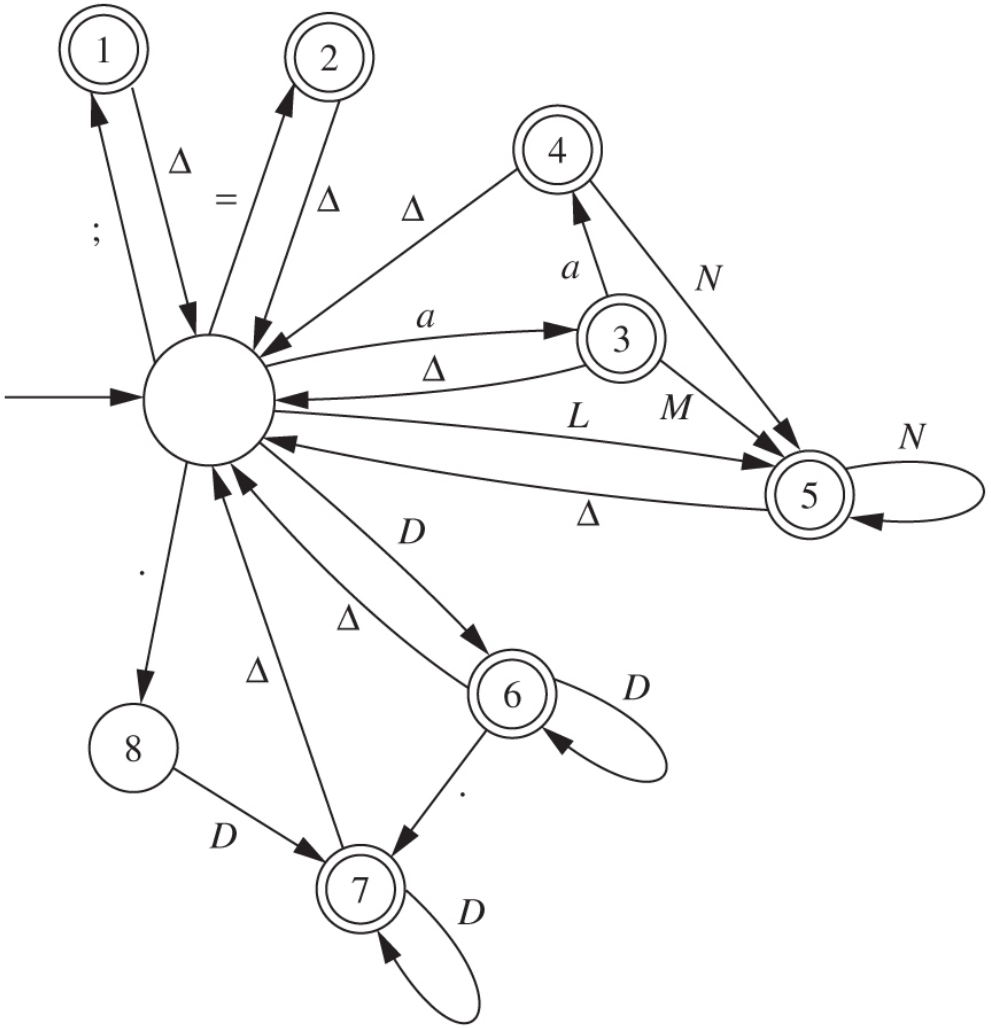
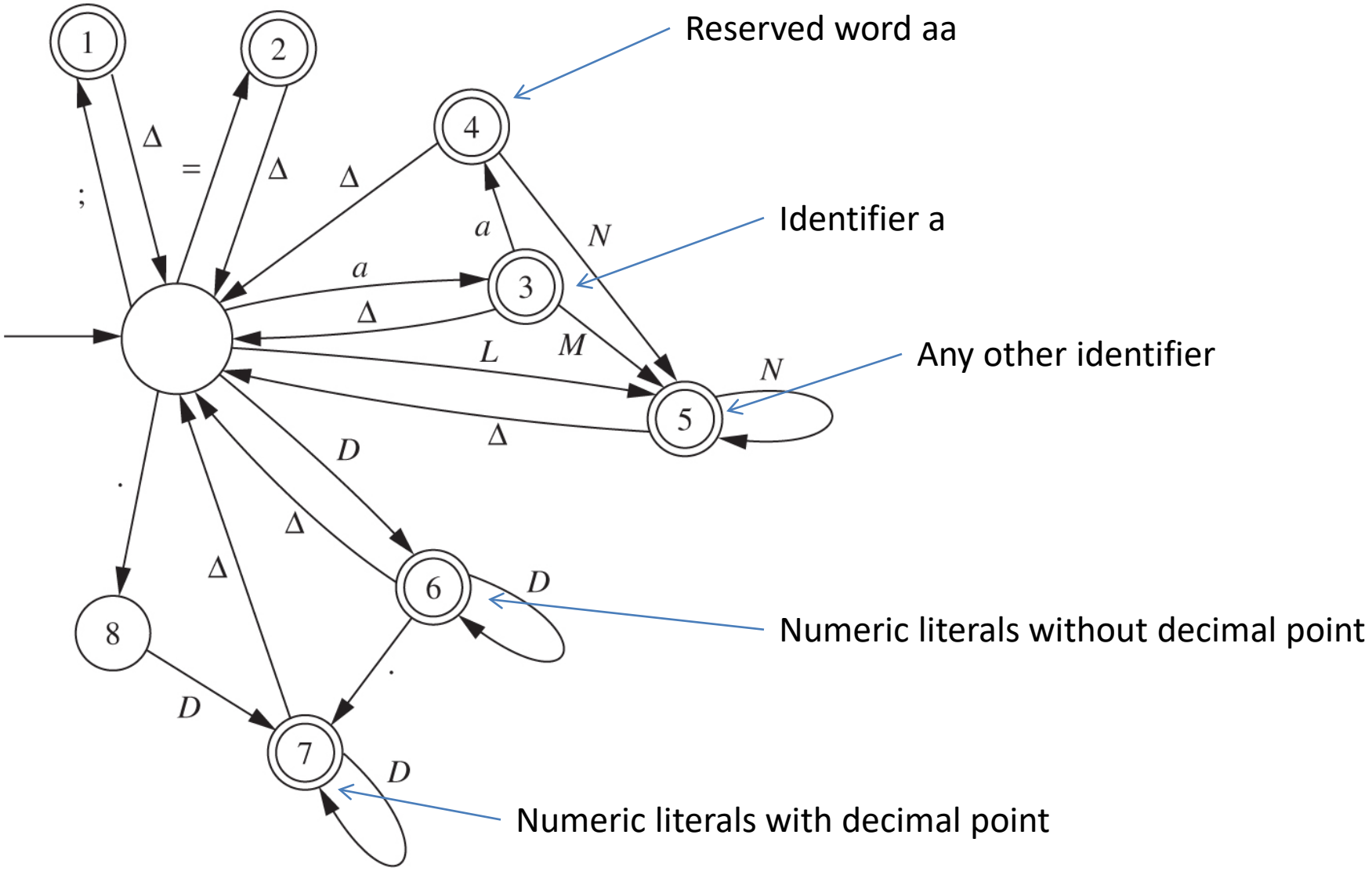
Finite Automata: Lexical Analysis Example

