Deterministic Finite Automaton and Non-deterministic Finite Automaton

DFA and NFA

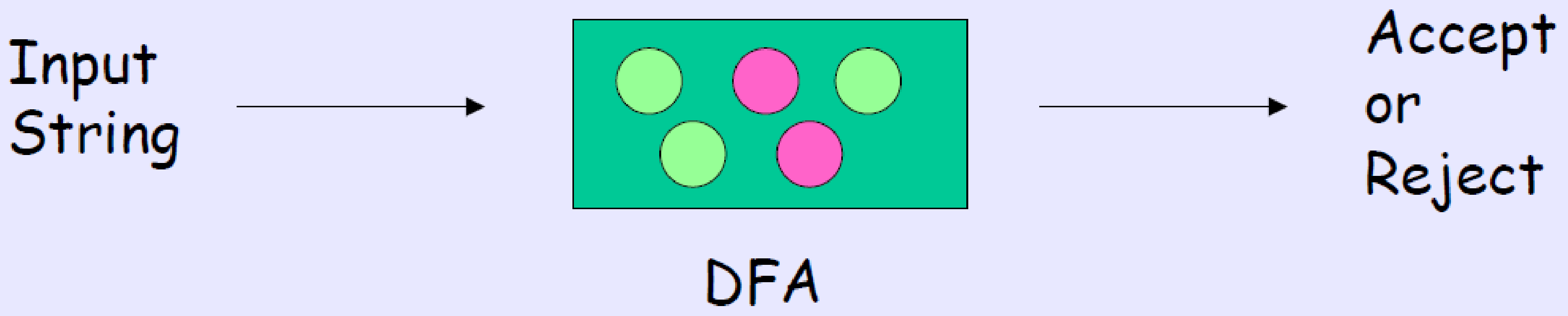
(Finite State Machines)

Notation and Definitions

- Alphabet
- String
- Language
- Operations on languages

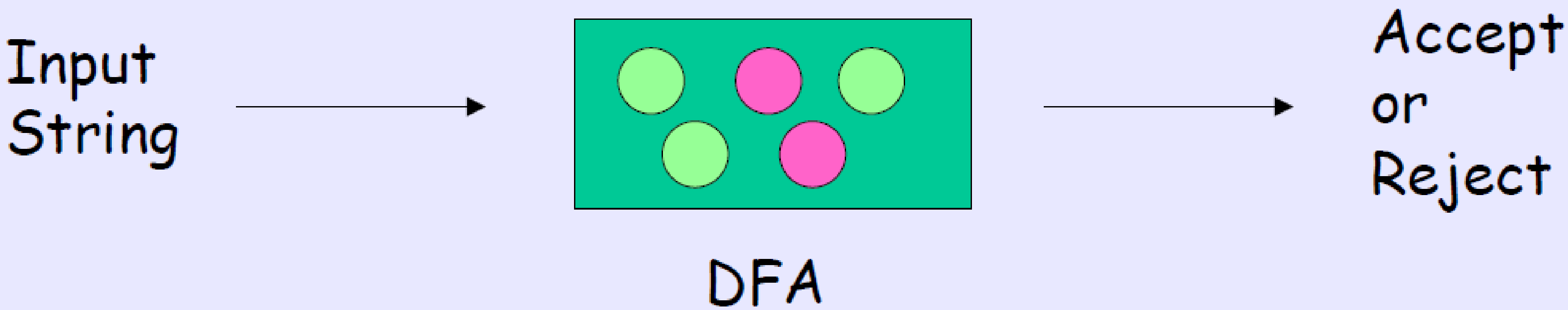
Finite State Machines

DFA



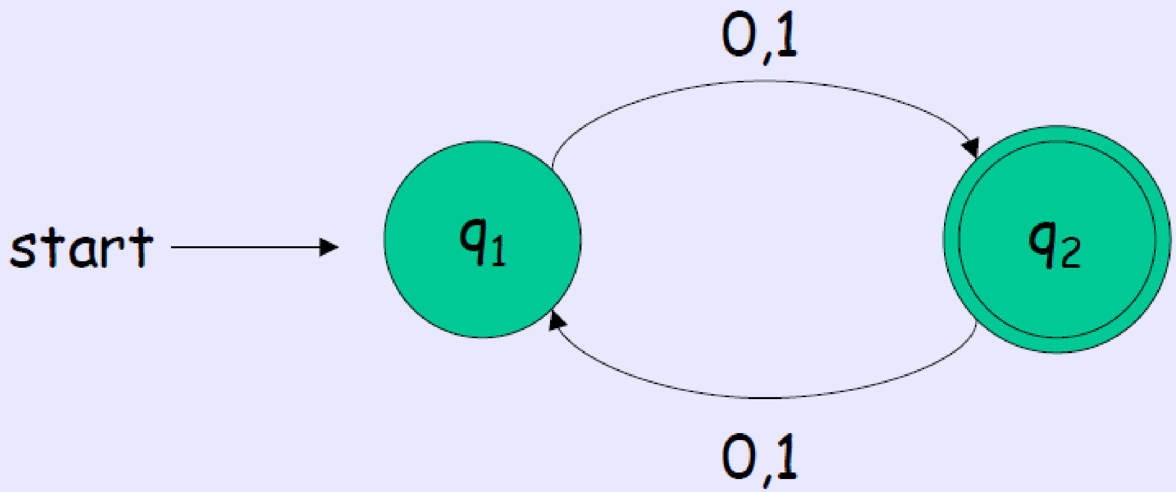
- A machine with finite number of states, some states are accepting states, others are rejecting states
- At any time, it is in one of the states
- It reads an input string, one character at a time

DFA



- After reading each character, it moves to another state depending on what is read and what is the current state
- If reading all characters, the DFA is in an accepting state, the input string is accepted.
- Otherwise, the input string is rejected.

