# Chapter 3

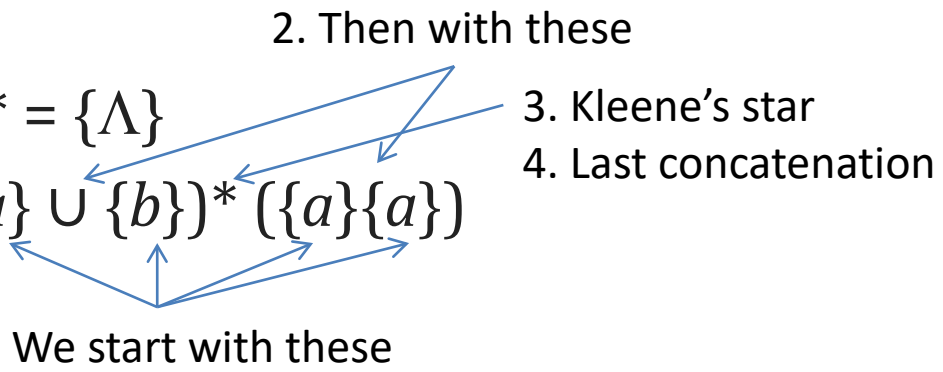
## *Regular Expressions, Regular Languages, Nondeterminism, and Kleene's Theorem*

# Regular Languages and Regular Expressions

- Many simple languages can be expressed by a formula involving languages containing a single string of length 1 and the operations of union, concatenation and Kleene star. Here are three examples
  - Strings ending in *aa*: $\{a, b\}^* \{aa\}$
  - (This is a simplification of $(\{a\} \cup \{b\})^*\{a\}\{a\}$ )
  - Strings containing *ab* or *bba*: $\{a, b\}^* \{ab, bba\} \{a, b\}^*$
- These are called *regular* languages

# Regular Languages and Regular Expressions (cont'd.)

- Definition: If $\Sigma$ is an alphabet**, the set *R* of regular languages** over $\Sigma$ is defined as follows:
  - The language $\varnothing$ is an element of ***R***, and for every $\sigma \in \Sigma$, the language $\{\sigma\}$ is in ***R***
  - For every two languages $L_1$ and $L_2$ in ***R***, the three languages $L_1 \cup L_2$, $L_1L_2$, and $L_1{}^*$ are elements of ***R***
- Examples:
  - $\{\Lambda\}$, because $\varnothing^* = \{\Lambda\}$
  - $\{a, b\}^*\{aa\} = (\{a\} \cup \{b\})^* (\{a\}\{a\})$



# Regular Languages and Regular Expressions (cont'd.)

- A *regular expression* for a language is a slightly more user-friendly formula which is similar to algebraic expressions
  - Parentheses replace curly braces, and are used only when needed, and the union symbol is replaced by +

| ***Regular language*** | ***Regular Expression*** |
|---|---|
| $\varnothing$ | $\varnothing$ |
| $\{\Lambda\}$ | $\Lambda$ |
| $\{a,b\}^*$ | $(a+b)^*$ |
| $\{aab\}^*\{a,ab\}$ | $(aab)^*(a+ab)$ |
| $(\{aa, bb\} \cup \{ab, ba\}\{aa, bb\}^*\{ab, ba\})^*$ | $(aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$ |

# Regular Languages and Regular Expressions (cont'd.)

- A regular expression describes a regular language, and a regular language can be described by a regular expression.
- Two regular expressions are equal if the languages they describe are equal. For example,
  - $(a^*b^*)^* = (a+b)^*$
  - $(a+b)^*ab(a+b)^*+b^*a^* = (a+b)^*$
    - The first half of the left-hand expression describes the strings that contain the substring *ab* and the second half describes those that don't

# Regular Languages and Regular Expressions (cont'd.)

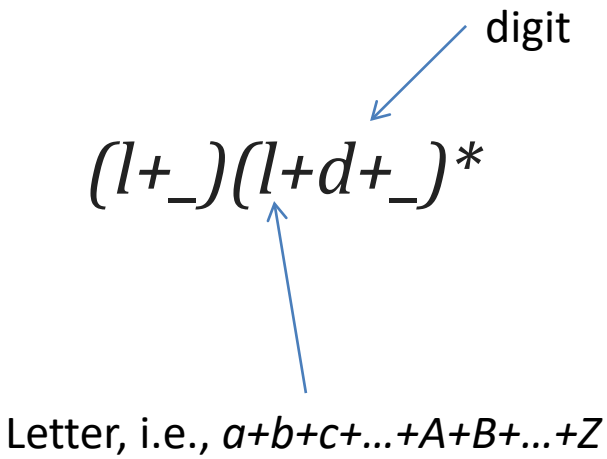
- The language in {a, b}* with an odd number of a's
- A string with an odd number of *a*'s has at least one *a*, and the additional *a*'s can be grouped into pairs. There can be arbitrarily many *b*'s before the first *a*, between any two consecutive *a*'s, and after the last *a*.

  - *b*ab*(ab*ab*)**
  - *b*a(b*ab*a)*b**
  - *b*a(b+ab*a)**
  - *(b+ab*a)*ab**

See more examples in the textbook!

# Regular Languages and Regular Expressions (cont'd.)

- An identifier in C is a string of length 1 or more that contains only letters, digits, and underscores ("_") and does not begin with a digit.



For the alphabet {0, 1} find regular expressions for languages

- All binary strings

  (0+1)* = (1+0)*

- All binary strings of even length

  ((0+1)(0+1))*

- All binary strings containing the substring 001

  (0+1)*001(0+1)*

- All binary strings with #1s = 0 mod 3

  0* + (0*10*10*10*)*

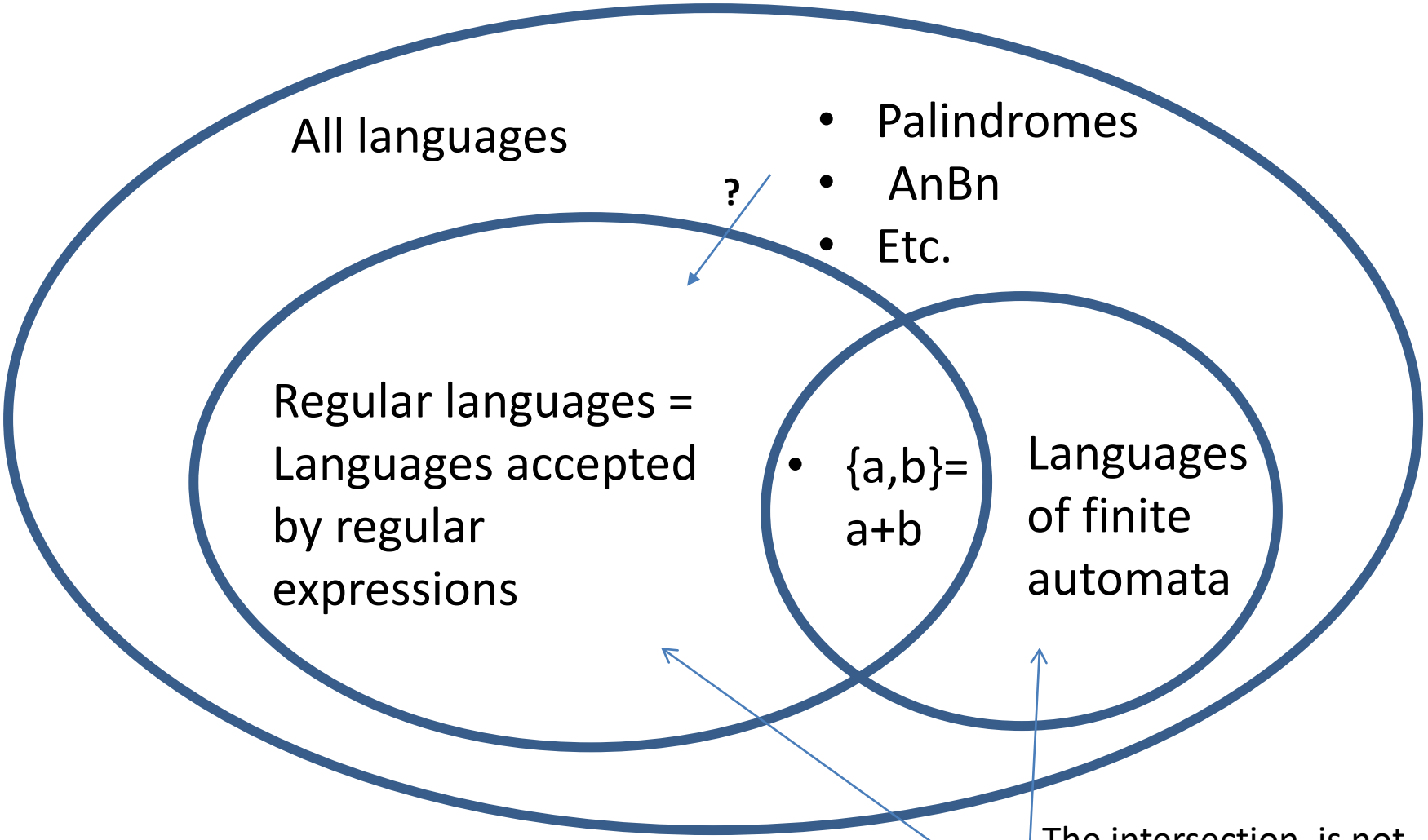
- All binary strings without two consecutive 0s

  (01+1)*(0+$\Lambda$)

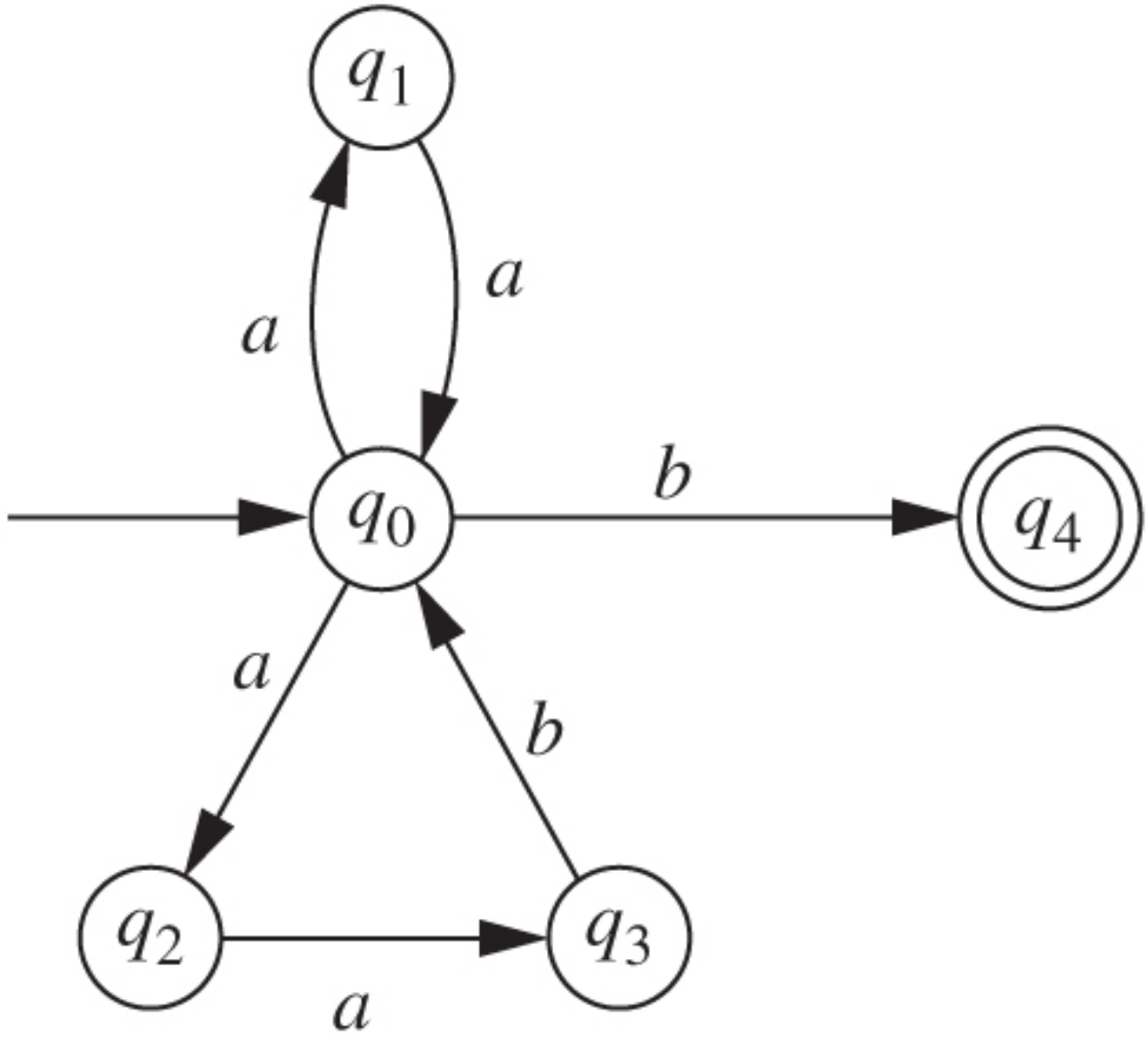
- All binary strings with either 001 or 100 occurring somewhere

  (0+1)*001(0+1)* + (0+1)*100(0+1)*

# This is what we know about languages …



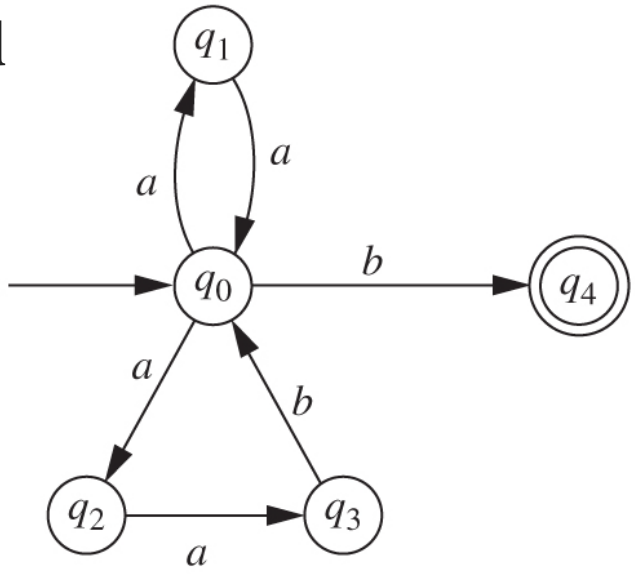
# Nondeterministic Finite Automata


$q_1$
$a$
$a$
$q_0$
$b$
$q_4$
$a$
$b$
$q_2$
$q_3$
$a$

# Nondeterministic Finite Automata

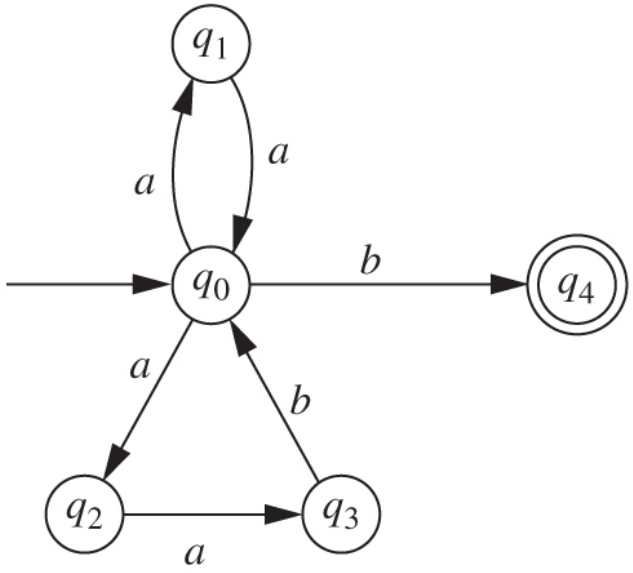
- This NFA closely resembles the regular expression **(*aa* + *aab*)****b***
  - The top loop is *aa*
  - The bottom loop is *aab*
  - By following the links we can generate any string in the language
- This is not the transition diagram for an FA; some nodes have more than one *a*-arc, some have none
- Example: *aaaabaab* can be either accepted (top-bottom-top-b) or

  not accepted (top-bottom-bottom loops).



# Nondeterministic Finite Automata

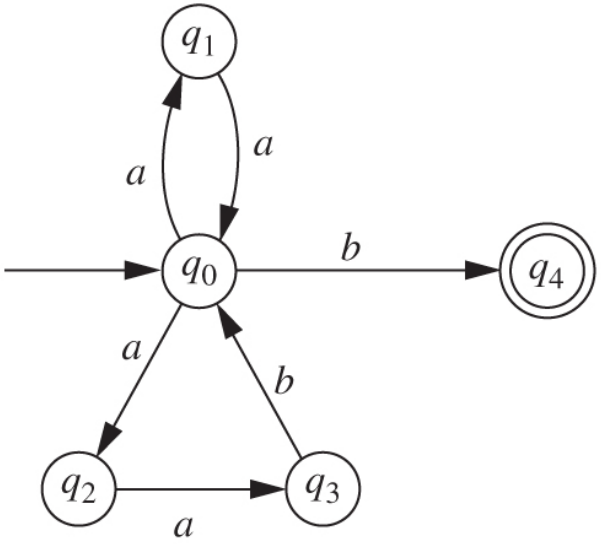
- For this reason, we should **not** think of an NFA as describing an algorithm for recognizing a language
- Instead, consider it as describing a number of different sequences of steps that *might* be followed

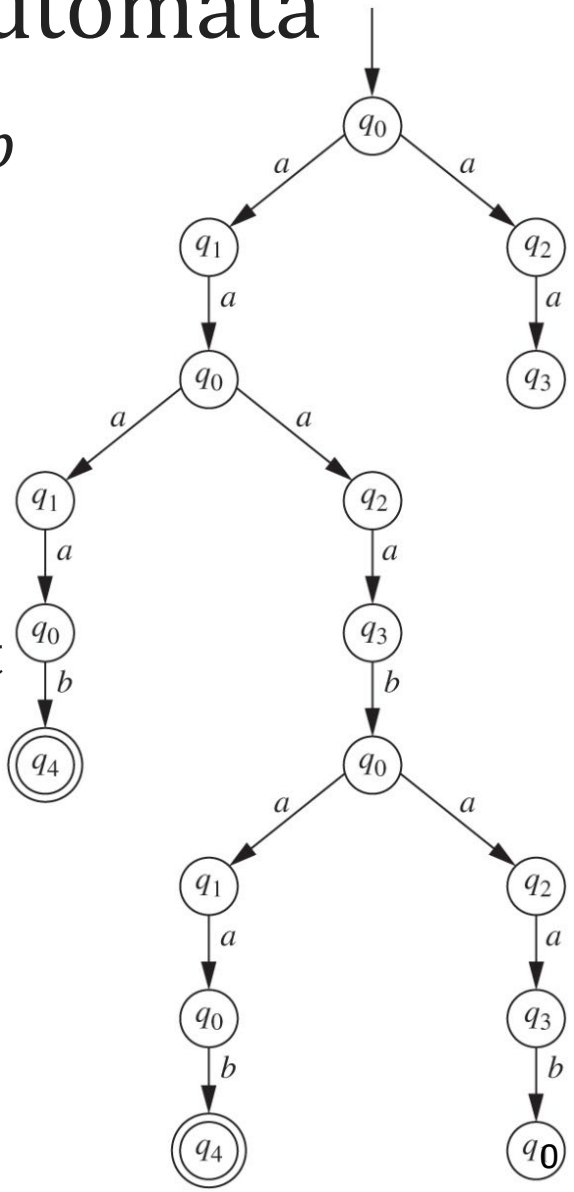


# Nondeterministic Finite Automata

This is the "computation tree" for *aaaabaab*

- Each level corresponds to a prefix of the input string
- Each state on a level is one  the machine *could* be in after processing that prefix
- There is an accepting path for the input string (as well as other paths that are not accepting)

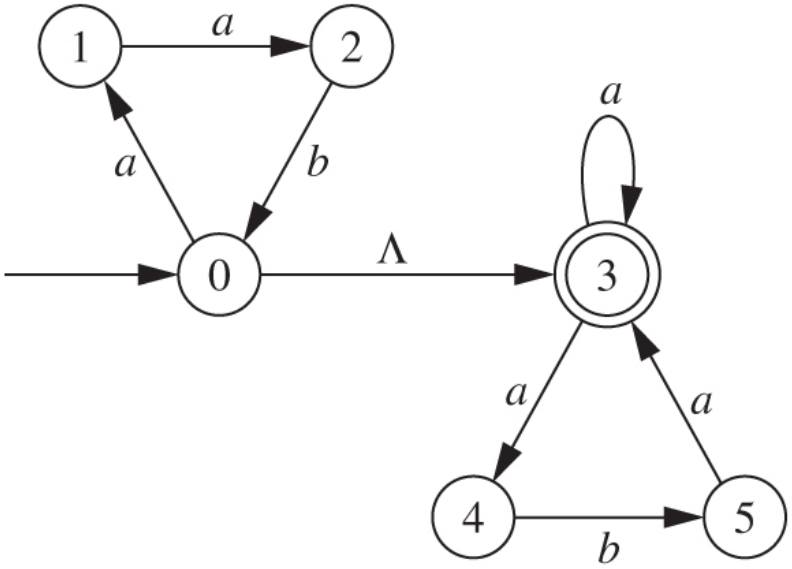


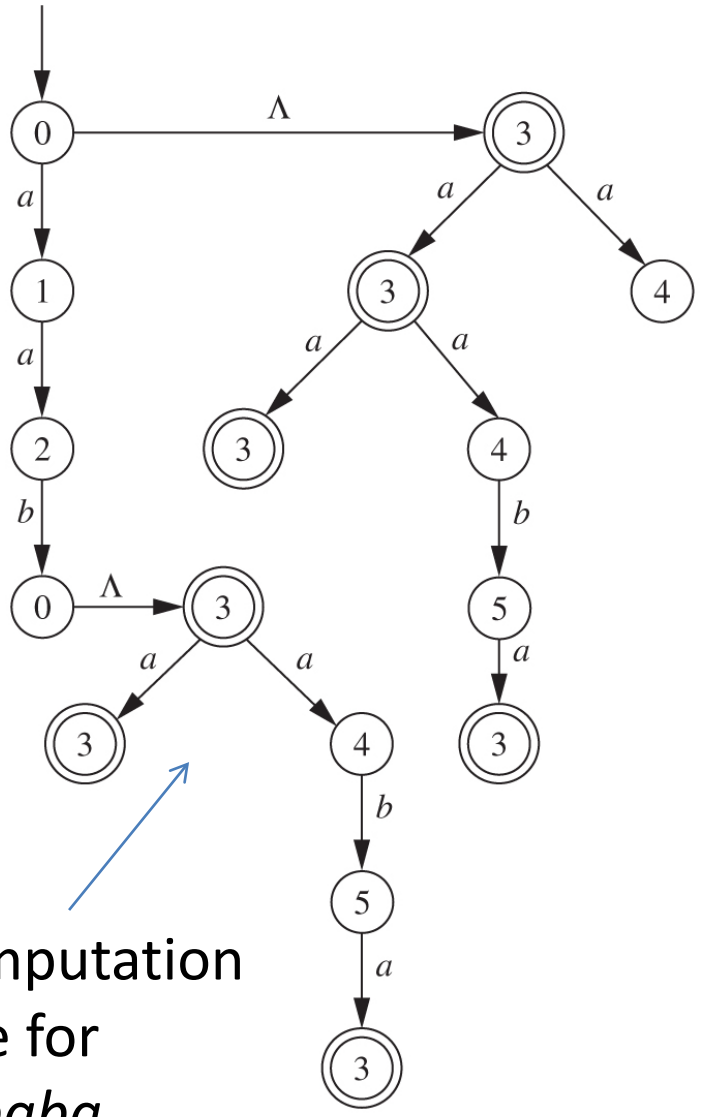



# NFA: **Λ-transitions**

The technique in previous example does not provide a simple way to draw a transition diagram for *(aab)*(a+aba)**

- We introduce a new feature called **Λ-transition.**
- **It allows the device to change state without reading the next symbol.**



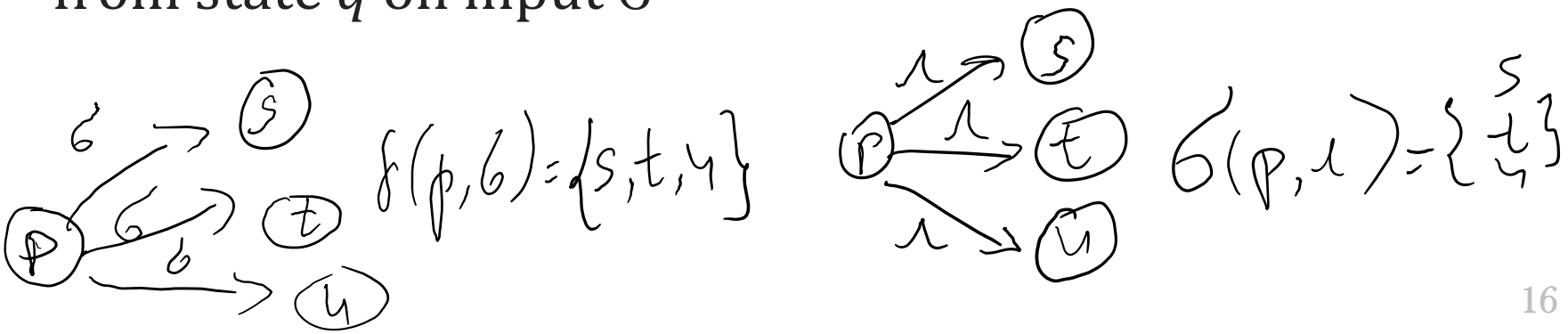




Computation tree for *aababa*

# Nondeterministic Finite Automata

- Definition: A *nondeterministic finite automaton* (NFA) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$, where:
  - $Q$ is a finite set of states,
  - $\Sigma$ is a finite input alphabet
  - $q_0 \in Q$ is the initial state
  - $A \subseteq Q$ is the set of accepting states
  - $\delta : Q \times (\Sigma \cup \{\Lambda\}) \rightarrow 2^Q$ is the transition function.

