NONDETERMINISM

- Useful concept, has great impact on ToC/algorithms.
- 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.

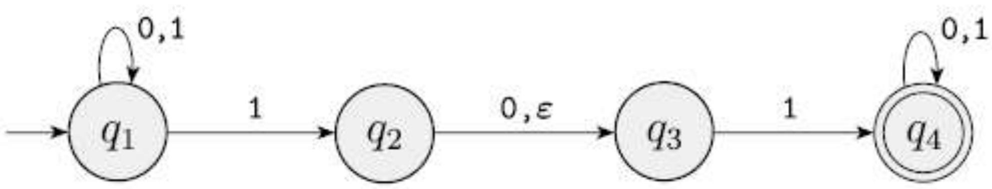


FIGURE 1.27
The nondeterministic finite automaton N_1

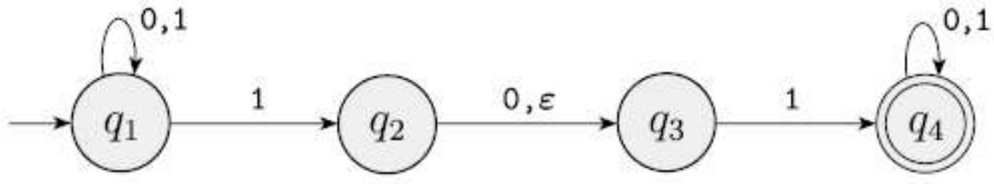


FIGURE 1.27
The nondeterministic finite automaton N_1

- 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?

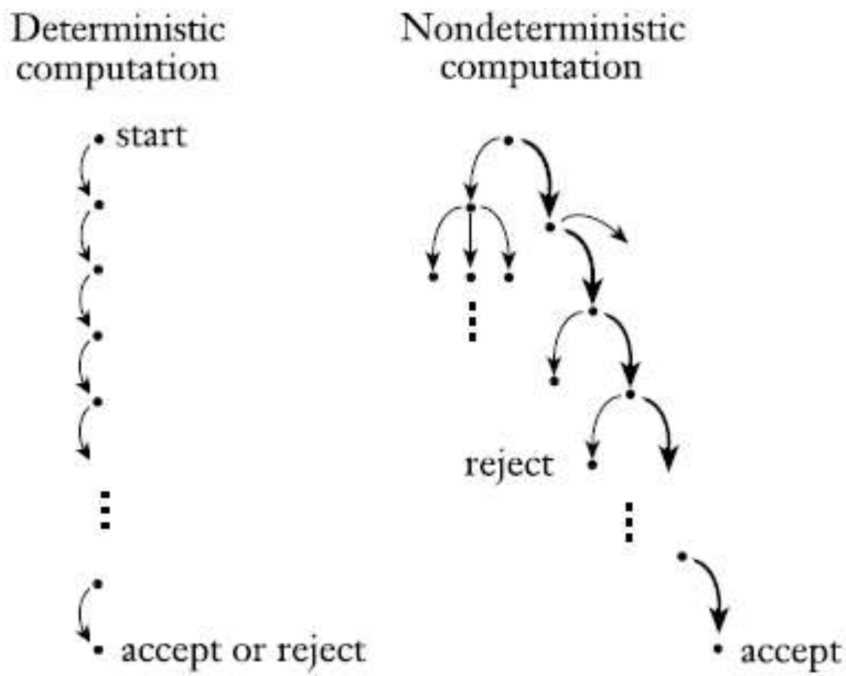


FIGURE **1.28**
Deterministic and nondeterministic computations with an accepting branch

