Computing Theory

COMP 147 (4 units)

Chapter 1: Regular Languages
Section 1.2: Nondeterminism

Last Time

- Finite State Machines (or DFA)
    - Formal Definition
    - Designing DFAs
- Regular Languages
- In-class activity

More Examples of DFAs

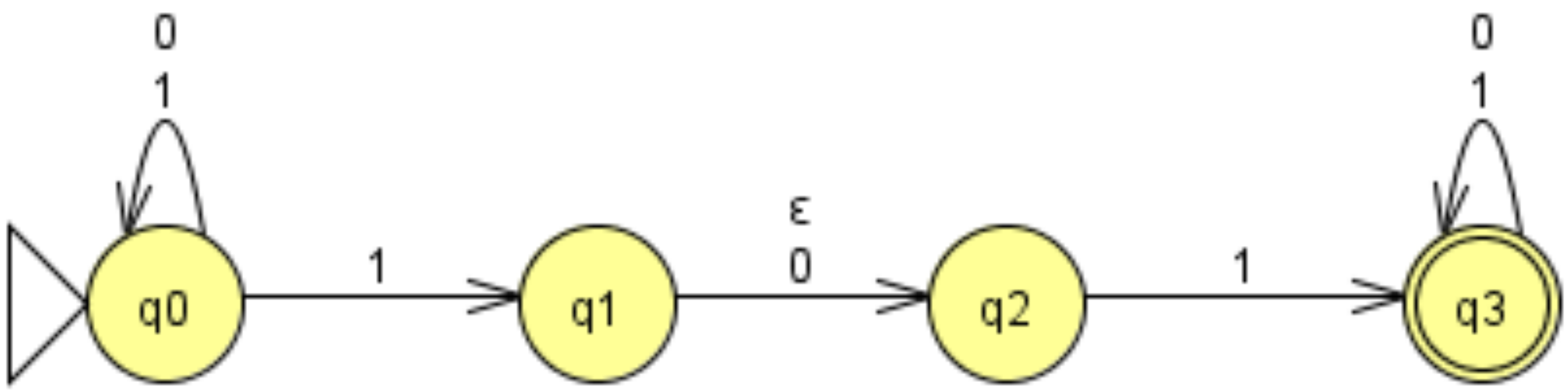
- L1= { w | w contains the substring 1010 }
- L2={ w | in w every 1 is directly followed by two 0’s}
- L3 = { w | w is divisible by 3} (a bit trickier!)

Nondeterministic Finite Automata

- An NFA can have more than one transition for a member of the alphabet Σ.
- An NFA can transition to a new state without reading any symbol. These are called ε transitions.
- Allows threads of execution in parallel.
  Each thread is searching for a match with the input string.

Example of an NFA

NFA – Nondeterministic Finite Automaton



1. A state may have 0 or more transitions labeled with the same symbol.
2. ε transitions are possible.

Computation of an NFA

- When several transitions with the same label exist, an input word may induce **several** paths.

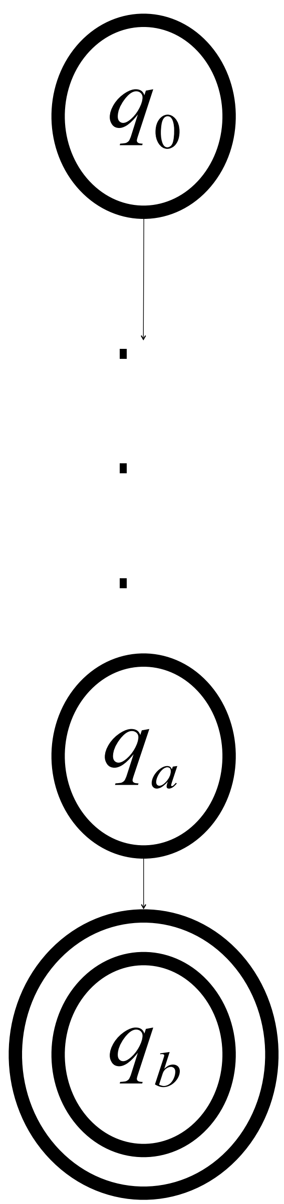
- When no transition is possible a computation is "stuck".

Q: Which words are accepted and which are not?

A: If word w induces (at least) a single accepting path, the automaton "chooses" this **accepting path** and w is accepted.