Example of DFA



- The circles indicates the states
- If accepting state is marked with double circle
- The arrows pointing from a state q indicates how to move on reading a character when current state is q

Formal Definition of DFA

- A DFA is a 5-tuple $(Q, \Sigma, \delta, q_{start}, F)$, where
  - Q is a set consisting finite number of states
  - Σ is an alphabet consisting finite number of characters
  - $\delta: Q \times \Sigma \rightarrow Q$ is the transition function
  - q_{start} is the start state
  - F is the set of accepting states
- Note: only 1 start state, and can have many accepting states

Some Terminology

Let M be a DFA

- Among all possible strings, M will accept some of them, and M will reject the remaining
- The set of strings which M accepts is called the language recognized by M
- That is, M recognizes A if

$$A = \{ w \mid M \text{ accepts } w \}$$

Designing a DFA (Quick Quiz)

- How to design a DFA that accepts all binary strings representing a multiple of 5? (E.g., 101, 1111, 11001, ...)

DFA for complement of a language

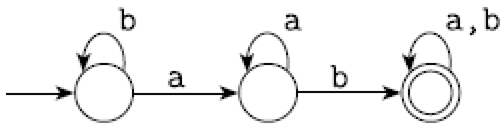
- Flip final and non-final states.

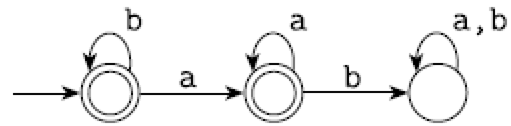
1.5 Each of the following languages is the complement of a simpler language. In each part, construct a DFA for the simpler language, then use it to give the state diagram of a DFA for the language given. In all parts, $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

[A] **a.** $\{w \mid w \text{ does not contain the substring } \mathtt{ab}\}$

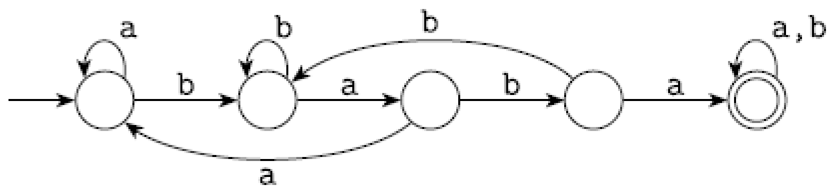
[A] **b.** $\{w \mid w \text{ does not contain the substring } \mathtt{baba}\}$

1.5 (a) The left-hand DFA recognizes $\{w \mid w \text{ contains ab}\}$. The right-hand DFA recognizes its complement, $\{w \mid w \text{ doesn't contain ab}\}$.

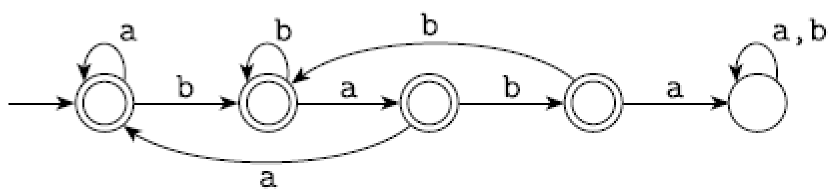




(b) This DFA recognizes $\{w \mid w \text{ contains baba}\}$.



This DFA recognizes $\{w \mid w \text{ does not contain baba}\}$.



$L(M)$

If A is the set of all strings that machine M accepts, we say that A is the ***language of machine M*** and write $L(M) = A$. We say that ***M recognizes A*** or that ***M accepts A***.

- Can you find what is the language accepted by M_2

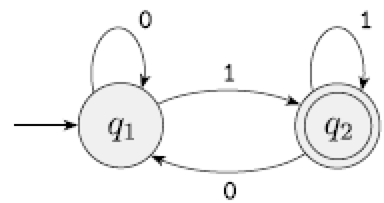


FIGURE 1.8
State diagram of the two-state finite automaton M_2

In the formal description, M_2 is $(\{q_1, q_2\}, \{0,1\}, \delta, q_1, \{q_2\})$. The transition function δ is

	0	1
q_1	q_1	q_2
q_2	q_1	q_2.

$L(M_2) = \{w \mid w \text{ ends in a } \mathtt{1}\}.$

- Can you find $L(M_4)$

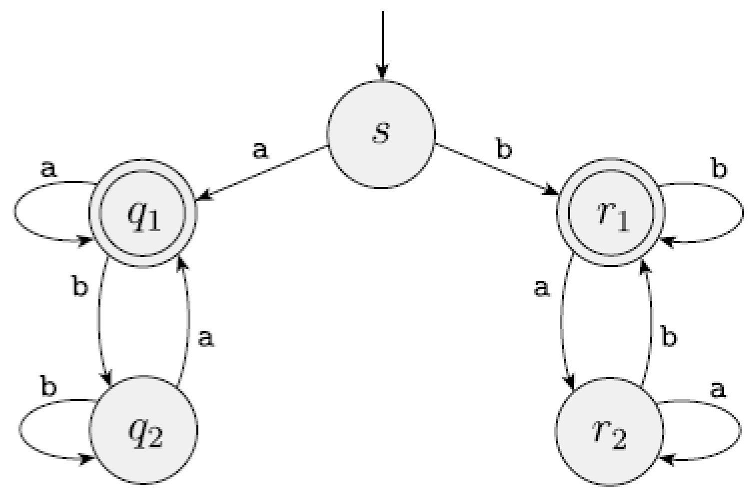


FIGURE 1.12
Finite automaton M_4

- $L(M_4) = \{w \in \Sigma^* | w \text{ starts and ends in same symbol}\}$

Formally

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,

2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and

3. $r_n \in F$.

