# DFA = NFA

Sipser pages 54-58

# 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

# Every DFA is an NFA

The first direction is trivial

**A DFA** is a quintuple **A** = (**Q**,**S**,**T**,$q_0$,**F**) where

- **Q** is a set of ***states***
- **S** is the **alphabet** (of *input symbols)*
- **T**: **Q** $\times$ S $\rightarrow$ **Q** is the ***transition function***
- $q_0 \in$ **Q** -- the ***start state***
- **F** $\subseteq$ **Q** -- ***final states***

An NFA is a quintuple **A= (Q,S,T,$q_0$,F)** , where

- **Q** is a set of ***states***
- **S** is the **alphabet** (of *input symbols)*
- **T**: **Q** $\times$ S $\rightarrow$ P(**Q**) is the ***transition function***
- $q_0 \in$ **Q** -- the ***start state***
- **F** $\subseteq$ **Q** -- ***final states***

Make a new transition function that returns a singleton set!

```
dfaToNfa (DFA states alphabet trans start accept)
       = (NFA states alphabet delta start accept)
  where delta s c = [trans s c]
```

# Every NFA is a DFA

Here we will compare the powers of DFAs and NFAs. Since every DFA can be turned into an equivalent NFA, the latter are at least as powerful. Surprisingly, they are not more powerful.

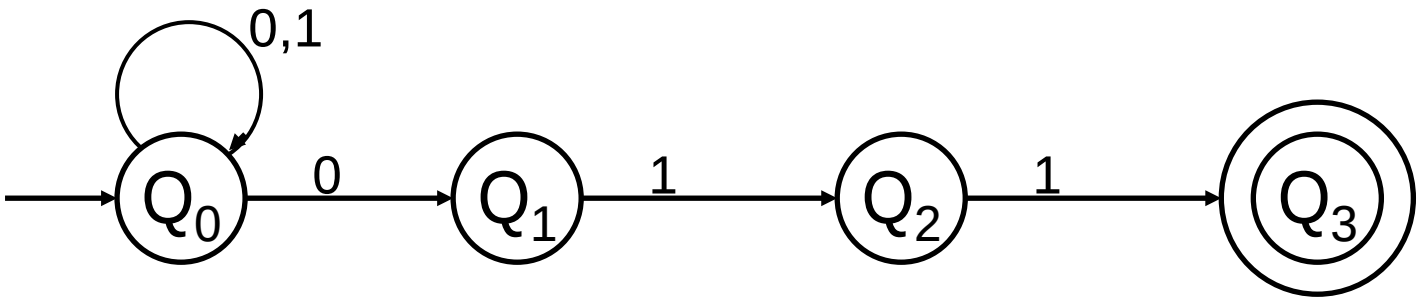
**Theorem.** For every NFA $N$, there exists a DFA $D$ such that $L(D)=L(N)$.

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 $D$.

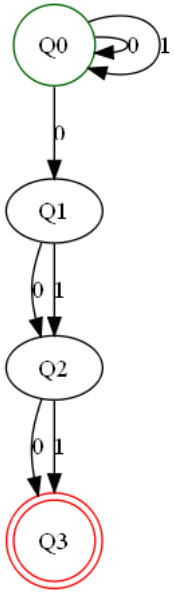
# Example

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.