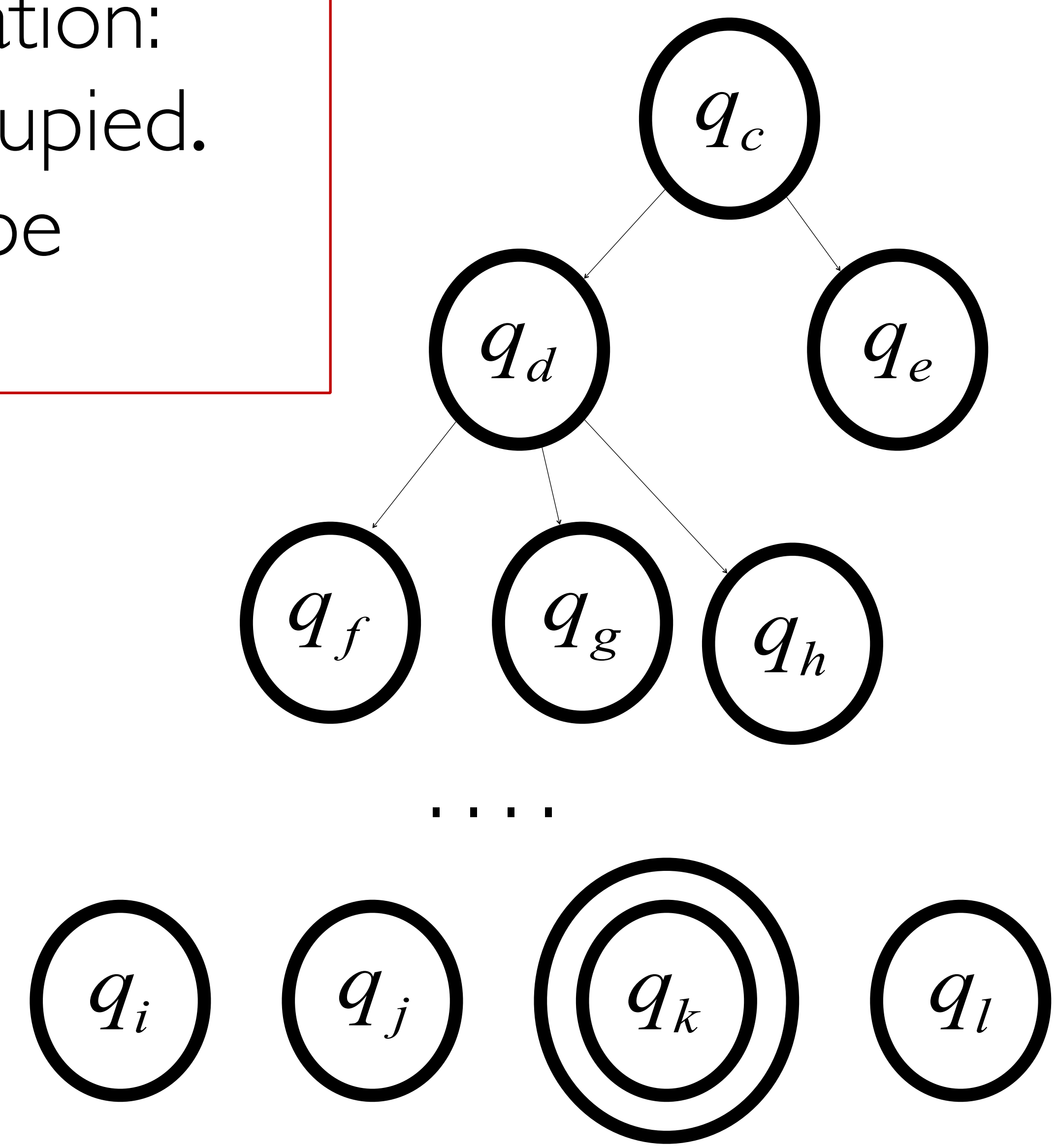
Possible Computations

DFA

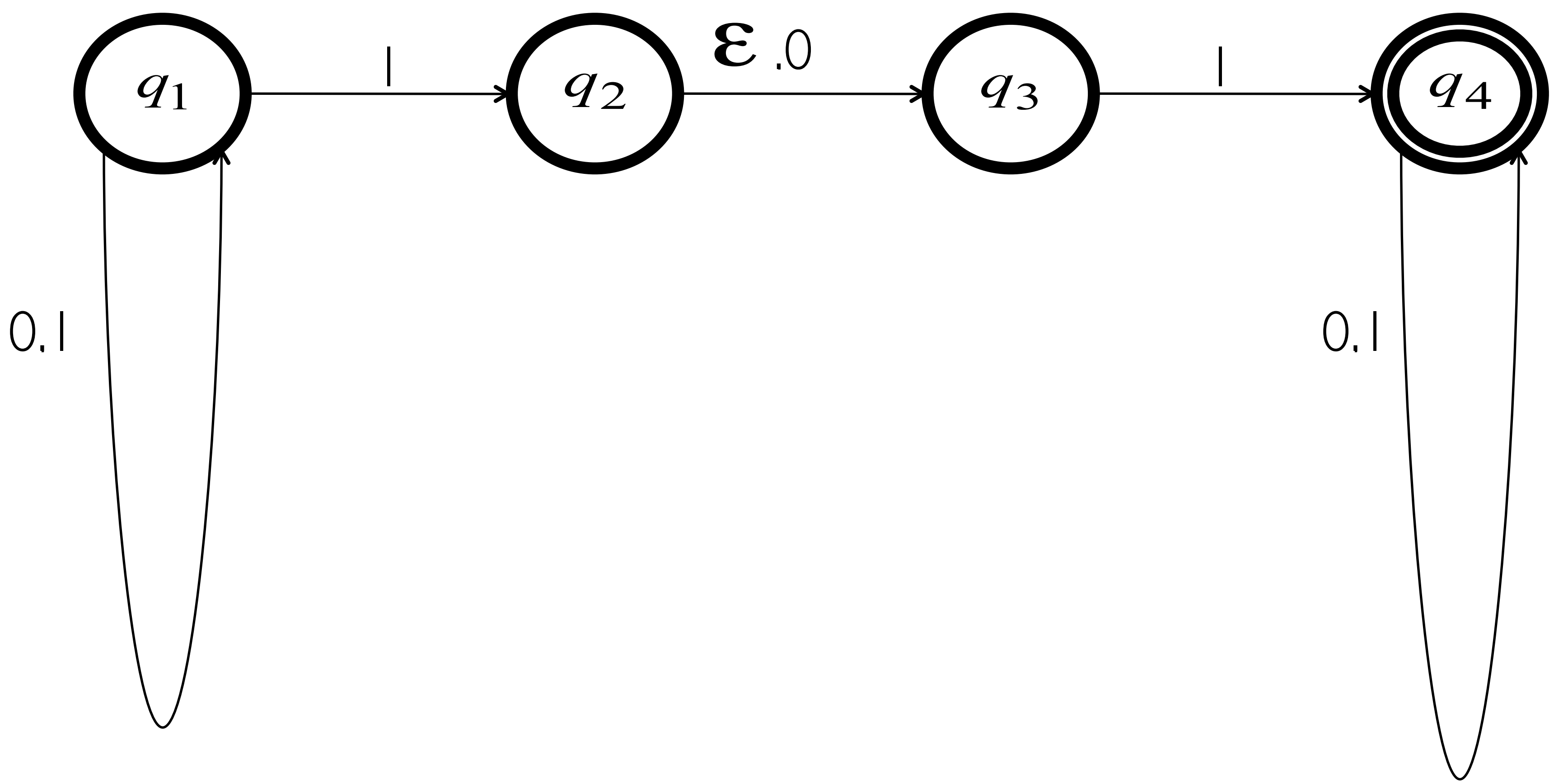


At each step of the computation:
DFA - A **single state** is occupied.
NFA - **Several states** may be occupied.