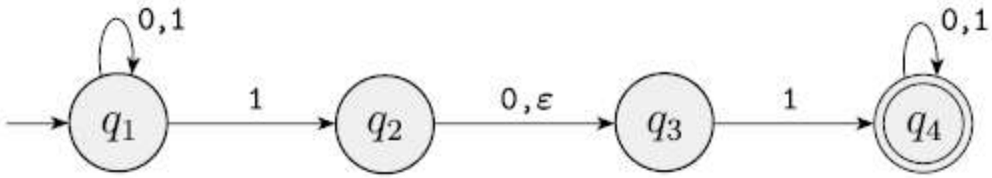


FIGURE 1.27
The nondeterministic finite automaton N_1

On input **010110**

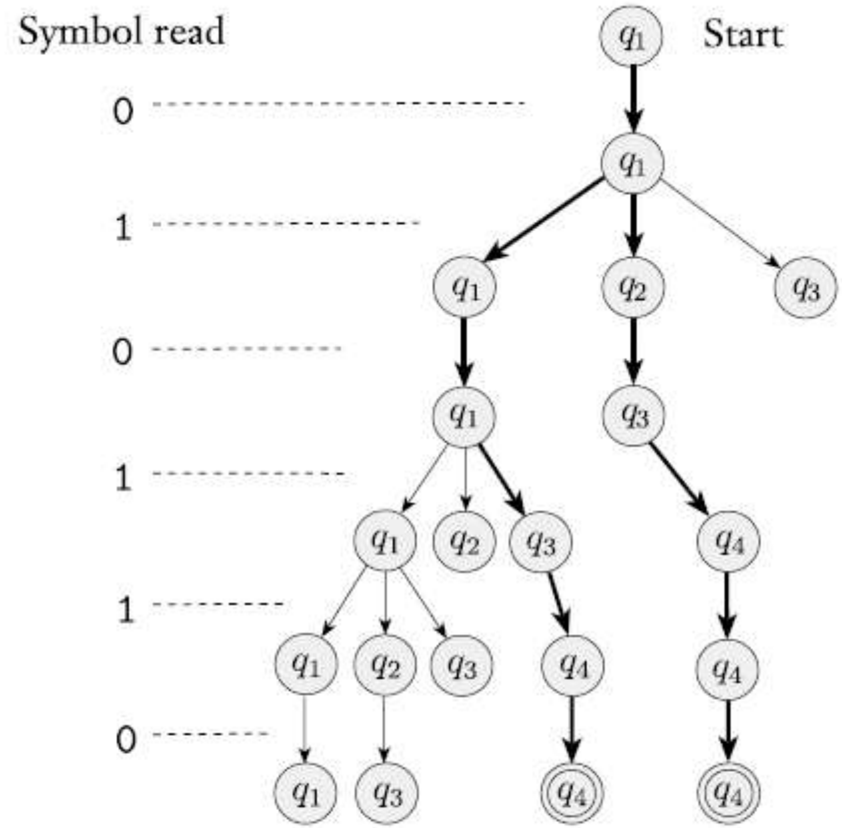


FIGURE 1.29
The computation of N_1 on input 010110

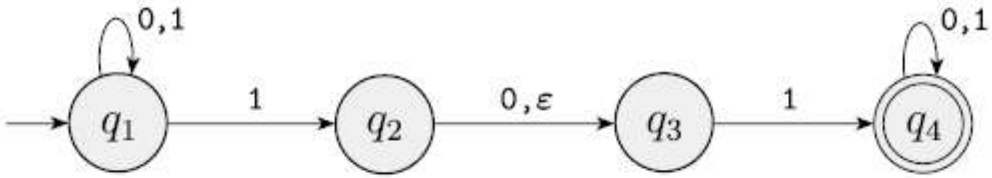


FIGURE 1.27
The nondeterministic finite automaton N_1

- What is the language accepted by this NFA?

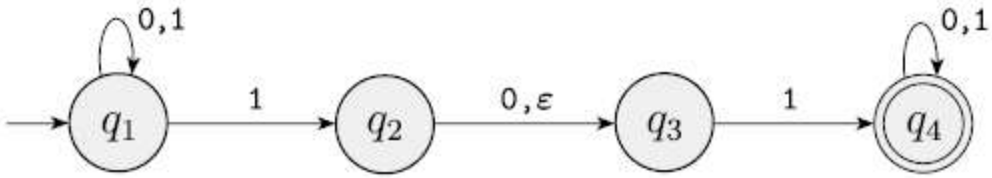


FIGURE **1.27**
The nondeterministic finite automaton N_1

- 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.