  (The values of $\delta$ are not single states, but *sets* of states)
- For every $q \in Q$ and every $\sigma \in \Sigma \cup \{\Lambda\}$, we interpret $\delta(q, \sigma)$ as the set of states to which the NFA can move from state $q$ on input $\sigma$



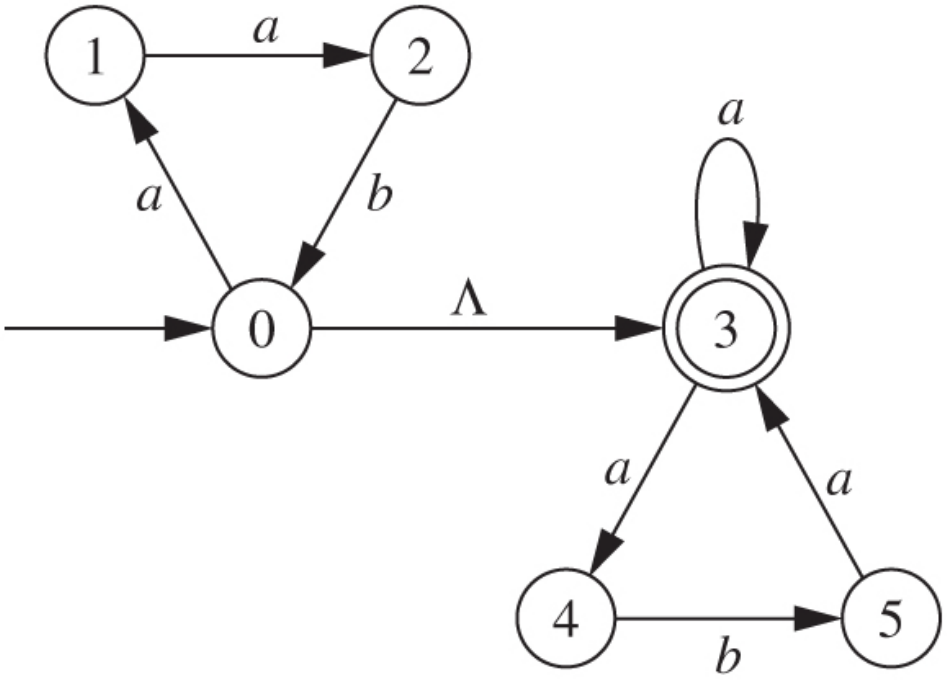
- Example:

$\delta(0,a) = \{1\}$

$\delta(0,\Lambda) = \{3\}$

$\delta(0,b) = \varnothing$

$\delta(3,a) = \{3, 4\}$



# Nondeterministic Finite Automata

How to define $\delta^*(q, x\sigma)$?

Defining $\delta^*$ is a little harder than for an FA, since $\delta^*(q, x)$ is a set, as is $\delta(p, \sigma)$ for any $p$ in the first set:

$\cup \{ \delta(p, \sigma) \mid p \in \delta^*(q, x) \}$ is a first step towards $\delta^*$

We must also consider $\Lambda$-transitions, which could potentially occur at any stage

- Definition: Suppose $M = (Q, \Sigma, q_0, A, \delta)$ is an NFA, and $S \subseteq Q$ is a set of states
  - **The $\Lambda$-*closure* of $S$ is the set $\Lambda(S)$** that can be defined recursively as follows:
    - $S \subseteq \Lambda(S)$
    - For every $q \in \Lambda(S)$, $\delta(q, \Lambda) \subseteq \Lambda(S)$

- Definition: Suppose $M = (Q, \Sigma, q_0, A, \delta)$ is an NFA, and $S \subseteq Q$ is a set of states
  - **The $\Lambda$-*closure* of $S$ is the set $\Lambda(S)$** that can be defined recursively as follows:
    - $S \subseteq \Lambda(S)$
    - For every $q \in \Lambda(S)$, $\delta(q, \Lambda) \subseteq \Lambda(S)$

- Definition: Suppose $M = (Q, \Sigma, q_0, A, \delta)$ is an NFA, and $S \subseteq Q$ is a set of states
  - **The $\Lambda$-*closure* of $S$ is the set $\Lambda(S)$** that can be defined recursively as follows:
    - $S \subseteq \Lambda(S)$
    - For every $q \in \Lambda(S)$, $\delta(q, \Lambda) \subseteq \Lambda(S)$

- As for any finite set that is defined recursively, we can easily formulate **an algorithm to calculate $\Lambda(S)$**:
  - Initialize $T$ to be $S$, as in the basis part of the definition
  - Make a sequence of passes, in each pass considering every $q \in T$ and adding every state in $\delta(q, \Lambda)$ not already there
  - Stop after the first pass in which $T$ does not change
  - The final value of $T$ is $\Lambda(S)$

# Nondeterministic Finite Automata

- Definition: Let $M=(Q, \Sigma, q_0, A, \delta)$ be an NFA

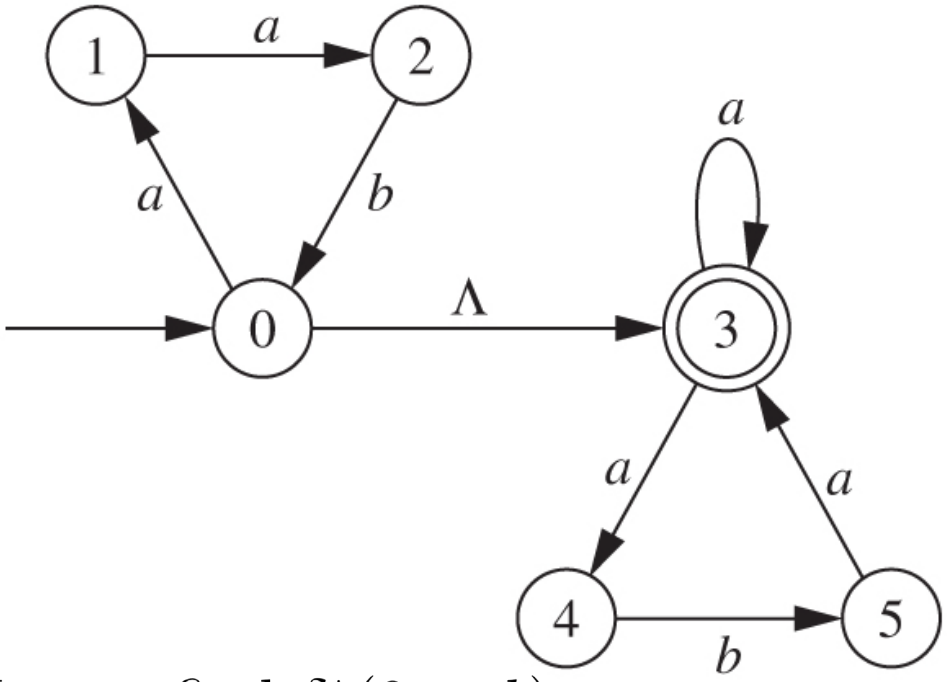
  Define the **extended transition function**

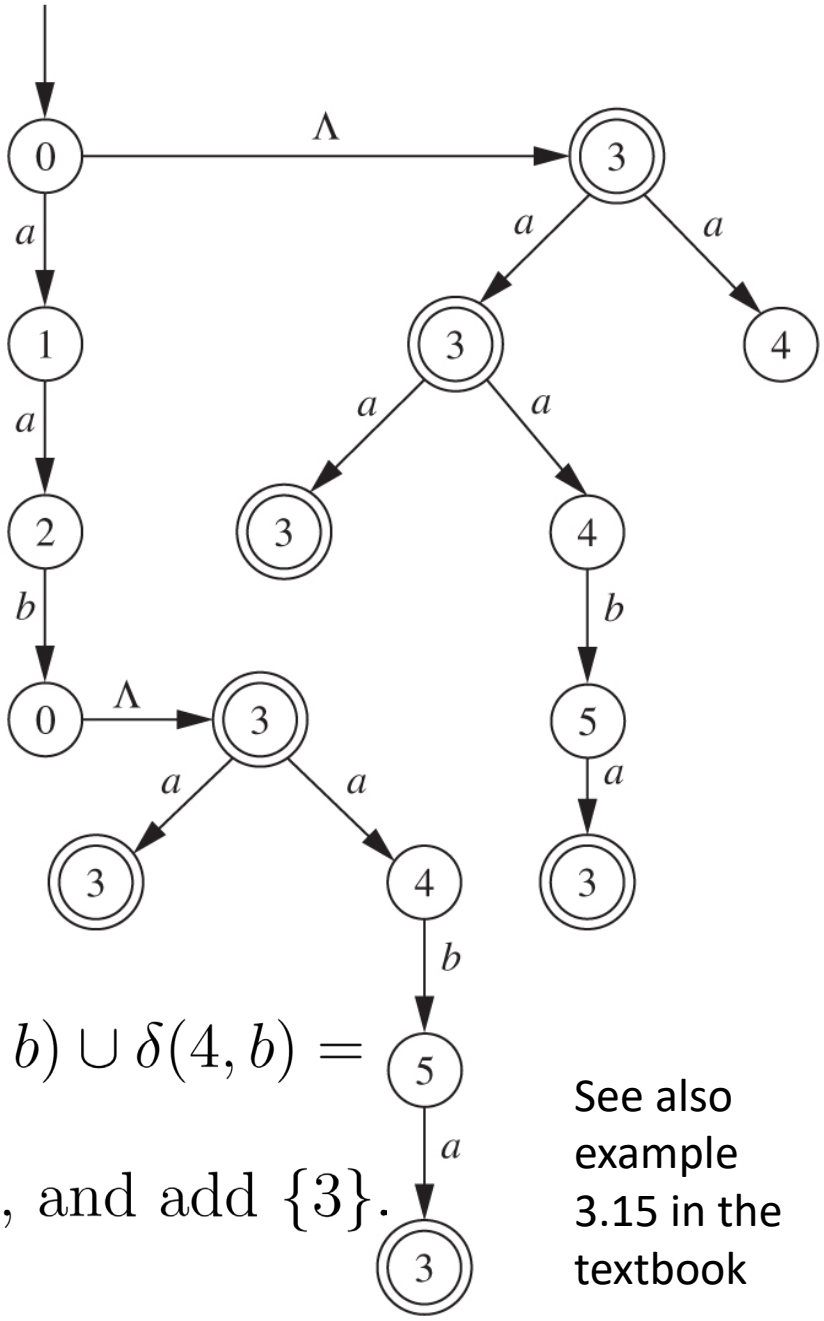
  $$\delta^* : Q \times \Sigma^* \rightarrow 2^Q \text{ as follows:}$$

  - For every $q \in Q$, $\delta^*(q,\Lambda) = \Lambda(\{q\})$
  - For every $q \in Q$, every $y \in \Sigma^*$, and every $\sigma \in \Sigma$
    - $\boldsymbol{\delta^*(q, y\sigma) = \Lambda(\cup \{\delta(p, \sigma) \mid p \in \delta^*(q, y)\})}$
  - A string $x \in \Sigma^*$ is accepted by $M$ if $\delta^*(q_0, x) \cap A \neq \varnothing$

  (i.e., some sequence of transitions **involving the symbols of $x$ and $\Lambda$'s** leads from $q_0$ to an accepting state)

- The language $L(M)$ accepted by $M$ is the set of all strings accepted by $M$




Let us find $\delta^*(0, aab)$

$\Lambda(\{0\}) = \{0, 3\}$;

Then, $\delta^*(0, aa) = \{2, 3, 4\}$;

Now we add $b$ to $aa$

$\cup\{\delta(p, b)|p \in \{2, 3, 4\}\} = \delta(2, b) \cup \delta(3, b) \cup \delta(4, b) =$

$= \{0\} \cup \emptyset \cup \{5\} = \{0, 5\}$;

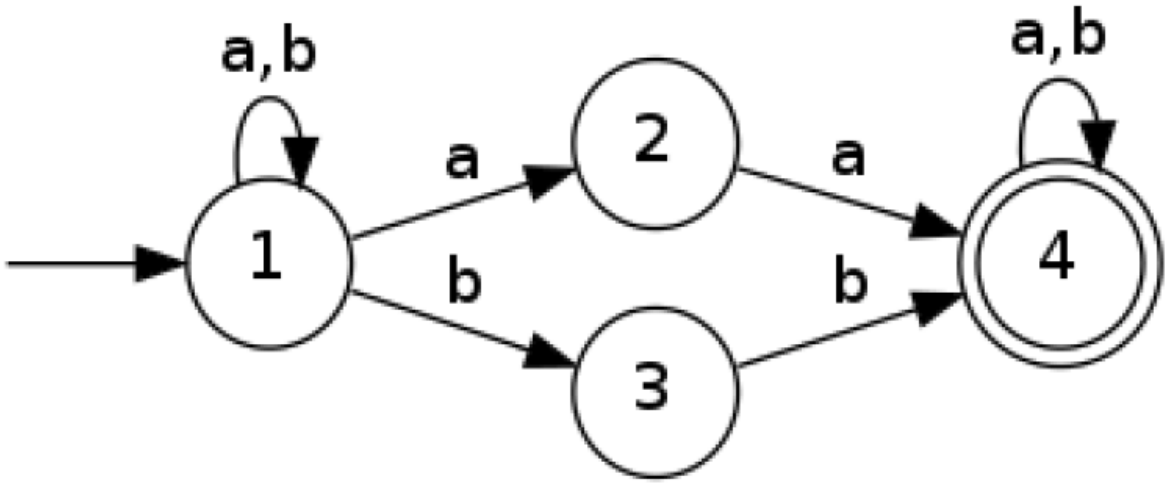
Now we compute $\Lambda$-closure of this set, and add $\{3\}$.

The answer is $\delta^*(0, aab) = \{0, 5, 3\}$.

See also example 3.15 in the textbook

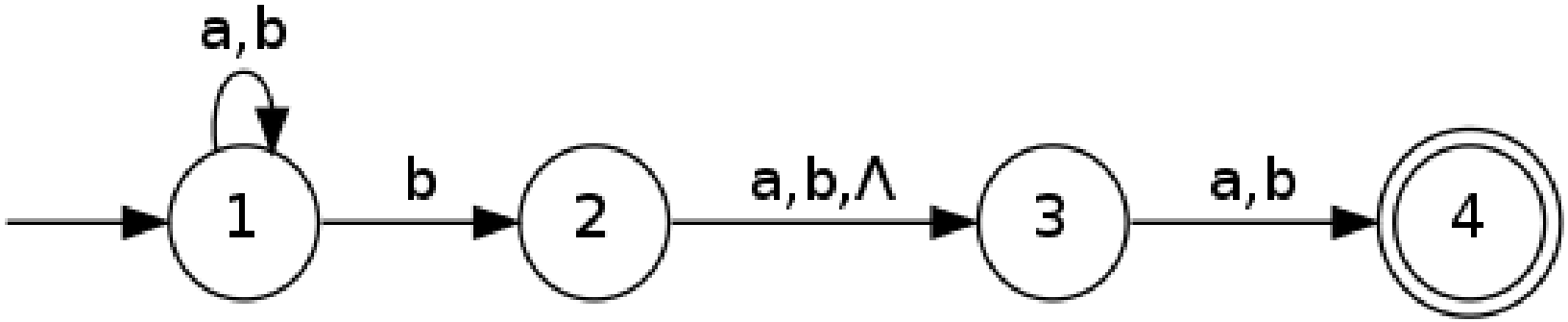
An NFA that accepts strings that contain aa or bb as a substring.

An NFA that accepts strings that contain aa or bb as a substring.



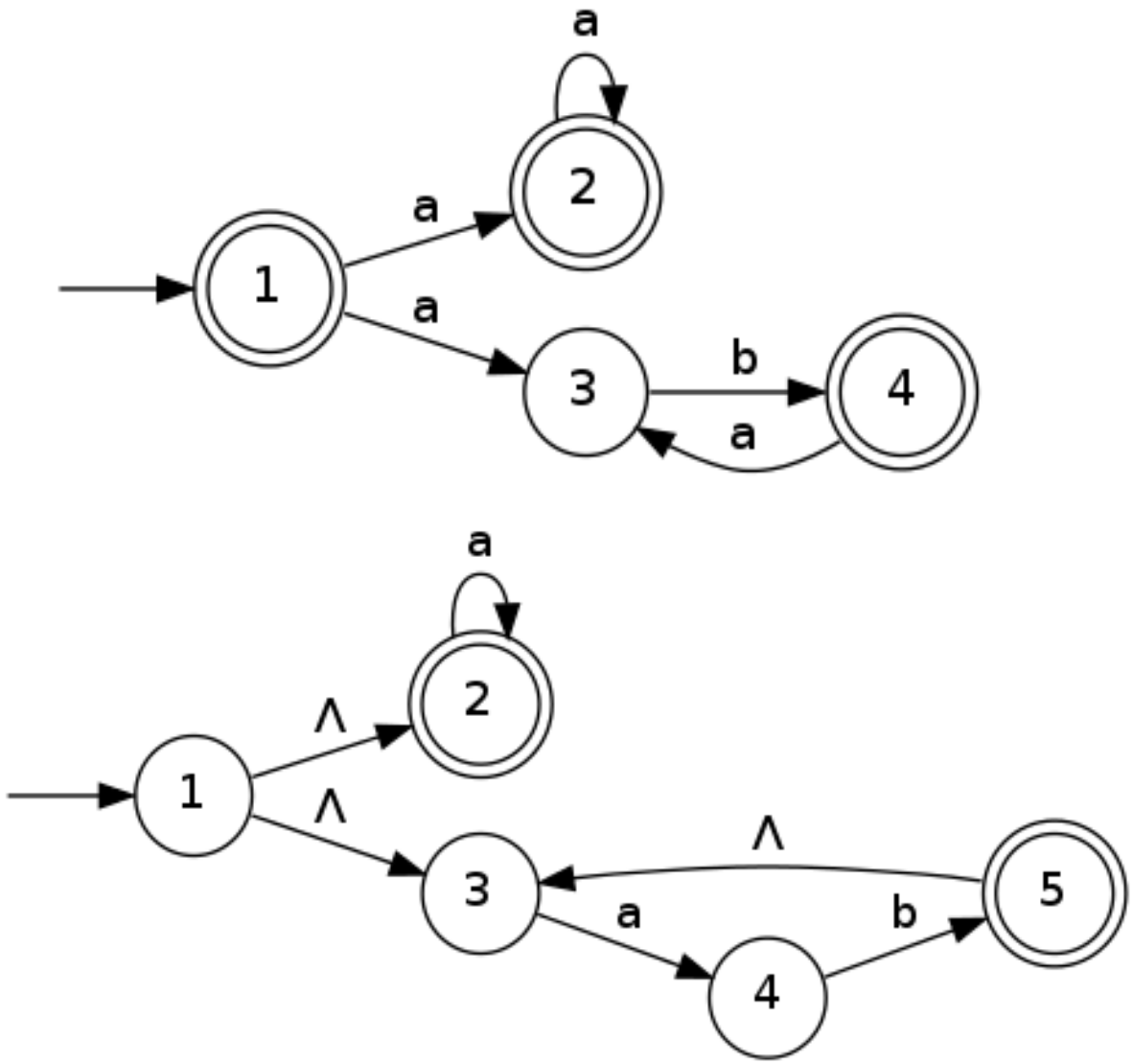
An NFA that accepts strings over {a,b} that contain b either at the third position from the right or at the second position from the right.

An NFA that accepts strings over {a,b} that contain b either at the third position from the right or at the second position from the right.