NFA



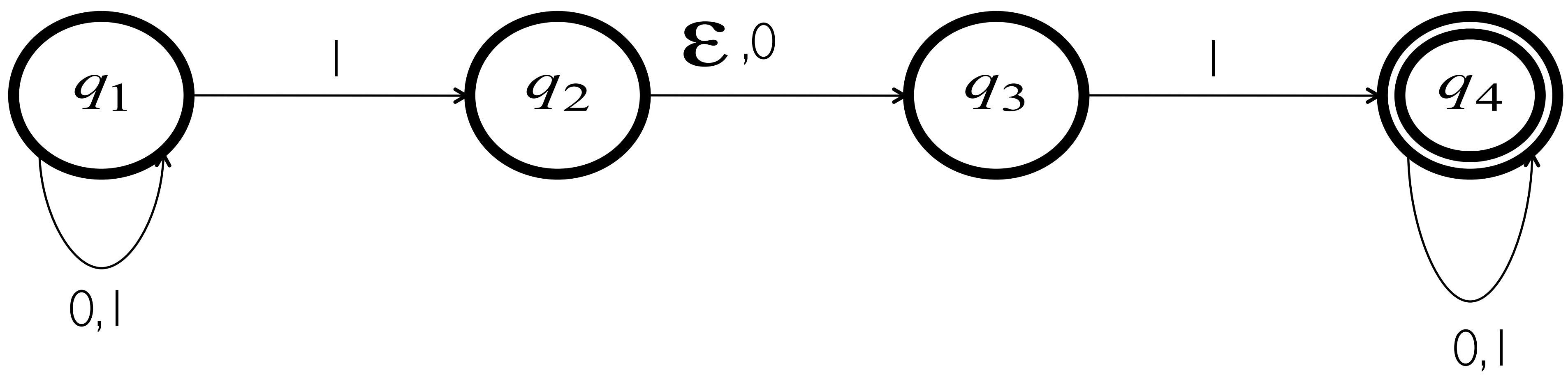
Example NFA Computation

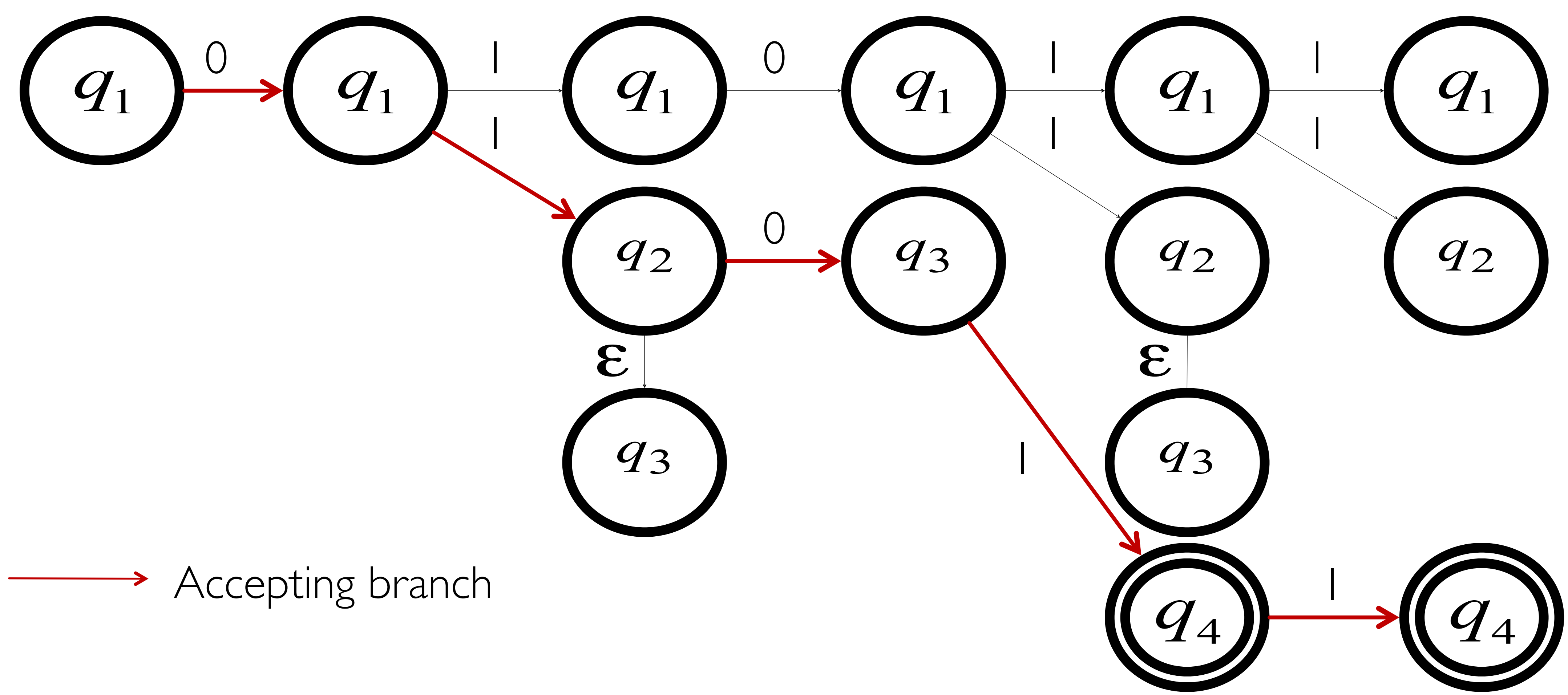


On the input word w=01011 there exists an accepting path and w is accepted.

Can we characterize (find) the language recognized by this automaton?