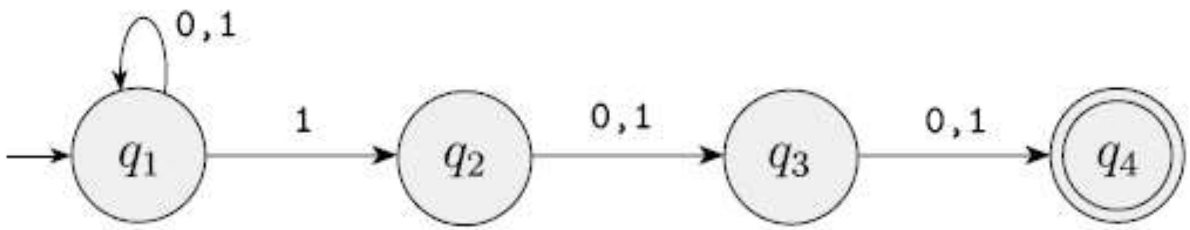


FIGURE 1.31
The NFA N_2 recognizing A

DFA for A

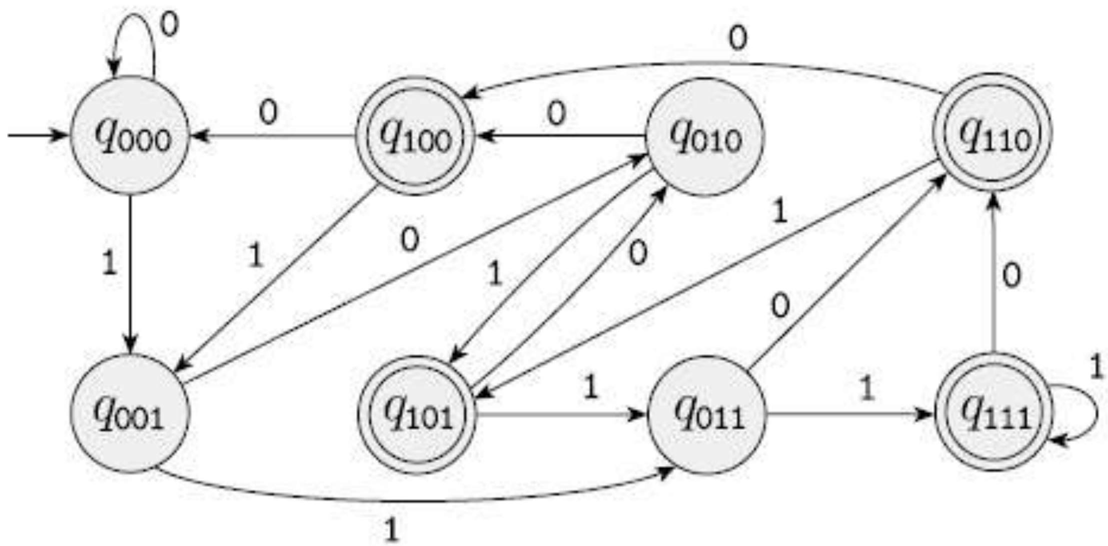


FIGURE 1.32
A DFA recognizing A

- See number of states and complexity !

Formal definition of NFA

We use $\sum_{\varepsilon}$ to mean $\sum \cup \{\varepsilon\}$

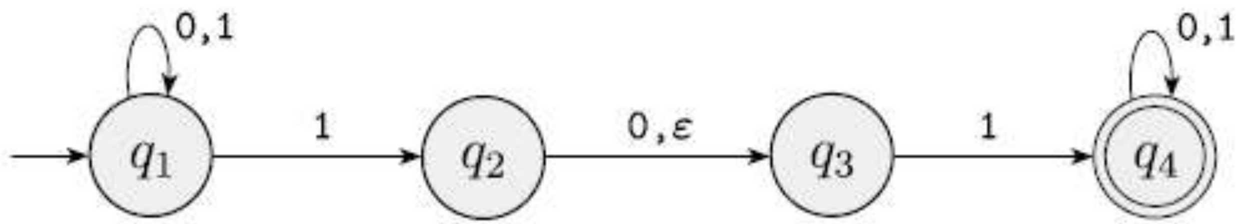
DEFINITION 1.37

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. $\delta \colon Q \times \Sigma_{\varepsilon} \longrightarrow \mathcal{P}(Q)$ 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 $\delta'(R, a) = \{q \in Q |\ q \in \delta(r, a) \text{ for some } r \in R\}$.