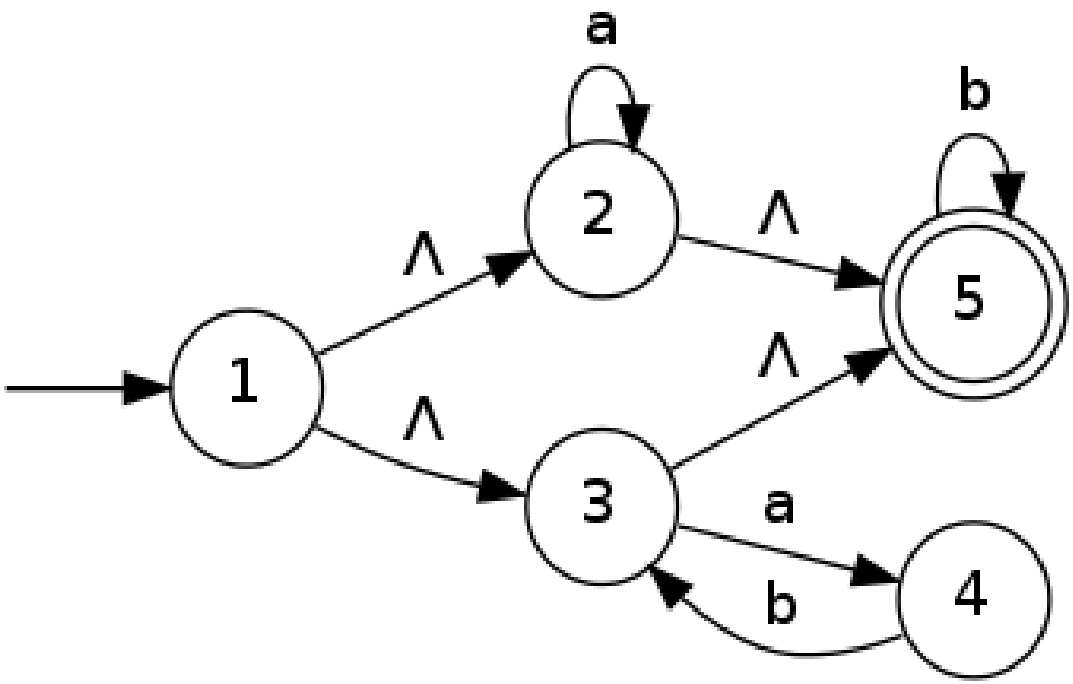
# Simultaneous Pattern: NFA for a*+(ab)*

# Simultaneous Pattern: NFA for a*+(ab)*



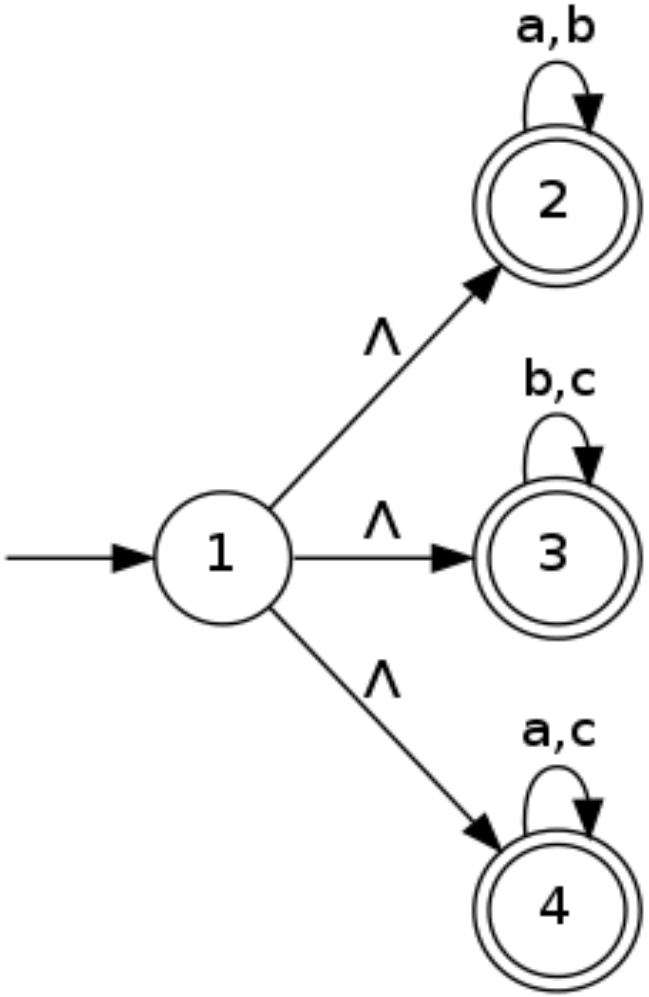
# Simultaneous Pattern: NFA for (a*+(ab)*)b*

# Simultaneous Pattern: NFA for (a*+(ab)*)b*

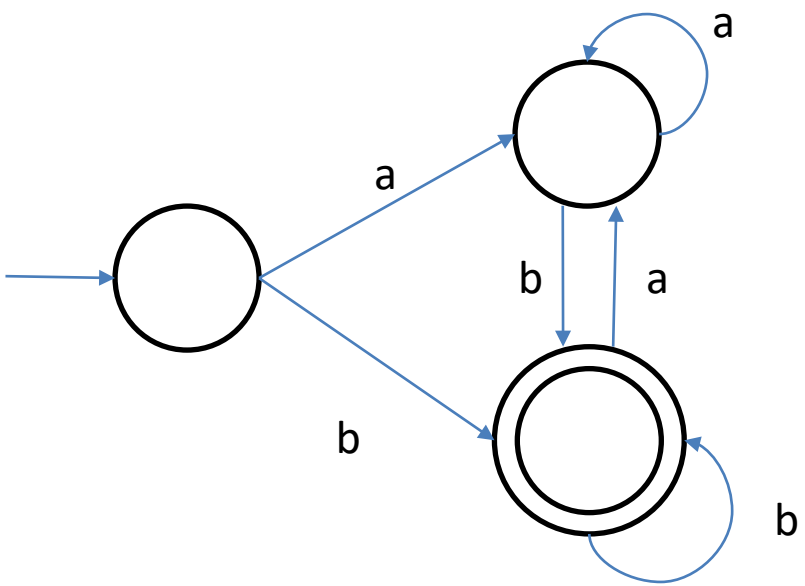


Simultaneous Pattern: NFA for all strings over {a,b,c} that are missing at least one letter. For example: ab,ccccc, bcbcbb, cacaaa

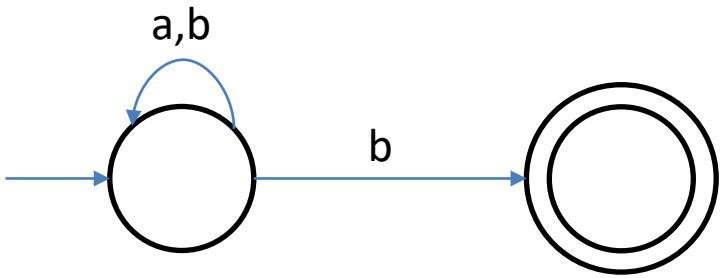
Simultaneous Pattern: NFA for all strings over {a,b,c} that are missing at least one letter. For example: ab,ccccc, bcbcbb, cacaaa



# L = (a+b)*b




FA




NFA

**Claim.** *Let* $M = (Q, \Sigma, q_0, A, \delta)$ *be an NFA. Show that if* $S$ *and* $T$ *are subsets of* $Q$ *for which* $S \subseteq T$ *, then* $\Lambda(S) \subseteq \Lambda(T)$.

**Claim.** *Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. Show that if $S$ and $T$ are subsets of $Q$ for which $S \subseteq T$ , then $\Lambda(S) \subseteq \Lambda(T)$.*

*Proof.* We need to show that if $s \in \Lambda(S)$ then $s \in \Lambda(T)$.

- $\forall s \in S$, it is true that $s \in T$ and then (by def of $\Lambda(T)$) $s \in \Lambda(T)$
- if $s \notin S$ then it was added to $\Lambda(S)$ by the recursive rule

$$\delta(q, \Lambda) \subseteq \Lambda(S) \text{ for some } q \in \Lambda(S)$$

. We prove by induction on the number of recursive applications of $\delta(\cdot, \Lambda)$, i.e., by structural induction.

**Claim.** *Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. Show that if $S$ and $T$ are subsets of $Q$ for which $S \subseteq T$ , then $\Lambda(S) \subseteq \Lambda(T)$.*

*Proof.* We need to show that if $s \in \Lambda(S)$ then $s \in \Lambda(T)$.

- $\forall s \in S$, it is true that $s \in T$ and then (by def of $\Lambda(T)$) $s \in \Lambda(T)$
- if $s \notin S$ then it was added to $\Lambda(S)$ by the recursive rule

$$\delta(q, \Lambda) \subseteq \Lambda(S) \text{ for some } q \in \Lambda(S)$$

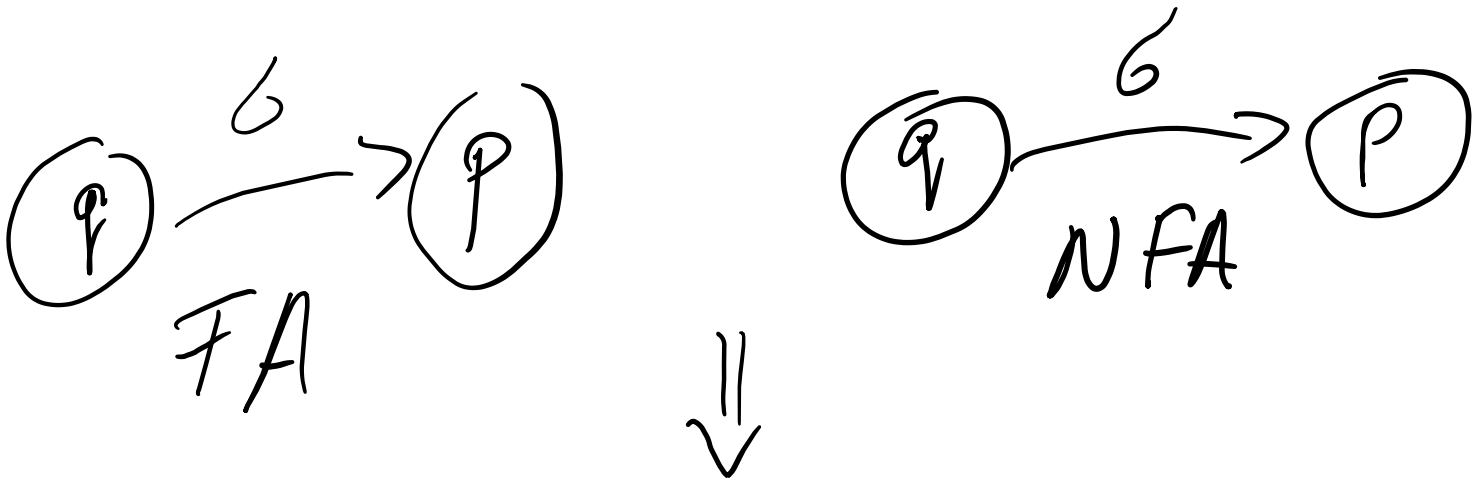
. We prove by induction on the number of recursive applications of $\delta(\cdot, \Lambda)$, i.e., by structural induction.

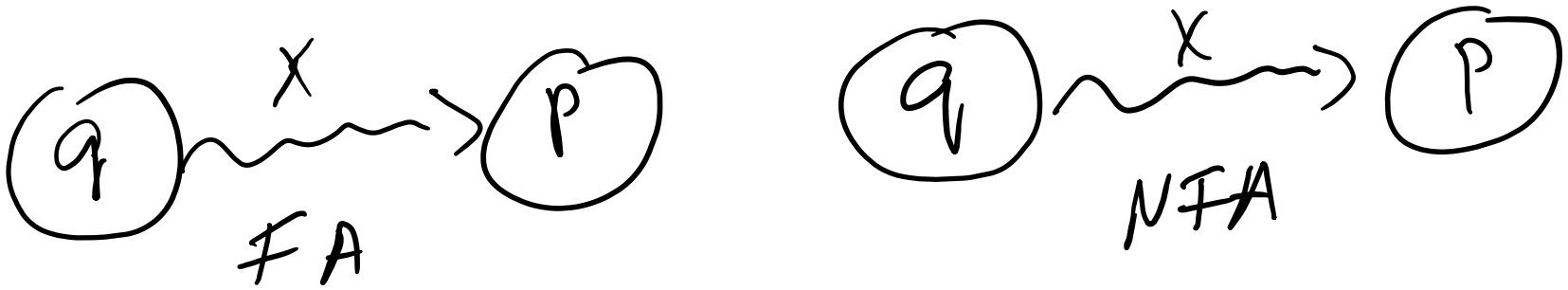
- IH: all elements in $\Lambda(S)$ that were added by less than $k$ applications of $\delta(\cdot, \Lambda)$ are in $\Lambda(T)$
- IS: We add $s$ to $\Lambda(S)$ by $k$th application of $\delta$. W.l.o.g., there exists $q \in \Lambda(S)$ (such that $s \in \delta(q, \Lambda)$) that was added by less than $k$ applications of $\delta$ to $\Lambda(S)$ and by IH $q \in \Lambda(T)$. By def of $\Lambda(T)$ $\delta(q, \Lambda) \subset \Lambda(T)$ because $q \in \Lambda(T)$.

**Claim.** *Let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA, and let $M_1 = (Q, \Sigma, q_0, A, \delta_1)$ be the NFA with no $\Lambda$-transitions for which*

$$\forall q \in Q \ \forall \sigma \in \Sigma \quad \delta_1(q, \sigma) = \{\delta(q, \sigma)\}.$$

*Then* $\forall q \in Q \ \forall \sigma \in \Sigma \quad \delta_1^*(q, x) = \{\delta^*(q, x)\}$.



**Claim.** *Let* $M = (Q, \Sigma, q_0, A, \delta)$ *be an FA, and let* $M_1 = (Q, \Sigma, q_0, A, \delta_1)$ *be the NFA with no* $\Lambda$*-transitions for which*

$$\forall q \in Q \ \forall \sigma \in \Sigma \quad \delta_1(q, \sigma) = \{\delta(q, \sigma)\}.$$

*Then* $\forall q \in Q \ \forall \sigma \in \Sigma \quad \delta_1^*(q, x) = \{\delta^*(q, x)\}.$

*Proof.* The proof is by structural induction.

- Observe there are no $\Lambda$-transitions in $M_1$, i.e., $\delta_1^*(q, \Lambda) = \{q\}$, and $\delta_1^*(q, xa) = \bigcup\{\delta_1(p, a) \mid p \in \delta_1^*(q, x)\}, \ \forall q \in Q, \ \forall a \in \Sigma.$
- BS: $\delta_1^*(q, \Lambda) = \{q\}$, and $\delta^*(q, \Lambda) = q$ by definition of $\delta^*$.
- IH: Suppose that for some $y$, $\delta_1^*(q, y) = \{\delta^*(q, y)\}$, for every $q$.
- IS: Then for $a \in \Sigma$,

$$\begin{aligned} \delta_1(q, ya) &= \cup\{\delta_1(p, a) \mid p \in \delta_1^*(q, y)\} \\ &= \cup\{\delta_1(p, a) \mid p \in \{\delta^*(q, y)\}\} \\ &= \delta_1(\delta^*(q, y), a) \\ &= \{\delta(\delta^*(q, y), a)\} = \{\delta^*(q, ya)\}. \end{aligned}$$

□

**Claim.** *Let* $M = (Q, \Sigma, q_0, A, \delta)$ *be an NFA. For every* $q \in Q$ *and every* $x, y \in \Sigma^*$,

$$\delta^*(q, xy) = \bigcup \{\delta^*(r, y) \mid r \in \delta^*(q, x)\}.$$

Proof by structural induction is given in Exercise 3.30. Learn it!

# The Nondeterminism in an NFA Can Be Eliminated

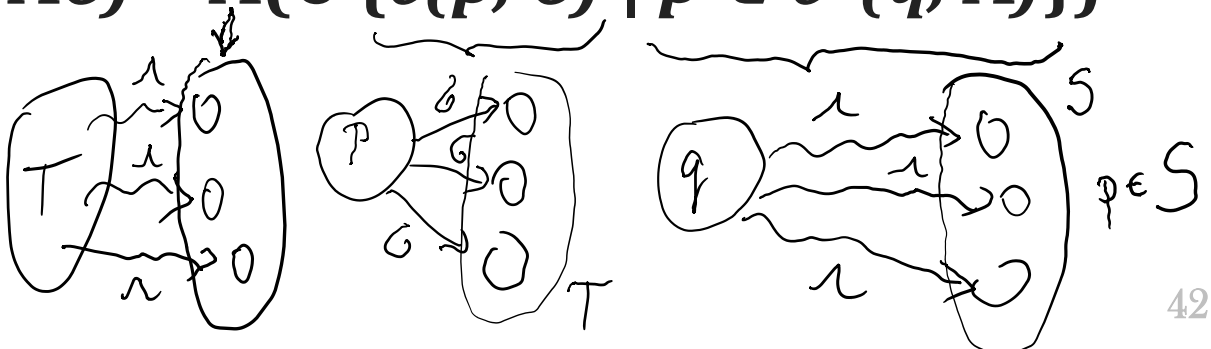
- Two types of nondeterminism have arisen:
  - 1) Different arcs for the same input symbol (or no arcs), and 2) Λ-transitions
  - Both can be eliminated
  - For the **second type**, introduce new transitions so that we no longer need the Λ-transitions
    - When there is no σ-transition from *p* to *q* but the NFA can go from *p* to *q* by using one or more Λ-transitions as well as σ, we introduce the σ-transition
    - The resulting NFA may have more nondeterminism of the first type, but it will have no Λ-transitions

# The Nondeterminism in an NFA Can Be Eliminated (cont'd.)

- **Theorem: For every language $L \subseteq \Sigma^*$ accepted by an NFA $M = (Q, \Sigma, q_0, A, \delta)$, there is an NFA $M_1$ with no $\Lambda$-transitions that also accepts $L$**
- Define $M_1 = (Q, \Sigma, q_0, A_1, \delta_1)$, where
  - for every $q \in Q$, $\delta_1(q, \Lambda) = \emptyset$, and
  - for every $q \in Q$ and every $\sigma \in \Sigma$, $\delta_1(q, \sigma) = \delta^*(q, \sigma)$

original definition of $\delta^*$ → $\delta^*(q, y\sigma) = \Lambda(\cup \{\delta(p, \sigma) \mid p \in \delta^*(q, y)\})$

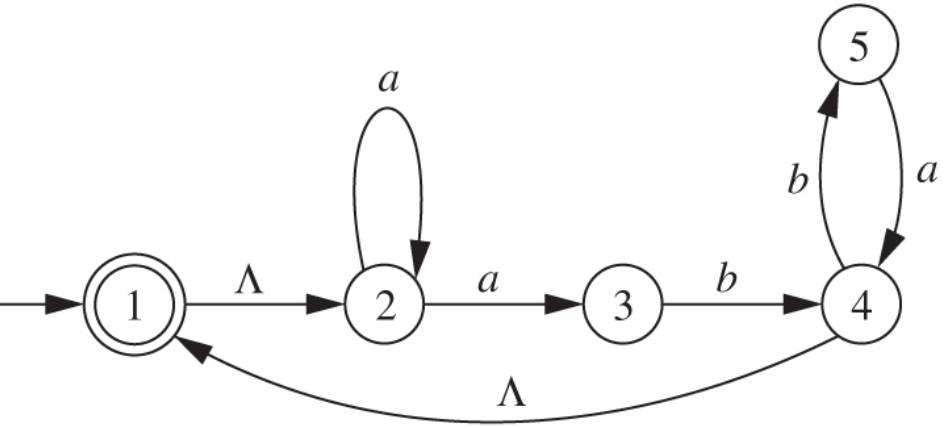
here we need y= $\Lambda$ → $\delta^*(q, \Lambda\sigma) = \Lambda(\cup \{\delta(p, \sigma) \mid p \in \delta^*(q, \Lambda)\})$



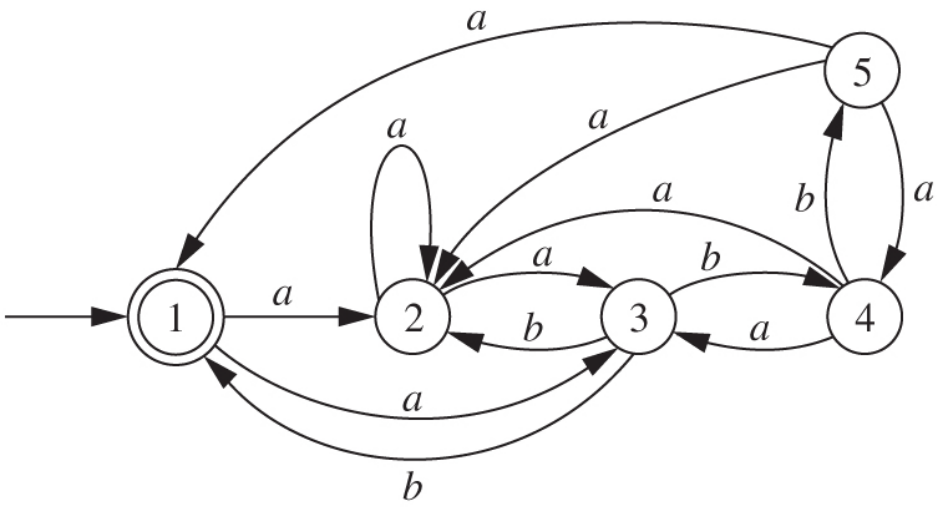
# The Nondeterminism in an NFA Can Be Eliminated (cont'd.)

- **Theorem: For every language $L \subseteq \Sigma^*$ accepted by an NFA $M = (Q, \Sigma, q_0, A, \delta)$, there is an NFA $M_1$ with no $\Lambda$-transitions that also accepts $L$**
- Define $M_1 = (Q, \Sigma, q_0, A_1, \delta_1)$, where
  - for every $q \in Q$, $\delta_1(q, \Lambda) = \emptyset$, and
  - for every $q \in Q$ and every $\sigma \in \Sigma$, $\delta_1(q, \sigma) = \delta^*(q, \sigma)$
- Define $A_1 = A \cup \{q_0\}$ if $\Lambda \in L$, and $A_1 = A$ otherwise
- We can prove, by structural induction on $x$, that for every $q$ and every $x$ with $|x| \geq 1$, $\delta_1{}^*(q, x) = \delta^*(q, x)$

Homework: prove this theorem (see Theorem 3.17 in the textbook)




(a)



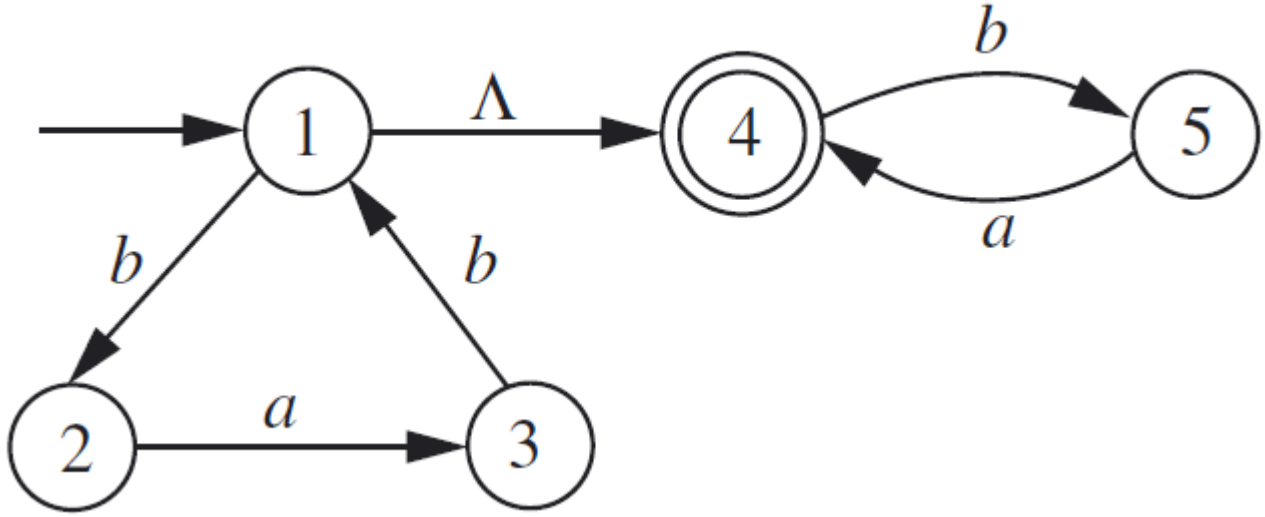

(b)

