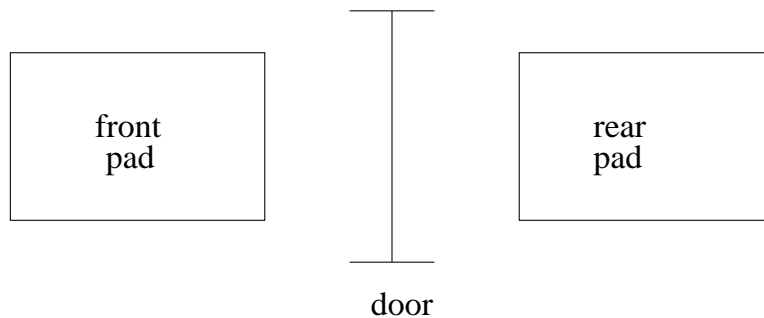


# Finite Automata

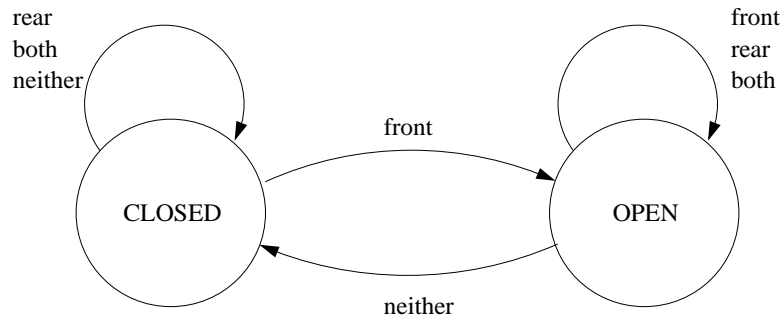
- An *automaton* is a simple machine designed to respond to encoded instruction.
- The key component of a *finite automaton* is its *finite control*, which represents a “processing unit” of fixed, finite capacity.
- A finite automaton has an *input tape* that contains a string, the symbols of which are processed in sequence, one at a time.
- The only output is an indication as to whether the input is *accepted* or *rejected*. Finite automata are thus examples of formal-language recognition devices.
- There is no separate memory component, but the finite control represents implicit memory capacity.
- More general computation models provide unbounded memory.

# Automatic Door Control

- The following controller for an automatic door displays the main characteristics of a finite automaton.



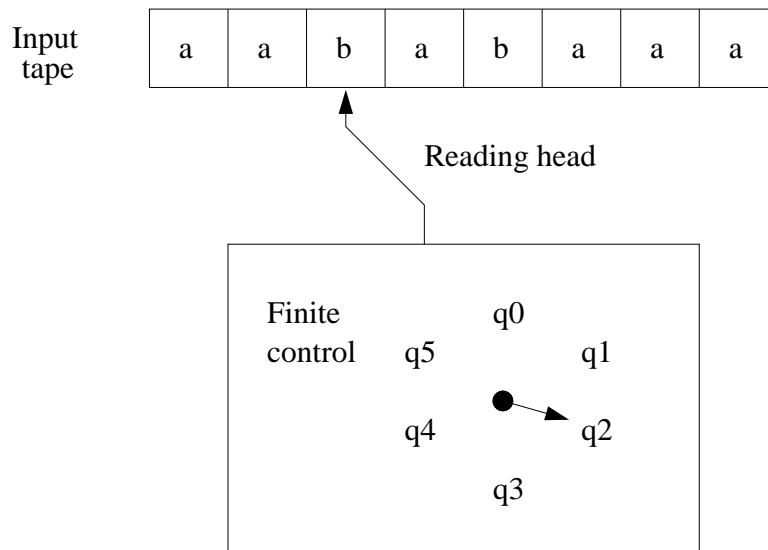
Top view of an automatic door



State diagram for automatic door controller

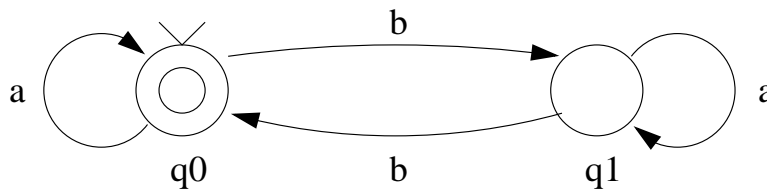
# State Transitions

- The finite control of an automaton consists of (a finite number of) *states* and an encoding of *transitions* between states.
- Operation begins with the control in a specified *start state* and proceeds with a sequence of *state transitions*, each step depending on the current state and the current input symbol.
- The last state, reached when the automaton has processed all input symbols, determines whether or not the input string is accepted.