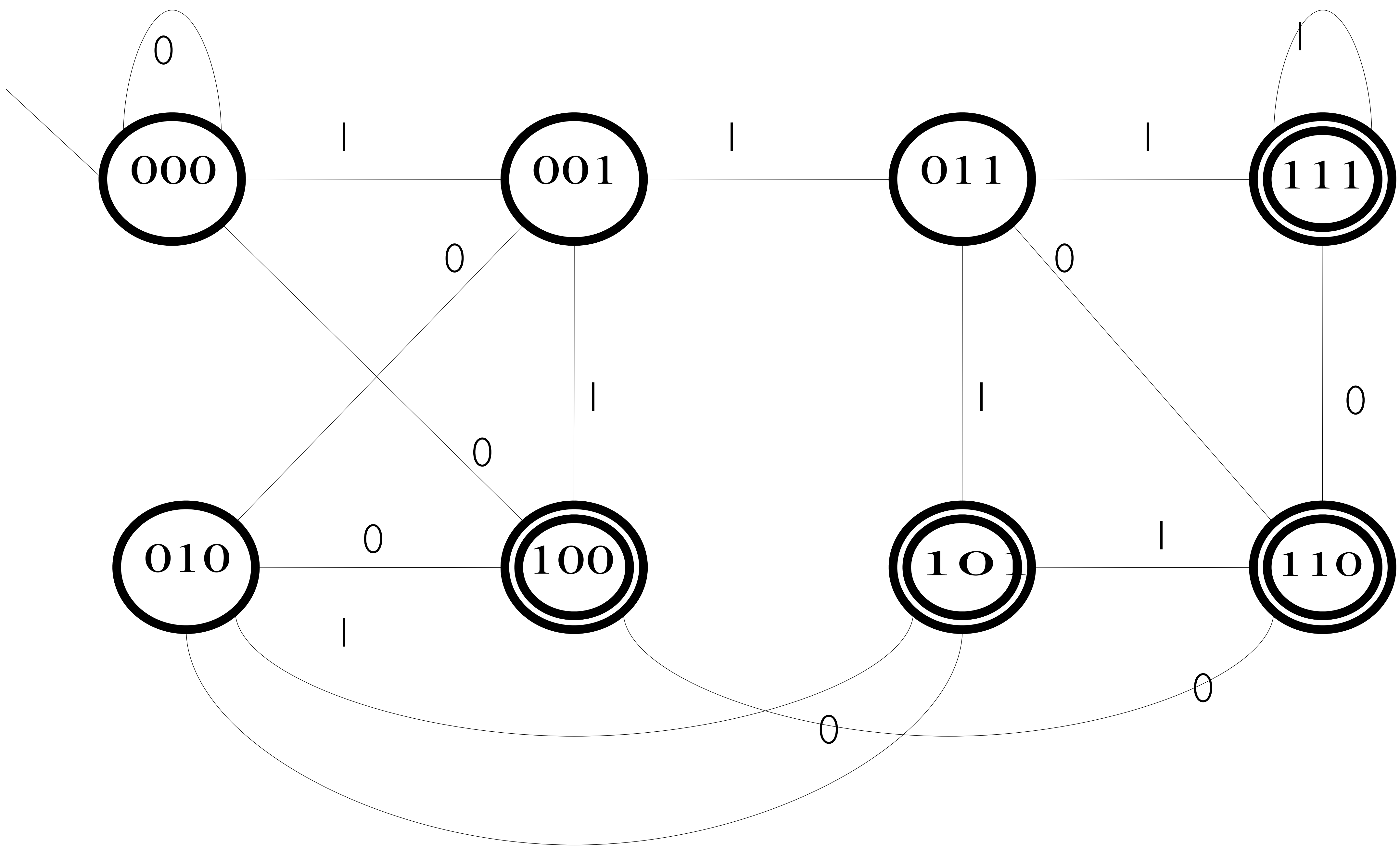
Computation tree for 01011





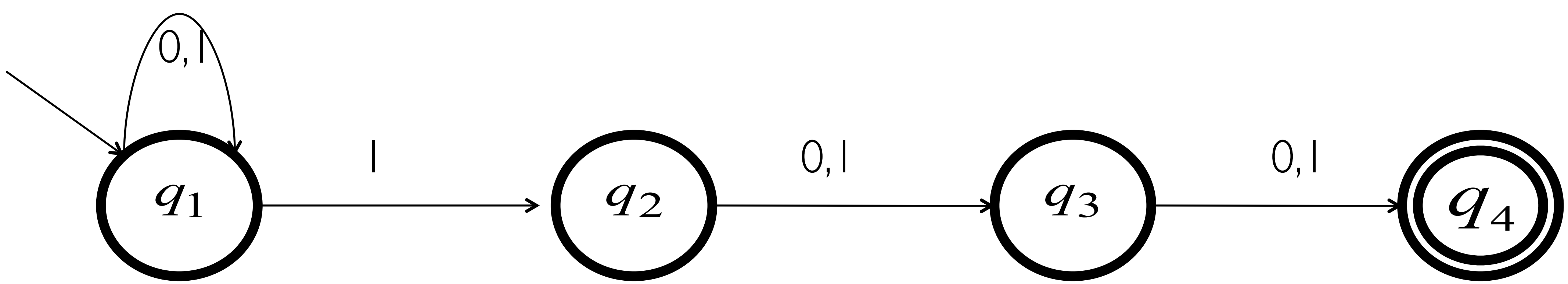
Example - A Complicated DFA

What does this DFA recognize?