Regular language [Ref: Sipser Book]

DEFINITION 1.16

A language is called a ***regular language*** if some finite automaton recognizes it.

The regular operations

DEFINITION 1.23

Let A and B be languages. We define the regular operations ***union***, ***concatenation***, and ***star*** as follows:

- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

- These are similar to arithmetic operations.
- Note, * is a unary operator.

THEOREM 1.25

The class of regular languages is closed under the union operation.

In other words, if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

- The proof is by construction.
- We build a DFA for the union from the individual DFAs.

- The idea is simple: While reading the input simultaneously follow both machines.
  - Put a finger on current state. You need two fingers. You can move these two fingers as per the respective transition function.

PROOF

Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and
M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$.

1. $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
   This set is the ***Cartesian product*** of sets Q_1 and Q_2 and is written $Q_1 \times Q_2$.
   It is the set of all pairs of states, the first from Q_1 and the second from Q_2.

2. Σ, the alphabet, is the same as in M_1 and M_2. In this theorem and in all subsequent similar theorems, we assume for simplicity that both M_1 and M_2 have the same input alphabet Σ. The theorem remains true if they have different alphabets, Σ_1 and Σ_2. We would then modify the proof to let $\Sigma = \Sigma_1 \cup \Sigma_2$.

3. δ, the transition function, is defined as follows. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let

$$\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big).$$

Hence δ gets a state of M (which actually is a pair of states from M_1 and M_2), together with an input symbol, and returns M's next state.

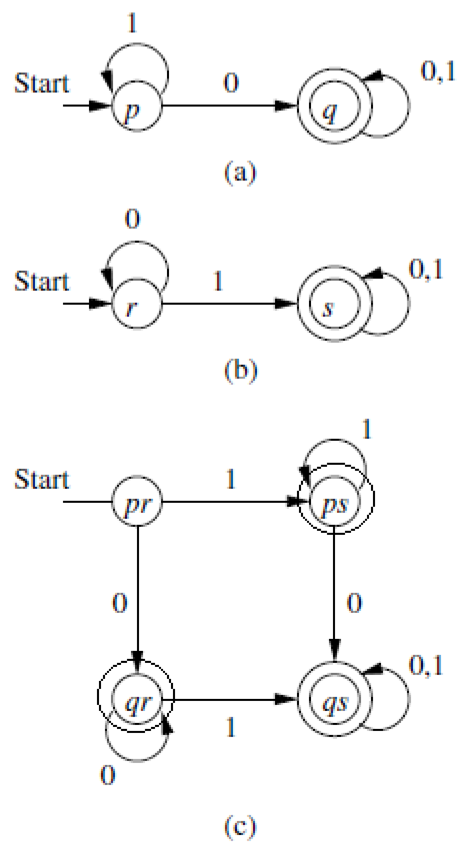
4. q_0 is the pair (q_1, q_2).

5. F is the set of pairs in which either member is an accept state of M_1 or M_2. We can write it as

$$F = \{(r_1, r_2) | \; r_1 \in F_1 \text{ or } r_2 \in F_2\}.$$

This expression is the same as $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. (Note that it is *not* the same as $F = F_1 \times F_2$. What would that give us instead?[3])

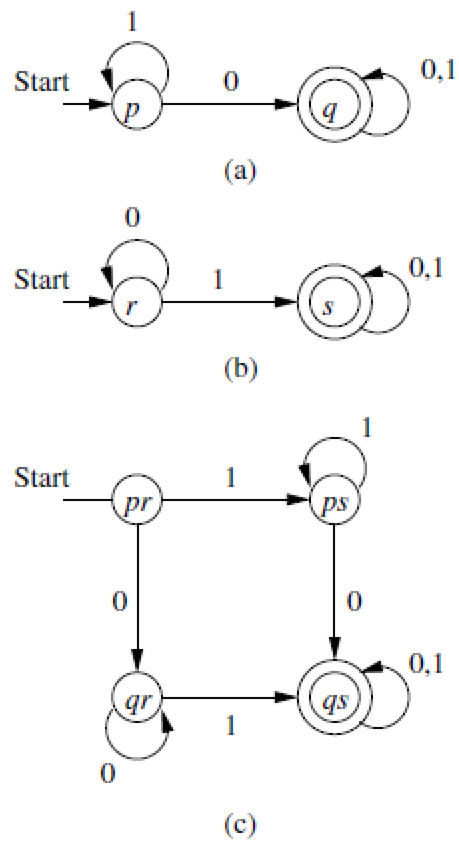
Union Example



What about intersection?

- Intersection of two regular languages is also regular.
- Proof: by construction. Similar. Only final states will change.

Intersection



What else we can do with product principle?

- Set difference.
  - How?