- FAs are ideally suited for lexical analysis, the first stage in compiling a computer program
- A lexical analyzer takes a string of characters and provides a string of "tokens" (indecomposable units)
- Tokens have a simple structure: e.g., "41.3", "main", "="
- The next slide shows an FA that accepts tokens for a simple language based on C
  - The only tokens are identifiers, semicolons, =, *aa*, and numeric literals; tokens are separated by spaces
  - Accepting states represent scanned tokens; each accepting state represents a category of token

- The input alphabet contains the 26 lowercase letters, the 10 digits, a semicolon, an equals sign, a decimal point, and the blank space .
- *D* is any digit
- *L* is a lowercase letter other than *a*
- *M* is *D* or *L*
- *N* is *D* or *L* or *a*
- Δ is a space
- All transitions not shown explicitly go to an error state and stay there




1
2
3
4
5
6
7
8
;
Δ
=
Δ
Δ
a
a
N
Δ
L
M
N
Δ
D
.
Δ
Δ
D
8
D
.
7
D
Reserved word aa
Identifier a
Any other identifier
Numeric literals without decimal point
Numeric literals with decimal point

Definition (Finite Automata) A finite automaton (FA) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$, where

- Q is a finite set of states;
- Σ is a finite input alphabet;
- $q_0 \in Q$ is the initial state;
- $A \subseteq Q$ is the set of accepting states;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

