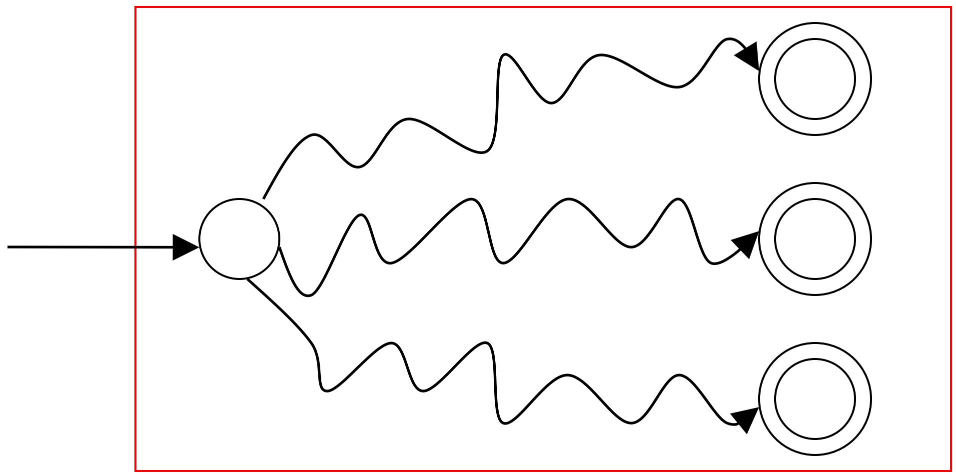


Equivalent NFA

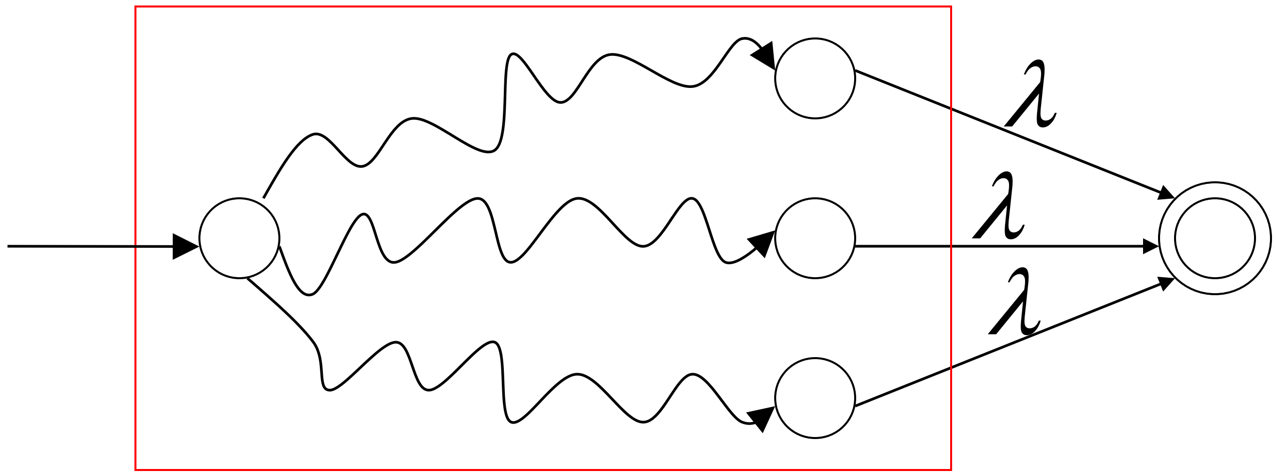


In General

NFA



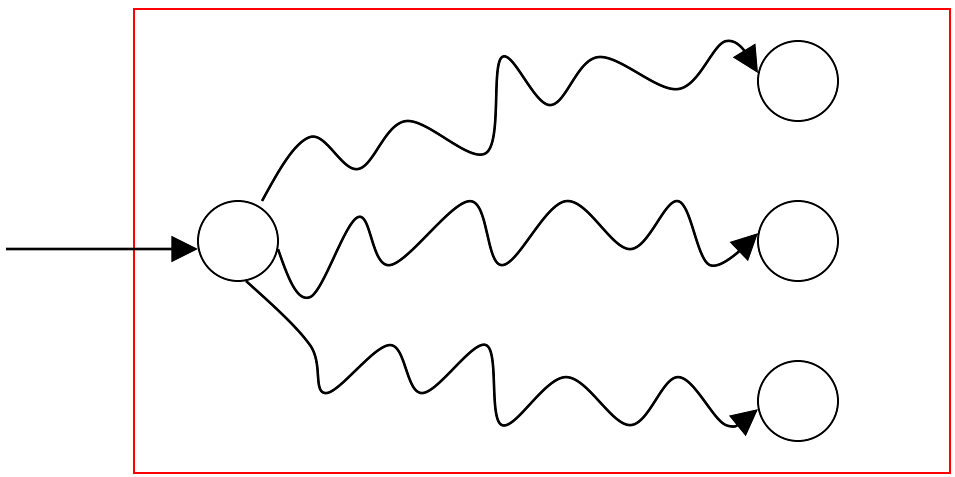
Equivalent NFA

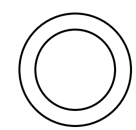


Single final state

Extreme Case

NFA without final state





Add a final state
Without transitions

Properties of Regular Languages

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

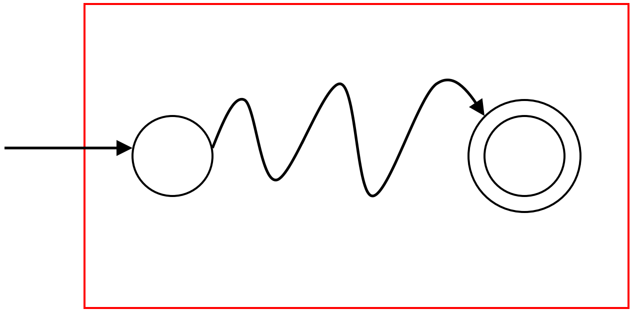