1.4 Each of the following languages is the intersection of two simpler languages. In each part, construct DFAs for the simpler languages, then combine them using the construction discussed in footnote 3 (page 46) to give the state diagram of a DFA for the language given. In all parts, $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

a. $\{w \mid w$ has at least three a's and at least two b's$\}$

[A]b. $\{w \mid w$ has exactly two a's and at least two b's$\}$

c. $\{w \mid w$ has an even number of a's and one or two b's$\}$

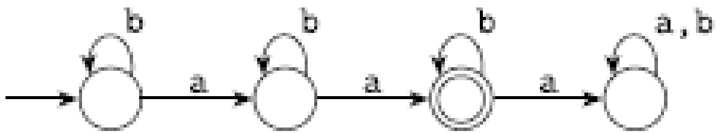
[A]d. $\{w \mid w$ has an even number of a's and each a is followed by at least one b$\}$

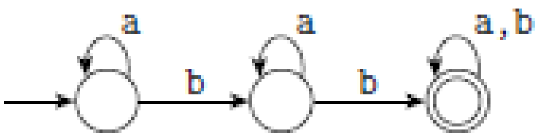
e. $\{w \mid w$ starts with an a and has at most one b$\}$

f. $\{w \mid w$ has an odd number of a's and ends with a b$\}$

g. $\{w \mid w$ has even length and an odd number of a's$\}$

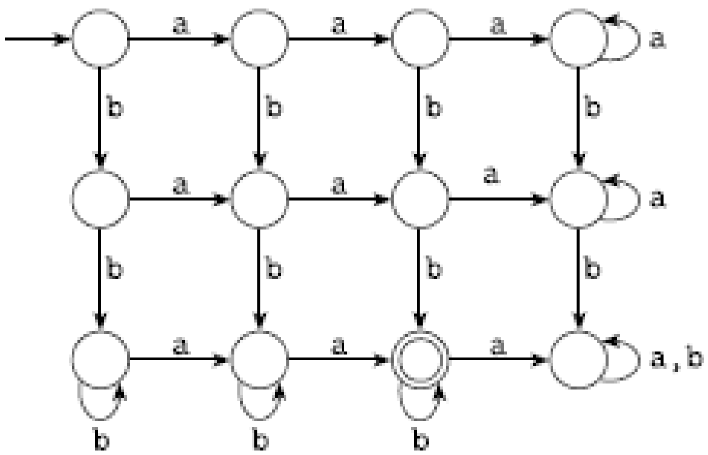
1.4 (b) The following are DFAs for the two languages $\{w \mid w \text{ has exactly two a's}\}$ and $\{w \mid w \text{ has at least two b's}\}$.





- Now find product machine.

Combining them using the intersection construction gives the following DFA.



- This can be minimized. {Some states are redundant}.

NONDETERMINISM

- Useful concept, has great impact on ToC.
- DFA is deterministic: every step of a computation follows in a unique way from the preceding step.
  - When the machine is in a given state, and upon reading the next input symbol, we know deterministically what would be the next state.
  - Only one next state.
  - No choice !!

NONDETERMINISM

- In a nondeterministic machine, several choices may exist for the next state at any point.
- Nondeterminism is a generalization of determinism.

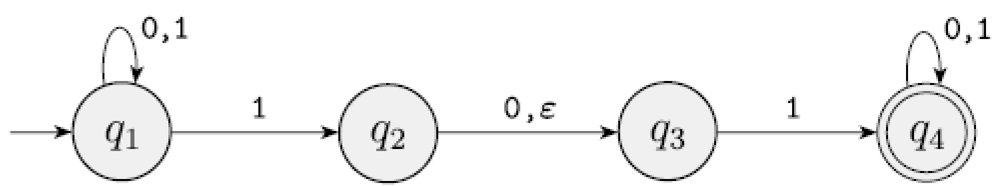


FIGURE **1.27**
The nondeterministic finite automaton N_1

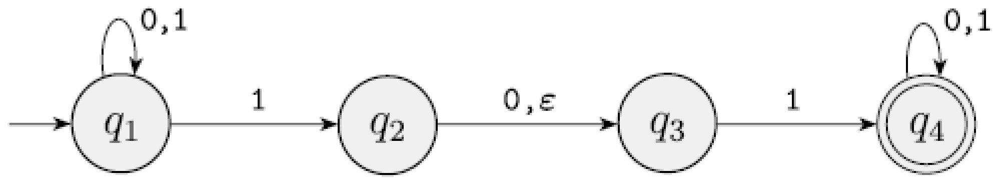


FIGURE **1.27**
The nondeterministic finite automaton N_1

- More than one arrow from from q_1 on symbol 1.
- No arrow at all from q_3 on 0.
- There is Ɛ over an arrow !

How does an NFA compute?

