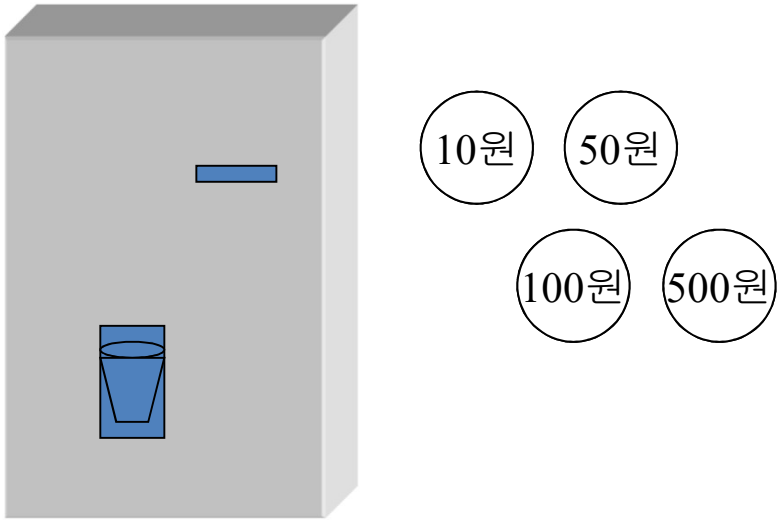
Chapter 3

2. 유한 오토마타 : Part I

목차

커피 자판기는 어떻게 동작할까?

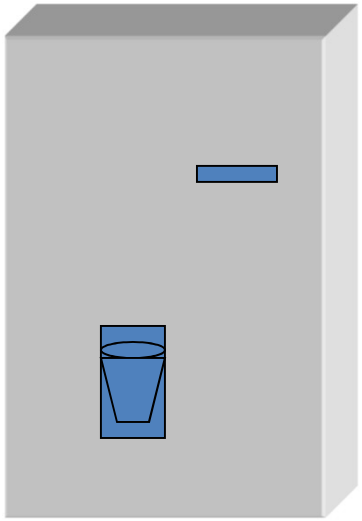
coffee vending machine



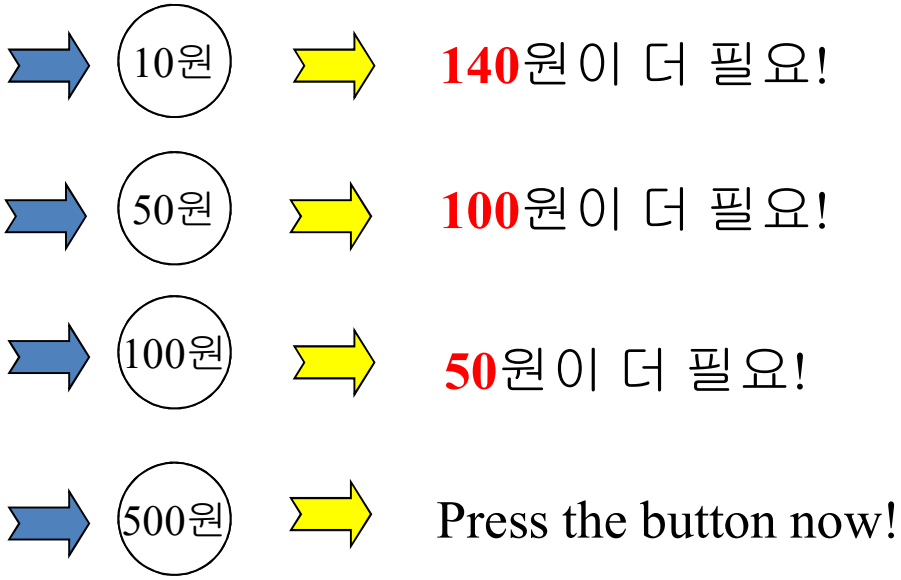
밀크 커피 **1**잔=**150**원

자판기는 입력에 따라 상태가 바뀐다

coffee vending machine



밀크 커피
1잔=150원



Finite State Automata

A model of computation

- **A set of *states*** and **how to get** from some state to other states

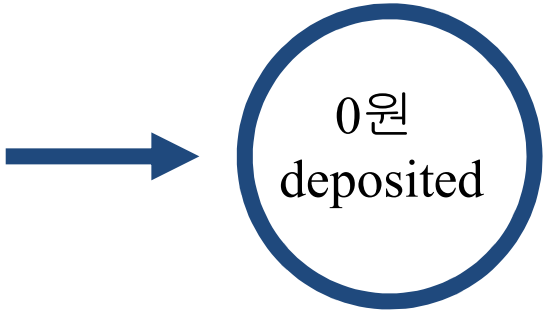
Computation model 이란?

