

# Are DFAs and NFAs Equivalent

It turns out DFAs and NFAs accept exactly the same languages.

To show this we must prove every DFA can be converted into an NFA which accepts the same language, and vice-versa

# Power of a class of machines

We can simulate DFA's by  $\mathcal{C}$  programs: For every DFA  $A$  over (say)  $\{0, 1\}$ , there exists a  $\mathcal{C}$  program which takes binary strings  $w$  as inputs and returns a boolean value `True` iff  $w \in L(A)$ .

Thus,  $\mathcal{C}$  programs are at least as powerful as DFAs (big surprise).

We will see later that the converse is not true: there exist  $\mathcal{C}$  programs whose languages of accepted strings are not the languages of any DFA.

# Comparing DFA and NFA

Here we will compare the powers of DFAs and NFAs. Since every DFA is at the same time an NFA, the latter are at least as powerful. Surprisingly, they are not more powerful.

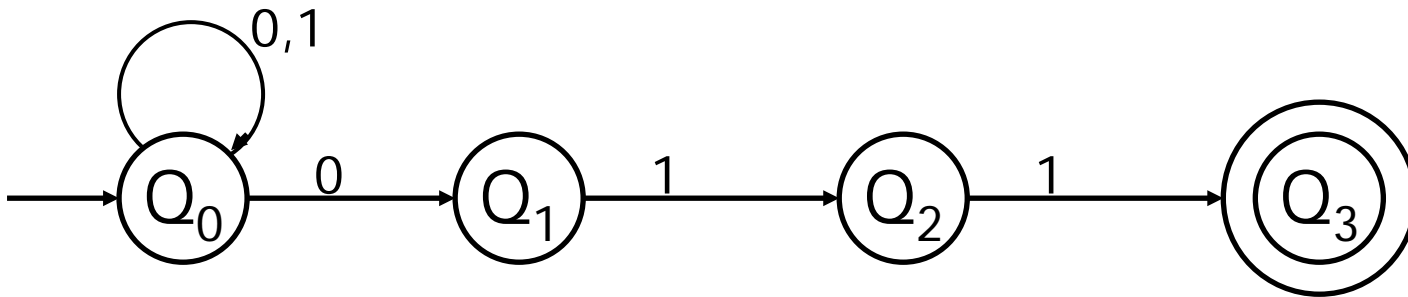
**Theorem.** For every NFA  $N$ , there exists a DFA  $D$  such that  $L(D) = L(N)$ .

*Thus, a language  $L$  is accepted by some NFA if and only if it is accepted by some DFA.*