Complement: $\overline{L_1}$

Intersection: $L_1 \cap L_2$

Regular language L_1

$$L(M_1) = L_1$$

NFA M_1

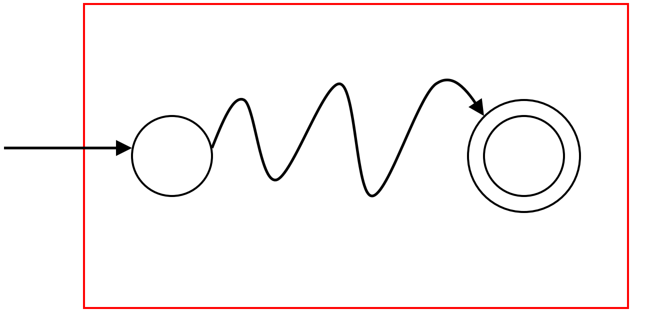


Single final state

Regular language L_2

$$L(M_2) = L_2$$

NFA M_2



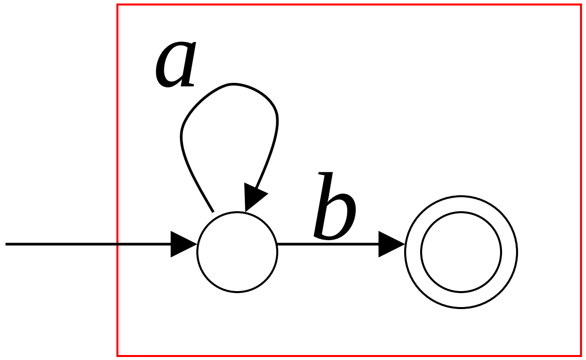
Single final state

Example

$n \geq 0$

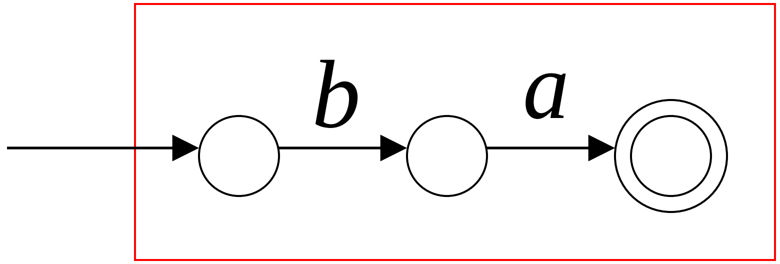
$L_1 = \{a^n b\}$

M_1



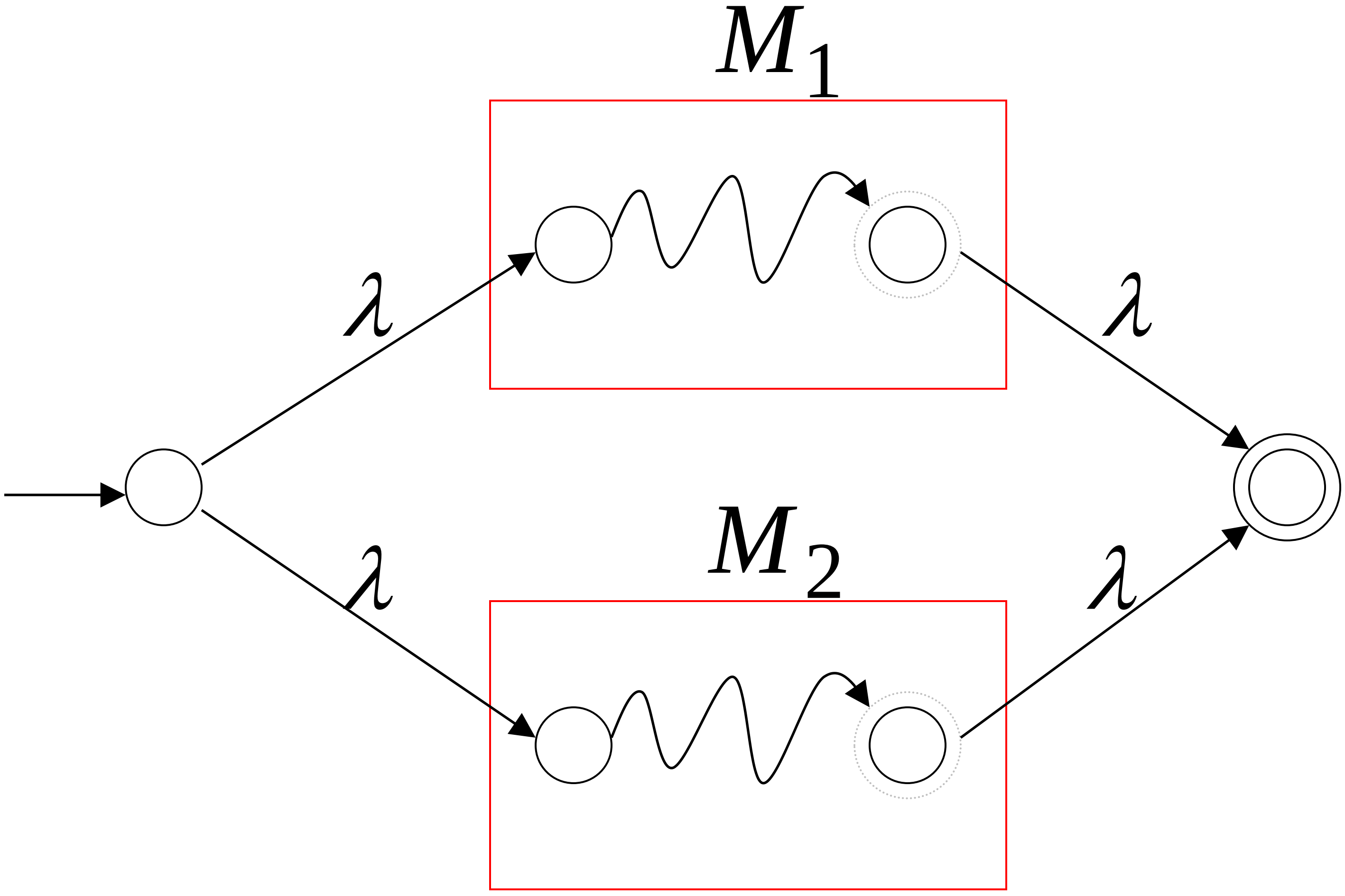
$L_2 = \{ba\}$