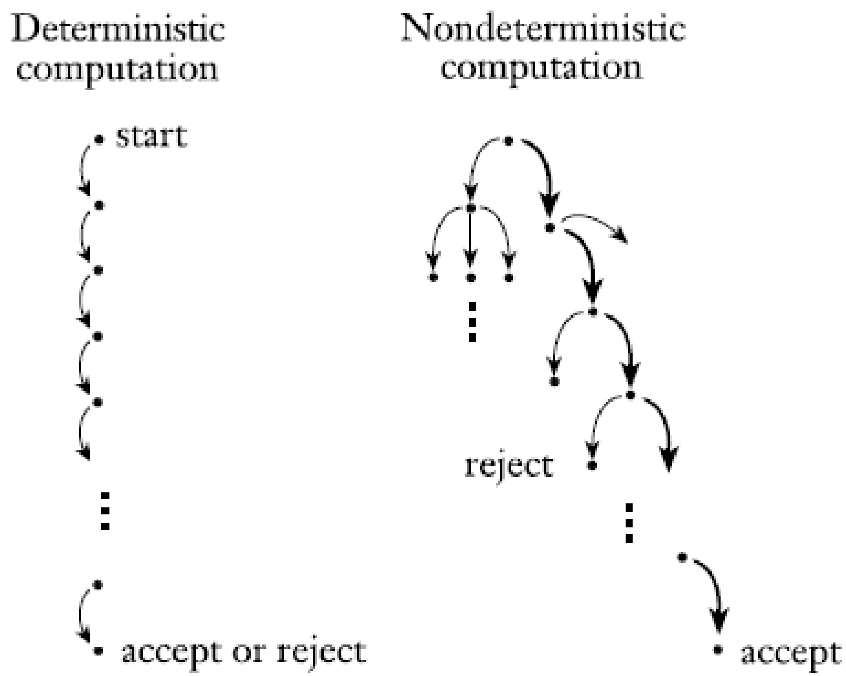


FIGURE 1.28
Deterministic and nondeterministic computations with an accepting branch

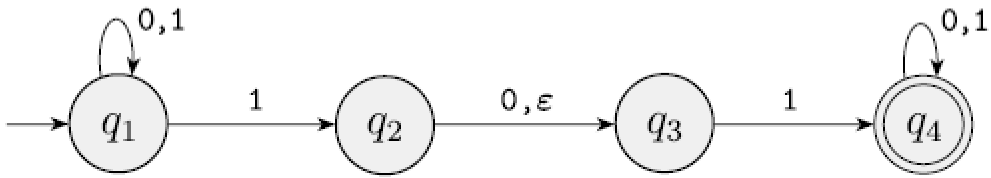


FIGURE 1.27
The nondeterministic finite automaton N_1

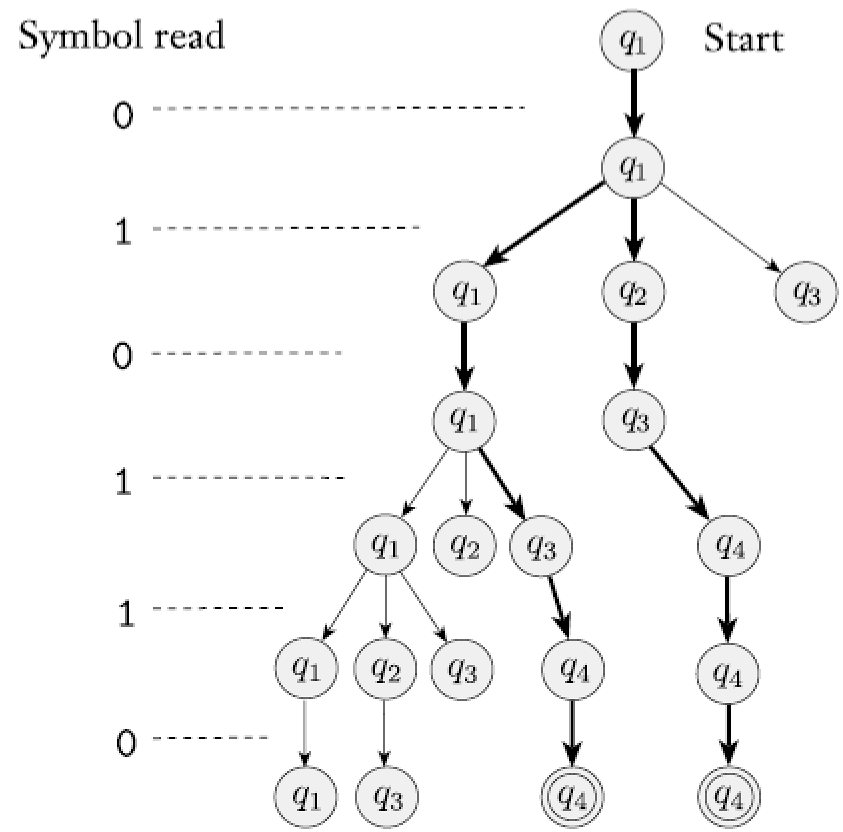


FIGURE 1.29
The computation of N_1 on input 010110

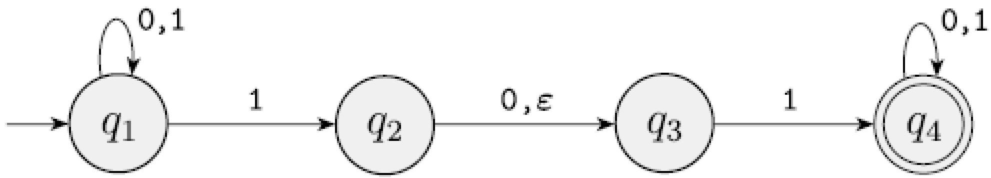


FIGURE 1.27
The nondeterministic finite automaton N_1

- What is the language accepted by this NFA?

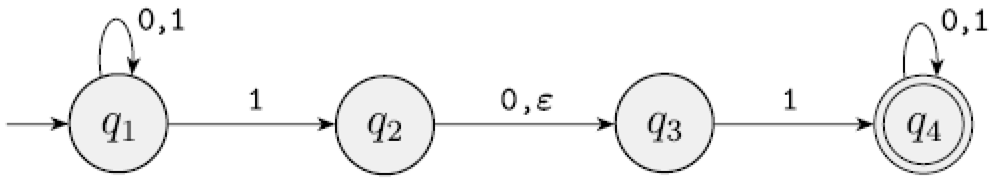


FIGURE 1.27
The nondeterministic finite automaton N_1

- It accepts all strings that contain either 101 or 11 as a substring.

- Constructing NFAs is sometimes easier than constructing DFAs.
  - Later we see that every NFA can be converted into an equivalent DFA.