# Example: Λ-transition elimination

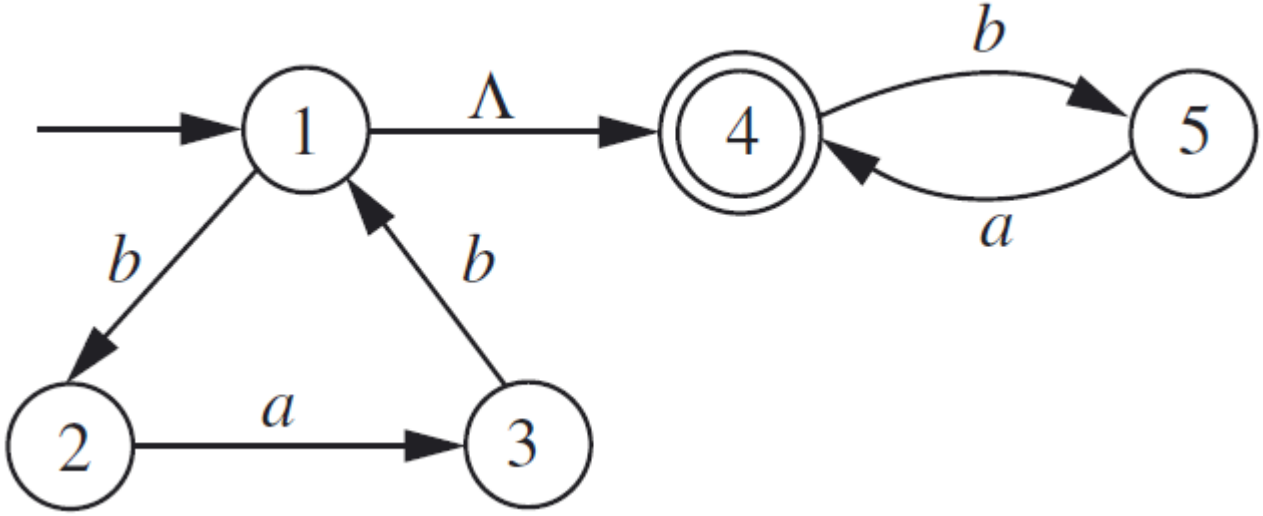
means 5 will be connected to 1, 2, and 4

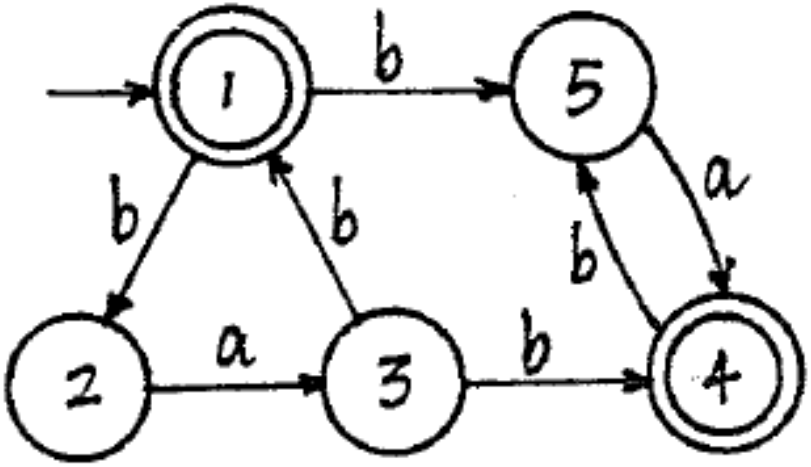
| $q$ | $\delta(q, a)$ | $\delta(q, b)$ | $\delta(q, \Lambda)$ | $\delta^*(q, a)$ | $\delta^*(q, b)$ |
|---|---|---|---|---|---|
| 1 | $\emptyset$ | $\emptyset$ | {2} | {2, 3} | $\emptyset$ |
| 2 | {2, 3} | $\emptyset$ | $\emptyset$ | {2, 3} | $\emptyset$ |
| 3 | $\emptyset$ | {4} | $\emptyset$ | $\emptyset$ | {1, 2, 4} |
| 4 | $\emptyset$ | {5} | {1} | {2, 3} | {5} |
| 5 | {4} | $\emptyset$ | $\emptyset$ | {1, 2, 4} | $\emptyset$ |

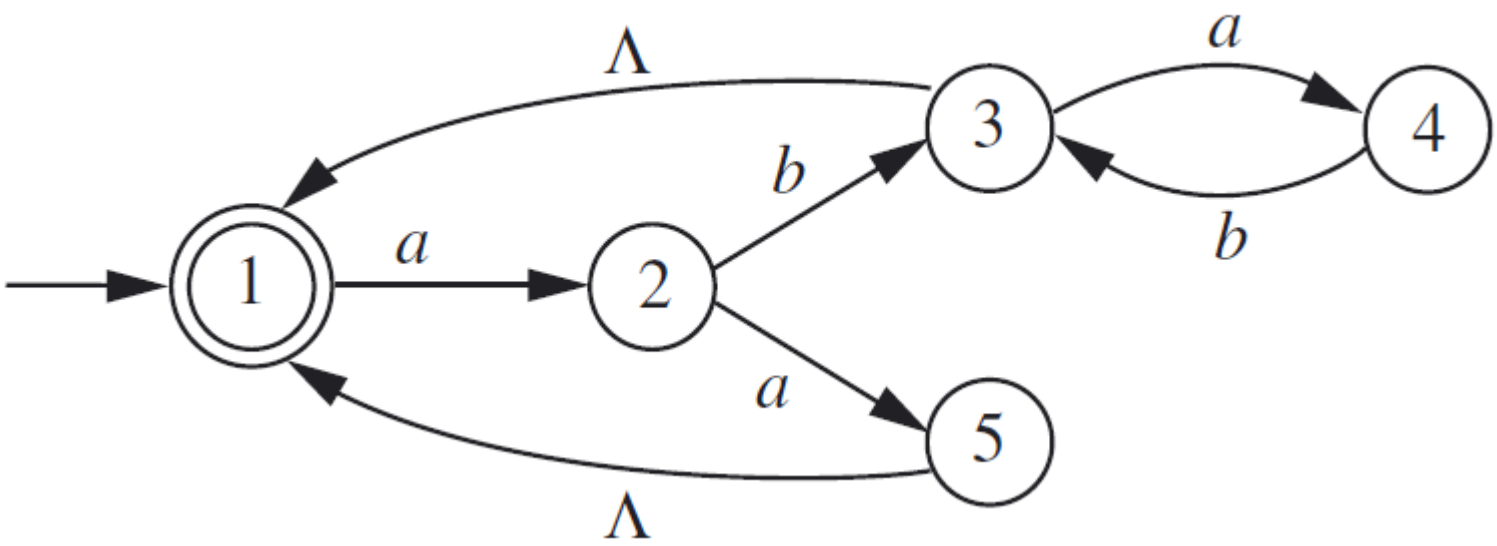
# Eliminate Lambda-transition

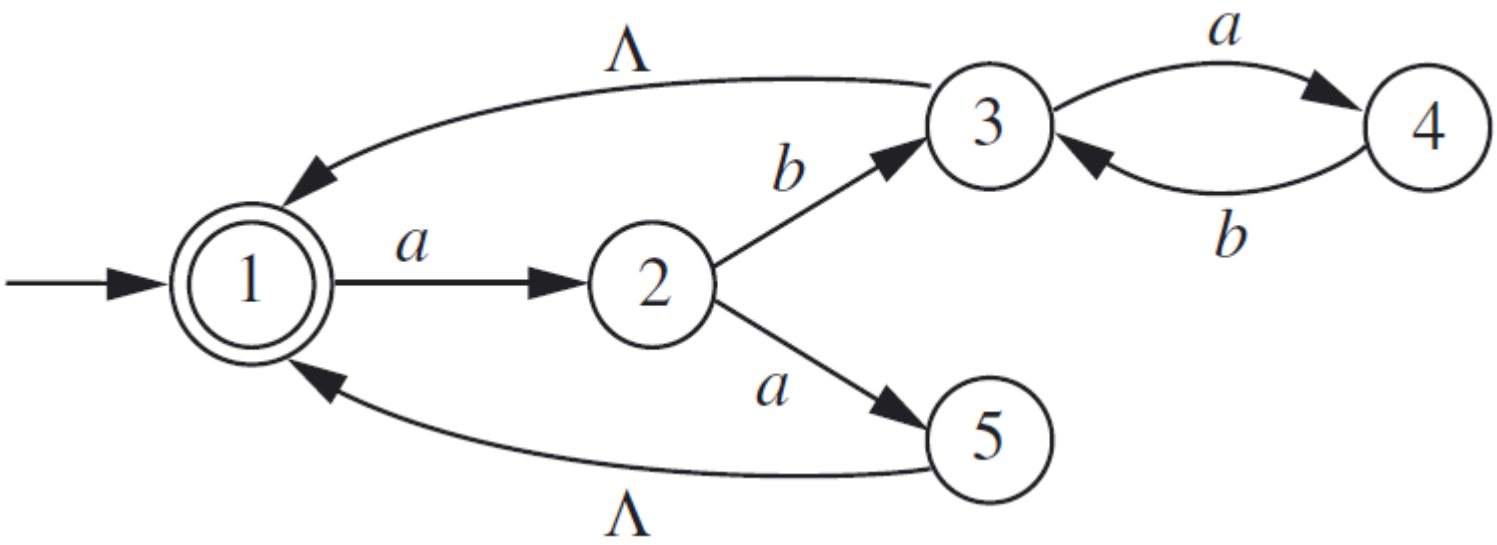


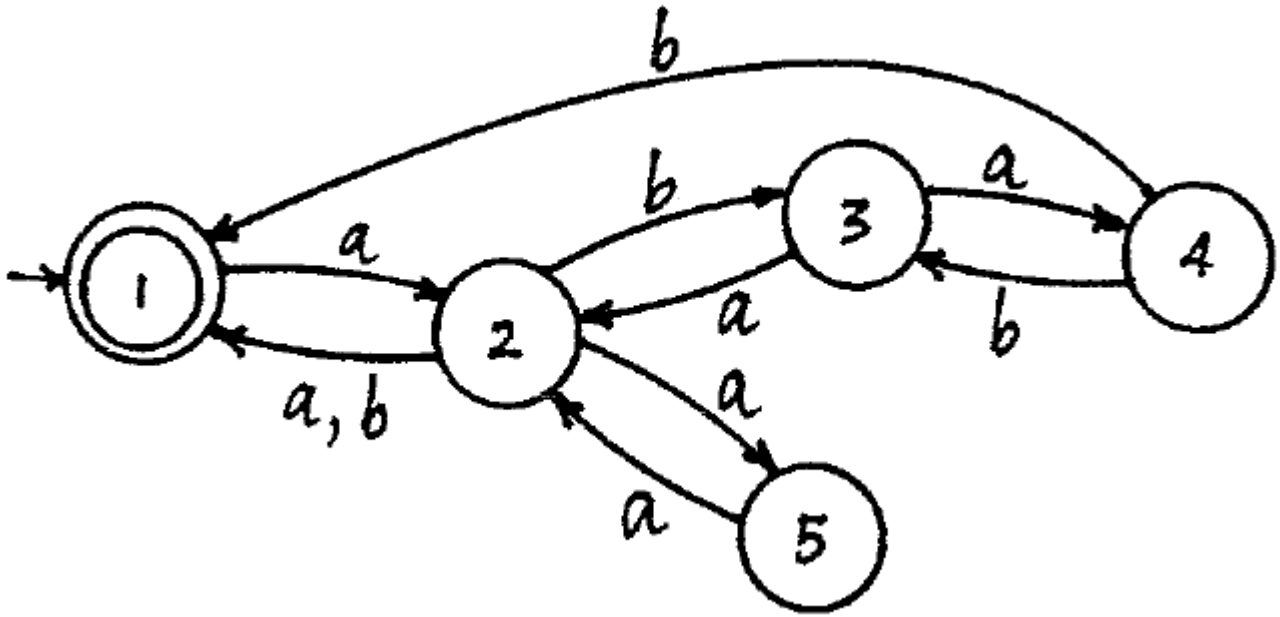
# Eliminate Lambda-transition




Λ
a
3
4
b
b
a
1
2
5
Λ


Λ
a
3
4
b
b
1
a
2
a
5
Λ

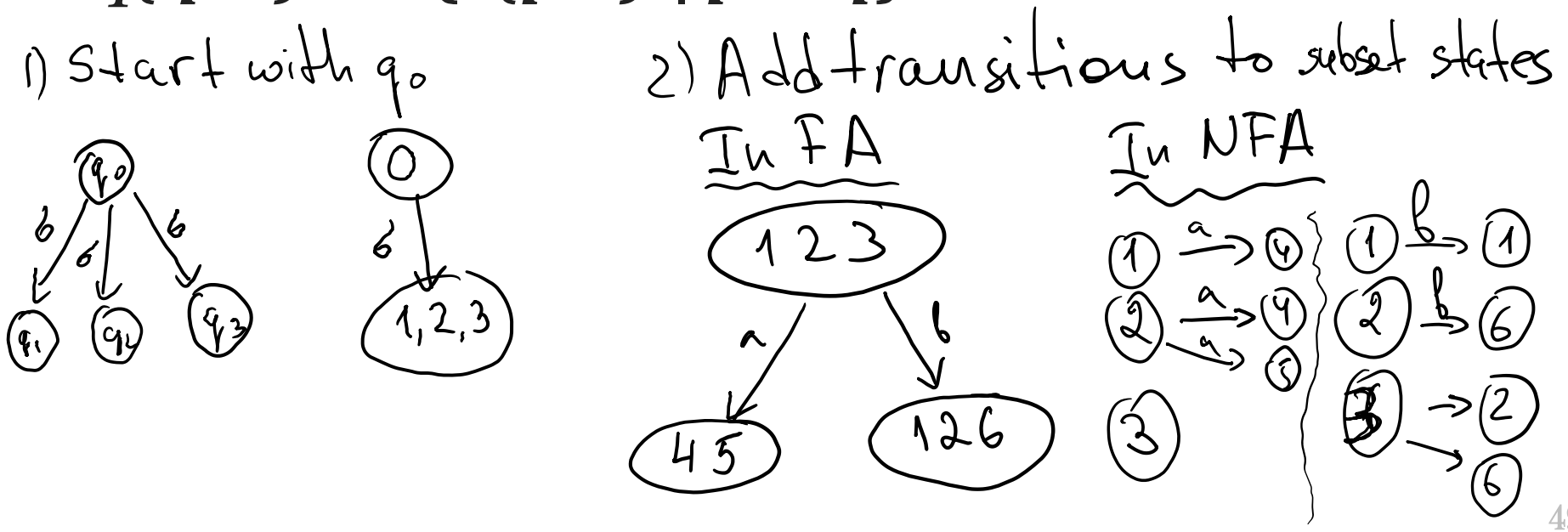

b
b
a
3
a
4
1
a
2
a
b
a, b
a
a
5

# The Nondeterminism in an NFA Can Be Eliminated

- **Theorem: For every language $L \subseteq \Sigma^*$ accepted by an NFA $M = (Q, \Sigma, q_0, A, \delta)$, there is an FA $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ that also accepts $L$**
- We can assume M has no $\Lambda$-transitions. Let $Q_1 = 2^Q$ (for this reason, this is called the *subset construction*); $q_1 = \{q_0\}$; for every $q \in Q_1$ and $\sigma \in \Sigma$,

subset of Q

$$\delta_1(q, \sigma) = \cup\{\delta(p,\sigma) \mid p \in q\} \ldots$$



# The Nondeterminism in an NFA Can Be Eliminated

- **Theorem: For every language $L \subseteq \Sigma^*$ accepted by an NFA $M = (Q, \Sigma, q_0, A, \delta)$, there is an FA $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ that also accepts $L$**
- We can assume M has no $\Lambda$-transitions. Let $Q_1 = 2^Q$ (for this reason, this is called the *subset construction*); $q_1 = \{q_0\}$; for every $q \in Q_1$ and $\sigma \in \Sigma$,

subset of Q

$$\delta_1(q, \sigma) = \cup\{\delta(p,\sigma) \mid p \in q\}$$

$$A_1 = \{q \in Q_1 \mid q \cap A \neq \varnothing\}$$

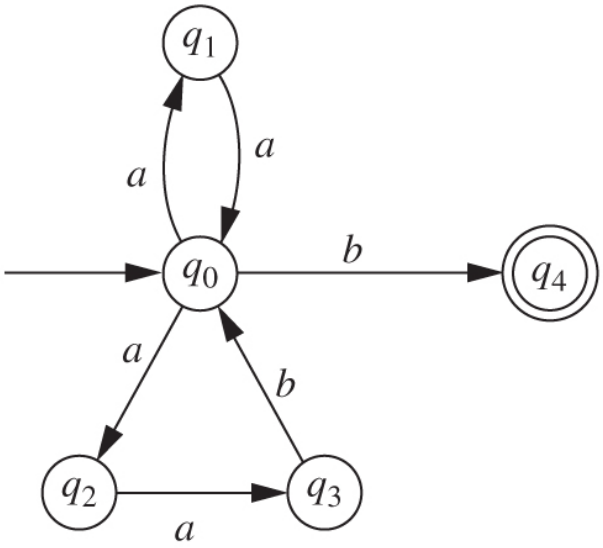
- $M_1$ is clearly an FA
  - It accepts the same language as $M$ because for every $x \in \Sigma^*$, $\delta_1^*(q_1, x) = \delta^*(q_0, x)$
- The proof is by structural induction on $x$

Homework: prove this theorem (see Thm 3.18 in the textbook)

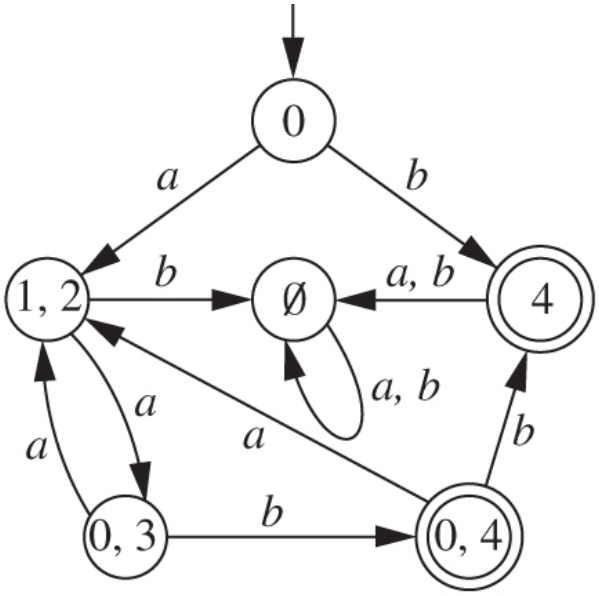
# Example: Subset construction to eliminate nondeterminism

$M = (Q, \Sigma, q_0, A, \delta)$ ⟶ $M_1 = (2^Q, \Sigma, \{q_0\}, A_1, \delta_1)$

NFA to accept *{aa,aab}*{b}*

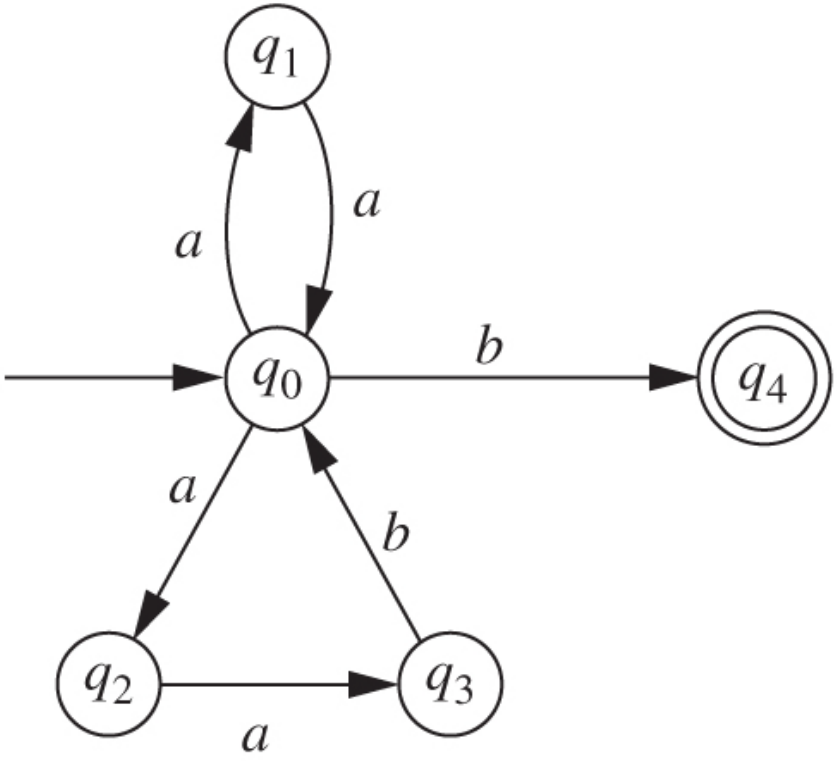


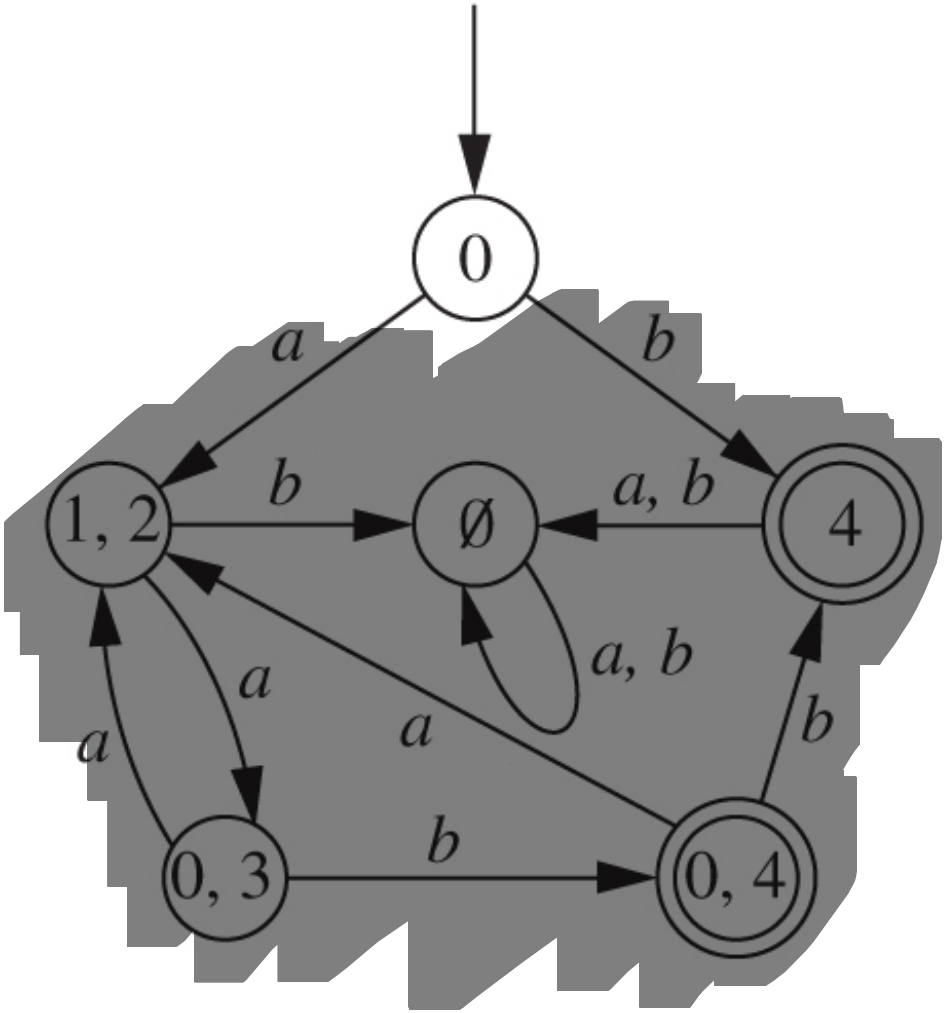

FA to accept *{aa,aab}*{b}*



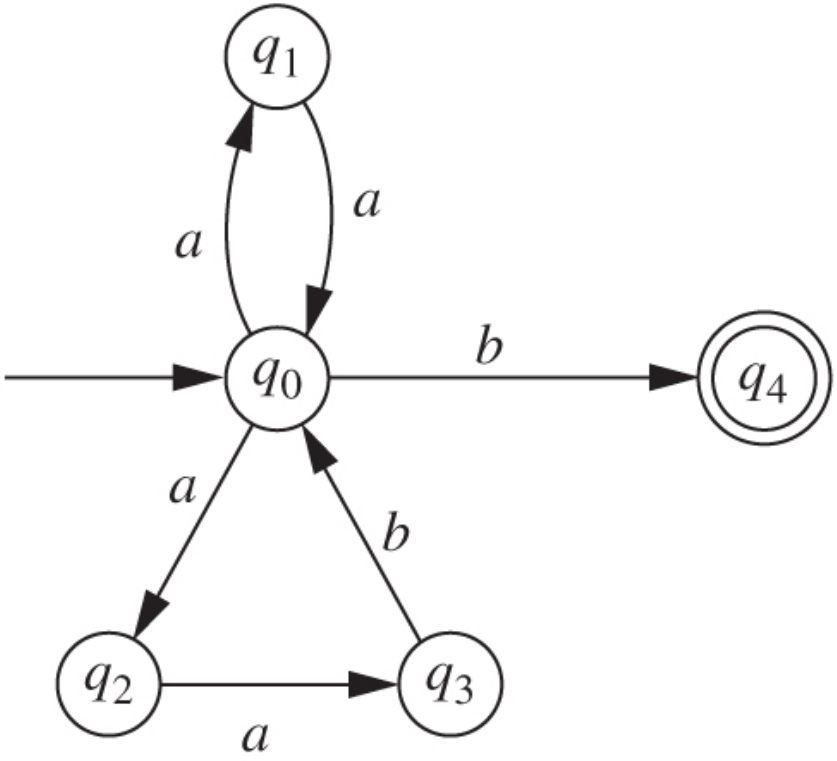

- No need to generate $2^n$ subsets; consider only reachable states
- It is recommended to use a transition table
- Example: $\delta_1(\{1,2\},a) = \delta(1,a) \cup \delta(2,a) = \{0,3\}$
- All reachable states that contain elements from A are in $A_1$