# Consider processing “0011”



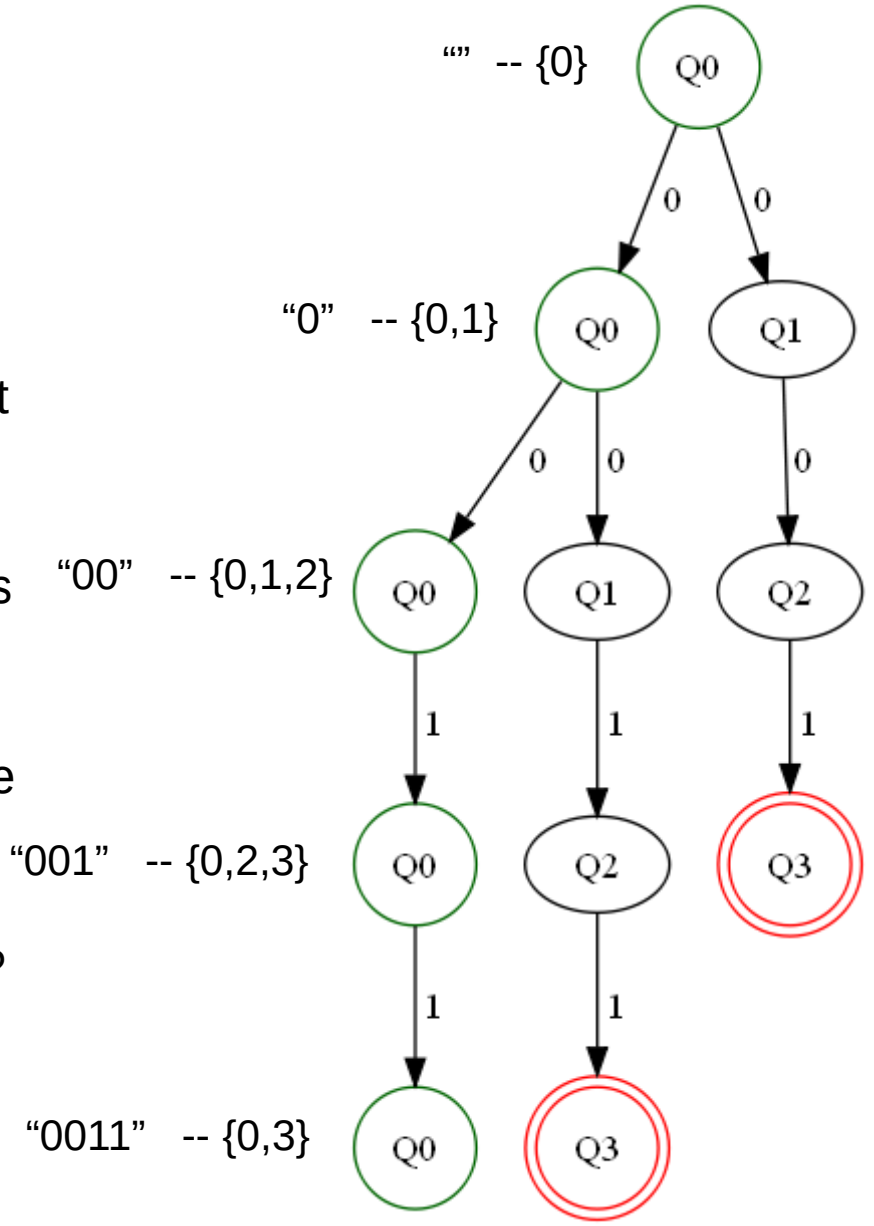

One possible strategy is motivated by the path tree.

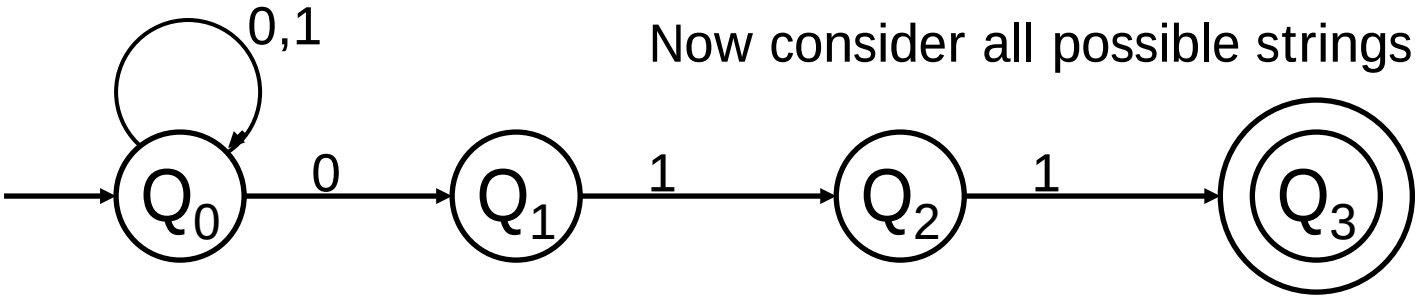
For each prefix keep track of the set of states the system might be in.