EXAMPLE 1.30

Let A be the language consisting of all strings over $\{0,1\}$ containing a 1 in the third position from the end (e.g., 000100 is in A but 0011 is not). The following four-state NFA N_2 recognizes A.

- Building DFA for this is possible, but difficult.
- Try this.

But NFA is easy to build.

EXAMPLE 1.30

Let A be the language consisting of all strings over $\{0,1\}$ containing a 1 in the third position from the end (e.g., 000100 is in A but 0011 is not). The following four-state NFA N_2 recognizes A.

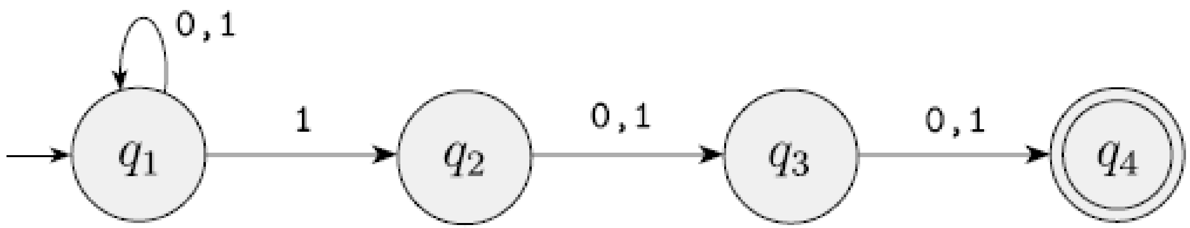


FIGURE 1.31
The NFA N_2 recognizing A

DFA for A

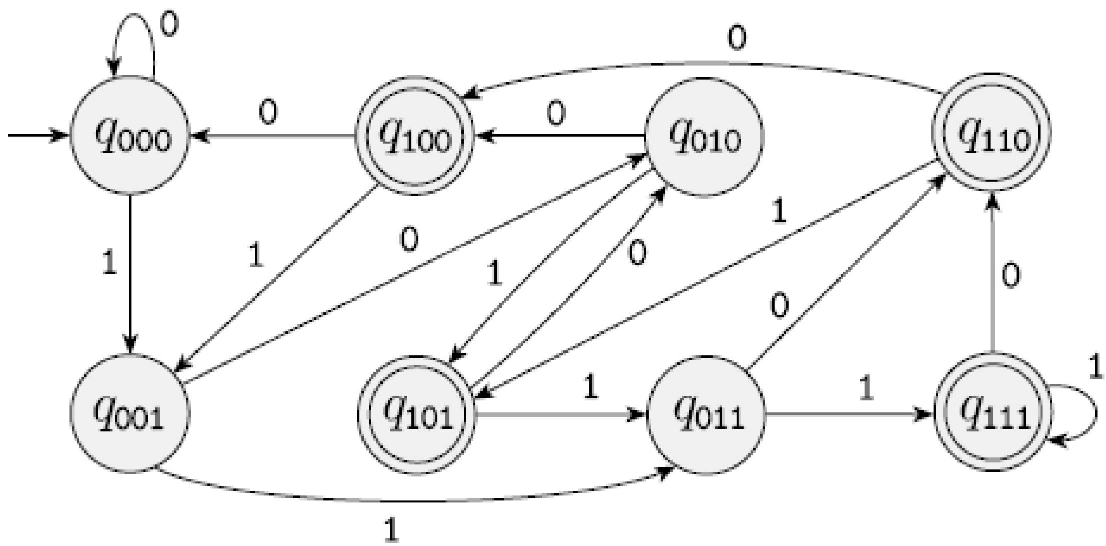


FIGURE 1.32
A DFA recognizing A

- See number of states and complexity !

Formal definition of NFA

We use Σ_ε to mean $\Sigma \cup \{\varepsilon\}$

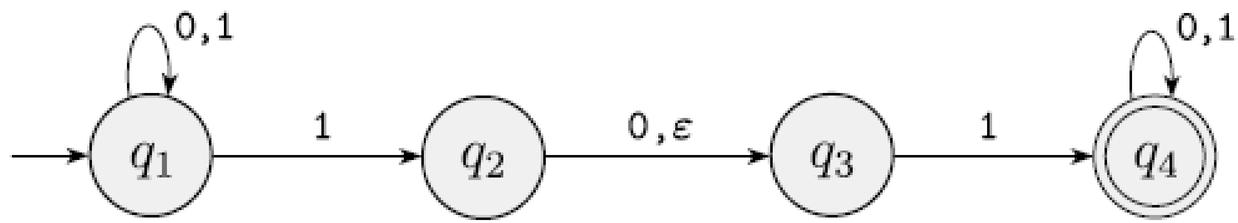
DEFINITION 1.37

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

EXAMPLE 1.38

Recall the NFA N_1:



The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

4. q_1 is the start state, and
5. $F = \{q_4\}$.

The formal definition of computation for an NFA is similar to that for a DFA. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over the alphabet Σ. Then we say that N ***accepts*** w if we can write w as $w = y_1 y_2 \cdots y_m$, where each y_i is a member of Σ_ε and a sequence of states $r_0, r_1, \ldots, r_m$ exists in Q with three conditions:

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, \ldots, m-1$, and
3. $r_m \in F$.

Equivalence of NFAs and DFAs

- We say two machines are equivalent if they recognize the same language.

THEOREM 1.39

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Proof

- Proof by construction.
  - We build a equal DFA for the given NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A.

We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A.

- First, for understanding purpose, we assume that there are no edges with Ɛ transitions.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A.