# State Diagrams

- Finite automata can be conveniently represented by labeled directed graphs called *state diagrams*.
- The nodes of the diagram represent states, the edges represent transitions between states. Edges are labeled by input symbols to indicate the dependency of transitions on the input.
- Each node has one outgoing edge for each possible input symbol.
- One specially marked node represents the start state. Furthermore, nodes are classified as to whether or not they represent accept states.
- The following state diagram represents a simple automaton that accepts strings of symbols  $a$  and  $b$  that contain an even number of  $b$ 's.



# Deterministic Automata

- **Definition**

A *deterministic finite automaton* is a quintuple  $M = (Q, \Sigma, q_0, T, \delta)$ , where

$Q$  is a finite set, the elements of which are called *states*,

$\Sigma$  is a finite set, called an *alphabet*,

$q_0$  is an element of  $Q$ , called the *start state*,

$T$  is a subset of  $Q$ , the elements of which are called *accept states*, and

$\delta$  is a function from  $Q \times \Sigma$  to  $Q$ , called the *transition function*.

- The important component is the transition function: if  $q$  is the current state and  $\sigma$  the next input symbol, then  $\delta(q, \sigma)$  indicates the (unique) next state.
- Computations begin with the start state and stop when all input symbols have been read.
- If the automaton ends up in an accept state (i.e., a state of  $T$ ) the input is *accepted*, otherwise it is *rejected*.

## Example

- The automaton  $M$  depicted in the previous transition diagram corresponds to a quintuple  $(Q, \Sigma, q_0, T, \delta)$ , where

$$Q = \{q_0, q_1\},$$

$$\Sigma = \{a, b\},$$

$$T = \{q_0\},$$

and  $\delta$  is the following function:

$q$	$\sigma$	$\delta(q, \sigma)$
$q_0$	$a$	$q_0$
$q_0$	$b$	$q_1$
$q_1$	$a$	$q_1$
$q_1$	$b$	$q_0$

# Computations

- A *configuration* (for a finite automaton) is a pair  $(q, w)$  in  $Q \times \Sigma^*$ , where  $q$  represents the current state and  $w$  the unread portion of the input.

- A configuration  $(q, w)$  *yields*  $(q', w')$ , written

$$(q, w) \vdash_M (q', w'),$$

if  $w$  is a nonempty string  $aw'$  and  $\delta(q, a) = q'$ .

- The relation  $\vdash_M$  represents individual computation steps by an automaton  $M$ .
- We write  $(q, w) \vdash_M^* (q', w')$ , if there is a sequence of states  $q_1, \dots, q_{k+1}$ , and a string  $a_1 \dots a_k$ , such that

$$q = q_1 \text{ and } q' = q_{k+1},$$

$$w = a_1 \dots a_k w', \text{ and}$$

$$\delta(q_i, a_i) = q_{i+1} \text{ for all } i \text{ with } 1 \leq i \leq k.$$

- The relation  $\vdash_M^*$  is the reflexive-transitive closure of  $\vdash_M$ , and represents *computation sequences*.

# Language Recognition

- A *start configuration* for a finite automaton  $M$  is any configuration  $(q_0, w)$ , where  $q_0$  is the start state of  $M$ .
- Pairs  $(q, \epsilon)$  are called *accepting configurations*, if  $q$  is an accept state; and *rejecting configurations*, otherwise. (Recall that  $\epsilon$  denotes the empty string.)
- We say that a string  $w$  is *accepted* by a deterministic automaton  $M$  if there is a computation sequence from the start configuration for input  $w$  to an accepting configuration:

$$(q_0, w) \vdash_M^* (q, \epsilon)$$

for some accept state  $q$ .