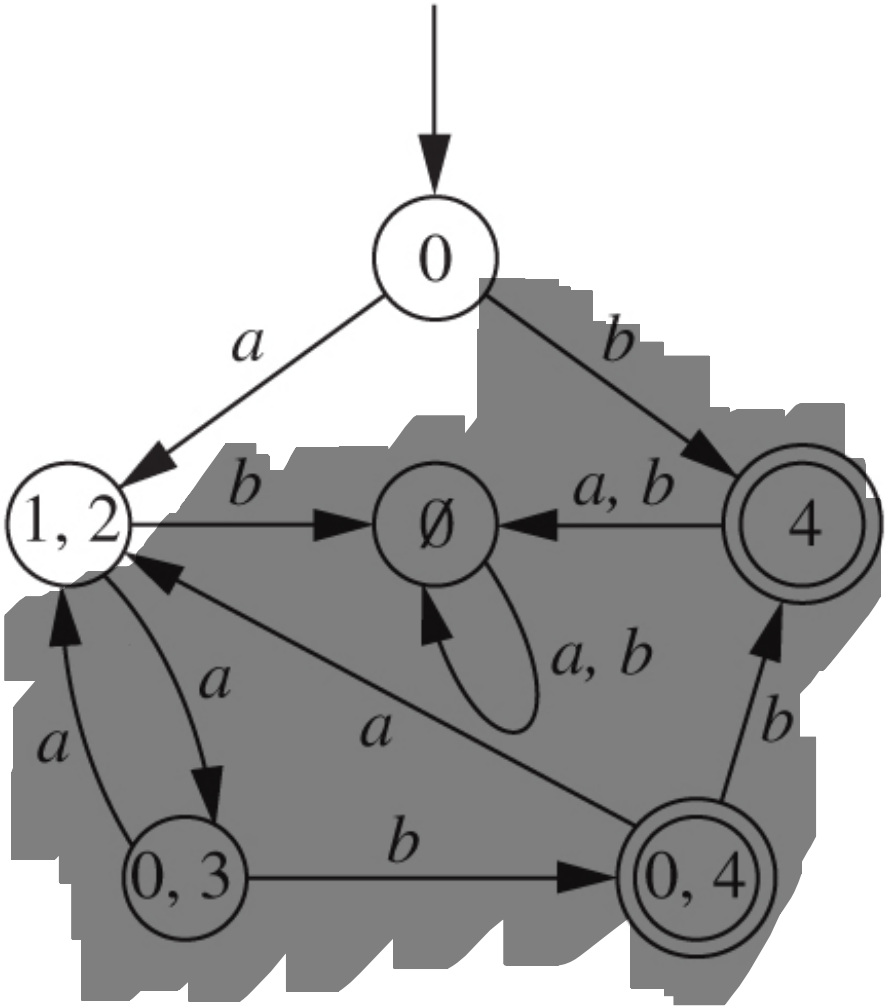
| $q$ | $\delta(q,a)$ | $\delta(q,b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | Ø |
| 2 | {3} | Ø |
| 3 | Ø | {0} |
| 4 | Ø | Ø |



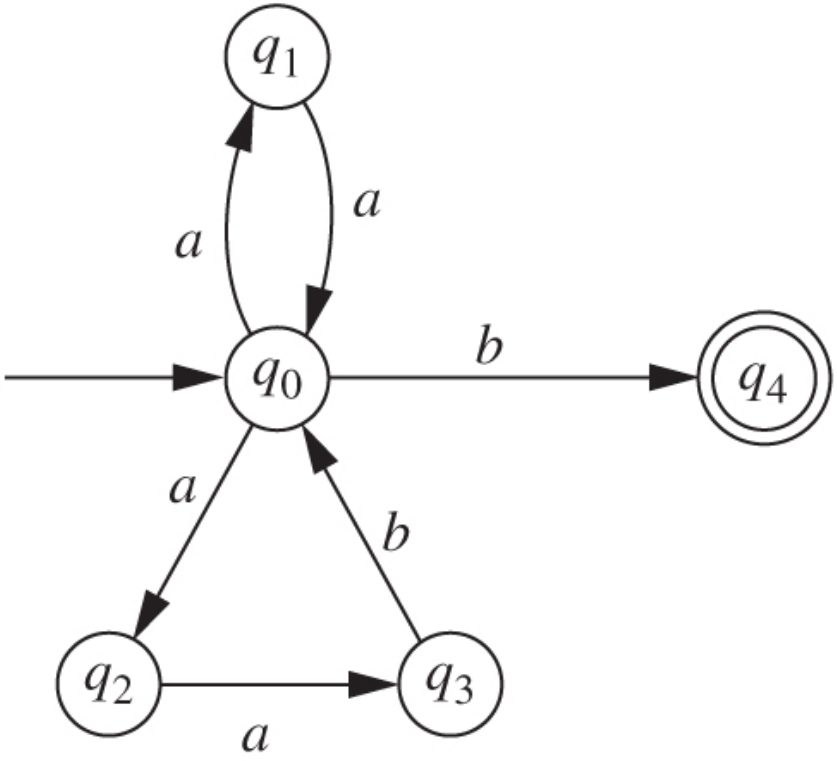



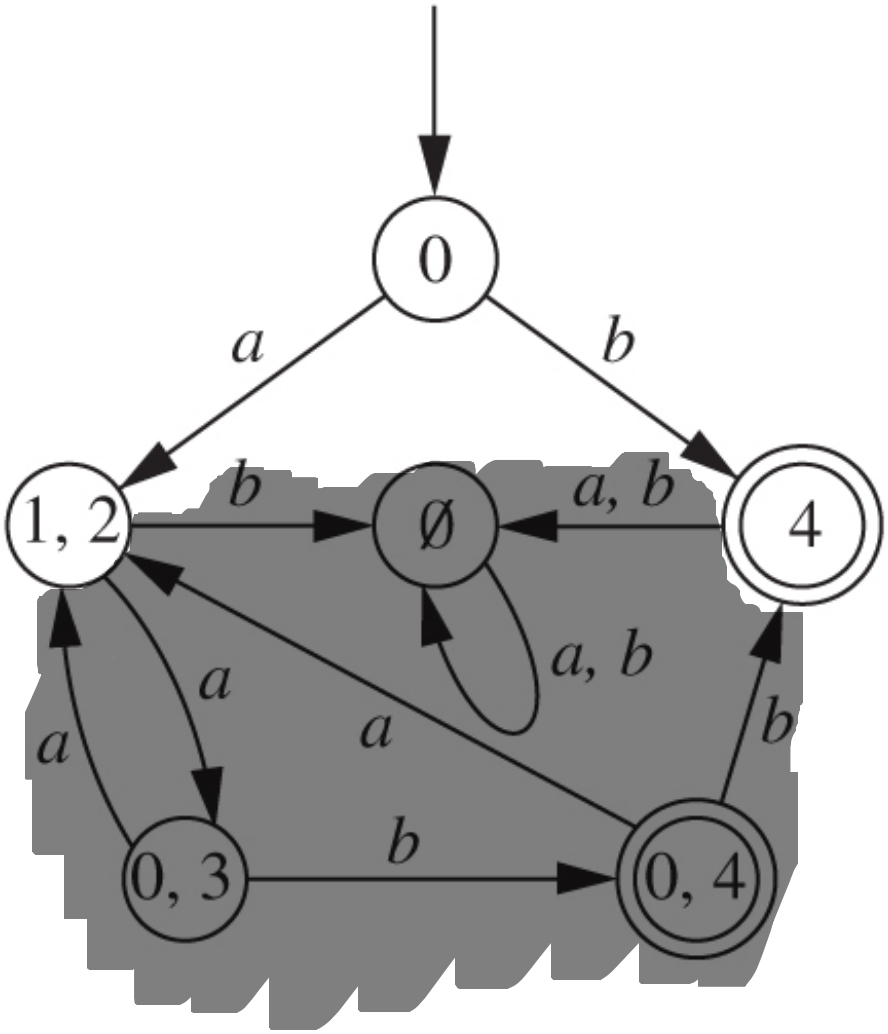

| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | ∅ |
| 2 | {3} | ∅ |
| 3 | ∅ | {0} |
| 4 | ∅ | ∅ |



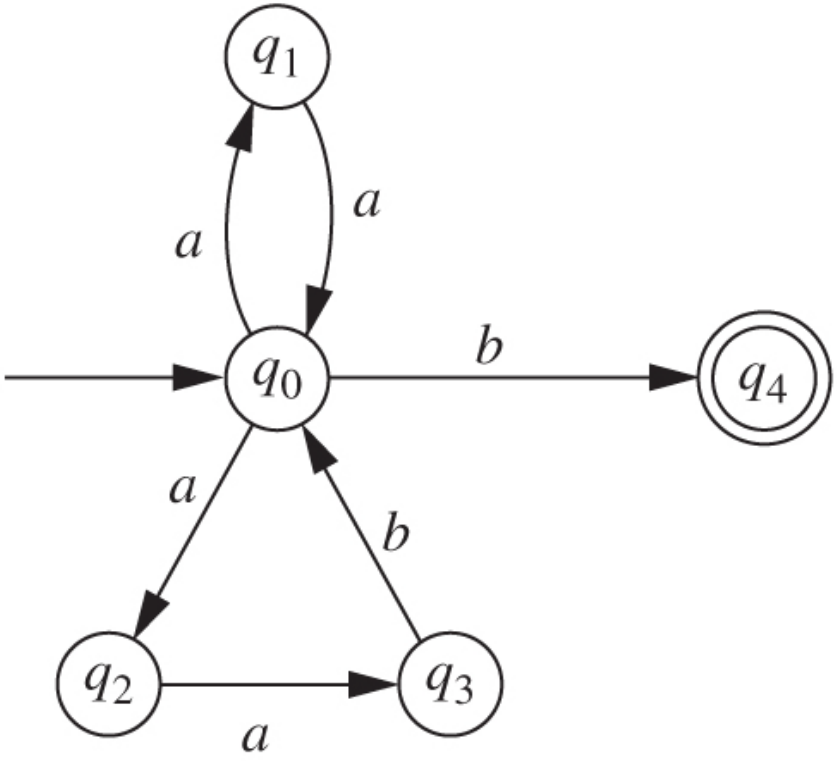



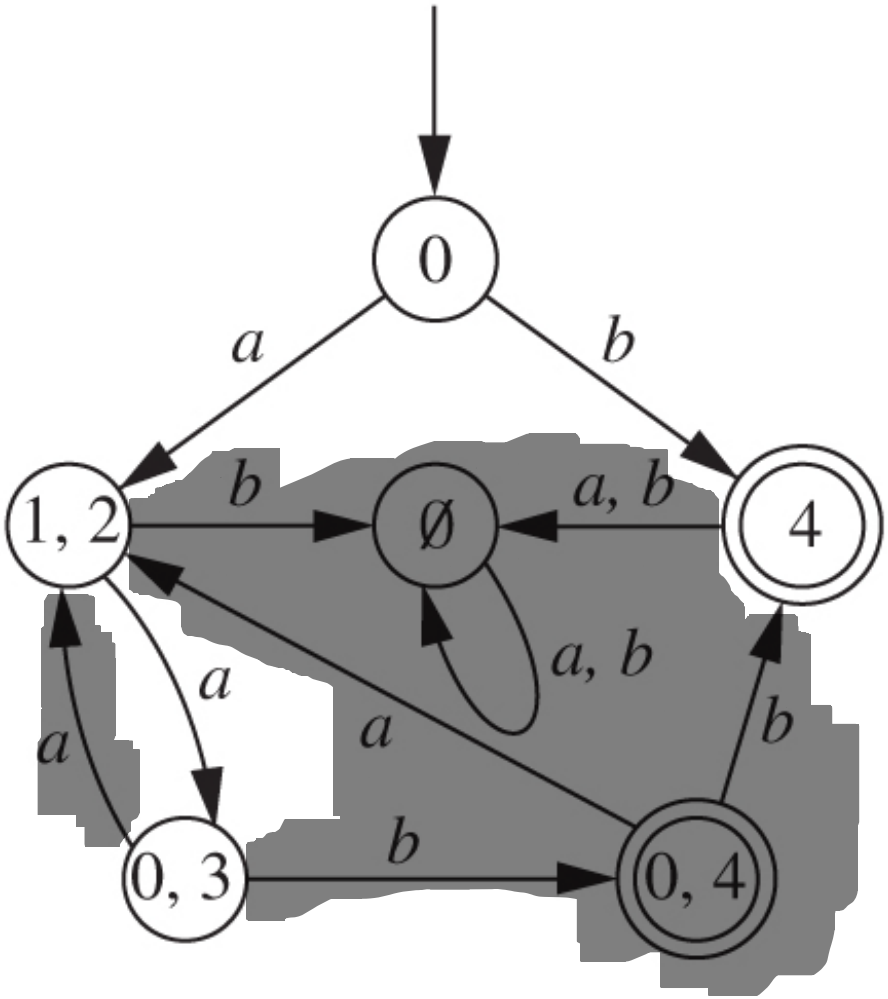

| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | ∅ |
| 2 | {3} | ∅ |
| 3 | ∅ | {0} |
| 4 | ∅ | ∅ |



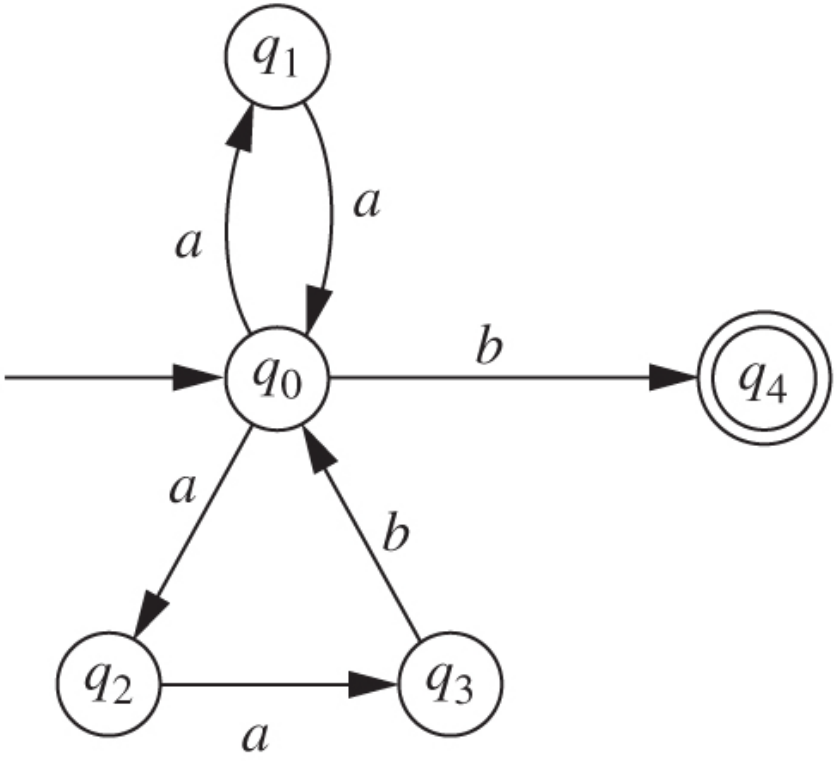



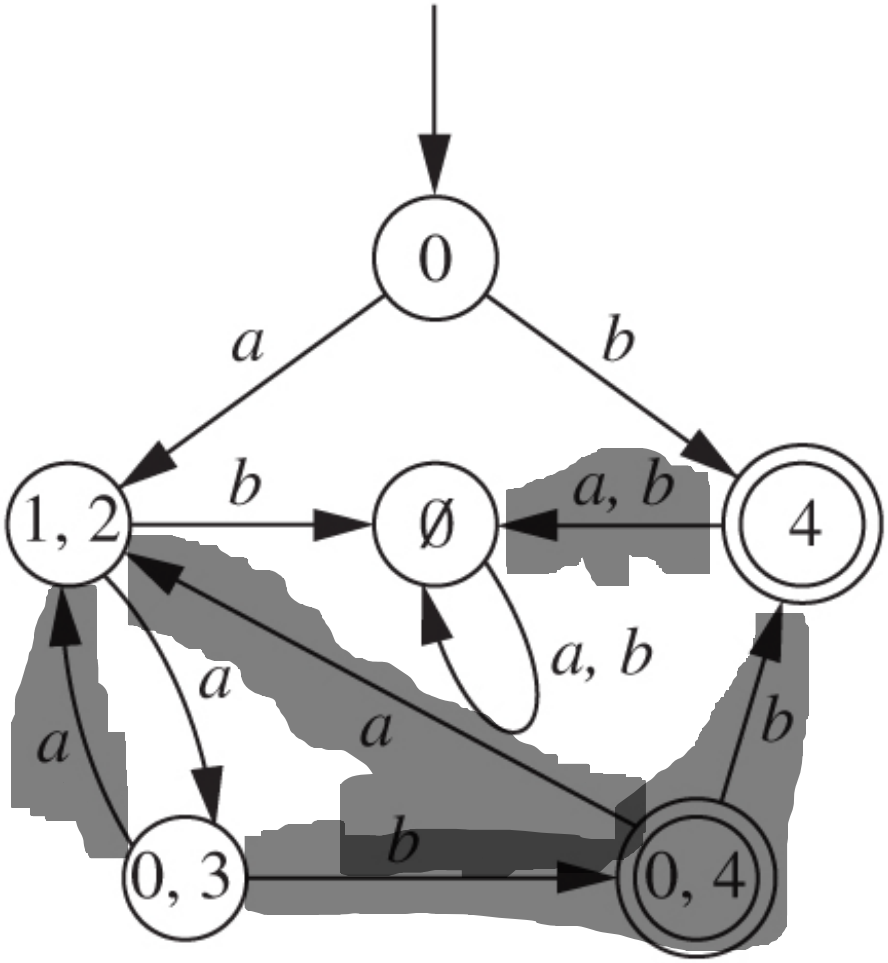

| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | ∅ |
| 2 | {3} | ∅ |
| 3 | ∅ | {0} |
| 4 | ∅ | ∅ |



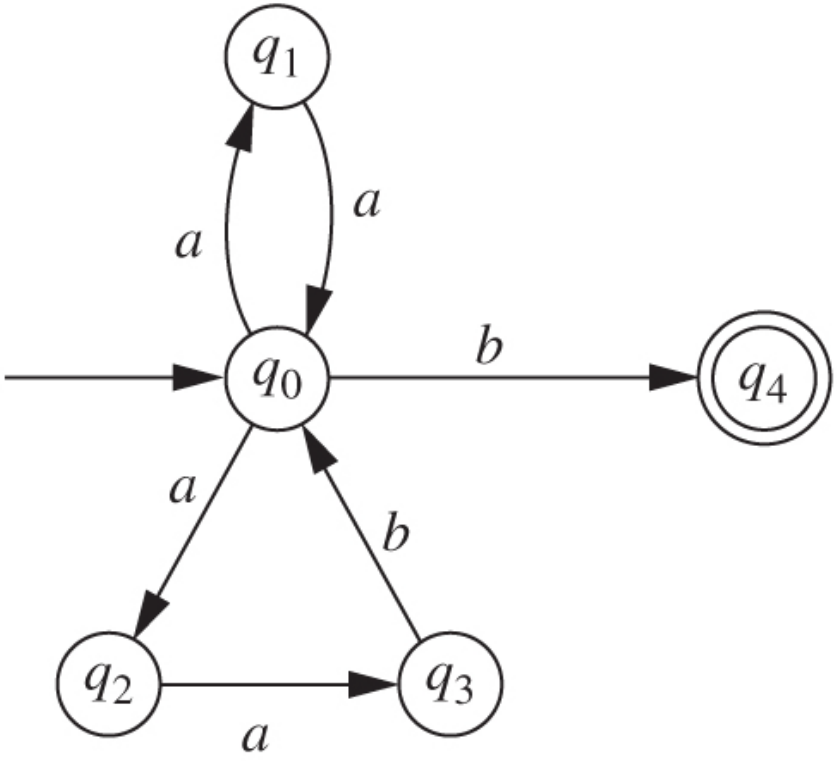



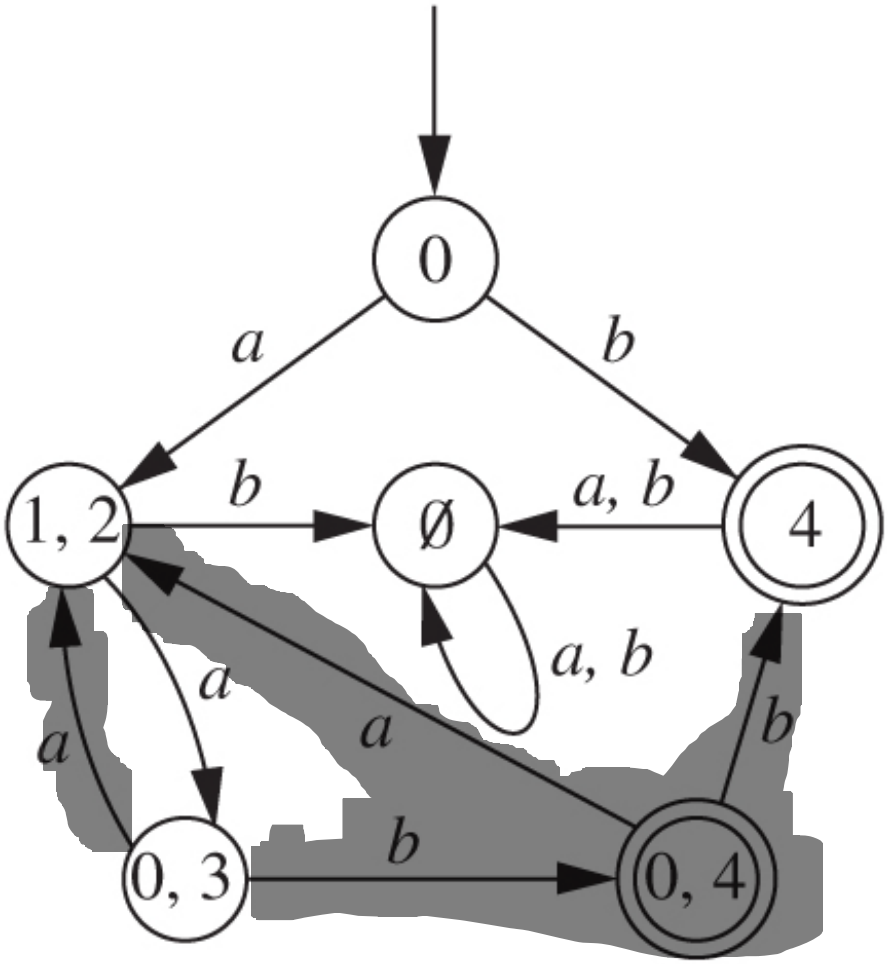

| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | ∅ |
| 2 | {3} | ∅ |
| 3 | ∅ | {0} |
| 4 | ∅ | ∅ |