1. $Q' = \mathcal{P}(Q)$.
   Every state of M is a set of states of N. Recall that $\mathcal{P}(Q)$ is the set of subsets of Q.

2. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a)$ for some $r \in R\}$.

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).$$

3. $q_0' = \{q_0\}$.
   M starts in the state corresponding to the collection containing just the start state of N.

4. $F' = \{R \in Q' \mid R$ contains an accept state of $N\}$.
   The machine M accepts if one of the possible states that N could be in at this point is an accept state.

Can you convert the following

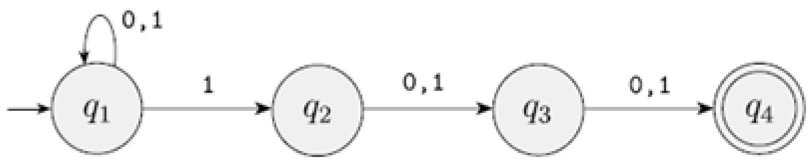


FIGURE 1.31
The NFA N_2

- What is the language accepted by this?

Now, considering Ɛ arrows

- For this purpose, we define Ɛ-CLOSURE of a set of states *R*.

Formally, for $R \subseteq Q$ let

$E(R) = \{q \mid q$ can be reached from R by traveling along 0 or more ε arrows$\}$.

- *E(R)* is Ɛ-CLOSURE of *R*.

- Then the transition is defined as,

  $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a))$ for some $r \in R\}$.

- Now the start state of the DFA should be

  $q_0{}' = E(\{q_0\})$

Example

-

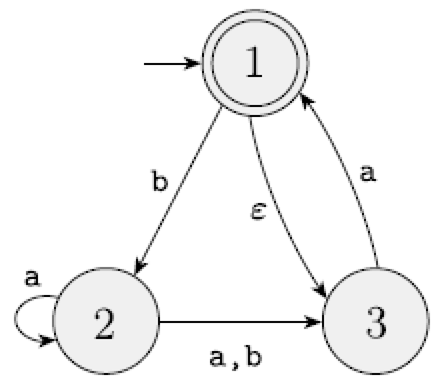
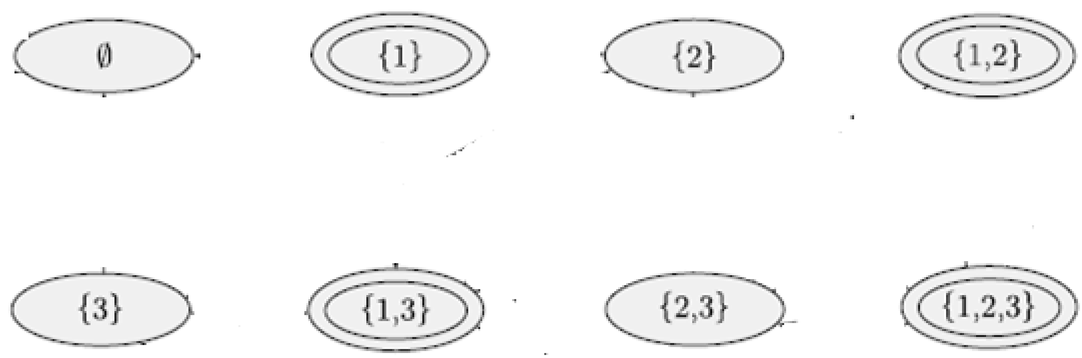


FIGURE 1.42
The NFA N_4



All possible states of the DFA.
(to be constructed; Final states are shown)

- Now we need to add edges, and
- identify the initial state.

•

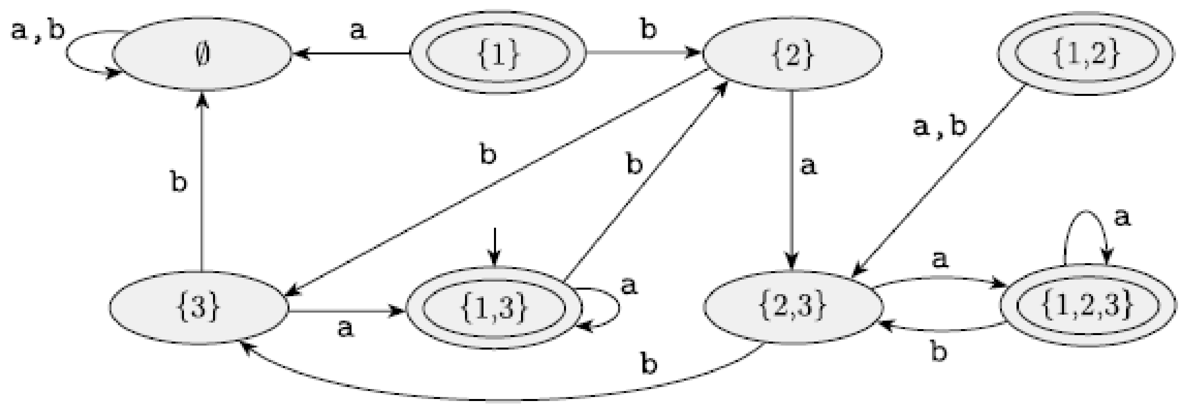


FIGURE 1.43
A DFA D that is equivalent to the NFA N_4

- But, some states are not reachable !
- Simplification can remove this.