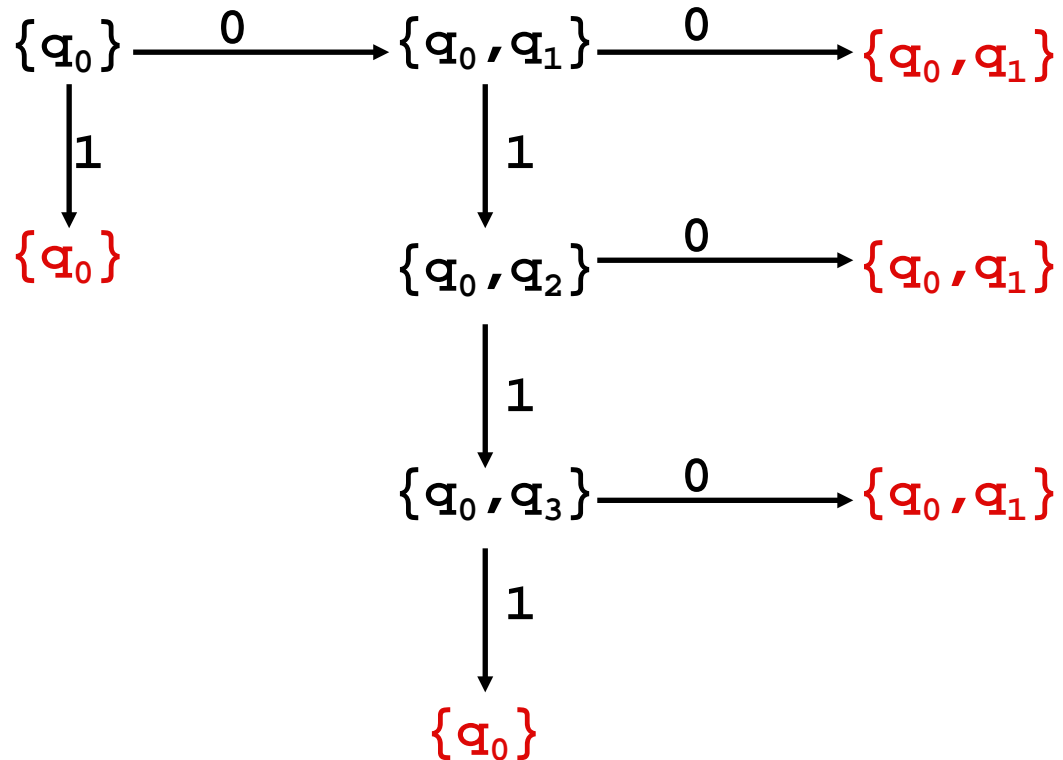
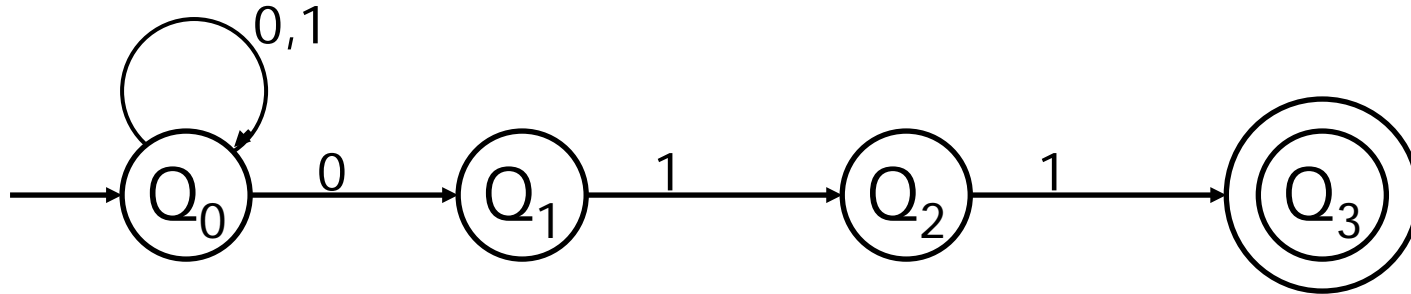
Given  $N$ , we can effectively construct the corresponding  $D$ .

# Example

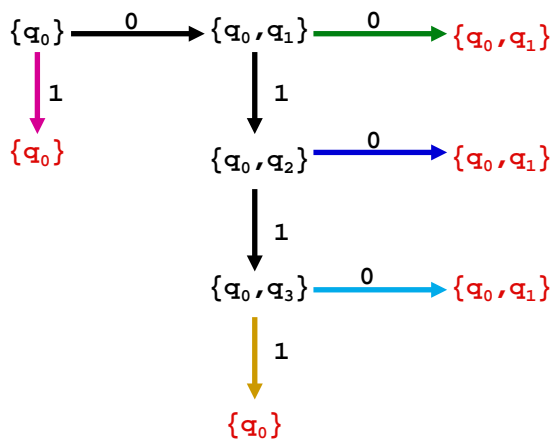
Consider the NFA that accepts binary strings ending with 011.



The key idea for building an equivalent DFA is to consider *the set of all states* this NFA can reach after reading any particular string. We'll examine all strings starting with the shortest.