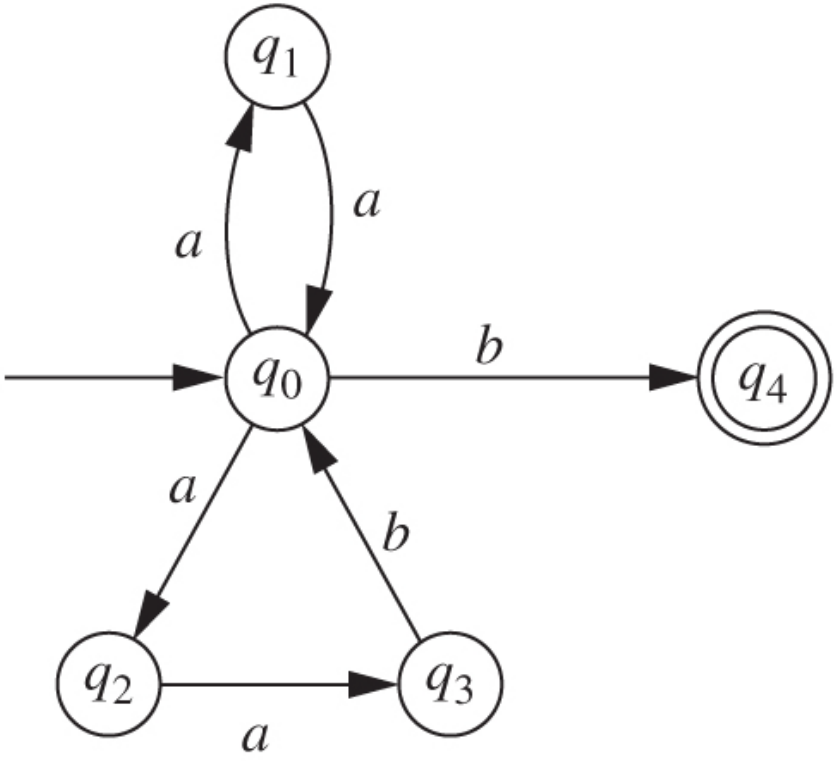



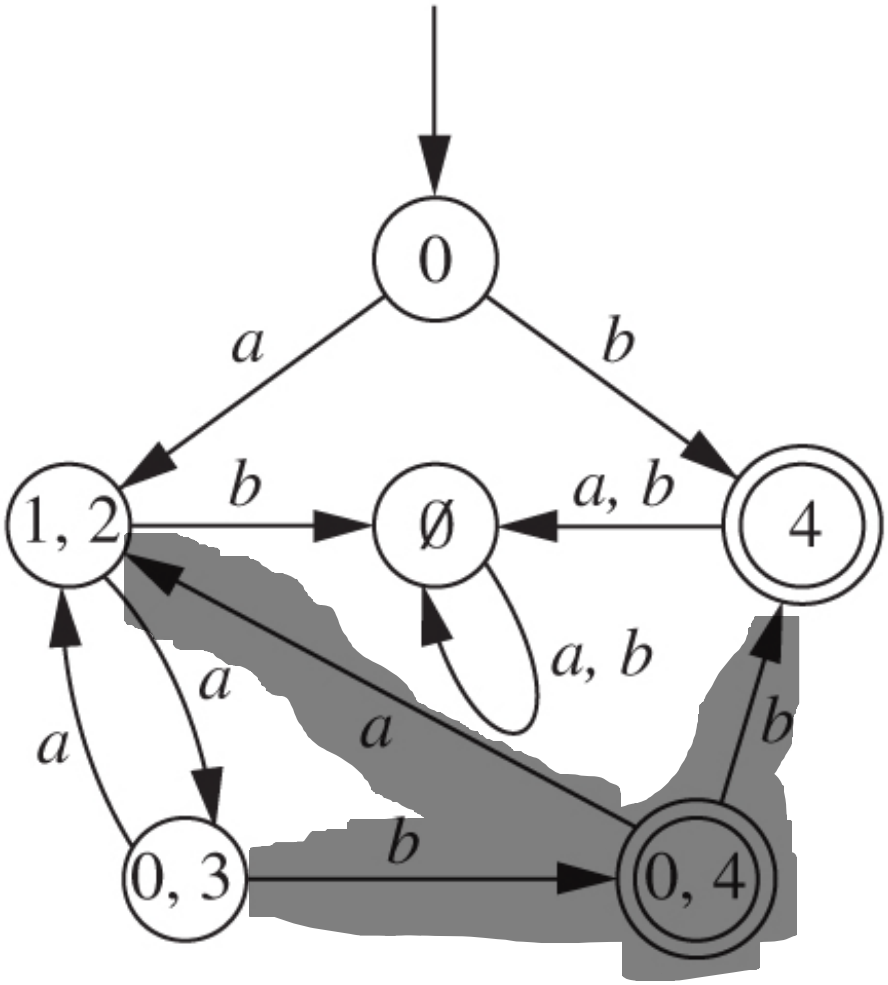

| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | ∅ |
| 2 | {3} | ∅ |
| 3 | ∅ | {0} |
| 4 | ∅ | ∅ |



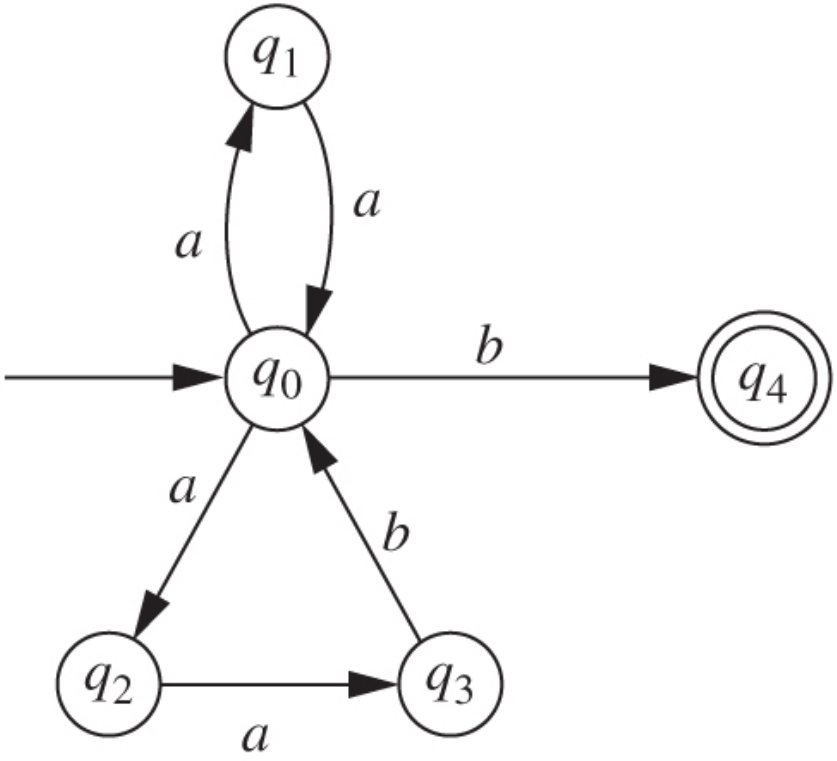



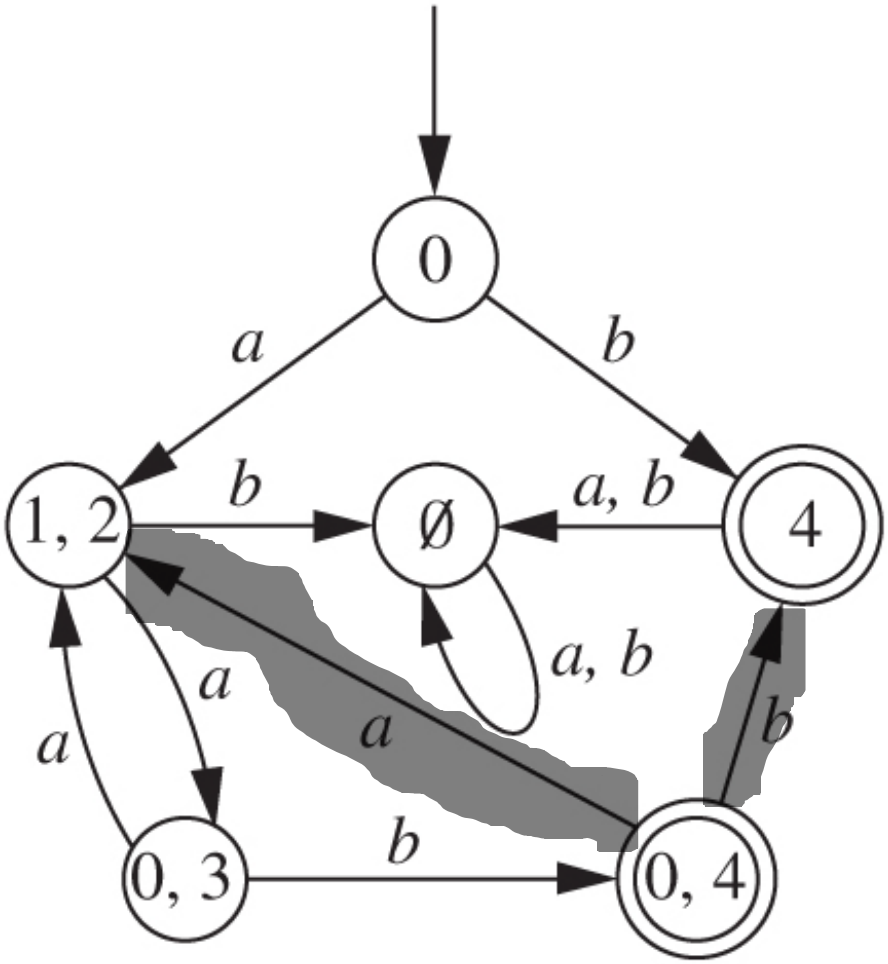

| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | ∅ |
| 2 | {3} | ∅ |
| 3 | ∅ | {0} |
| 4 | ∅ | ∅ |



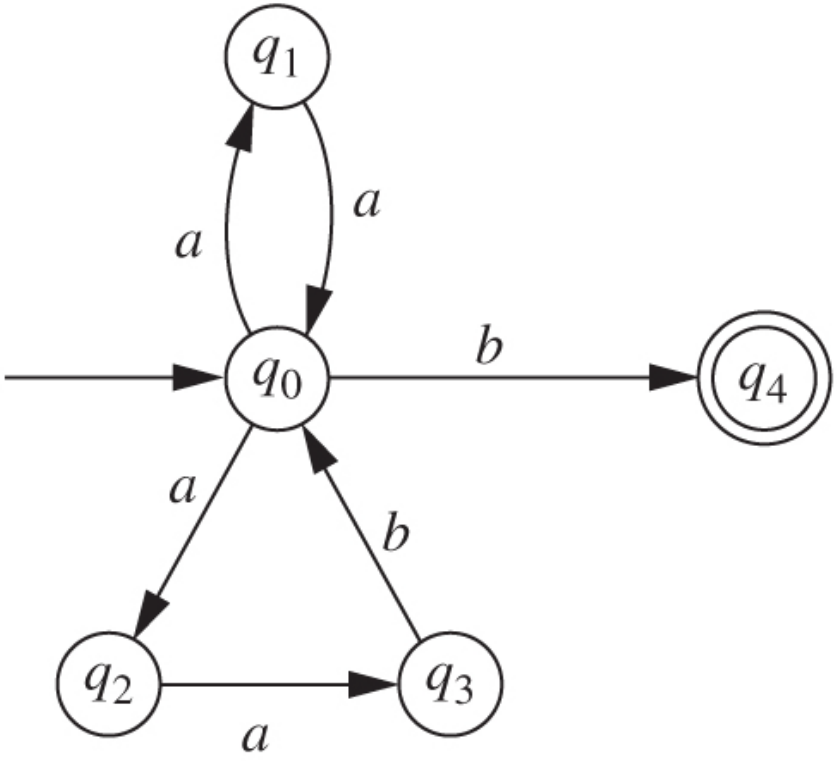



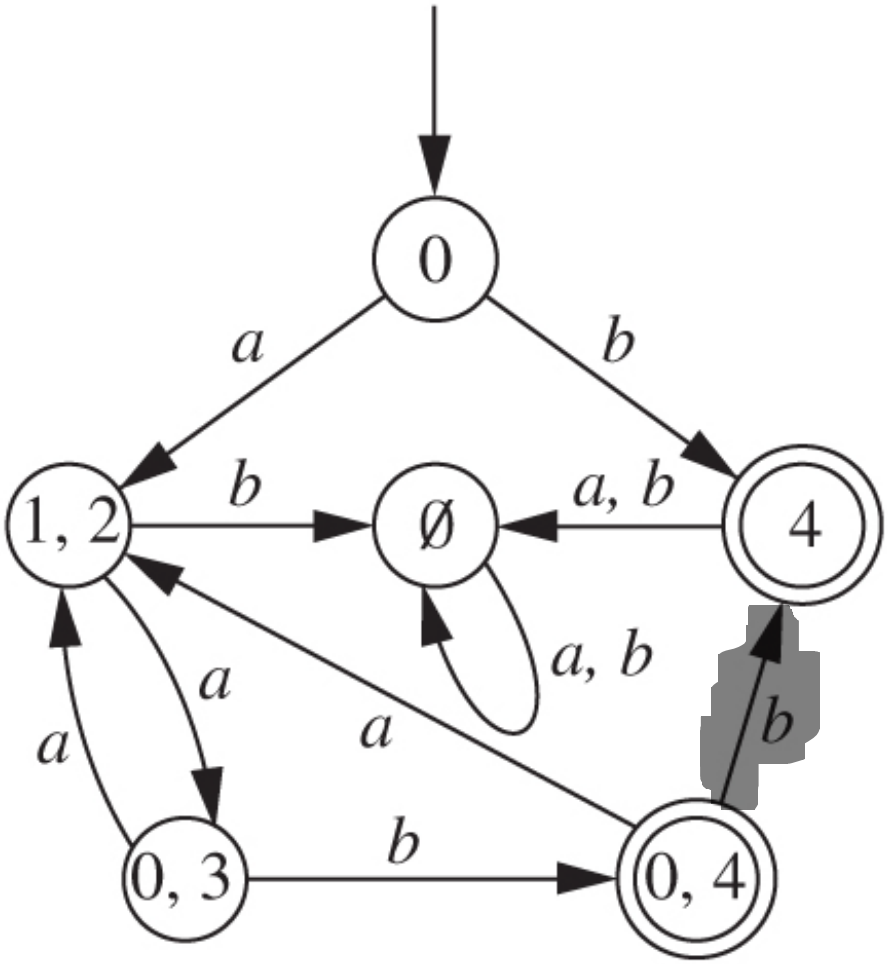

| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | Ø |
| 2 | {3} | Ø |
| 3 | Ø | {0} |
| 4 | Ø | Ø |



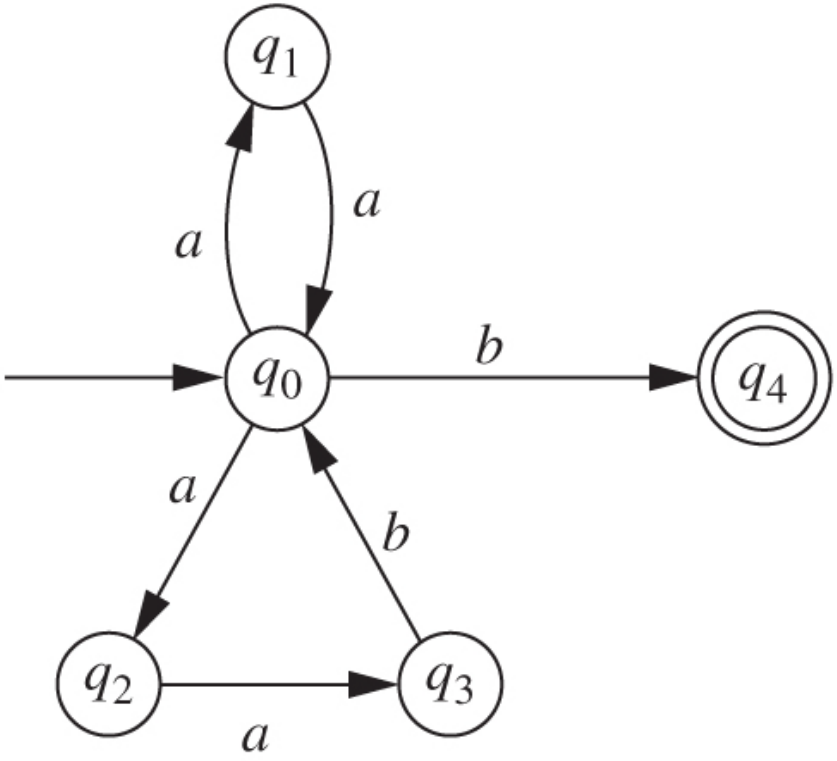



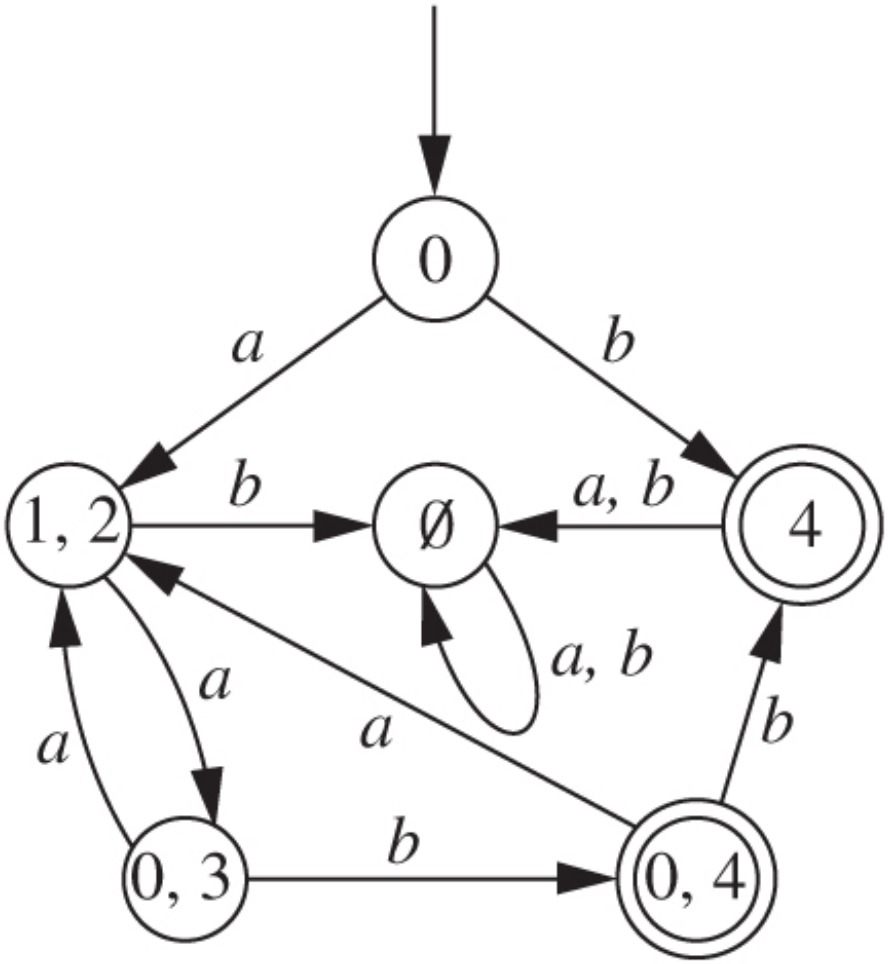

| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | ∅ |
| 2 | {3} | ∅ |
| 3 | ∅ | {0} |
| 4 | ∅ | ∅ |



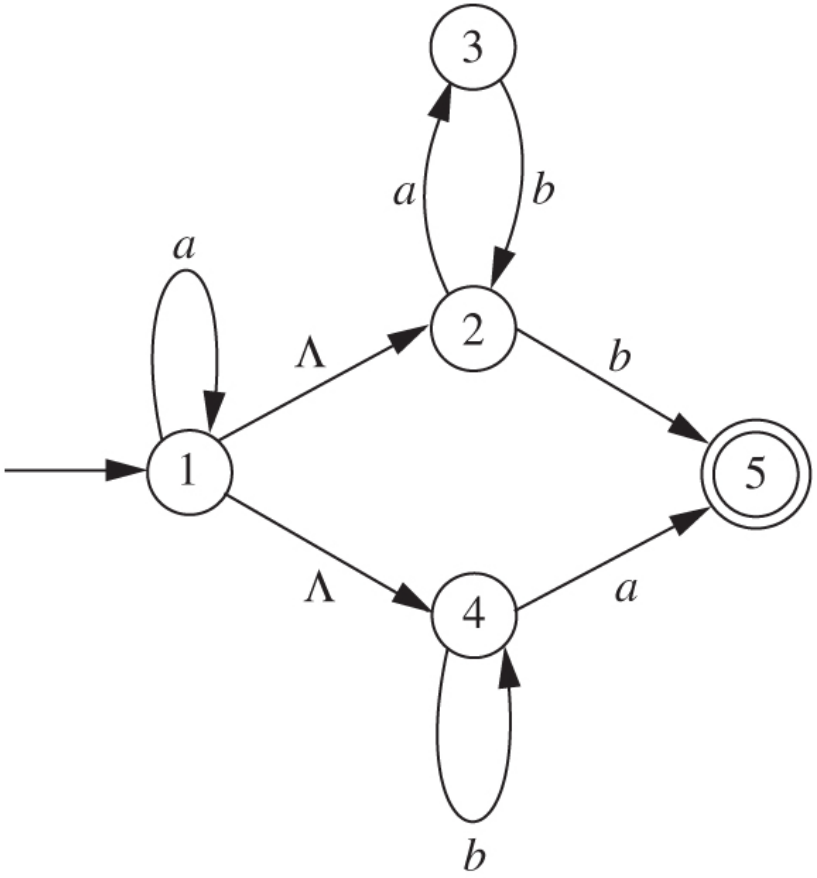

| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | ∅ |
| 2 | {3} | ∅ |
| 3 | ∅ | {0} |
| 4 | ∅ | ∅ |