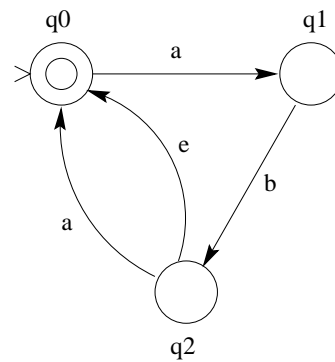
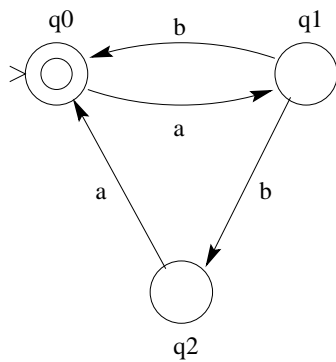
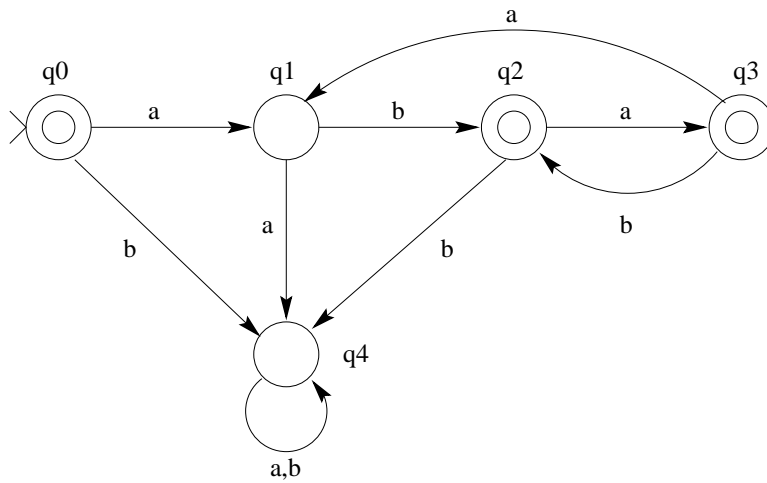
- The *language accepted by  $M$*  is the set

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

of all strings accepted by  $M$ .

# State Transition Diagrams

- Here are a few more examples of state transition diagrams. The first represents a deterministic automaton; the other two, a more general computation model:



# Nondeterministic Automata

- **Definition**

A *nondeterministic finite automaton* is a quintuple  $M = (Q, \Sigma, q_0, T, \Delta)$ , where

$Q$  is a finite set, the elements of which are called *states*,

$\Sigma$  is a finite set, called an *alphabet*,

$q_0$  is an element of  $Q$ , called the *initial state*,

$T$  is a subset of  $Q$ , the elements of which are called *accept states*, and

$\Delta$  is a subset of  $Q \times (\Sigma \cup \{\epsilon\}) \times Q$ , called the *transition relation*.

- The triples  $(q, u, p)$  in  $\Delta$  are called *transitions*.
- Deterministic automata are special cases of this definition, where the transition relation is a function.

# Nondeterministic Computations

- If  $M$  is a nondeterministic automaton we say that a configuration  $(q, w)$  *yields*  $(q', w')$ , written

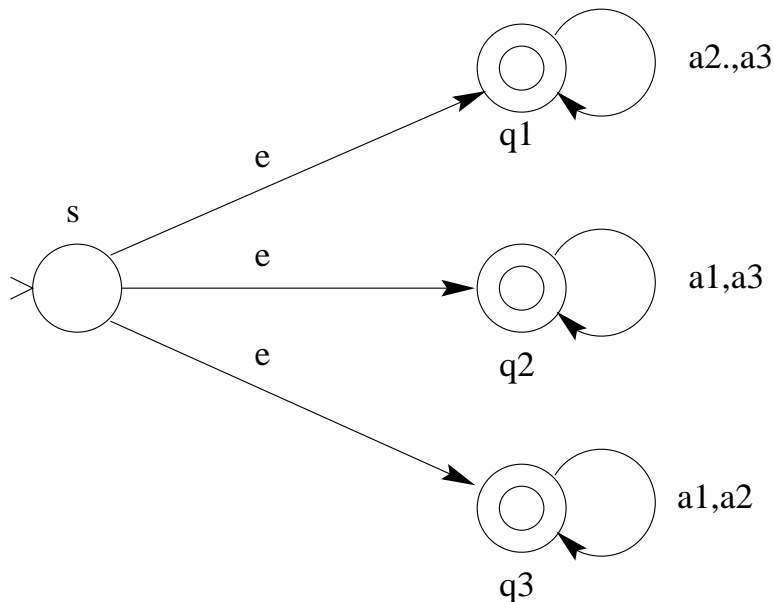
$$(q, w) \vdash_M (q', w'),$$

if, and only if, there is a  $u \in \Sigma \cup \{\epsilon\}$  such that  $w = uw'$  and  $(q, u, q') \in \Delta$ .

- This definition generalizes the deterministic case in that a configuration may yield more than one successor, or even no successor.
- The set of all possible computation sequences from a start configuration  $(q_0, w)$  can be represented by a *computation tree*.
- A string  $w$  is *accepted* by a nondeterministic automaton  $M$  if  $(s, w)$  yields some accepting configuration  $(q, \epsilon)$  (where  $q$  is an accept state) in zero or more steps.
- Note that there may be many computation sequences from the same start configuration: accepting, rejecting, even failing. Acceptance means that there is *at least one* accepting configuration.