M_2



Union

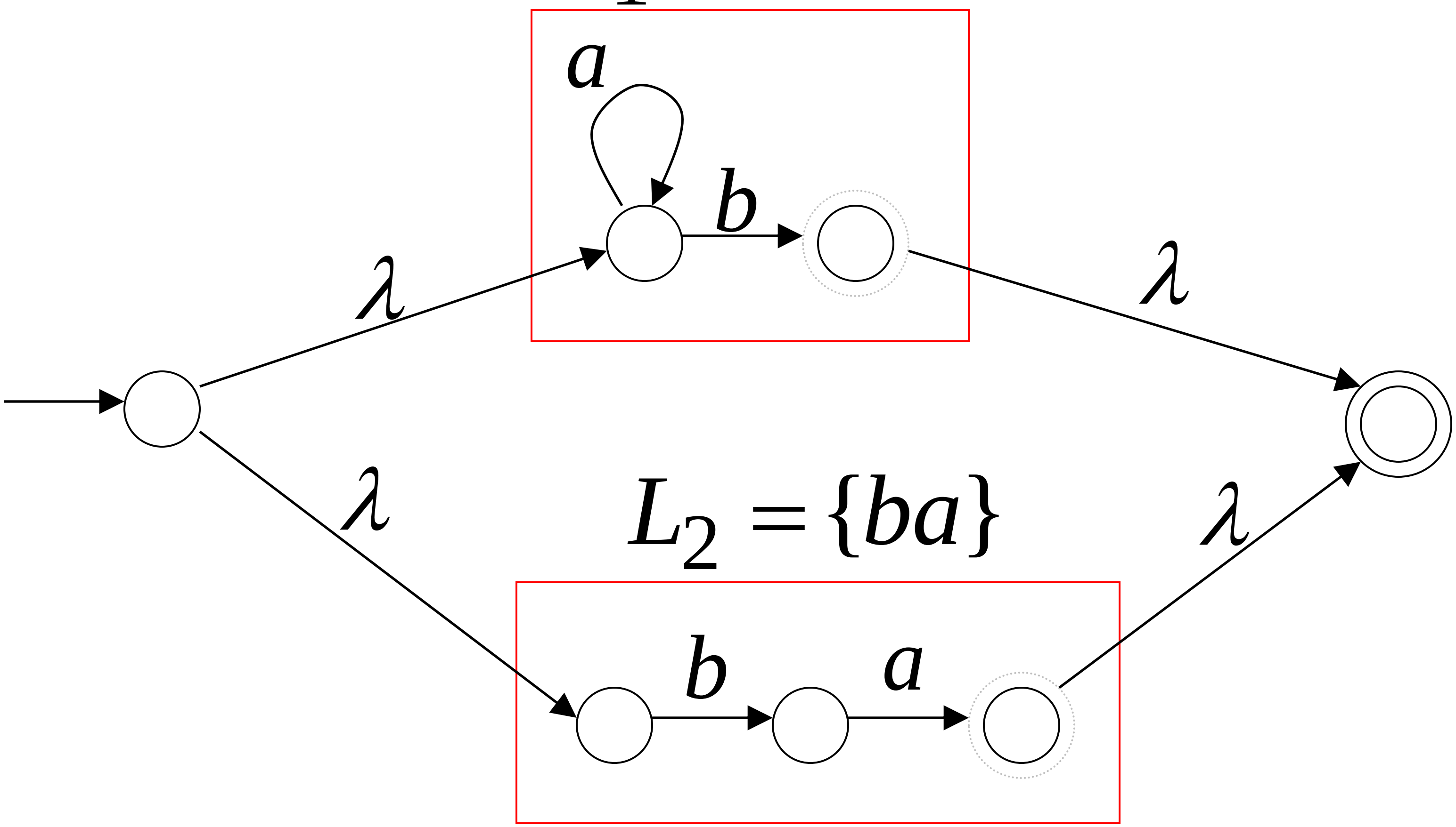
NFA for $L_1 \cup L_2$



Example

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

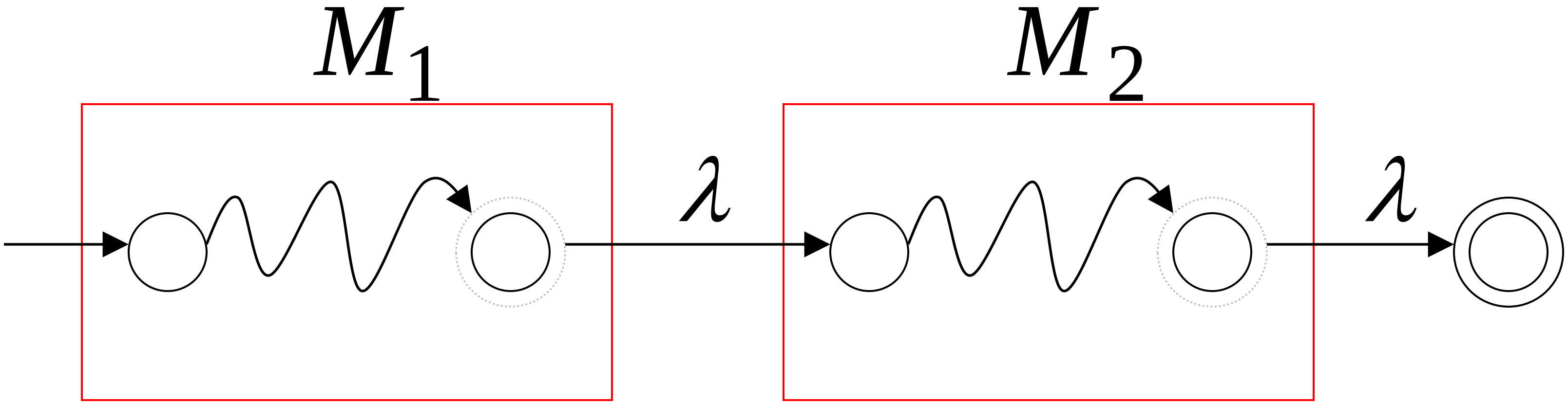
$L_1 = \{a^n b\}$



$L_2 = \{ba\}$

Concatenation

NFA for $L_1 L_2$

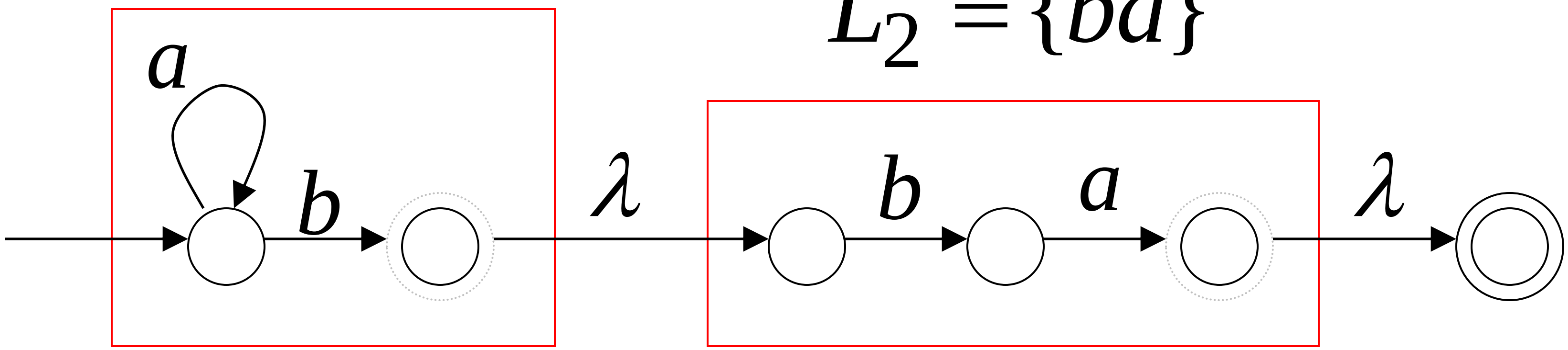


Example

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

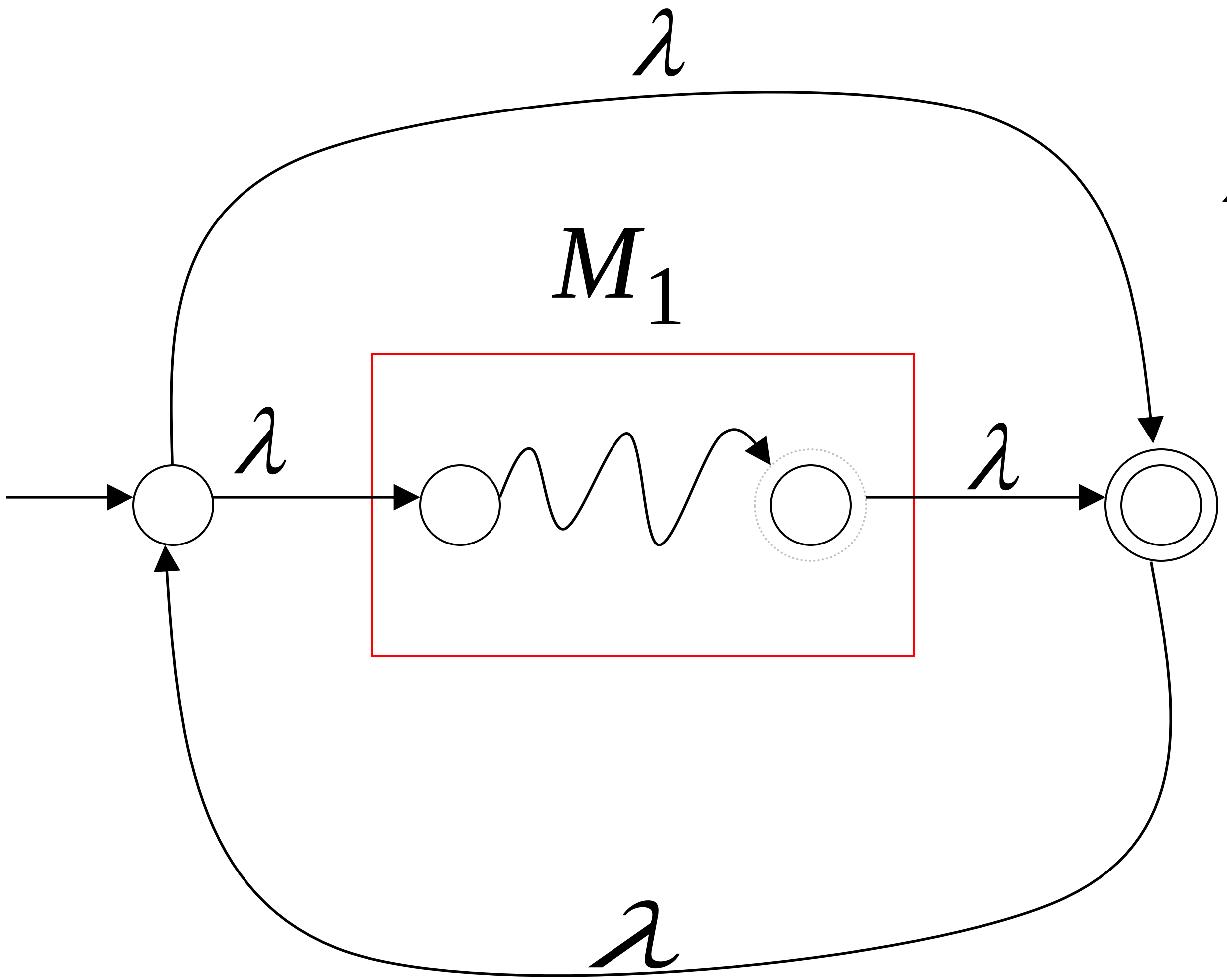
$L_1 = \{a^n b\}$

$L_2 = \{ba\}$