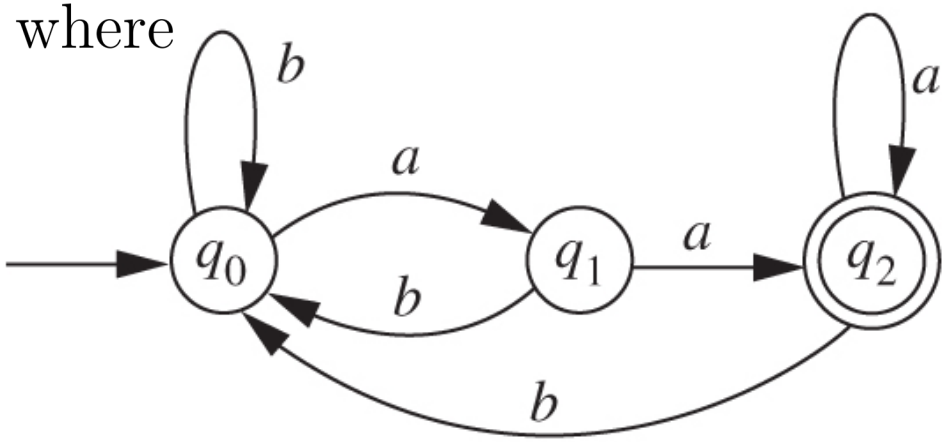
For any element $q \in Q$ and any symbol $\sigma \in \Sigma$, we interpret $\delta(q, \sigma)$ as the state to which the FA moves, if it is in state q and receives the input σ.

Definition (Finite Automata) A finite automaton (FA) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$, where

- Q is a finite set of states;
- Σ is a finite input alphabet;
- $q_0 \in Q$ is the initial state;
- $A \subseteq Q$ is the set of accepting states;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

Example: $M = (Q, \Sigma, q_0, A, \delta)$, where

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $A = \{q_2\}$



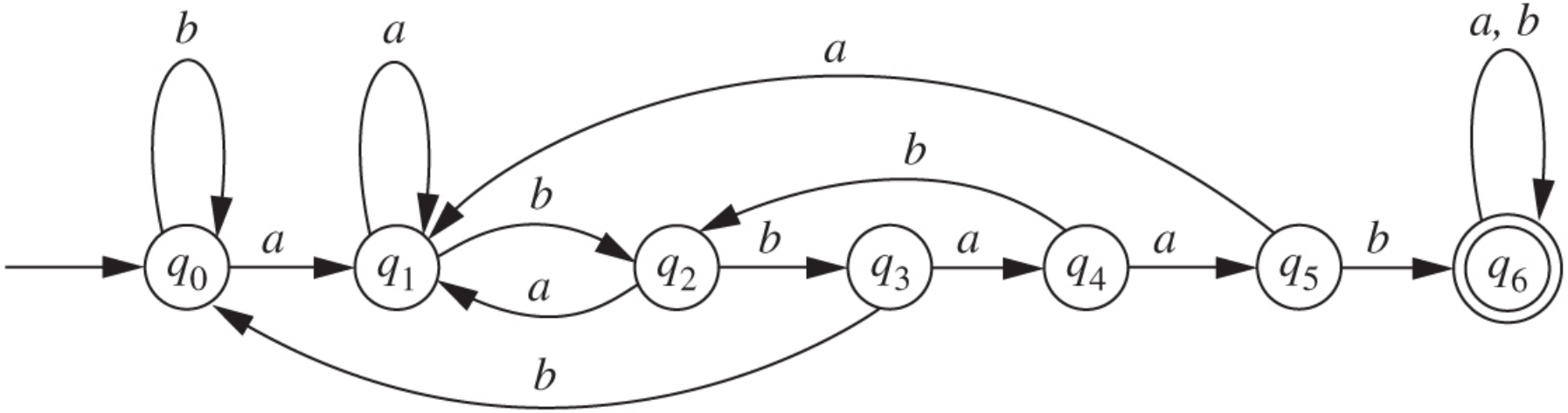
$\delta(q_0, a) = q_1;\ \delta(q_0, b) = q_0;\ \delta(q_1, a) = q_2;\ \delta(q_1, b) = q_0;$
$\delta(q_2, a) = q_2;\ \delta(q_2, b) = q_0;$

Definition (The Extended Transition Function δ^*)
Let $M = (Q, \Sigma, q_0, A, \delta)$ be FA. We define the extended transition function

$$\delta^* : Q \times \Sigma^* \to Q$$

- For every $q \in Q$, $\delta^*(q, \Lambda) = q$
- For every $q \in Q$, every $y \in \Sigma^*$, and every $\sigma \in \Sigma$,