Example – An Equivalent NFA

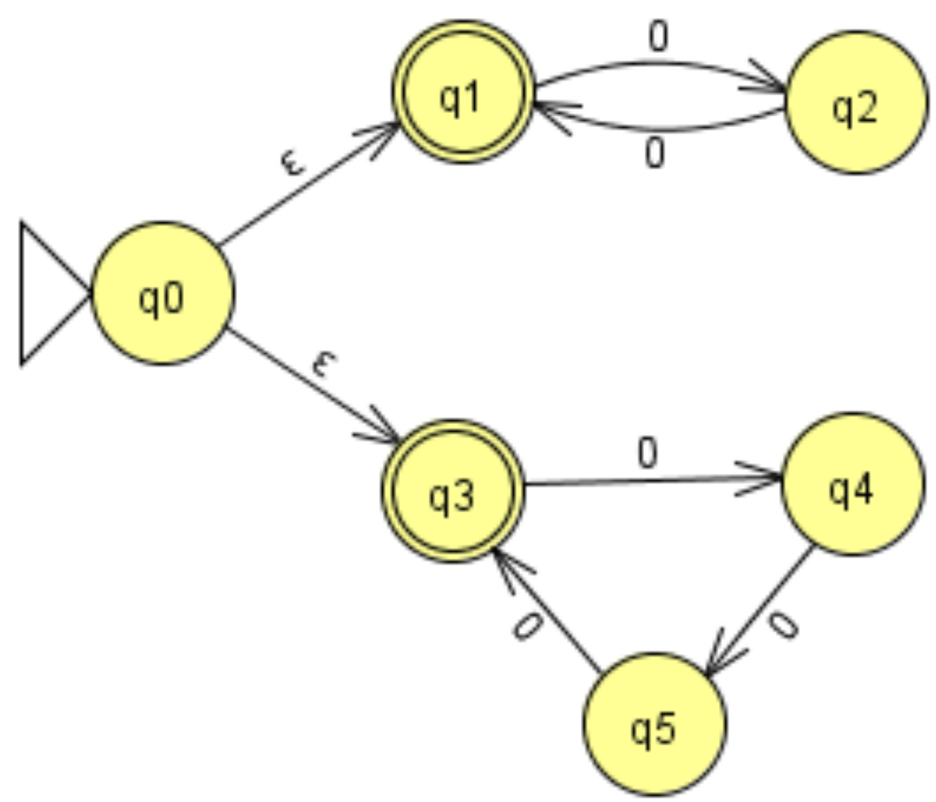
What does this NFA recognize?



bit strings with a 1 in third position from end

An NFA over a Unary Alphabet

- Let $\Sigma = \{0\}$.
  This NFA demonstrates the convenience of having ε transitions.
  What language does it accept?



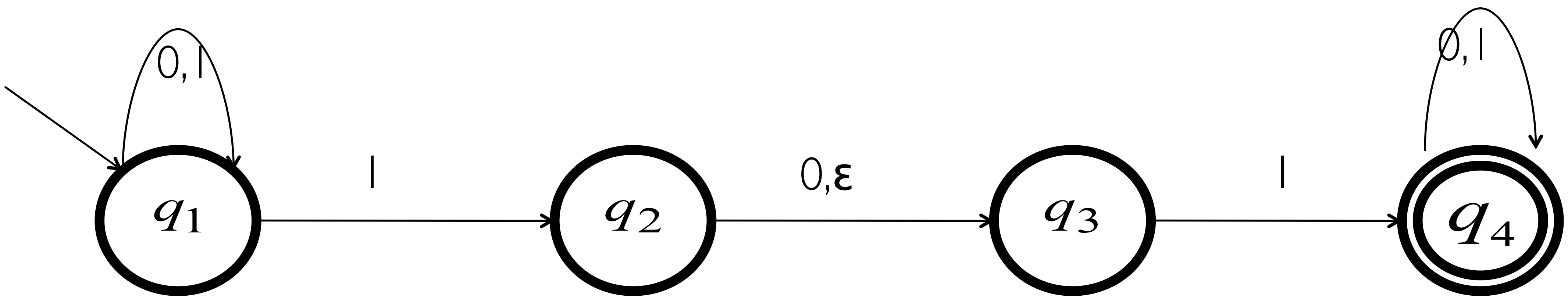
Formal Definition of an NFA

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. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \boxed{\mathcal{P}(Q)}$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