Star Operation

NFA for L_1*



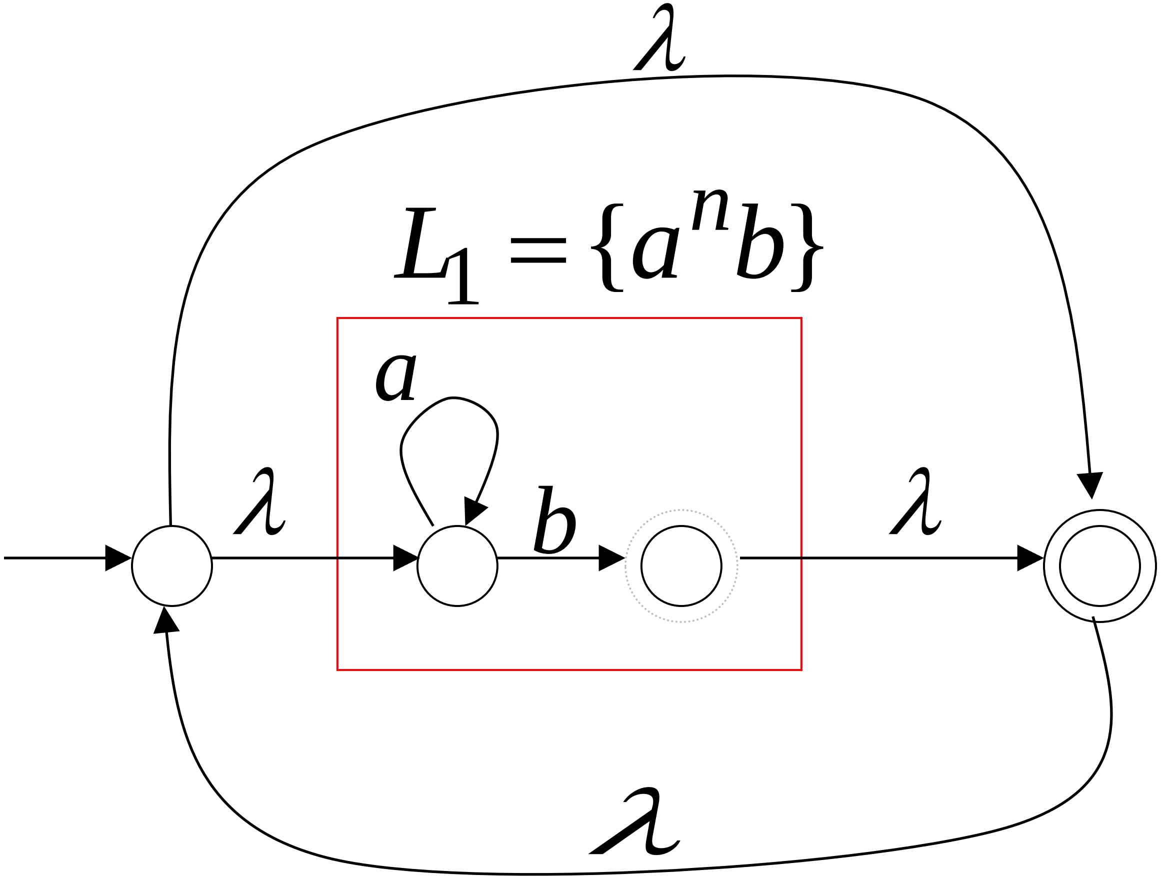
$\lambda \in L_1*$

Example

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

$w = w_1 w_2 \cdots w_k$

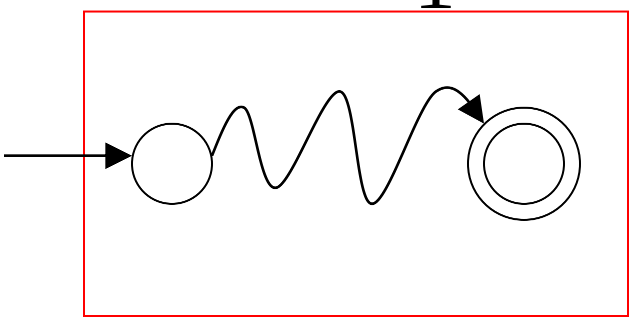
$w_i \in L_1$



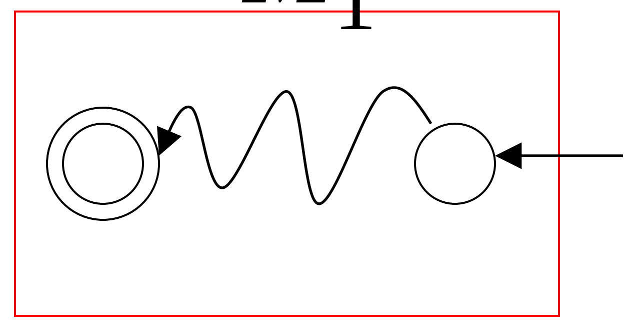
Reverse

NFA for L_1^R

L_1 M_1



M_1'



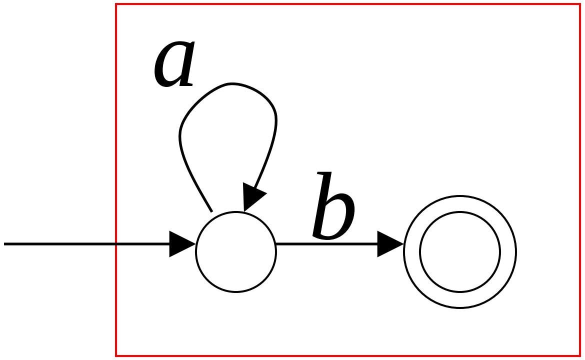
1. Reverse all transitions

2. Make initial state final state
   and vice versa