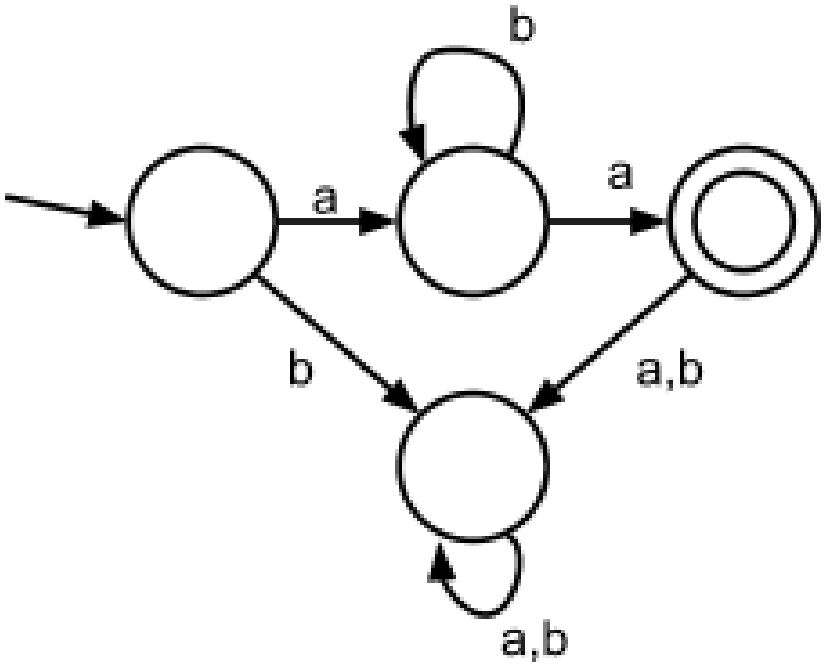
$$\delta^*(q, y\sigma) = \delta(\delta^*(q, y), \sigma).$$



Evaluation of $\delta^*(q_0, baa)$

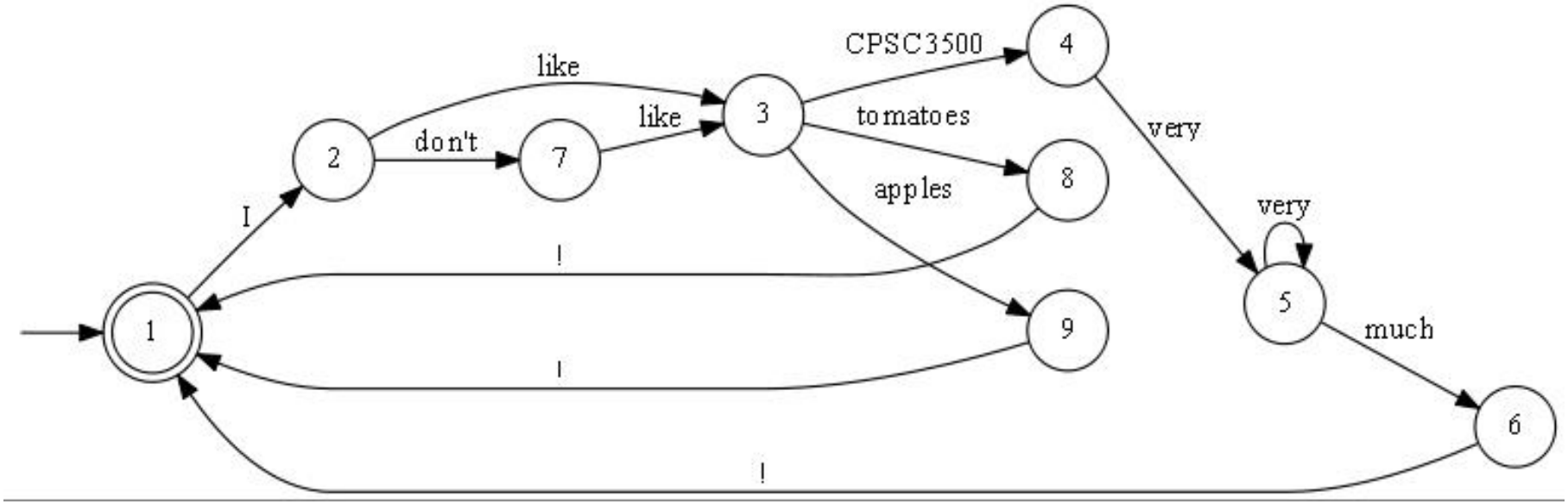
- $\delta^*(q_0, baa) = \delta(\delta^*(q_0, ba), a) = \delta(\delta(\delta^*(q_0, b), a), a)$
  $= \delta(\delta(\delta^*(q_0, \Lambda b), a), a)$
  $= \delta(\delta(\delta(\delta^*(q_0, \Lambda), b), a), a)$
  $= \delta(\delta(\delta(q_0, b), a), a) = \delta(\delta(q_0, a), a)$
  $= \delta(q_1, a) = q_1$

What language is accepted by this finite automaton?