P(Q) is the power set of ***Q***

Formal NFA Example



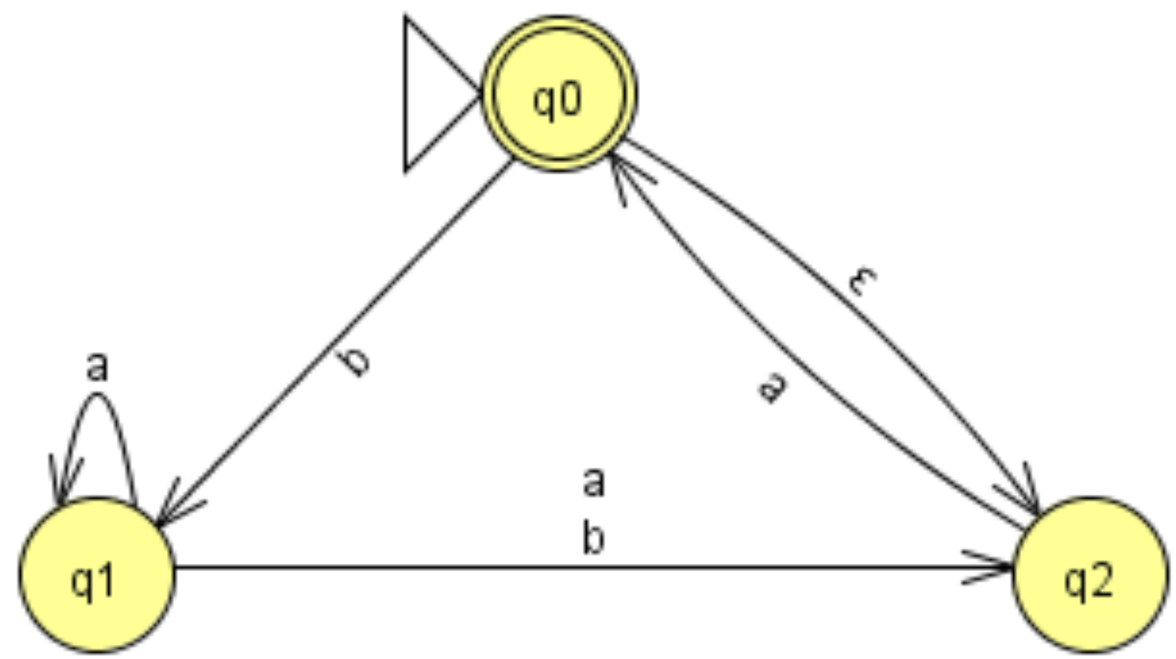
1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

	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. $F = \{q_4\}$.

Another NFA

- Does this NFA accepts the following strings
- ε
- a
- baba
- baa
- b
- bb
- babba



NOTE

- Soon we will see that the language accepted by the previous NFA is the same language generated by the *regular expression*

$$(\varepsilon + ba^*(b + a))a$$

DFA, NFA Equivalence

- Definition: A language is regular is some finite automata recognizes it.

- Theorem: Every NFA has an equivalent DFA.

- Corollary: A language is regular if and only of some NFA recognizes it.

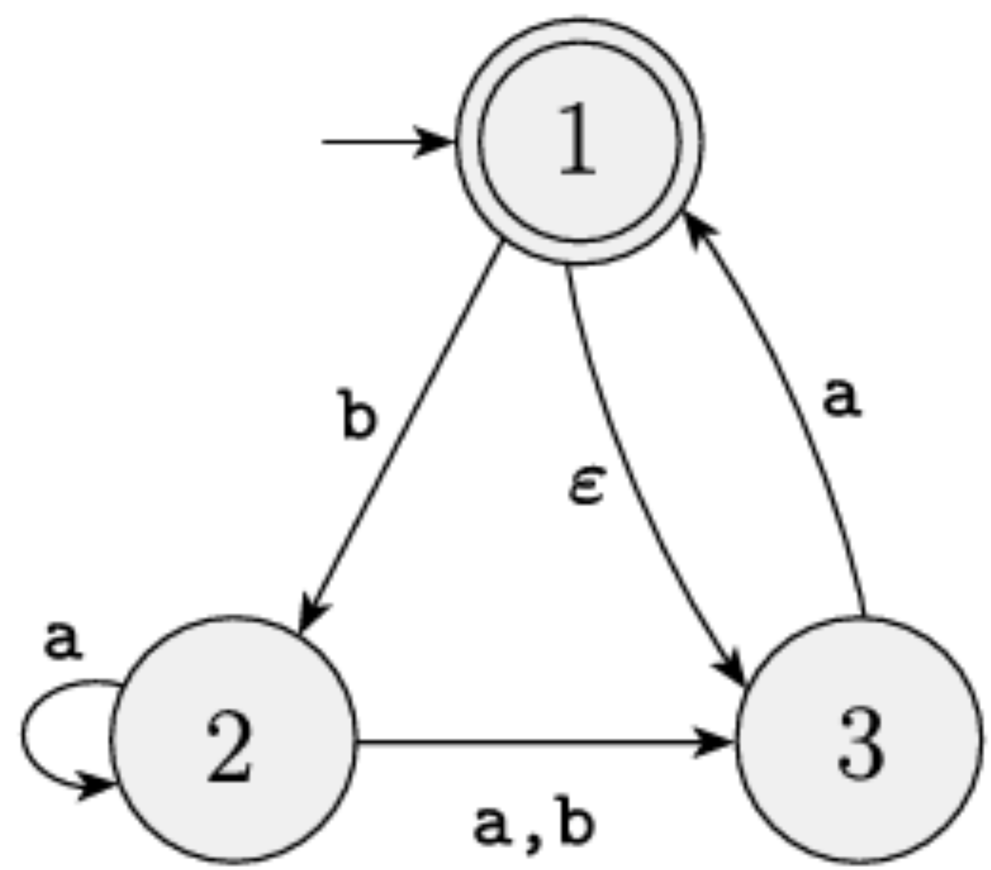
DFA, NFA Equivalence Proof

- Proof by construction:
  - Give algorithm that will convert any NFA to a DFA
  - States in the DFA defined by powerset of states in NFA
  - Start state of DFA is the state containing only the start state of NFA
  - Accept states of DFA are all states that contain any accept state of NFA
  - Transition function needs some explanation