Example

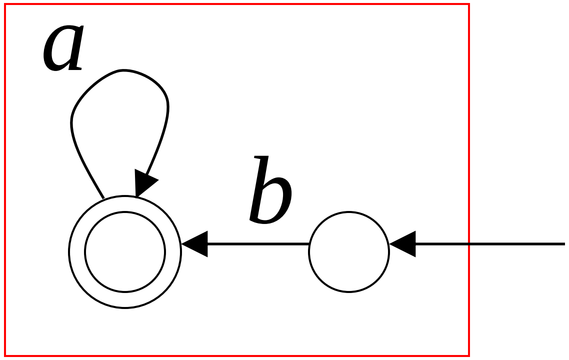
$$L_1 = \{a^n b\}$$

M_1

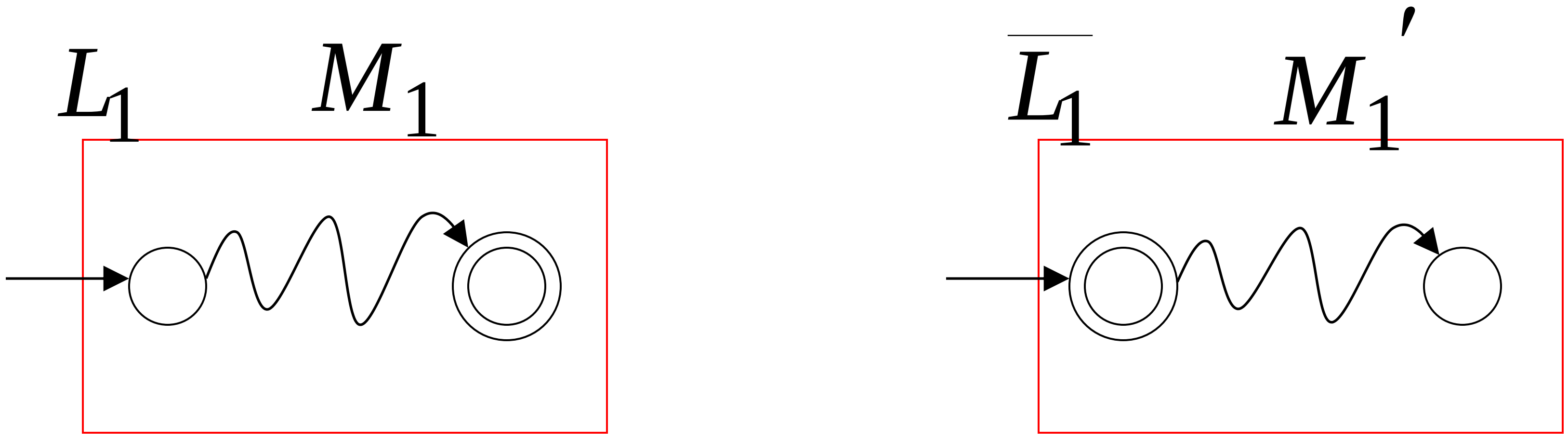


$$L_1{}^R = \{b a^n\}$$

M_1'



Complement



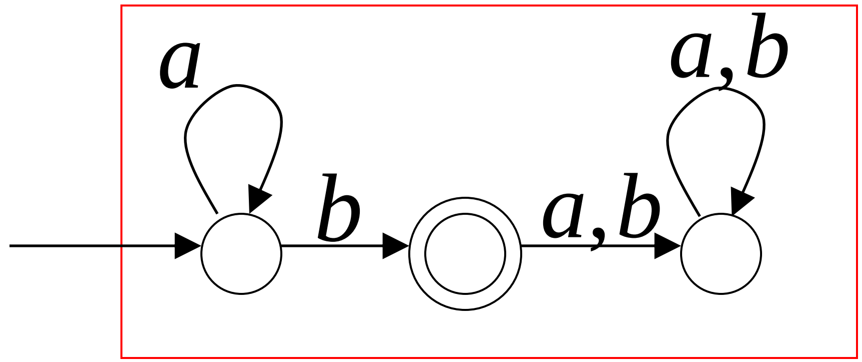
1. Take the **DFA** that accepts L_1

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

Example

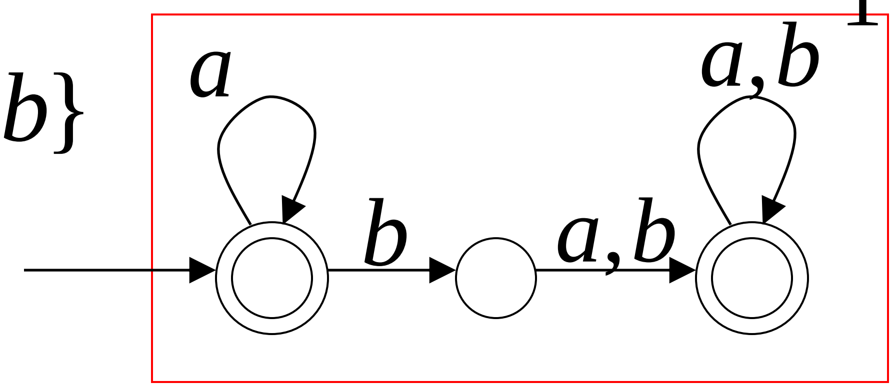
$$L_1 = \{a^n b\}$$

M_1



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

M_1'



