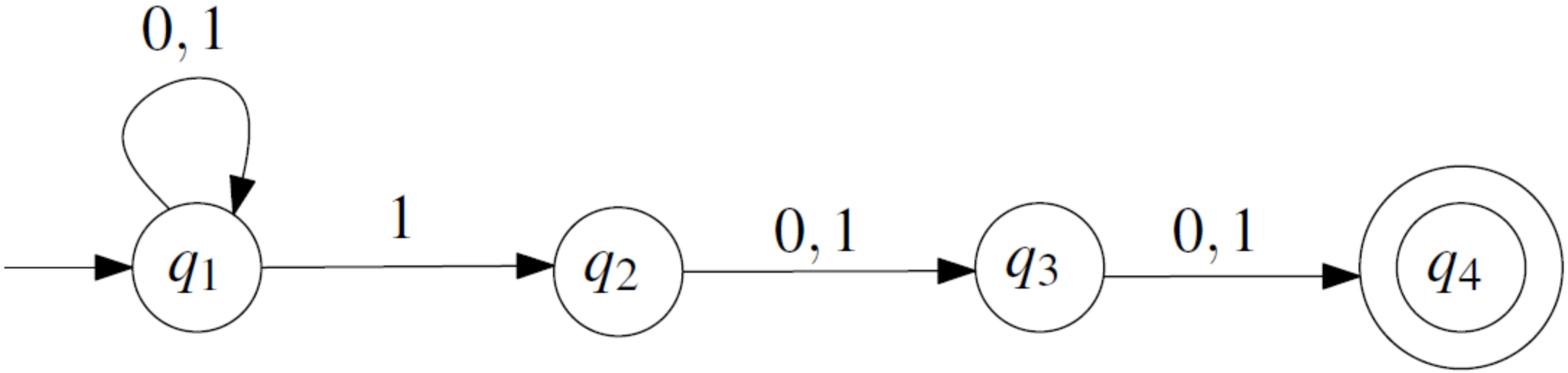


Theory of Computation

Lesson 2

Non-Deterministic Finite State Automata

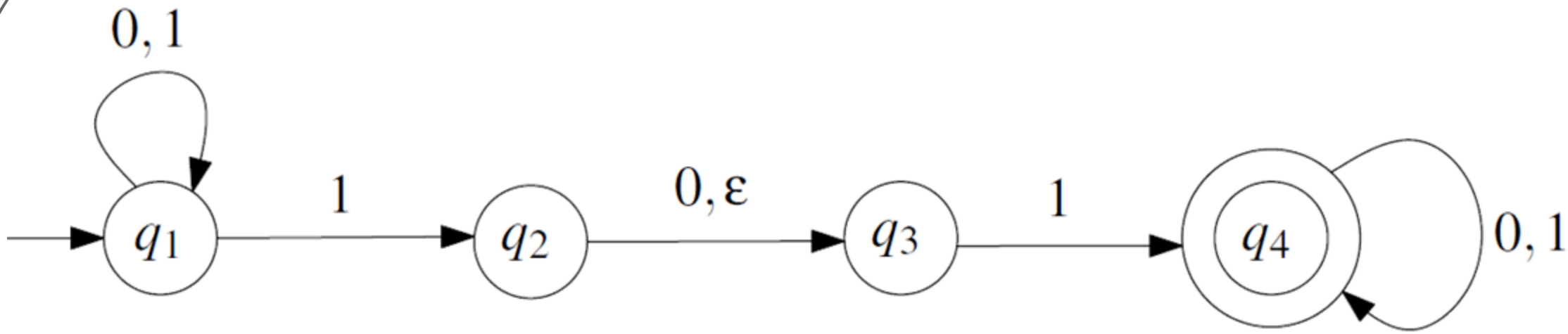
Why we need Non-Deterministic Automata ?

- Non-deterministic finite automata (NFA) focus on how they accept the inputs. For this reason, NFA are smaller and easier to construct than deterministic finite automata (DFA).
- NFA are used for humans, DFA are prepared for machines.
- Besides, they are equivalent to each other. Thus, we can solve especially difficult problems in NFA, then convert into DFA.

Uncertainties in NFA

There are 3 types of uncertainty to use in NFA:

- 1. Additional out-goings edges,
- 2. Undefined out-going edges,
- 3. Arbitrarily out-going edges.



Formal Definition

We now come to the formal definition of a NFA.

Definition 2.4.1 A *nondeterministic finite automaton (NFA)* is a 5-tuple $M = (Q, \Sigma, \delta, q, F)$, where

1. Q is a finite set, whose elements are called *states*,
2. Σ is a finite set, called the *alphabet*; the elements of Σ are called *symbols*,
3. $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is a function, called the *transition function*,
4. q is an element of Q; it is called the *start state*,
5. F is a subset of Q; the elements of F are called *accept states*.