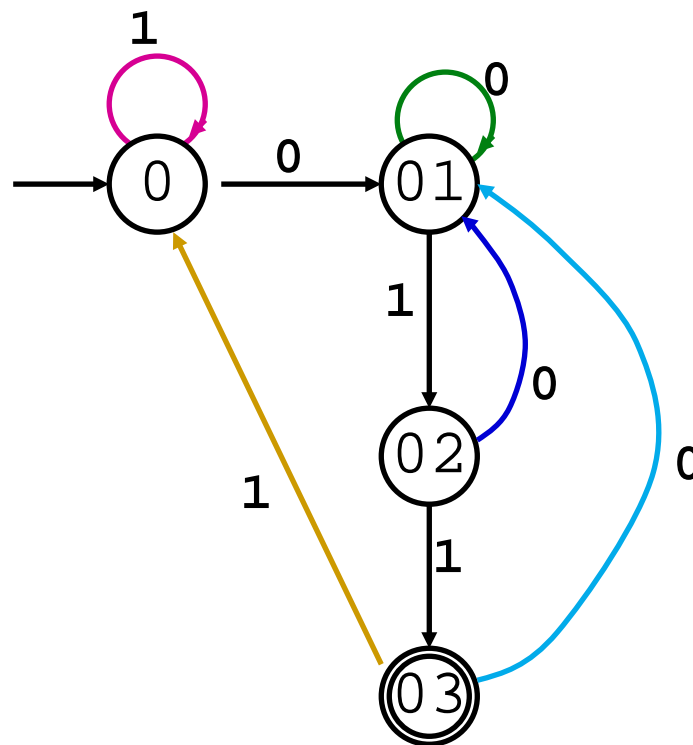


When processing if we see a set exactly the same as a set constructed earlier we mark it in red.



By “bending” the arrows to the **red** sets back to the first known set with those elements we construct a DFA.

Each state of the DFA corresponds to a set of states of the NFA



# General Construction

Given an NFA:

$$\mathbf{N} = (\mathbf{Q}, \Sigma, \mathbf{s}, \mathbf{F}, \Delta)$$

The associated DFA is

$$\mathbf{D} = (\mathbf{P}(\mathbf{Q}), \Sigma, \{\mathbf{s}\}, \mathbf{F}', \delta),$$

Where

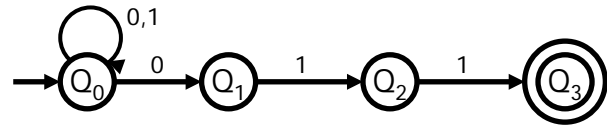
In the DFA constructed each state is labeled with a set of states from the NFA. Thus the **start state** is just the singleton set  $\{\mathbf{s}\}$

$\mathbf{F}'$  is the set of {subsets of  $\mathbf{Q}$ } that contain an element of  $\mathbf{F}$ . Thus  $\mathbf{F}' \subseteq \mathbf{P}(\mathbf{Q})$

$\delta$  is defined by  $\delta(\mathbf{s}, \mathbf{a}) = \bigcup_{\mathbf{q} \in \mathbf{s}} \Delta(\mathbf{q}, \mathbf{a})$

# Example

Let's compute two transitions of  $D$ , where  $N$  is as in the previous example.



$$\begin{aligned}\delta(\{q_0, q_2\}, 1) &= \Delta(q_0, 1) \cup \Delta(q_2, 1) \\ &= \{q_0\} \cup \{q_3\} \\ &= \{q_0, q_3\}\end{aligned}$$

$$\begin{aligned}\delta(\{q_0, q_1, q_3\}, 0) &= \Delta(q_0, 0) \cup \Delta(q_1, 0) \cup \Delta(q_3, 0) \\ &= \{q_0, q_1\} \cup \emptyset \cup \emptyset \\ &= \{q_0, q_1\}\end{aligned}$$