# Example

- Nondeterministic devices are not realistic computation models but often greatly simplify the description of automata.
- For example, let  $\Sigma = \{a_1, \dots, a_n\}$  be an alphabet of  $n$  elements. Consider the problem of designing an automaton that accepts exactly those strings that do not contain all the symbols of  $\Sigma$ .
- The design of a corresponding nondeterministic automaton is straightforward:



# Nondeterminism

- Nondeterminism does not increase the computational, or expressive, power of finite automata.
- Two finite automata  $M_1$  and  $M_2$  are *equivalent* if, and only if, they recognize the same language, i.e.,  $L(M_1) = L(M_2)$ .

- **Theorem**

*For each nondeterministic finite automaton there is an equivalent deterministic finite automaton.*

- *Proof sketch*

The basic idea is to design a deterministic automaton that keeps track of the various branches of a nondeterministic computation: a single state of the deterministic automaton represents a *set* of states of the nondeterministic automaton.

- Formally, for each nondeterministic automaton

$$M = (Q, \Sigma, q_0, T, \Delta)$$

define a corresponding deterministic automaton

$$M' = (Q', \Sigma, q'_0, T', \delta') :$$

$Q'$  is  $\mathcal{P}(Q)$ , the powerset of  $Q$ ,  
 $q'_0$  is the set  $E(q_0)$  [see the definition below],  
 $T'$  is the set  $\{K \subseteq Q : K \cap T \neq \emptyset\}$ , and  
 $\delta'(K, a) = \bigcup \{E(p) : p \in Q \text{ and } (q, a, p) \in \Delta$   
for some  $q \in K\}$ , for all sets  $K \subseteq Q$  and  
symbols  $a \in \Sigma$ .

- The expression  $E(q)$  denotes the set of all states of  $M$  that can be reached from  $q$  without reading any input symbol. That is,

$$E(q) = \{p \in Q : (q, \epsilon) \vdash_M^* (p, \epsilon)\}.$$

- It can be shown that
  1.  $M'$  is a deterministic finite automaton and
  2.  $M$  and  $M'$  are equivalent.

# Closure Properties

- **Theorem**

The class of languages accepted by finite automata is closed under

- (a) union,
- (b) concatenation,
- (c) Kleene star,
- (d) complementation, and
- (e) intersection.

- *Proof sketch*

Let  $M_1$  and  $M_2$  be finite automata,

$$M_1 = (Q_1, \Sigma, s_1, T_1, \Delta_1)$$

and

$$M_2 = (Q_2, \Sigma, s_2, T_2, \Delta_2),$$

where  $Q_1$  and  $Q_2$  are disjoint sets.

1. Let  $M$  be the automaton  $(Q, \Sigma, s, T, \Delta)$ , where

$$Q = Q_1 \cup Q_2 \cup \{s\},$$

$$T = T_1 \cup T_2,$$

$$\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, \epsilon, s_1), (s, \epsilon, s_2)\},$$

and  $s \notin Q_1 \cup Q_2$ .

Then  $L(M) = L(M_1) \cup L(M_2)$ .

2. Let  $M$  be the automaton  $(Q, \Sigma, s_1, T, \Delta)$ , where

$$Q = Q_1 \cup Q_2,$$

$$T = T_2, \text{ and}$$

$$\Delta = \Delta_1 \cup \Delta_2 \cup \{(q, \epsilon, s_2) : q \in T_1\}.$$

Then  $L(M) = L(M_1)L(M_2)$ .

3. Let  $M$  be the automaton  $(Q, \Sigma, s, T, \Delta)$ , where

$$Q = Q_1 \cup \{s\},$$

$$T = T_1 \cup \{s\},$$

$$\Delta = \Delta_1 \cup \{(s, \epsilon, s_1)\} \cup \{(q, \epsilon, s) : q \in T_1\},$$

and  $s \notin Q_1$ .

Then  $L(M) = L(M_1)^*$ .

4. If  $M$  is a *deterministic* automaton  $(Q, \Sigma, s, T, \Delta)$ , then  $(Q, \Sigma, s, Q \setminus T, \Delta)$  accepts the language  $\Sigma^* \setminus L(M)$ .

5. Intersection can be expressed in terms of union and complement, i.e.,

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}.$$

Hence, closure under intersection follows from previous closure properties.