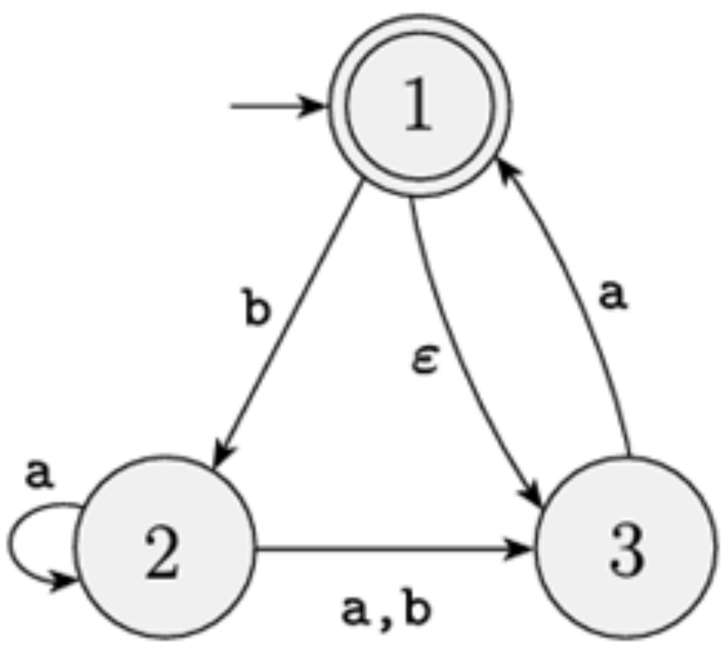
Let’s try an example for language L = {w | w that ends with 00}

DFA, NFA Equivalence Example

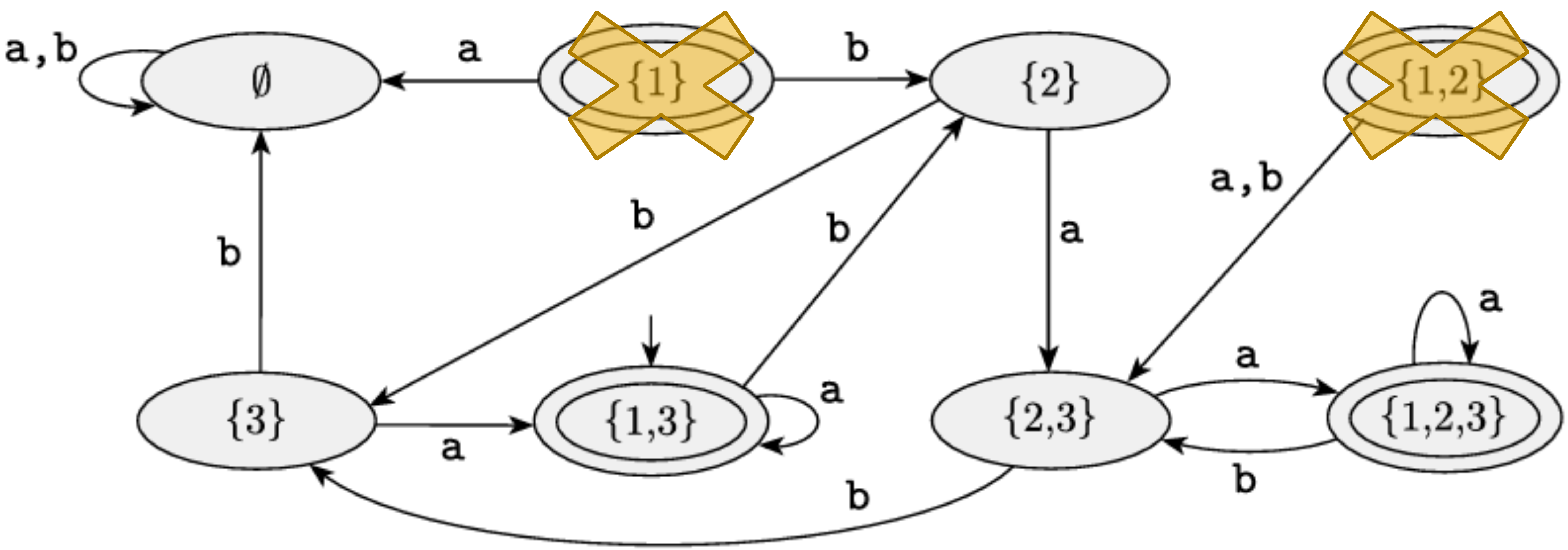
Convert this NFA to a DFA.



DFA, NFA Equivalence Example



Remove unreachable states



DFA, NFA Equivalence Example