The language of all strings that have exactly two letters “a”:
at the beginning and at the end of the string

What language is accepted by this finite automaton?



The alphabet is

- I
- like
- don’t
- CPSC3500
- tomatoes
- apples
- very
- much
- !

Add missing transitions; The language contains all chains (without spaces) of

- I like apples!
- I don’t like apples!
- I like tomatoes!
- I don’t like tomatoes!
- I like CPSC3500 very very* much!
- I don’t like CPSC3500 very very* much!

Can you propose a smaller FA that accepts the same language?

Merge states 8, 9 and 6

Claim. *For every* $x, y \in \Sigma^*$ $\quad \delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$

Proof. For every $y \in \Sigma^*$ we need to prove that $P(y)$ is true, where $P(y)$ is the statement "for every $x \in \Sigma^*$, $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$".

- BS: $\forall x \in \Sigma^* \ \delta^*(q, x\Lambda) = \delta^*(\delta^*(q, x), \Lambda)$. This is true beacause $\delta^*(\delta^*(q, x), \Lambda) = \delta^*(q, x)$, i.e., we have $\delta^*(q, x\Lambda) = \delta^*(q, x)$.
- IH: $y \in \Sigma^*$, and $\forall x \in \Sigma^*$, $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$.
- IS: $\forall x \in \Sigma^*$, $\delta^*(q, x(y\sigma)) = \delta^*(\delta^*(q, x), y\sigma)$

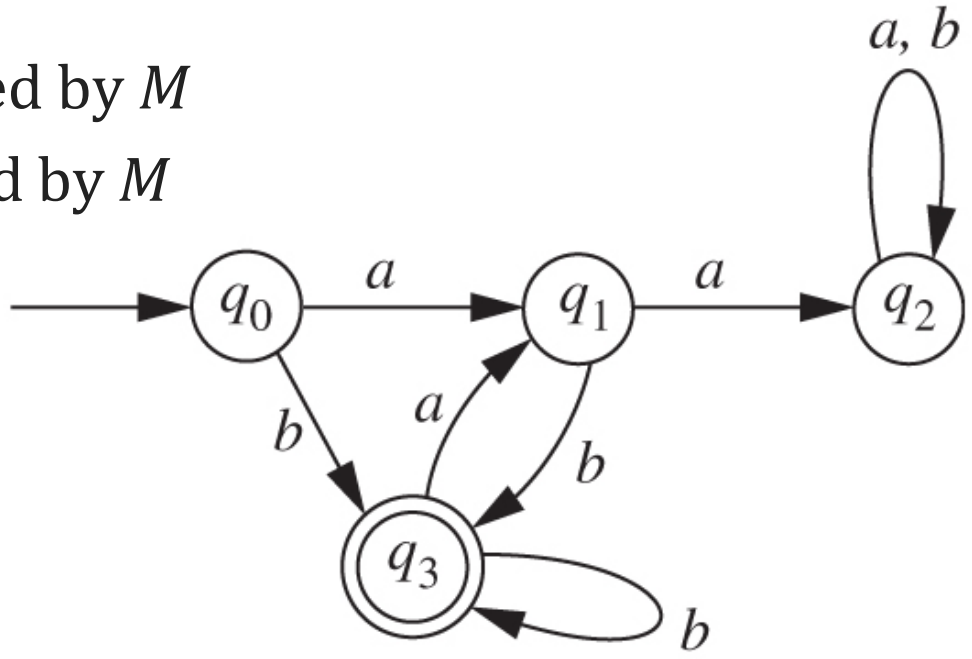
$$\begin{array}{ccccc} \delta^*(q, x(y\sigma)) & = & \delta^*(q, (xy)\sigma) & = & \\ \delta(\delta^*(q, xy), \sigma) & = & \delta(\delta^*(\delta^*(q, x), y), \sigma) & = & \\ \delta^*(\delta^*(q, x), y\sigma) & & & & \end{array}$$