Intersection

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

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

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

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

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

regular

Regular Expressions

Regular Expressions

Regular expressions
describe regular languages

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

describes the language

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

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

Recursive Definition

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

Given regular expressions r_1 and r_2

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

Examples

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

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

Languages of Regular Expressions

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

Example

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

Definition

For primitive regular expressions:

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

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

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

Definition (continued)

For regular expressions r_1 and r_2

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

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

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

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

Example

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

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

Example

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

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

Example

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

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

Example

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

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

Example

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

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

Equivalent Regular Expressions

Definition:

Regular expressions r_1 and r_2

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

Example

L = { all strings without two consecutive 0 }

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

$$r_2 = (1^*011^*)^*(0+\lambda)+1^*(0+\lambda)$$

$L(r_1) = L(r_2) = L$ $\Rightarrow$ r_1 and r_2 are equivalent regular expr.

Regular Expressions
and
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

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

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

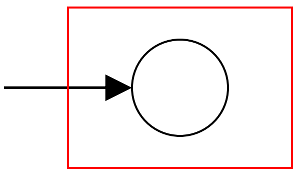
1. For any regular expression r
   the language $L(r)$ is regular

Proof by induction on the size of r

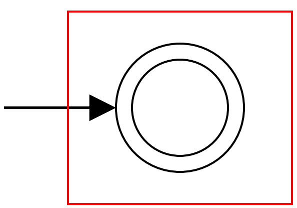
Induction Basis

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

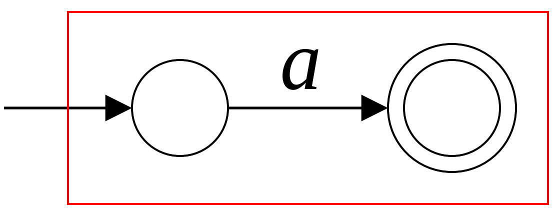
NFAs



$$L(M_1) = \emptyset = L(\emptyset)$$



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



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

regular languages

Inductive Hypothesis

Assume
for regular expressions r_1 and r_2
that
$L(r_1)$ and $L(r_2)$ are regular languages

Inductive Step

We will prove:

$$L(r_1 + r_2)$$

$$L(r_1 \cdot r_2)$$

$$L(r_1 *)$$

$$L((r_1))$$

Are regular Languages

By definition of regular expressions:

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

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

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

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

By inductive hypothesis we know:

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

We also know:

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

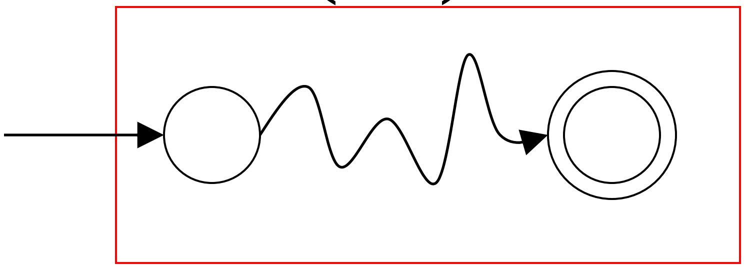
Proof - Part 2

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

Proof by construction of regular expression

Since L is regular take the NFA M that accepts it

$$L(M) = L$$



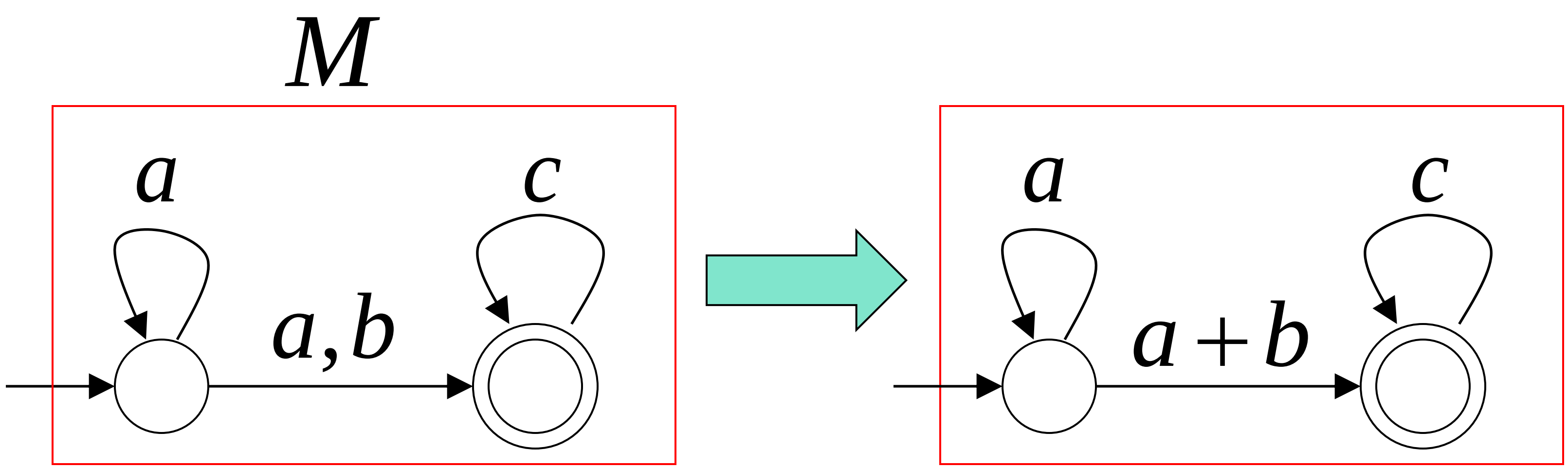
Single final state

From M construct the equivalent

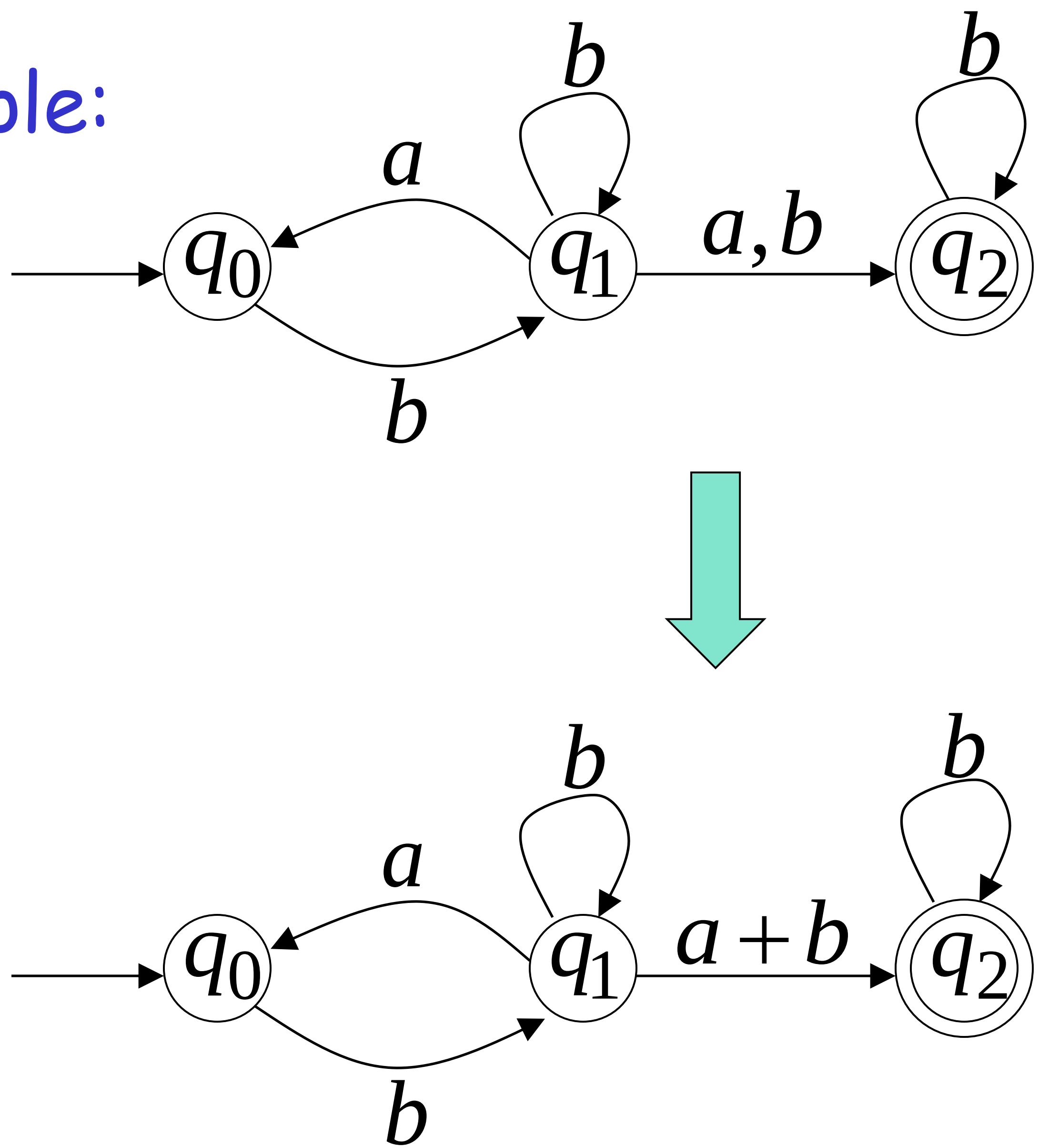
Generalized Transition Graph

in which transition labels are regular expressions

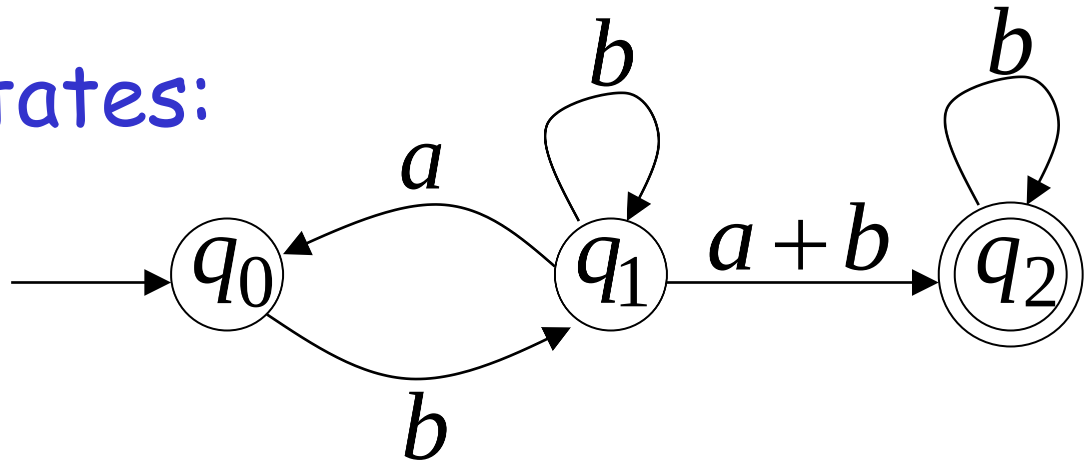
Example:

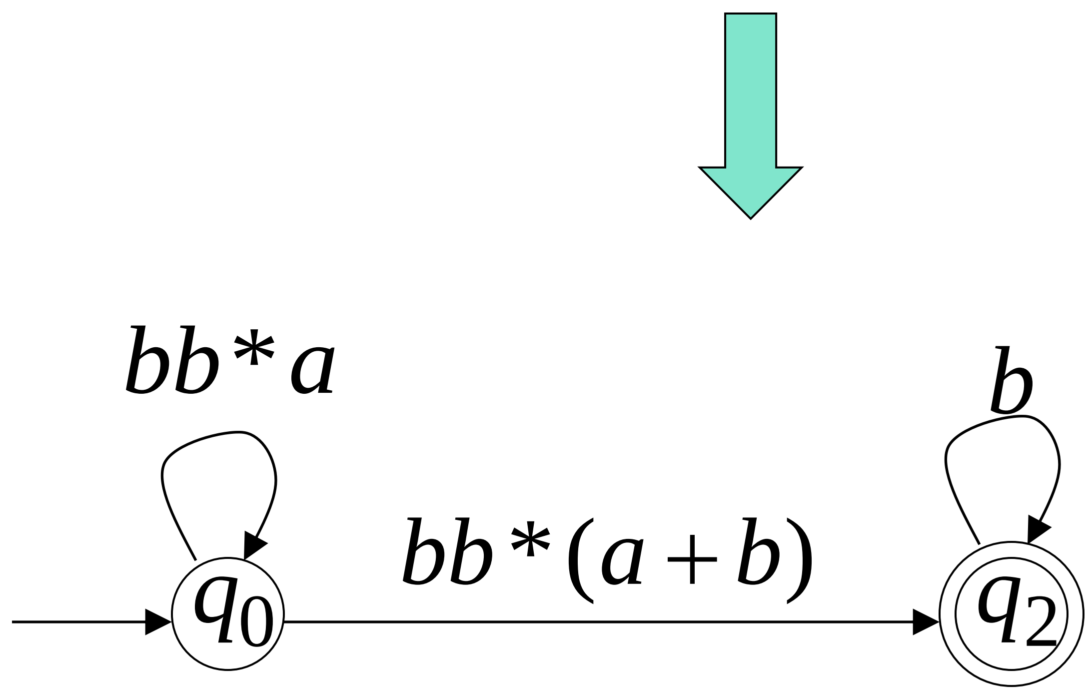


Another Example:

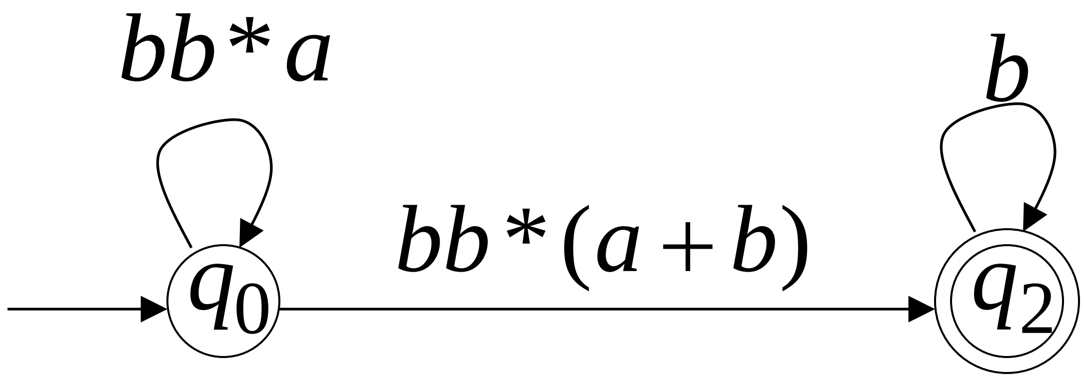


Reducing the states:





Resulting Regular Expression:

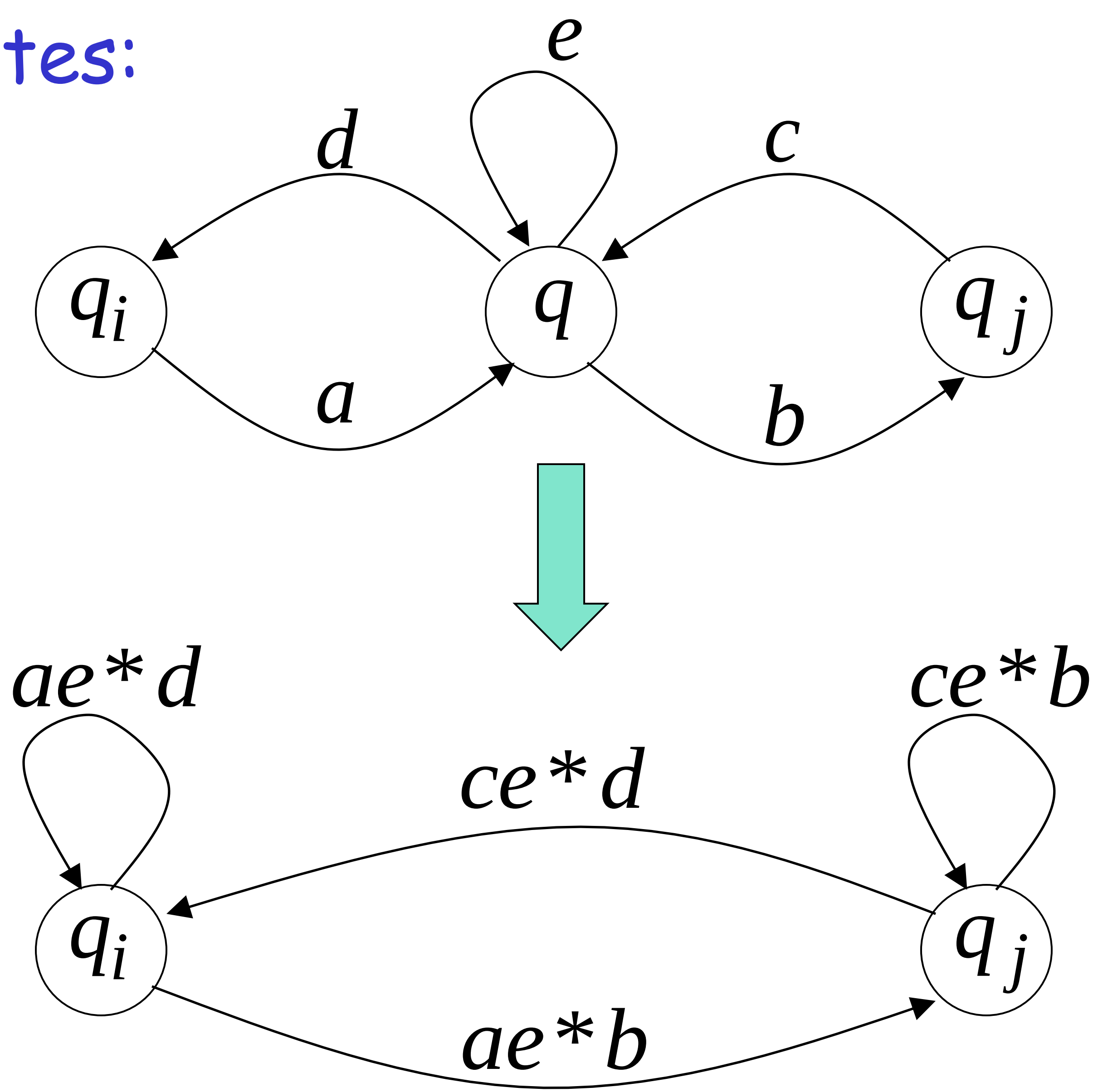


$$r = (bb^*a)^*bb^*(a+b)b^*$$

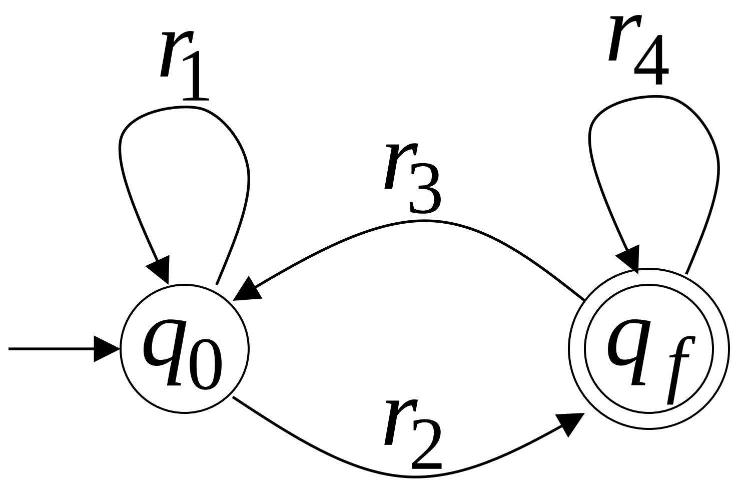
$$L(r) = L(M) = L$$

In General

Removing states:



The final transition graph:



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 ?

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.

```
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.

```
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>>