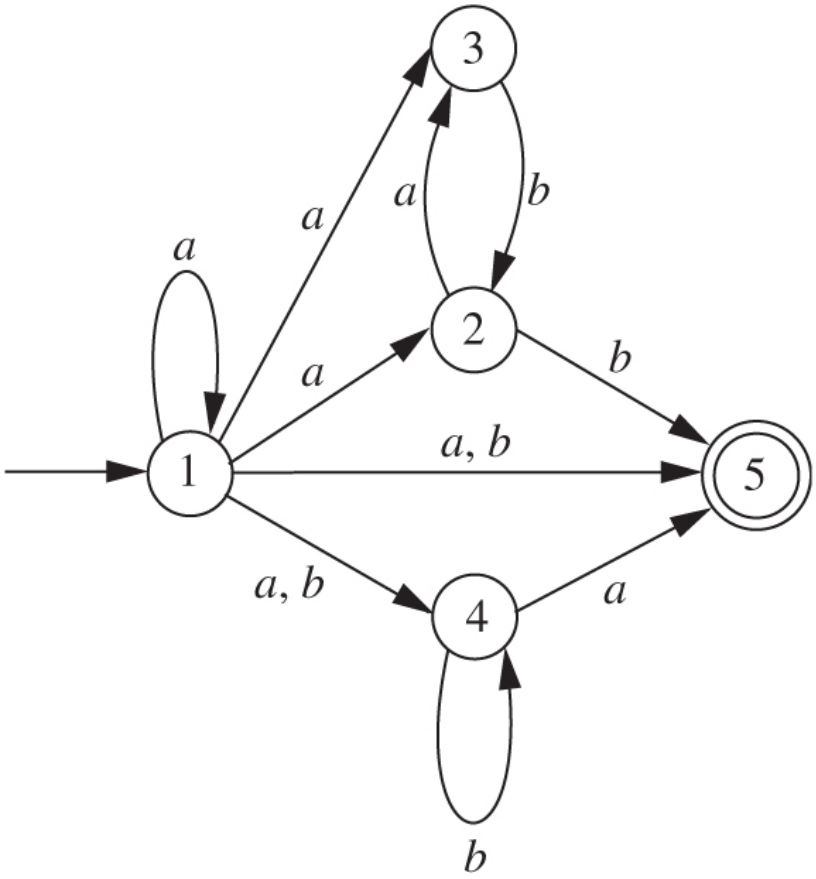




| $q$ | $\delta(q, a)$ | $\delta(q, b)$ |
|---|---|---|
| 0 | {1, 2} | {4} |
| 1 | {0} | ∅ |
| 2 | {3} | ∅ |
| 3 | ∅ | {0} |
| 4 | ∅ | ∅ |

# Construct FA from NFA: 1) eliminate all lambda transitions.



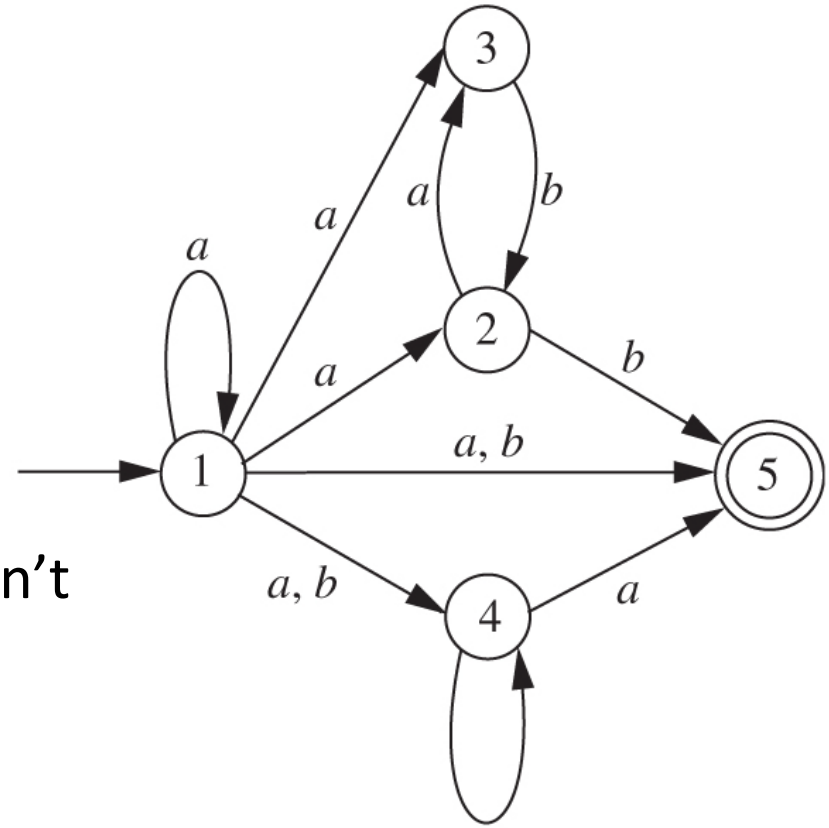

(a)




(b)

Construct FA from NFA:

2) create a table of transitions.

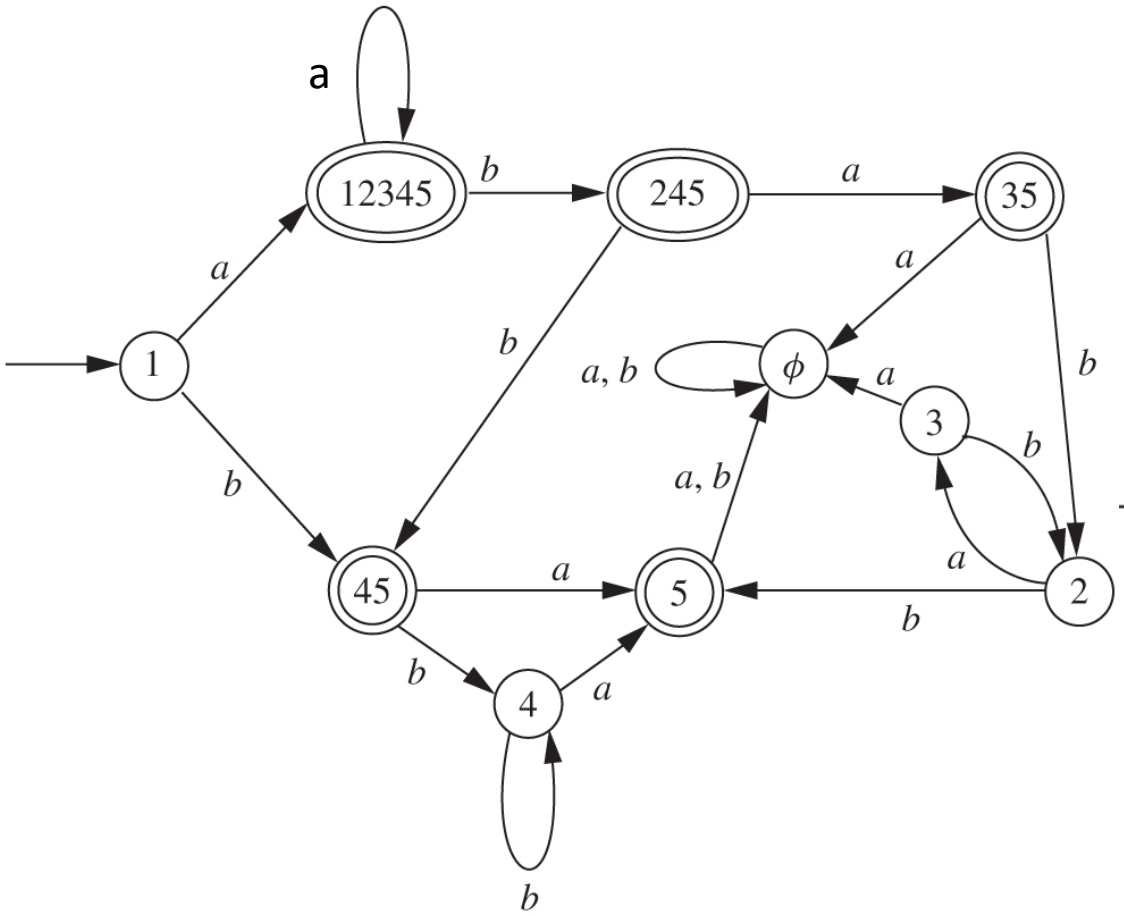



You need this only if you don't eliminate Lambda's

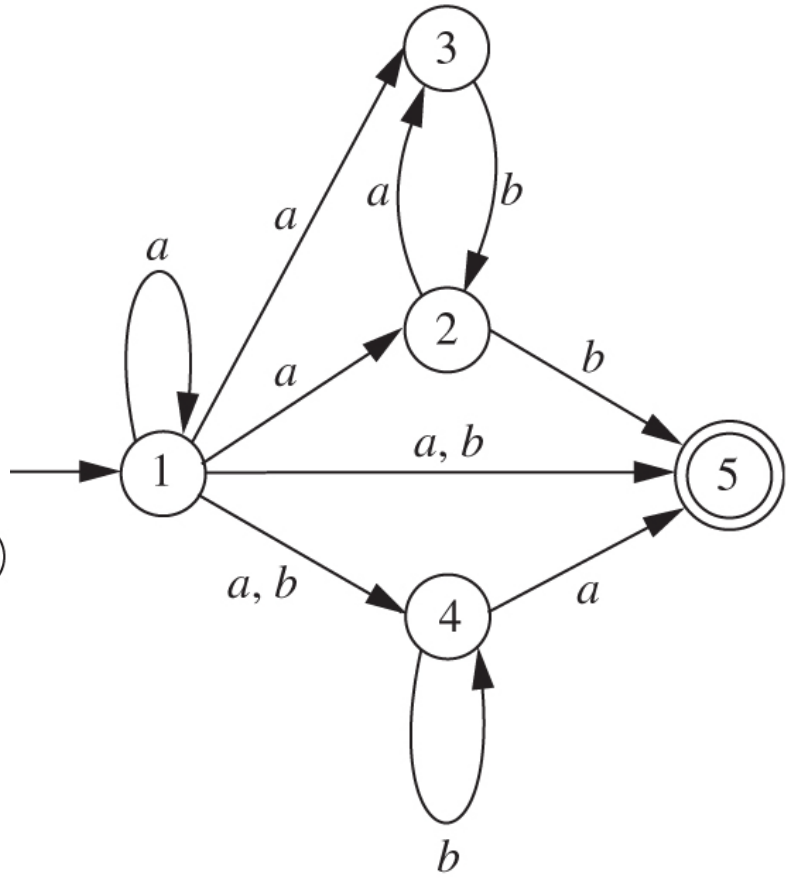
| $q$ | $\delta(q, a)$ | $\delta(q, b)$ | $\delta(q, \Lambda)$ | $\delta^*(q, a)$ | $\delta^*(q, b)$ |
|---|---|---|---|---|---|
| 1 | {1} | Ø | {2, 4} | {1, 2, 3, 4, 5} | {4, 5} |
| 2 | {3} | {5} | Ø | {3} | {5} |
| 3 | Ø | {2} | Ø | Ø | {2} |
| 4 | {5} | {4} | Ø | {5} | {4} |
| 5 | Ø | Ø | Ø | Ø | Ø |

3) Construct FA.

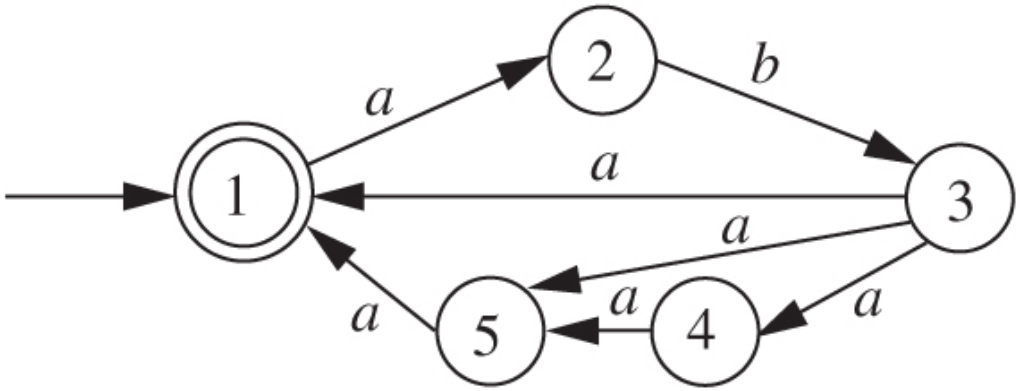
| $q$ | $\delta(q, a)$ | $\delta(q, b)$ | $\delta(q, \Lambda)$ | $\delta^*(q, a)$ | $\delta^*(q, b)$ |
|---|---|---|---|---|---|
| 1 | {1} | Ø | {2, 4} | {1, 2, 3, 4, 5} | {4, 5} |
| 2 | {3} | {5} | Ø | {3} | {5} |
| 3 | Ø | {2} | Ø | Ø | {2} |
| 4 | {5} | {4} | Ø | {5} | {4} |
| 5 | Ø | Ø | Ø | Ø | Ø |



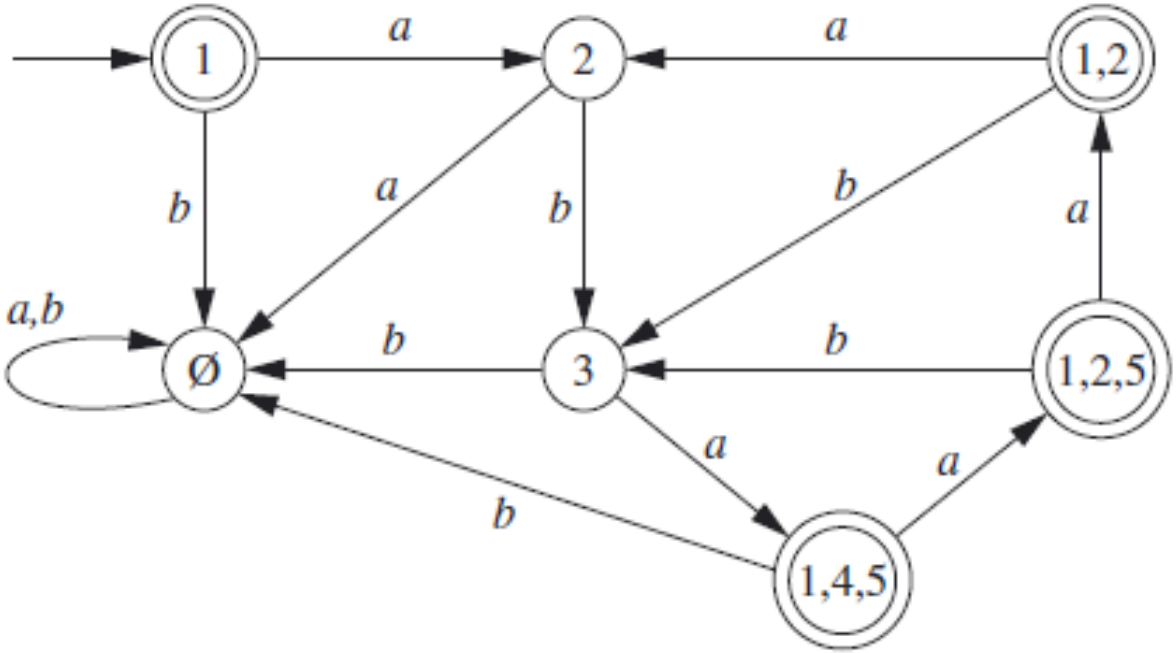

(c)



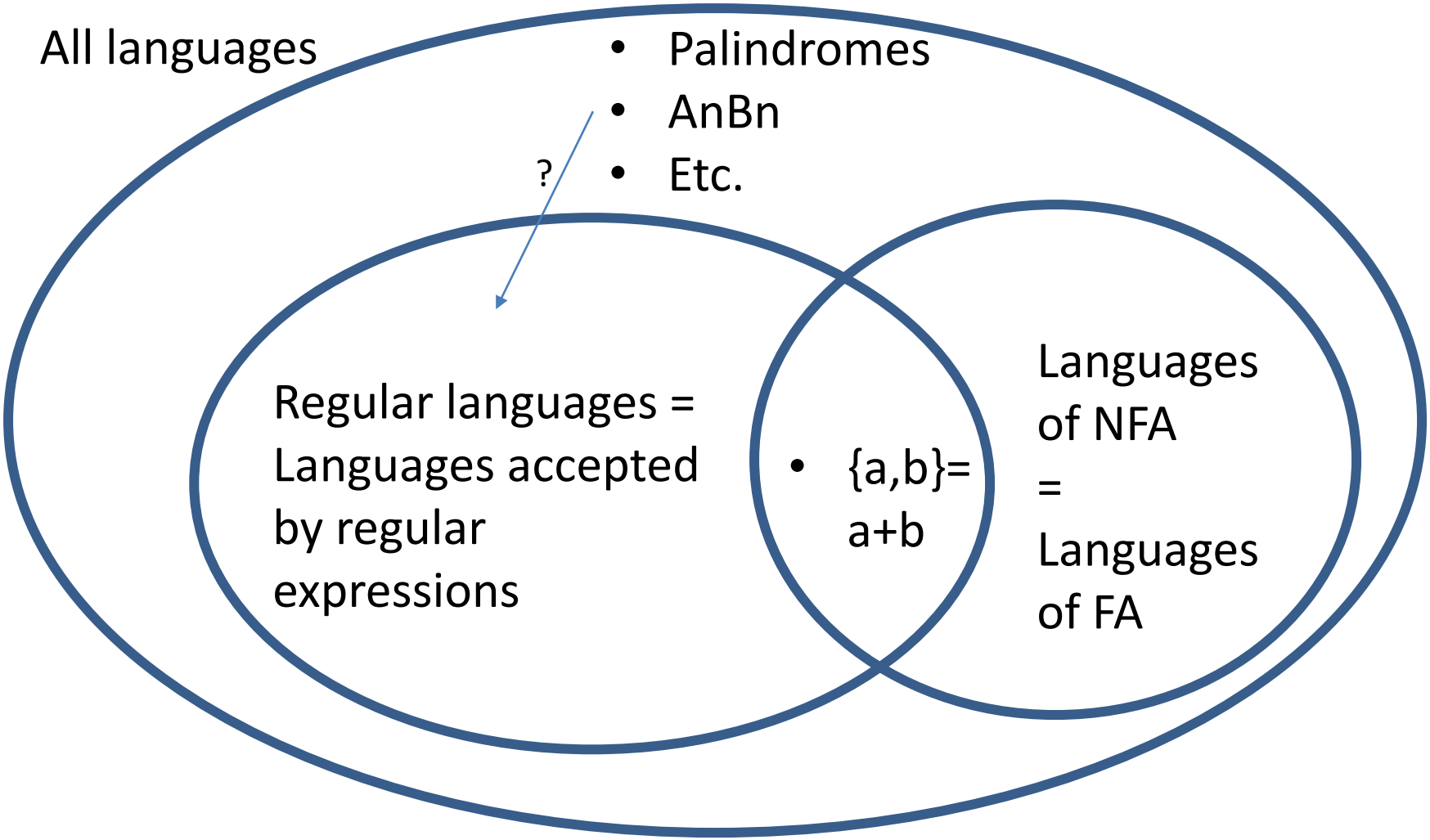

(b)




| q | δ(q,a) | δ(q,b) |
|---|---|---|
| 1 | {2} | ∅ |
| 2 | ∅ | {3} |
| 3 | {1,4,5} | ∅ |
| 4 | {5} | ∅ |
| 5 | {1} | ∅ |



# This is what we know about languages …



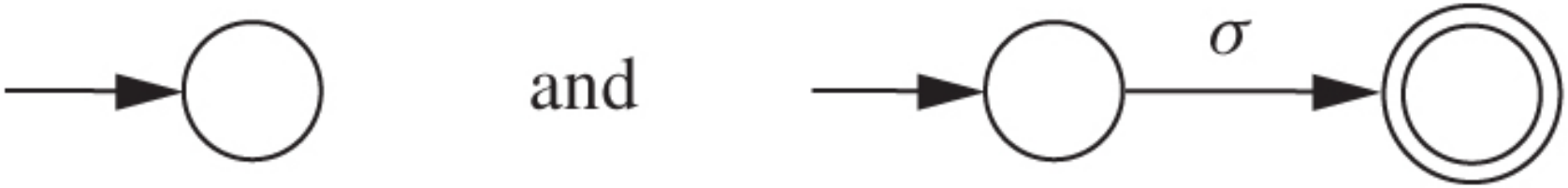
# Kleene's Theorem, Part 1

- **Theorem: For every alphabet Σ, every regular language over Σ can be accepted by a finite automaton**
- Because of what we have just shown, it is enough to show that every regular language over Σ can be accepted by an NFA
- The proof is by structural induction, based on the recursive definition of the set of regular languages over Σ

Homework: Learn both parts of Kleene's theorem (including proofs).

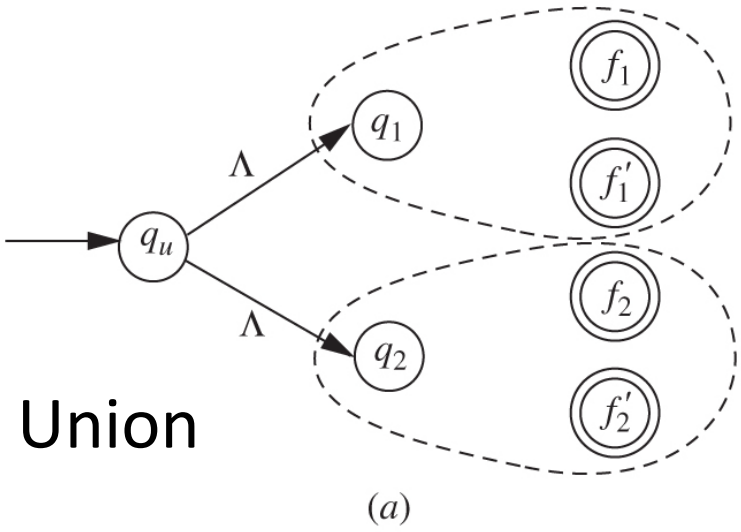
# Kleene's Theorem, Part 1 (cont'd.)