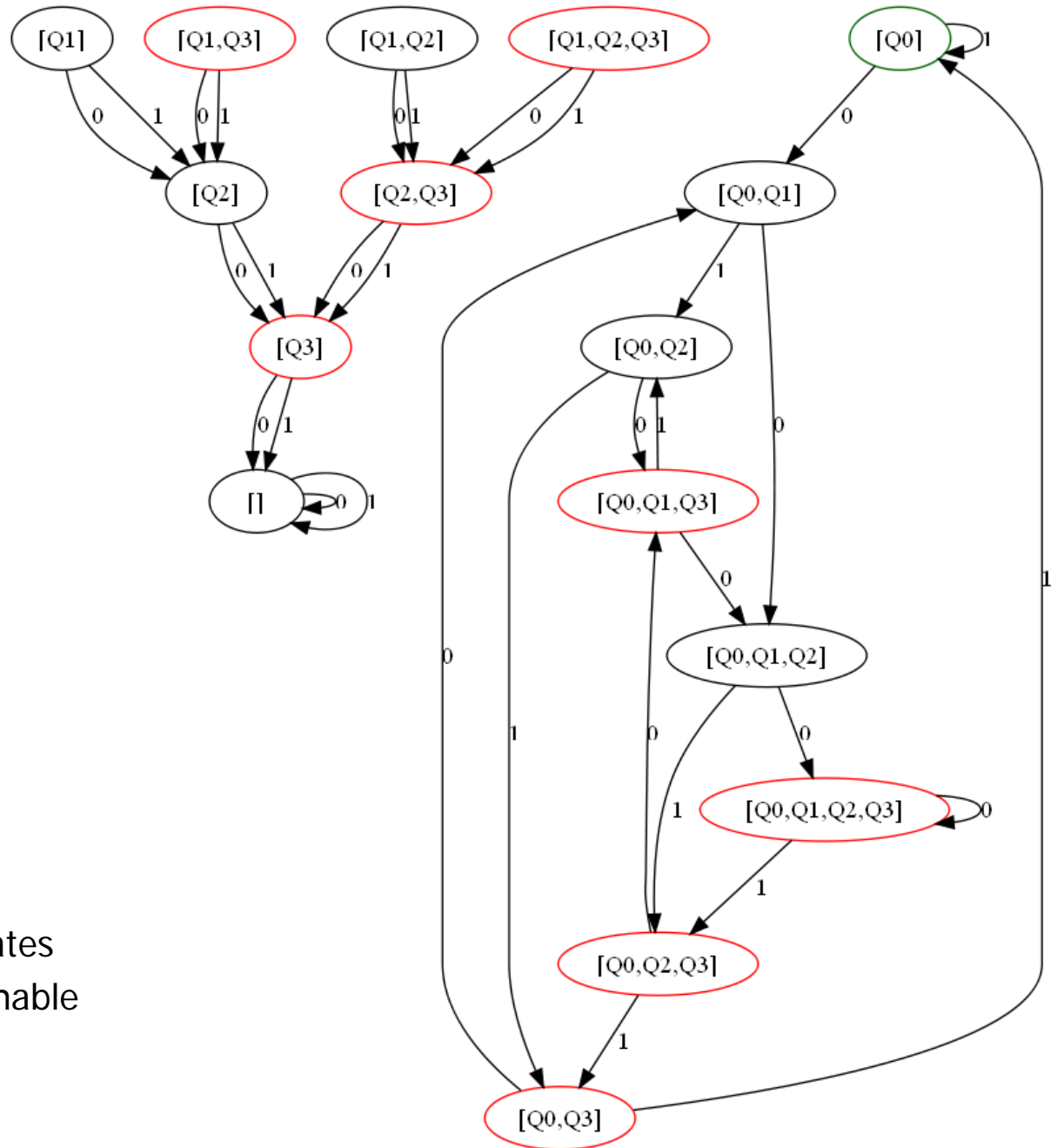
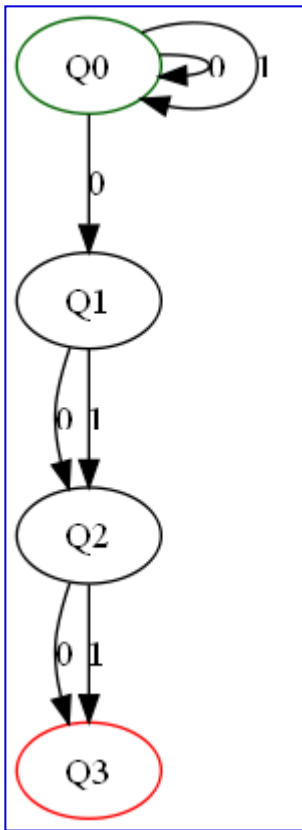
# Exponential Blowup

Note that if the NFA  $N$  has  $n$  states, then the corresponding DFA  $D$  has  $2^n$  states.

Many of those states can usually be discarded;

we must *keep only those states that are reachable from the initial state.*

There are cases, however, when there is no state to discard;



Note,  
 Original NFA has 4 states  
 The computed DFA has 16 states  
 Only some of the 16 are reachable  
 from the start state {Q0}