□

- Definition:
  - Let $M=(Q, \Sigma, q_0, A, \delta)$ be an FA, and let $x \in \Sigma^*$. Then x is *accepted* by M if $\delta^*(q_0, x) \in A$ and *rejected* otherwise

babbbbabb is accepted by *M*

bbbbaaaab is rejected by *M*



- The *language* accepted by *M* is

  $L(M) = \{x \in \Sigma^* \mid x \text{ is accepted by } M\}$

Accepting the Union, Intersection, or Difference of Two Languages

- Suppose that L_1 and L_2 are languages over Σ
  - Given an FA that accepts L_1 and another FA that accepts L_2, we can construct one that accepts $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$

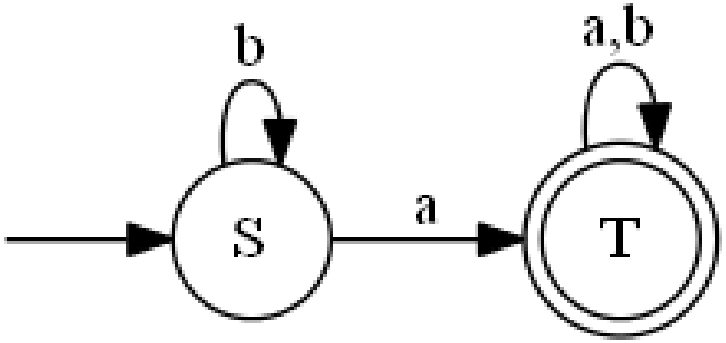
  How to construct an FA for $L_1 \cup L_2$?

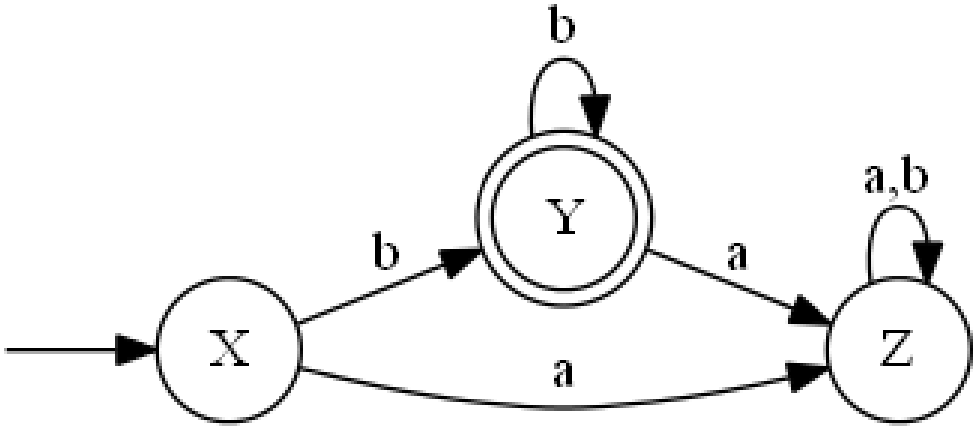
  - The idea is to construct an FA that executes both of the original FAs at the same time
  - This works because if $x \in \Sigma^*$, then knowing whether $x \in L_1$ and whether $x \in L_2$ is enough to determine whether $x \in L_1 \cup L_2$

Accepting the Union, Intersection, or Difference of Two Languages (cont'd.)

- Theorem: Suppose $M_1=(Q_1, \Sigma, q_1, A_1, \delta_1)$ and $M_2=(Q_2, \Sigma, q_2, A_2, \delta_2)$ are FAs accepting L_1 and L_2. Let $M=(Q, \Sigma, q_0, A, \delta)$ be defined as follows:
  - $Q = Q_1 \times Q_2$
  - $q_0 = (q_1, q_2)$
  - $\delta((p, q), \sigma) = (\delta_1(p, \sigma), \delta_2(q, \sigma))$
- Then, if :
  - $A = \{(p, q) \mid p \in A_1 \text{ or } q \in A_2\}$, M accepts $L_1 \cup L_2$
  - $A = \{(p, q) \mid p \in A_1 \text{ and } q \in A_2\}$, M accepts $L_1 \cap L_2$
  - $A = \{(p, q) \mid p \in A_1 \text{ and } q \notin A_2\}$, M accepts $L_1 - L_2$

L1 = all strings that include “a”





L2 = all strings that do not include “a”

Union of L1 and L2