- The basis cases are easy
- The automata pictured below accept the languages ∅ and {σ}, respectively



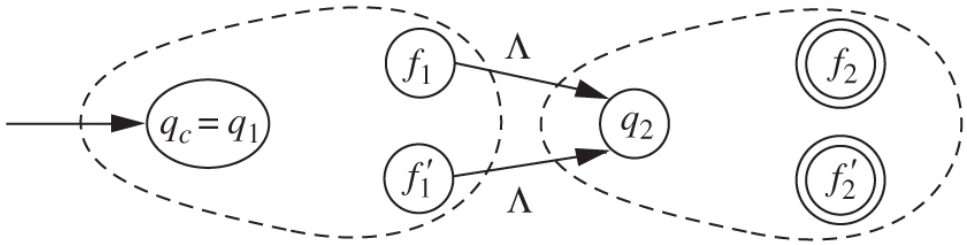

- Induction hypothesis: both $L_1$ and $L_2$ are regular languages can be accepted by NFAs
- Induction step: $L(M_1) \cup L(M_2)$, $L(M_1)L(M_2)$, and $L(M_1)^*$ can be accepted by NFAs

# Each FA is shown as having 2 accepting states

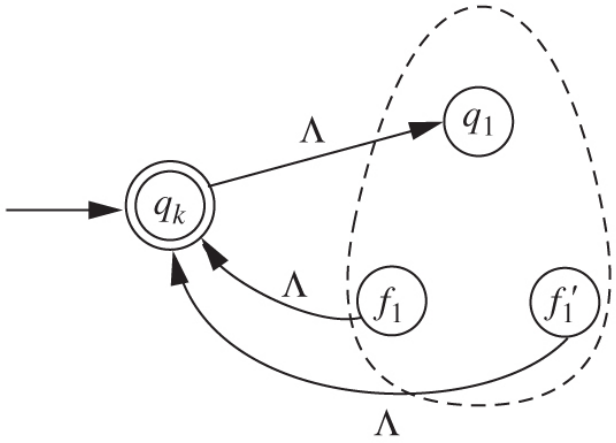



Union

(a)



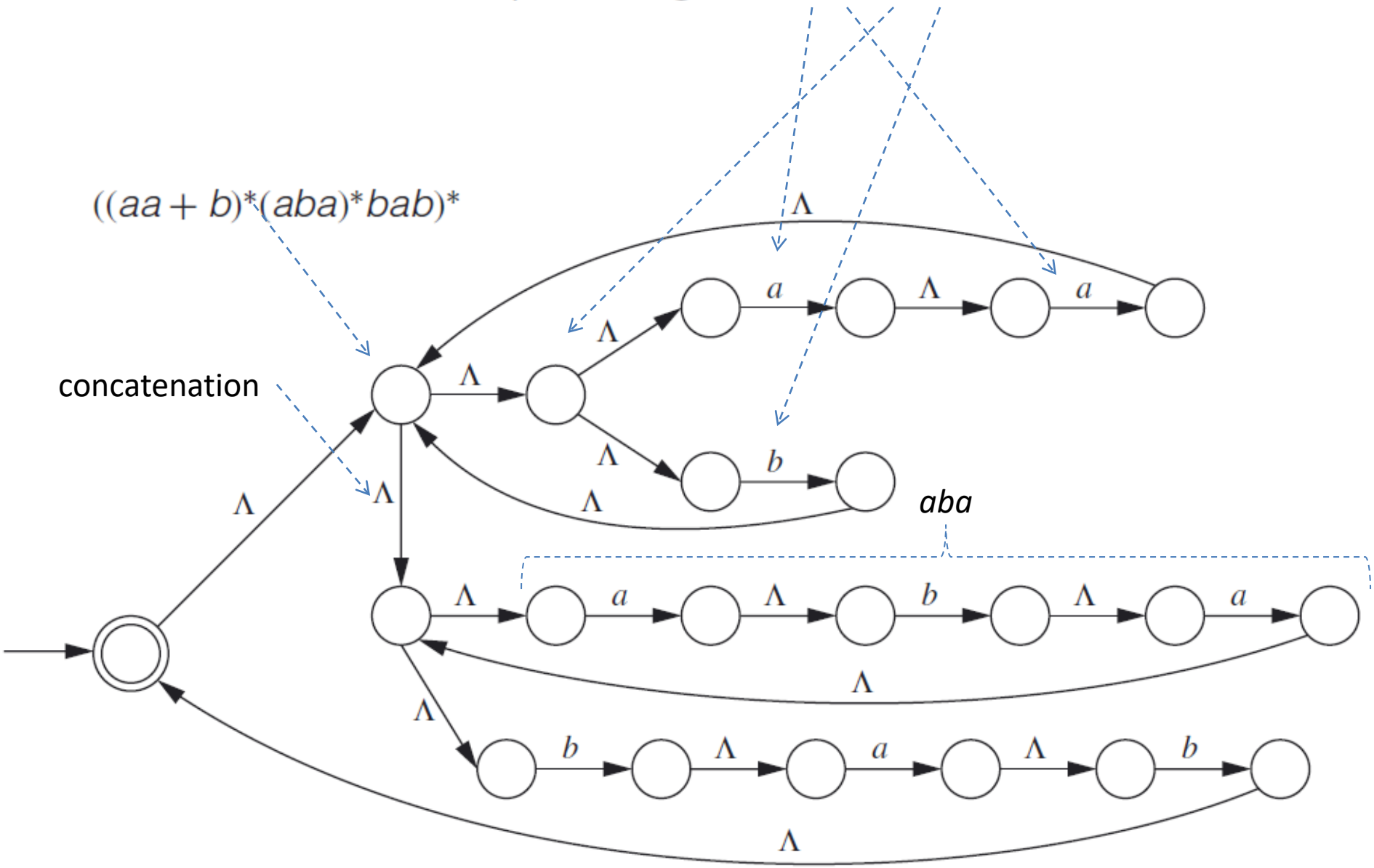

(b) Concatenation




(c)

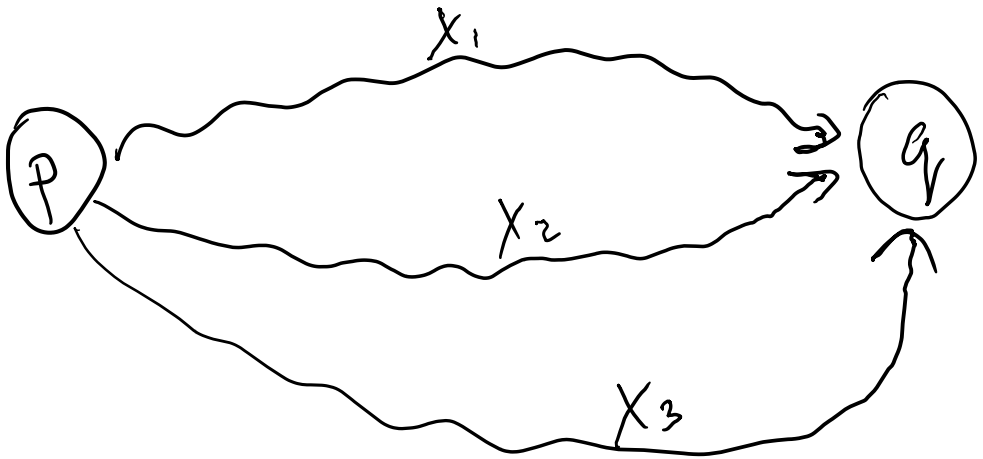
Kleene's *

# An NFA Corresponding to $((aa + b)^*(aba)^*bab)^*$



# Kleene's Theorem, Part 2

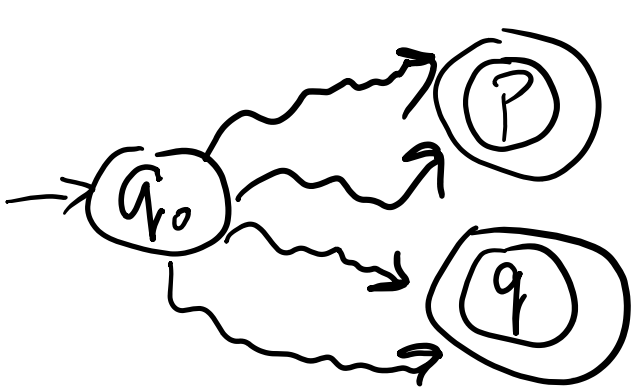
- **Theorem: For every finite automaton $M=(Q, \Sigma, q_0, A, \delta)$, the language $L(M)$ is regular**
- Proof: First, for two states $p$ and $q$, we define the language $L(p, q) = \{x \in \Sigma^* \mid \delta^*(p, x)=q\}$




$x_1, x_2, x_3 \in L(p,q)$

# Kleene's Theorem, Part 2

- **Theorem: For every finite automaton $M=(Q, \Sigma, q_0, A, \delta)$, the language $L(M)$ is regular**
- Proof: First, for two states $p$ and $q$, we define the language $L(p, q) = \{x \in \Sigma^* \mid \delta^*(p, x)=q\}$
- If we can show that for every $p$ and $q$ in $Q$, $L(p, q)$ is regular, then it will follow that $L(M)$ is, because …
  - $\boldsymbol{L(M) = \cup \{L(q_0, q) \mid q \in A\}}$



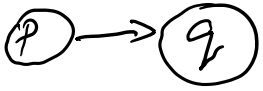

Each of these languages is regular, so is their union

# Kleene's Theorem, Part 2

- **Theorem: For every finite automaton $M=(Q, \Sigma, q_0, A, \delta)$, the language $L(M)$ is regular**
- Proof: First, for two states $p$ and $q$, we define the language $L(p, q) = \{x \in \Sigma^* \mid \delta^*(p, x)=q\}$
- If we can show that for every $p$ and $q$ in $Q$, $L(p, q)$ is regular, then it will follow that $L(M)$ is, because …
  - $L(M) = \cup \{L(q_0, q) \mid q \in A\}$
  - The union of a finite collection of regular languages is regular
- We will show that $L(p, q)$ is regular by expressing it in terms of simpler languages that are regular

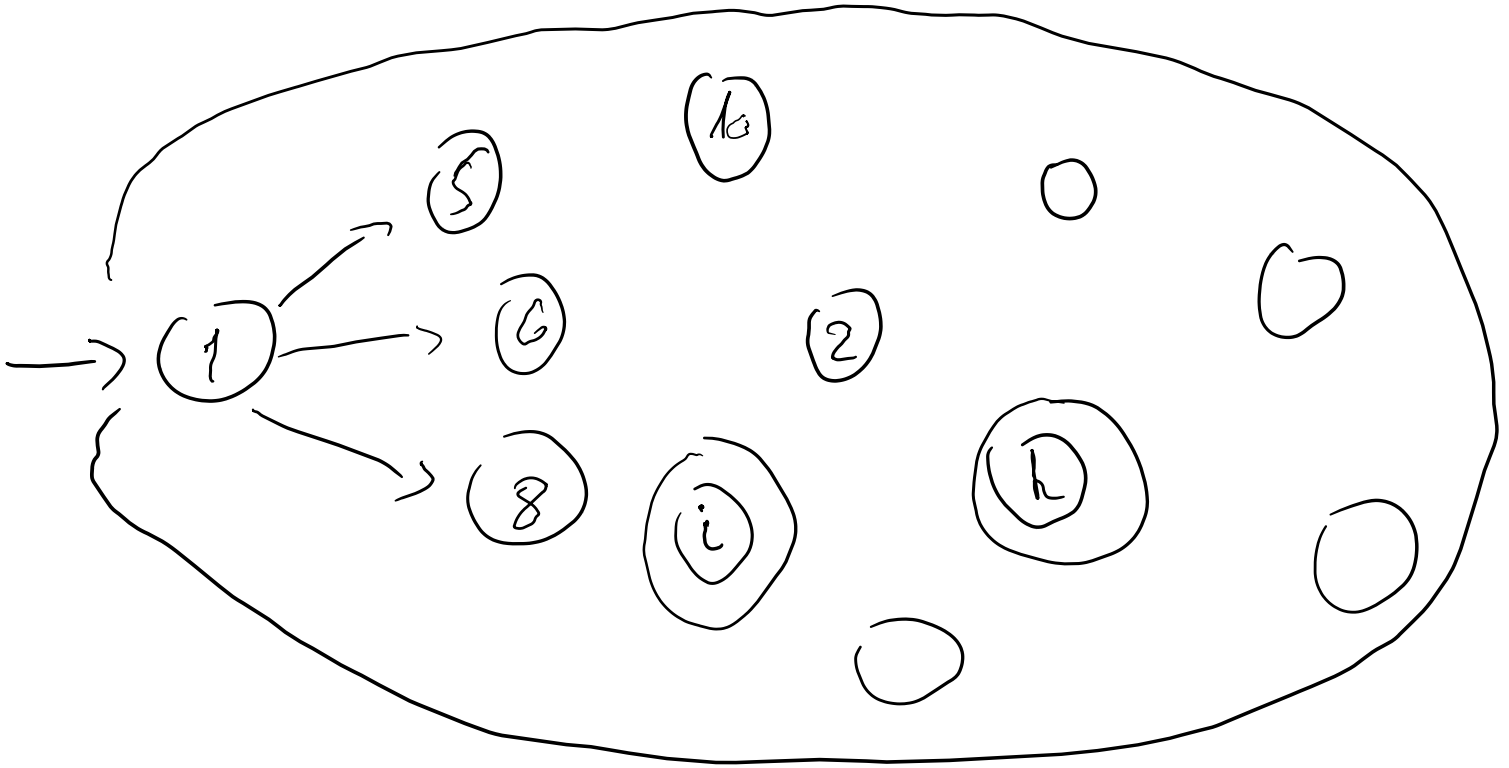
# Kleene's Theorem, Part 2 (cont'd.)

- We will consider the distinct states through which $M$ passes as it moves from $p$ to $q$
- If $x \in L(p, q)$, we say $x$ causes $M$ to go from $p$ to $q$ through a state $r$ if there are non-null strings $x_1$ and $x_2$ such that $x = x_1x_2$, $\delta^*(p, x_1) = r$, and $\delta^*(r, x_2) = q$


  - In using a string of length 1 to go from $p$ to $q$, $M$ does not go *through* any state 
  - How can we construct an inductive proof on what happens between $p$ and $q$?

# Kleene's Theorem, Part 2 (cont'd.)

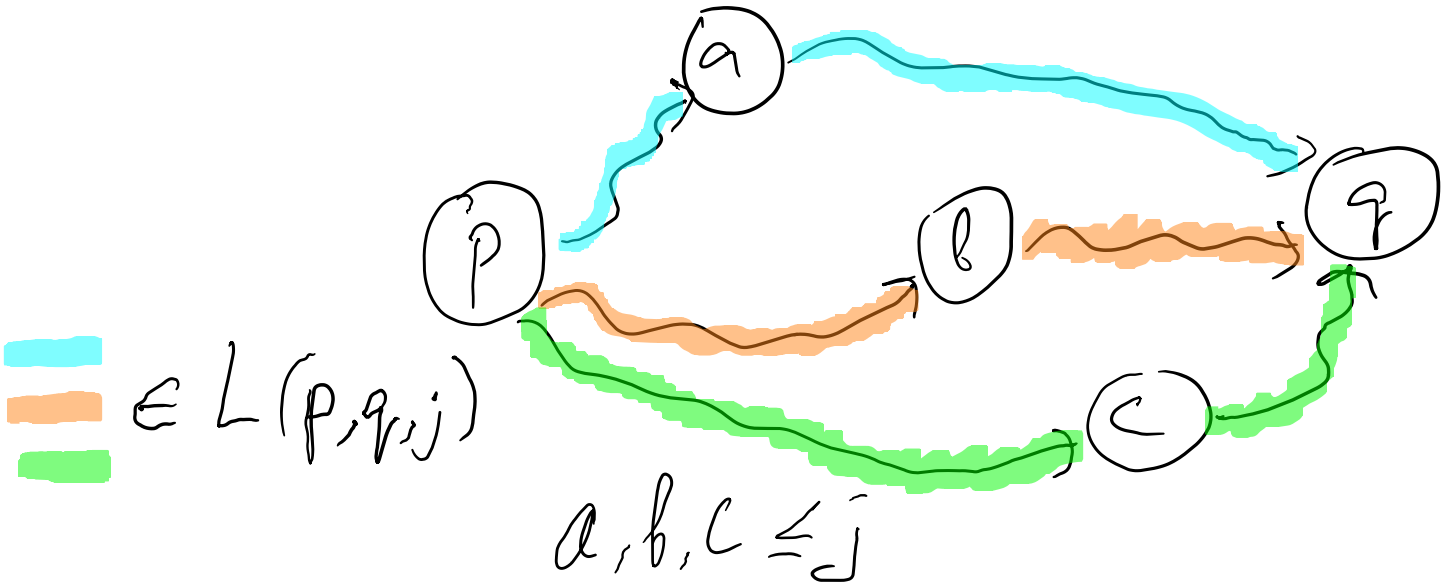
- Assume $Q$ has $n$ elements numbered 1 to $n$



# Kleene's Theorem, Part 2 (cont'd.)

- Assume $Q$ has $n$ elements numbered 1 to $n$
- For $p, q \in Q$ and $j \geq 0$

$L(p, q, j)$ = strings in $L(p, q)$ that cause $M$ to go from $p$ to $q$ without going through any state numbered higher than $j$

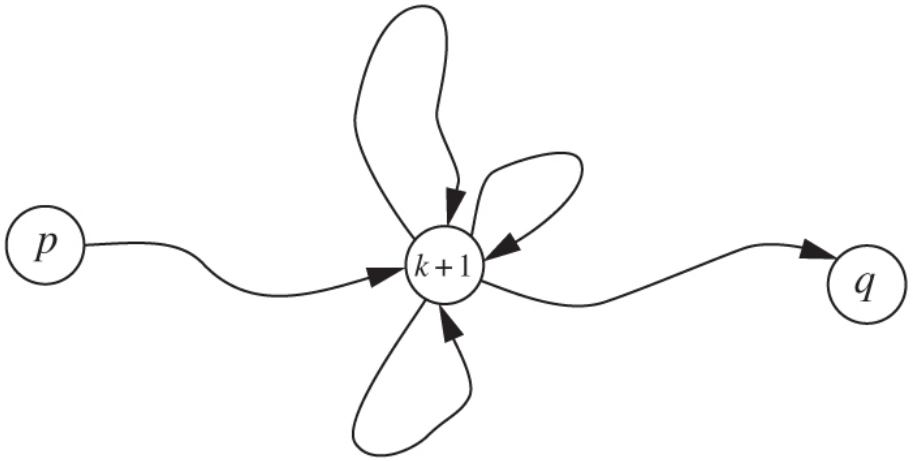


# Kleene's Theorem, Part 2 (cont'd.)

- Assume $Q$ has $n$ elements numbered 1 to $n$
- For $p, q \in Q$ and $j \geq 0$

  $L(p, q, j)$ = strings in $L(p, q)$ that cause $M$ to go from $p$ to $q$ without going through any state numbered higher than $j$
- Suppose that for some number $k \geq 0$, $L(p, q, k)$ is regular for every $p, q \in Q$ and consider how a string can be in $L(p, q, k+1)$
  - The easiest way is for it to be in $L(p, q, k)$
  - If not, it causes $M$ to go to $k+1$ one or more times, but $M$ goes through nothing higher (i.e., no state *k+2* for example)

# Kleene's Theorem, Part 2 (cont'd.)




- Every string in $L(p, q, k+1)$ can be described in one of those two ways and every string that has one of these two forms is in $L(p, q, k+1)$. This leads to the formula
  - $L(p, q, k+1) = L(p, q, k) \cup L(p, k+1, k)\, L(k+1, k+1, k)^*\, L(k+1, q, k)$
- This is the main point of a proof by induction on $k$ and for an algorithm

**Algorithm: FA→RE**. Let $r(i,j,k)$ denote a RE for $L(i,j,k)$. Then $L(M)$ is described by the RE

we need accepting states only

$$r(M) = r(1,1,3) + r(1,2,3)$$

The recursive formula with smaller $k$ in the proof tells

$$\begin{aligned} r(1,1,3) &= r(1,1,2) + r(1,3,2)r(3,3,2)^* r(3,1,2) \\ r(1,2,3) &= r(1,2,2) + r(1,3,2)r(3,3,2)^* r(3,2,2) \end{aligned}$$

Applying the formula to the expressions $r(i,j,2)$ we get

$$r(1,1,2) = r(1,1,1) + r(1,2,1)r(2,2,1)^* r(2,1,1)$$

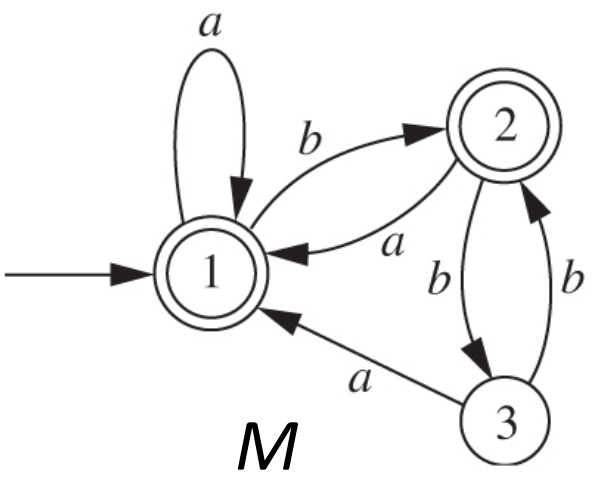
$$r(1,3,2) = r(1,3,1) + r(1,2,1)r(2,2,1)^* r(2,3,1)$$

$$r(3,3,2) = r(3,3,1) + r(3,2,1)r(2,2,1)^* r(2,3,1)$$

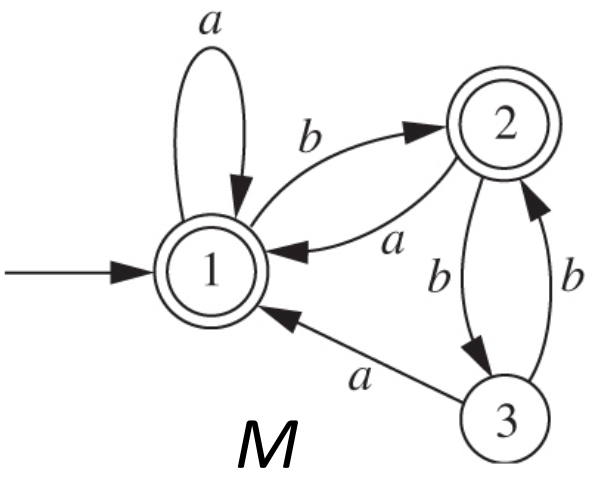
$$r(3,1,2) = r(3,1,1) + r(3,2,1)r(2,2,1)^* r(2,1,1)$$

$$r(1,2,2) = r(1,2,1) + r(1,2,1)r(2,2,1)^* r(2,2,1)$$

$$r(3,2,2) = r(3,2,1) + r(3,2,1)r(2,2,1)^* r(2,2,1)$$



| $p$ | $r(p,1,0)$ | $r(p,2,0)$ | $r(p,3,0)$ |
|---|---|---|---|
| 1 | $a+\Lambda$ | $b$ | $\emptyset$ |
| 2 | $a$ | $\Lambda$ | $b$ |
| 3 | $a$ | $b$ | $\Lambda$ |




| $p$ | $r(p,1,1)$ | $r(p,2,1)$ | $r(p,3,1)$ |
|---|---|---|---|
| 1 | $a^*$ | $a^*b$ | $\emptyset$ |
| 2 | $aa^*$ | $\Lambda+aa^*b$ | $b$ |
| 3 | $aa^*$ | $a^*b$ | $\Lambda$ |

| $p$ | $r(p,1,2)$ | $r(p,2,2)$ | $r(p,3,2)$ |
|---|---|---|---|
| 1 | $a^*(baa^*)^*$ | $a^*(baa^*)^*b$ | $a^*(baa^*)^*bb$ |
| 2 | $aa^*(baa^*)^*$ | $(aa^*b)^*$ | $(aa^*b)^*b$ |
| 3 | $aa^*+a^*baa^*(baa^*)^*$ | $a^*b(aa^*b)^*$ | $\Lambda+a^*b(aa^*b)^*b$ |

Example:

$$r(2,2,1)=r(2,2,0)+r(2,1,0)r(1,1,0)^*r(1,2,0)$$
$$=\Lambda+(a)(a+\Lambda)^*(b)$$
$$=\Lambda+aa^*b$$


Regular languages
=
Languages of regular expressions
=
Languages accepted by FA
=
Languages accepted by NFA

## Regular expressions and finite automata

- Tools such as grep, awk, and sed
- Email servers
- Pattern matching

Some motivation

## Finite automata

- Software testing/QC
- TCP/IP, HTTP, and other protocols
- Hardware

## Grammars, Automata, Regular Expressions

- GUI
- Lexical analysis in compilers of programming languages like C/C++, Java, and many more

## Future computers

- Biomolecular finite automata
- DNA/RNA Turing machines

# More questions

Some motivation

- Find duplicate occurrences of a phrase (Reg Exp).
- Does a program contain an assertion violation? Does a device driver respect certain protocols? (Properties of Lang)
- Can your software be stuck in an infinite loop? (Lang Incl)
- Does a distributed algorithm contain a livelock? (Lang Incl)
- Detect malicious Javascript entered into a web application. The set of malicious strings is a language. (Langs Inters)
- Run-time monitoring of reactive and mission-critical systems (nuclear reactors, chemical procs). (FA, Incl/Inters)
- Bioinformatics: pattern matching → build a language
- AI: FAs are used in simulation of character behavior