How can we compute a set of states from the transition function?

How can we extend the prefix to the next character in the input?

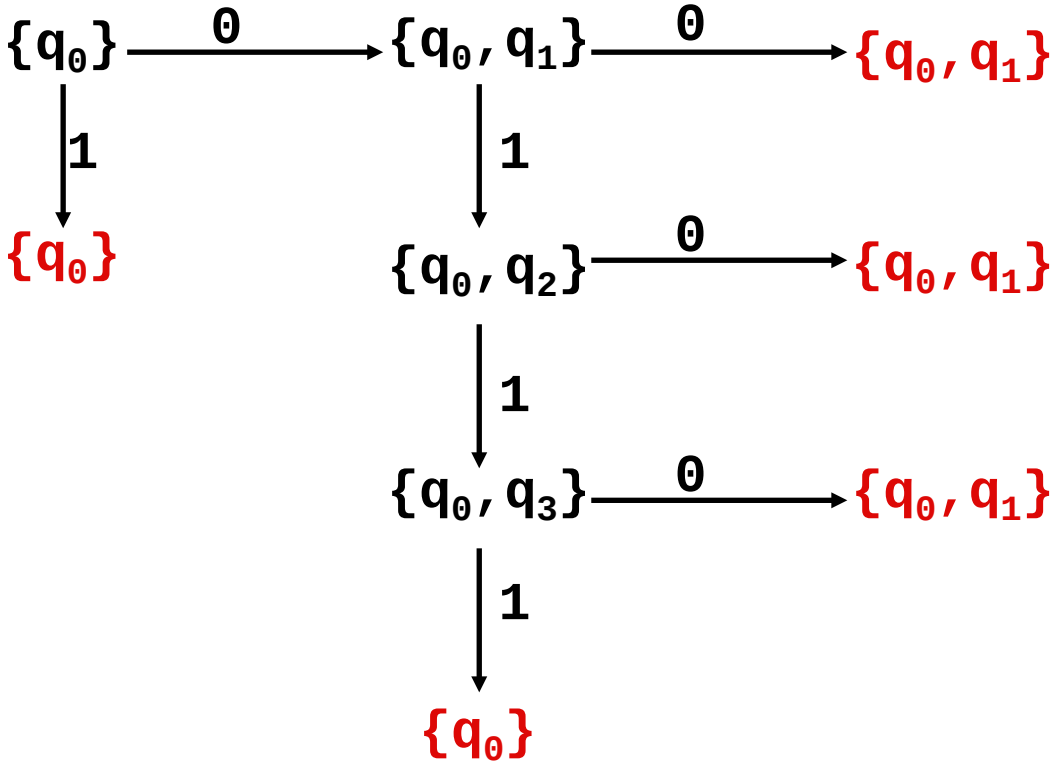
How do we know when we’re done?