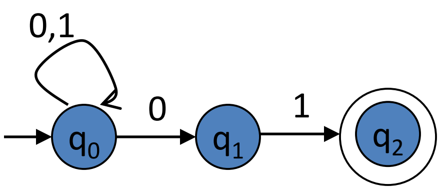
$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).$$

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

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

Subset construction: Example

NFA:

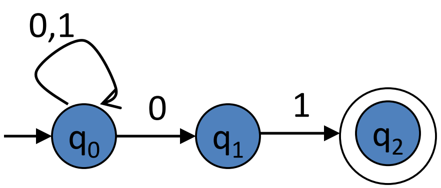


NFA to DFA construction: Example

- *L = {w | w ends in 01}*

Idea: To avoid enumerating all of power set, do "lazy creation of states"

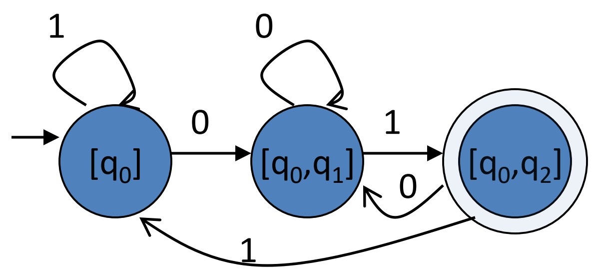
NFA:



δ_N	0	1
→q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
*q_2	Ø	Ø

DFA:

δ_D
~~Ø~~
→$\{q_0\}$
~~$\{q_1\}$~~
~~*$\{q_2\}$~~
$\{q_0,q_1\}$
*$\{q_0,q_2\}$
~~*$\{q_1,q_2\}$~~
~~*$\{q_0,q_1,q_2\}$~~



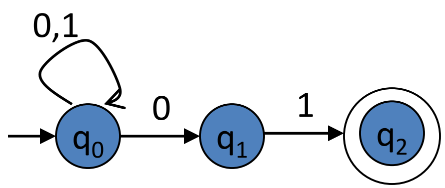
δ_D	0	1
→$[q_0]$	$[q_0,q_1]$	$[q_0]$
$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_2]$
*$[q_0,q_2]$	$[q_0,q_1]$	$[q_0]$

0. Enumerate all possible subsets
1. Determine transitions
2. Retain only those states reachable from $\{q_0\}$

NFA to DFA: Repeating the example using *LAZY CREATION*

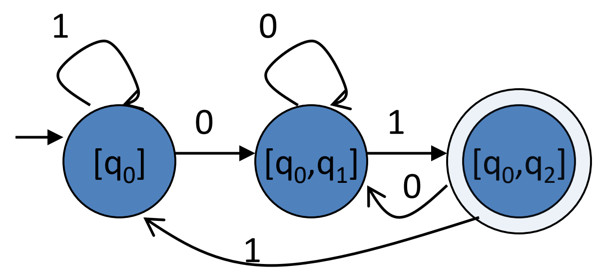
- *L = {w | w ends in 01}*

NFA:



δ_N	0	1
→ q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
$*q_2$	Ø	Ø

DFA:



δ_D	0	1
→ $[q_0]$	$[q_0,q_1]$	$[q_0]$

Main Idea:

Introduce states as you go (on a need basis)

Can you convert the following

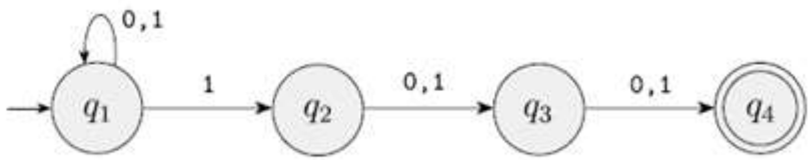


FIGURE 1.31
The NFA N_2

- What is the language accepted by this?