A first example

A = {w : w is a binary string containing 101 or 11 as a substring}.

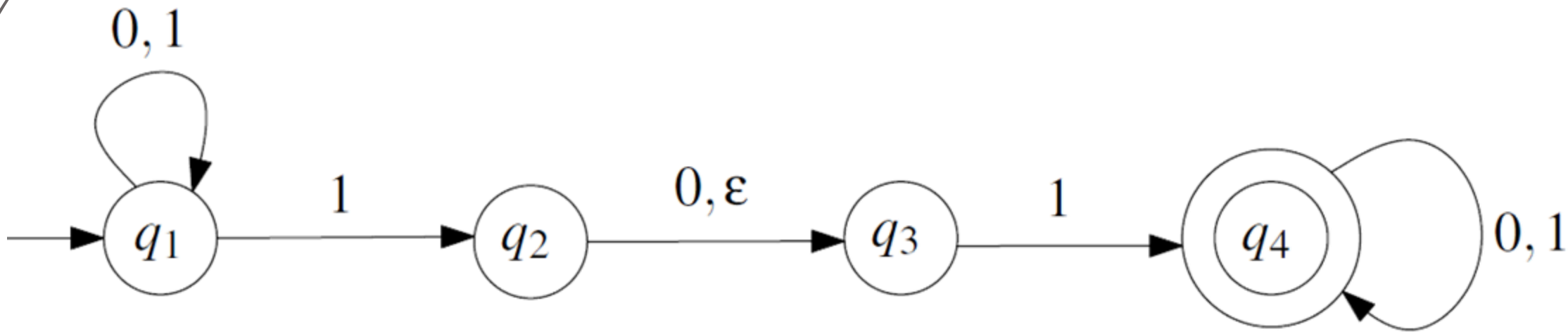
Here, our alphabet will be binary $\Sigma = \{0, 1\}$. Then we should find the states at first.

We can directly focus on 101 and 11 strings. We can design them two different sub solution, but if you can see, they have a common solution.

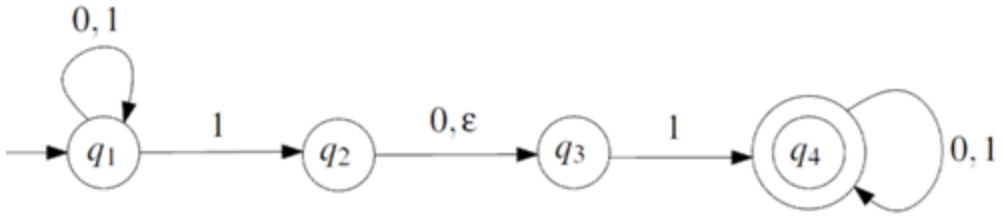
A first example

When we design states, we can define their workings as follows:

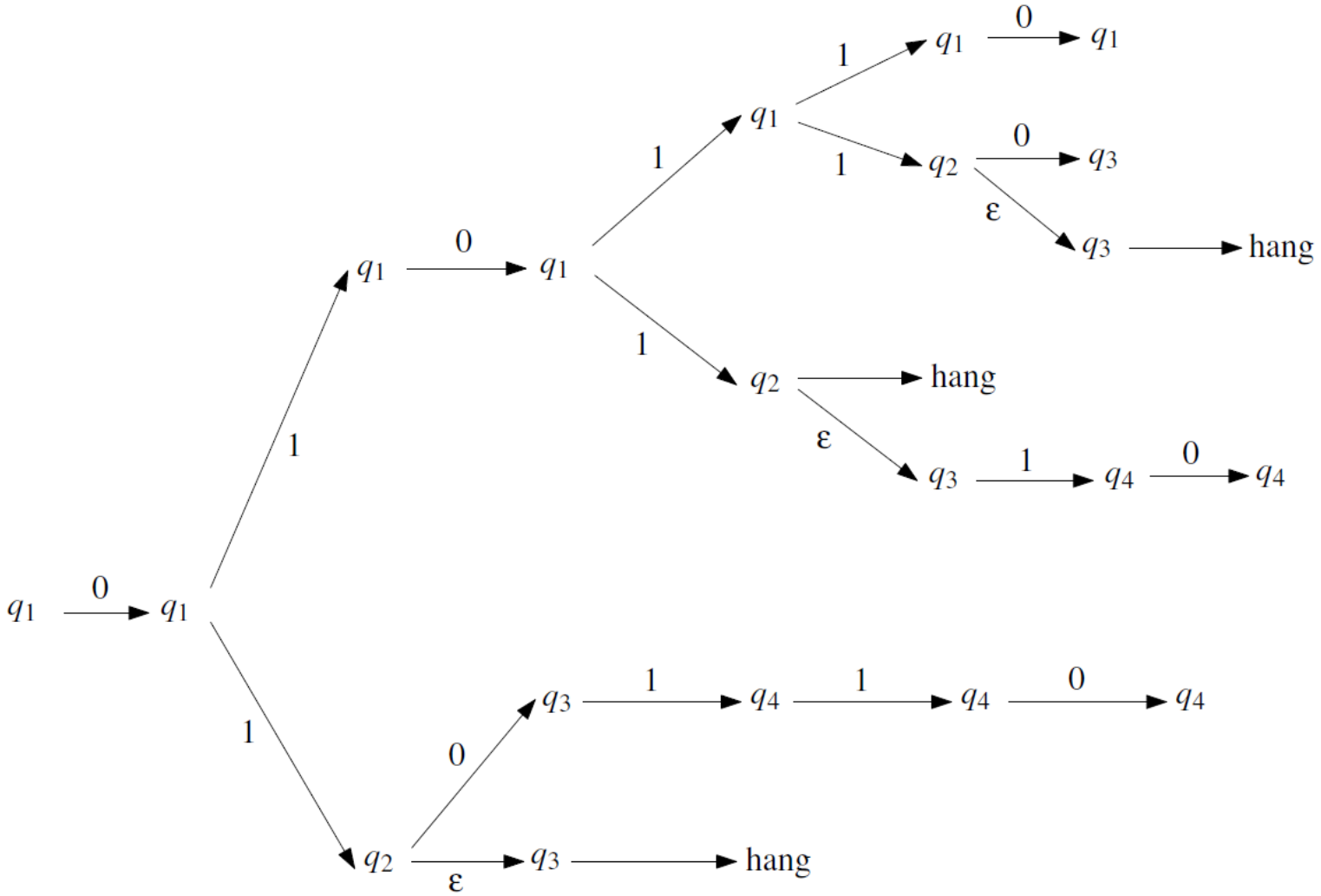
- q_1 : start state
- q_2 : first symbol is OK, it doesn't matter 101 or 11
- q_3 : second symbol is OK for 101 or directly pass for 11
- q_4 : last symbol is OK



A first example



Now, we can run the NFA defined for input 010110



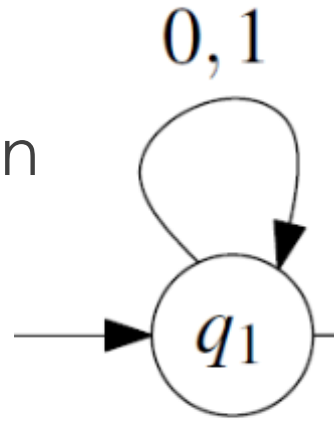
A second example

A = {w ∈ {0,1}* : w has a 1 in the third position from the **RIGHT**}

- This problem is easier than the one in DFA. Again, we can focus to design solution directly.

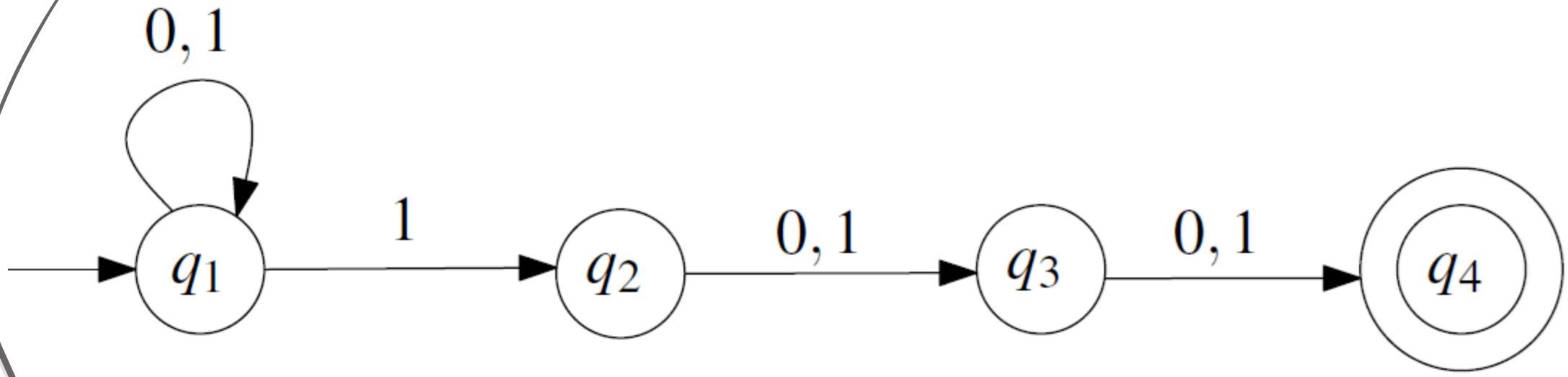
  {0,1}* 1 {0,1} {0,1}

- In the previous solution, there was a concept used to mean "whatever symbol comes" in start and end states. We will use this solution, shown on the right, in this question as well.