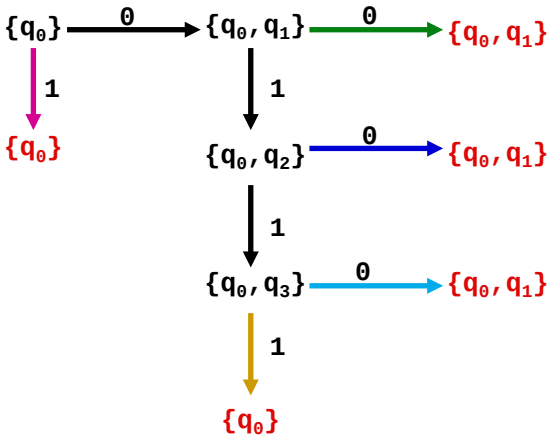


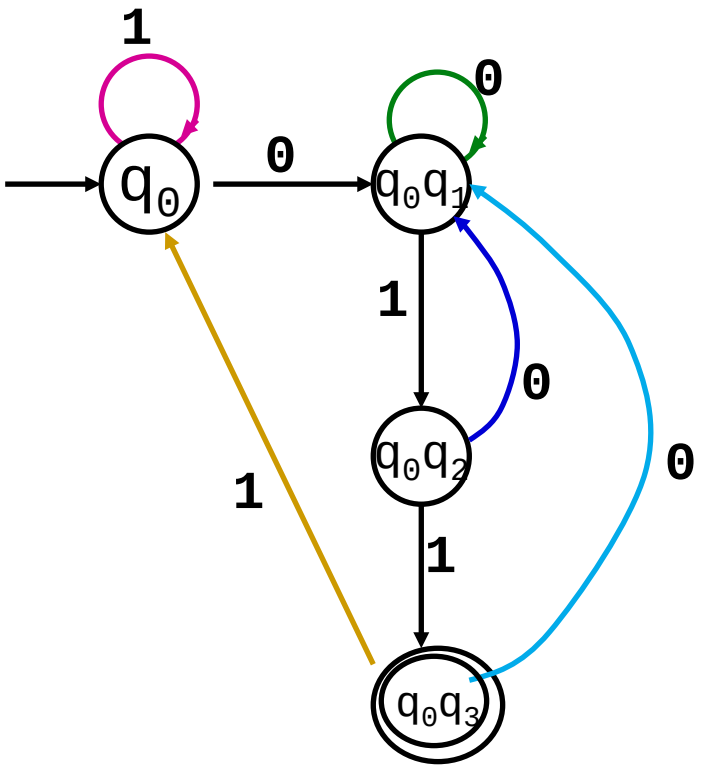

Now consider all possible strings

This is like path trees. But instead of tracking one string (each state has transitions on just one character), we track all possible strings (transitions on all possible characters).




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.

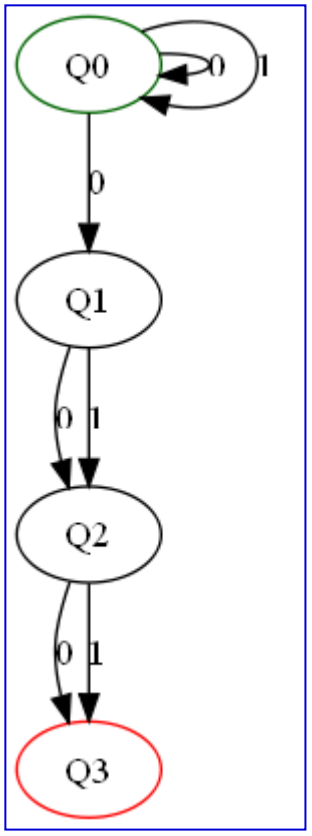
Each state of the DFA corresponds to a set of states of the NFA