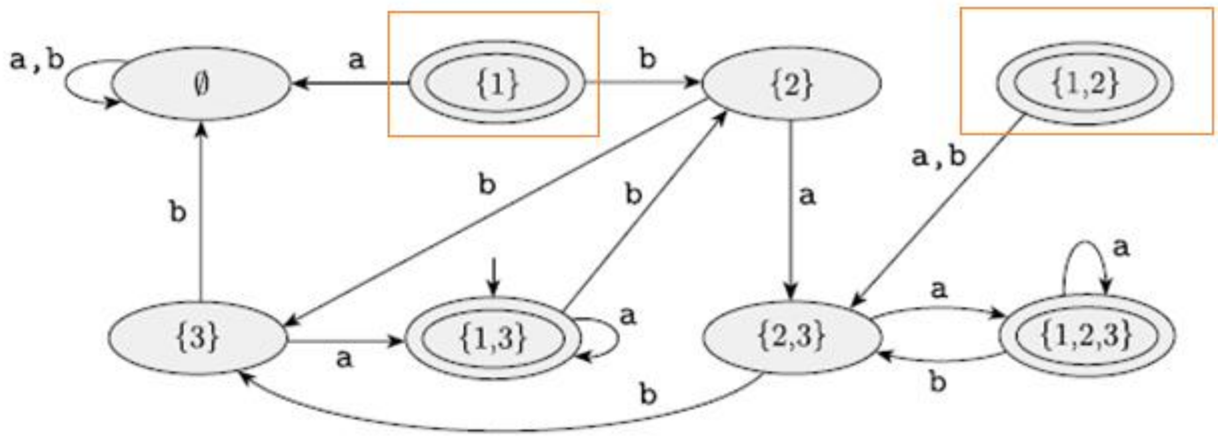


FIGURE 1.43
A DFA D that is equivalent to the NFA N_4

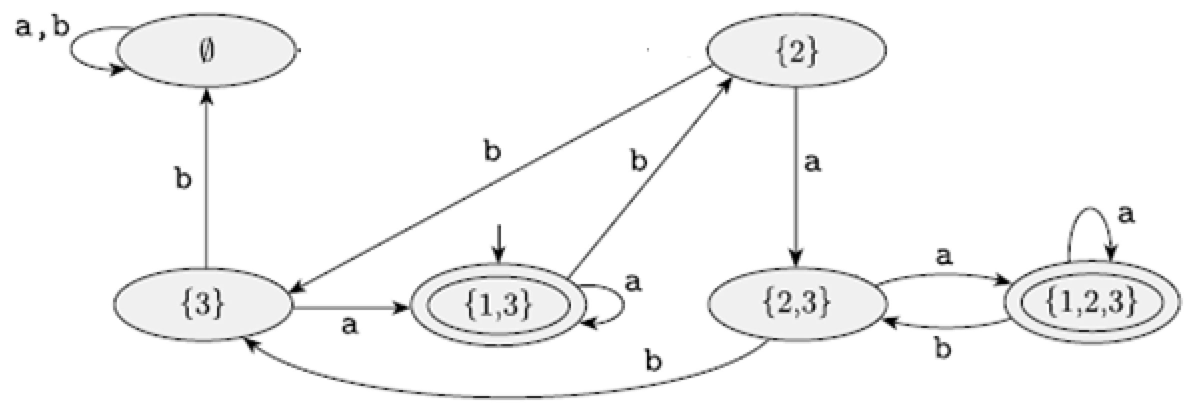
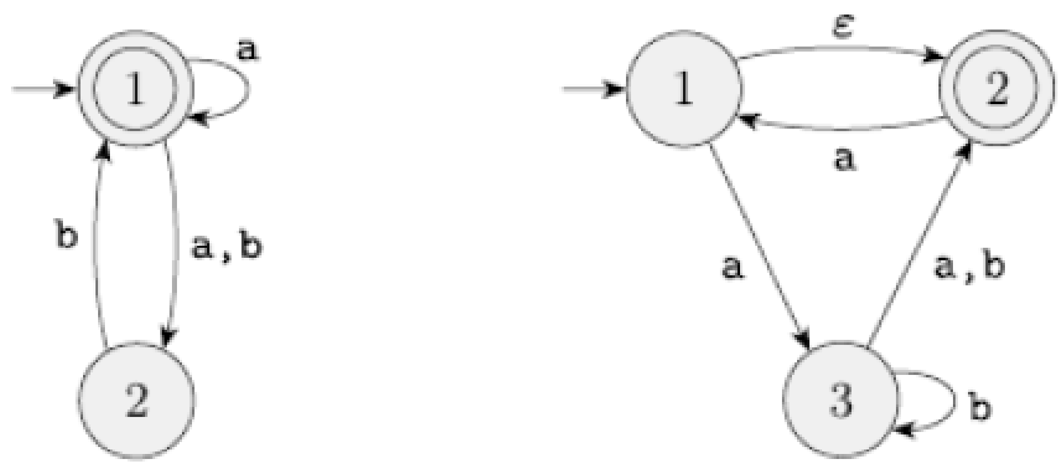


FIGURE 1.44
DFA D after removing unnecessary states

Exercise

Convert the following NFAs to equivalent DFAs.



(a) (b)

(Problem Source: Sipser's book exercise problem 1.16)

Exercise

- 1.7 Give state diagrams of NFAs with the specified number of states recognizing each of the following languages. In all parts, the alphabet is $\{0,1\}$.

  [A]**a.** The language $\{w \mid w \text{ ends with } 00\}$ with three states

  d. The language $\{0\}$ with two states

  g. The language $\{\varepsilon\}$ with one state

  h. The language 0^* with one state

- Can you convert each of above NFAs into a corresponding DFA.