A second example

- q_1: Both the initial state and the time elapsed state until the third letter from the right
- q_2: 3rd symbol from the right came
- q_3: 2nd symbol from the right came
- q_4: The last symbol from the right came



A third example

$A = \{0^k : k \equiv 0 \bmod 2 \text{ or } k \equiv 0 \bmod 3\}$

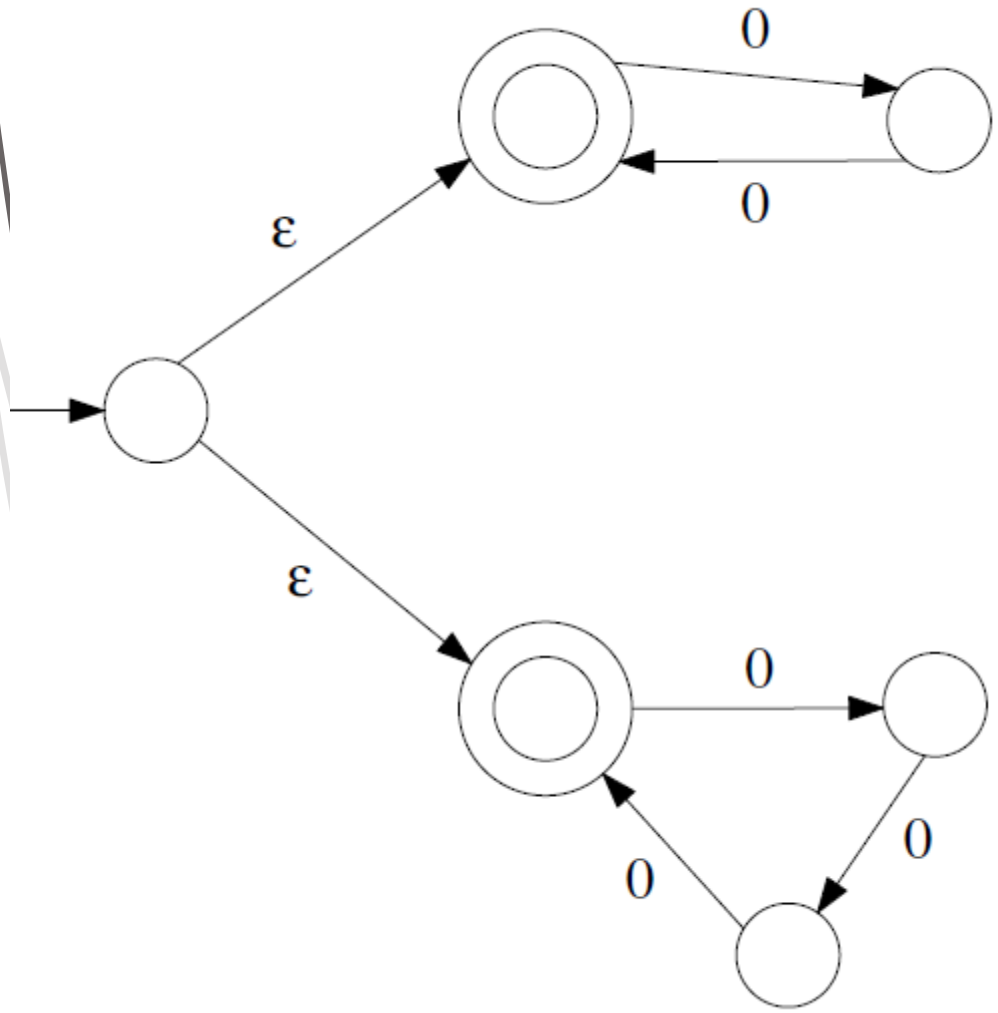
- The string can contain only 0 symbols. We can accept the string when its length is a multiple of two or three.

  00 00 00 00 00 … or 000 000 000 000 000 …

- If we misunderstand the question here and create a common solution to the two problems, the following problem may arise: {(00)or(000)}* Then this wrong solution may mistakenly accept the following string.

  00 000 00 00 00 X

A third example



In fact, this solution is the union of the two languages.

$$A = A_1 \cup A_2$$

A_1 accepts the strings when their lengths are multiple of two. A_2 accepts when the length is multiple of three.

- That’s all.
- Thanks for listening.