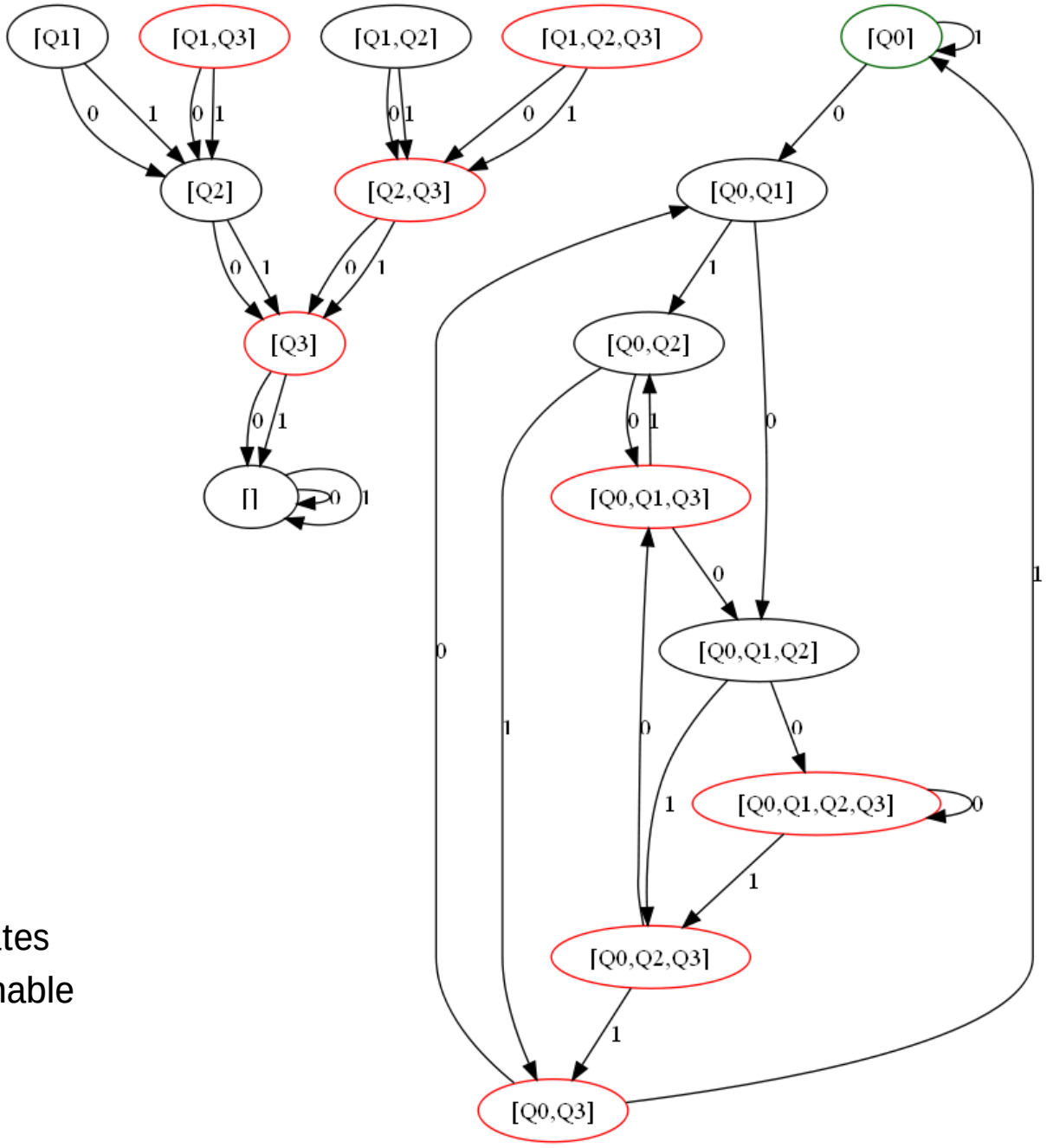
# Strategy

Write down all possible subsets of the set of states.

Each subset will be a possible state of the new machine.

A transition from one subset S to another T is added on character c, iff, trans s c = t, and $t \in T$ and $s \in S$




Note,

Original NFA has 4 states

The computed DFA has 16 states

Only some of the 16 are reachable from the start state {Q0}

# General Construction

Given an NFA:

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

The associated DFA is

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

Where

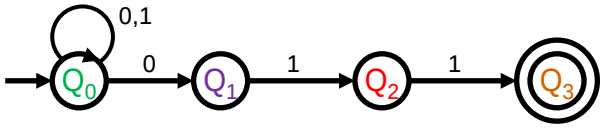
In the DFA constructed each state is labeled with a set of states from the NFA. Thus the start state is just the singleton set {s}

$F'$ is the set of {subsets of $Q$} that contain an element of $F$. Thus $F' \subseteq P(Q)$. $f \in F'$ iff exists $x \in f$ and $x \in F$

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

# Example

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




$$\delta(\{q_0,q_2\},1) = \Delta(q_0,1) \cup \Delta(q_2,1)$$
$$= \{q_0\} \cup \{q_3\}$$
$$= \{q_0,q_3\}$$

$$\delta(\{q_0,q_1,q_3\},0)$$
$$= \Delta(q_0,0) \cup \Delta(q_1,0) \cup \Delta(q_3,0)$$
$$= \{q_0,q_1\} \cup \varnothing \cup \varnothing$$
$$= \{q_0,q_1\}$$

# 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;