- 시스템 내부를 상세히 만들지 않았지만
  - S/W를 사용하여 시스템 동작을 확인

Automata 는 복수, **Automaton** 은 단수

An initial state

- **Start with 0원 deposited.**
  - Called the **initial state** of the machine.
  - A circle is drawn for the state



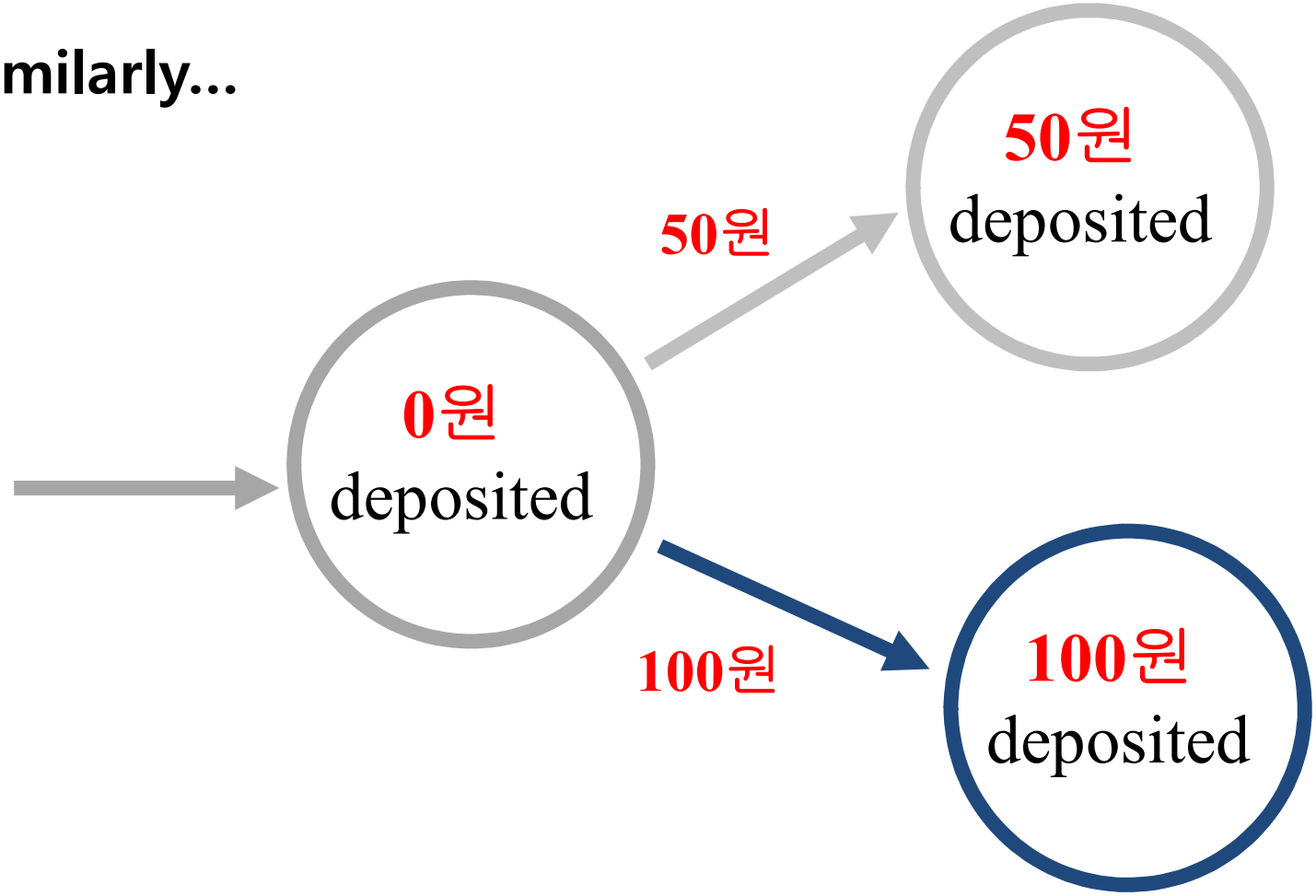
  - An arrow pointing to this state marks it as the initial state.

Depositing a 50원…

- **Takes us to a new state: 50원 deposited**
- **Add an arrow for the transition**
  - Label it with the **input** we gave to the machine

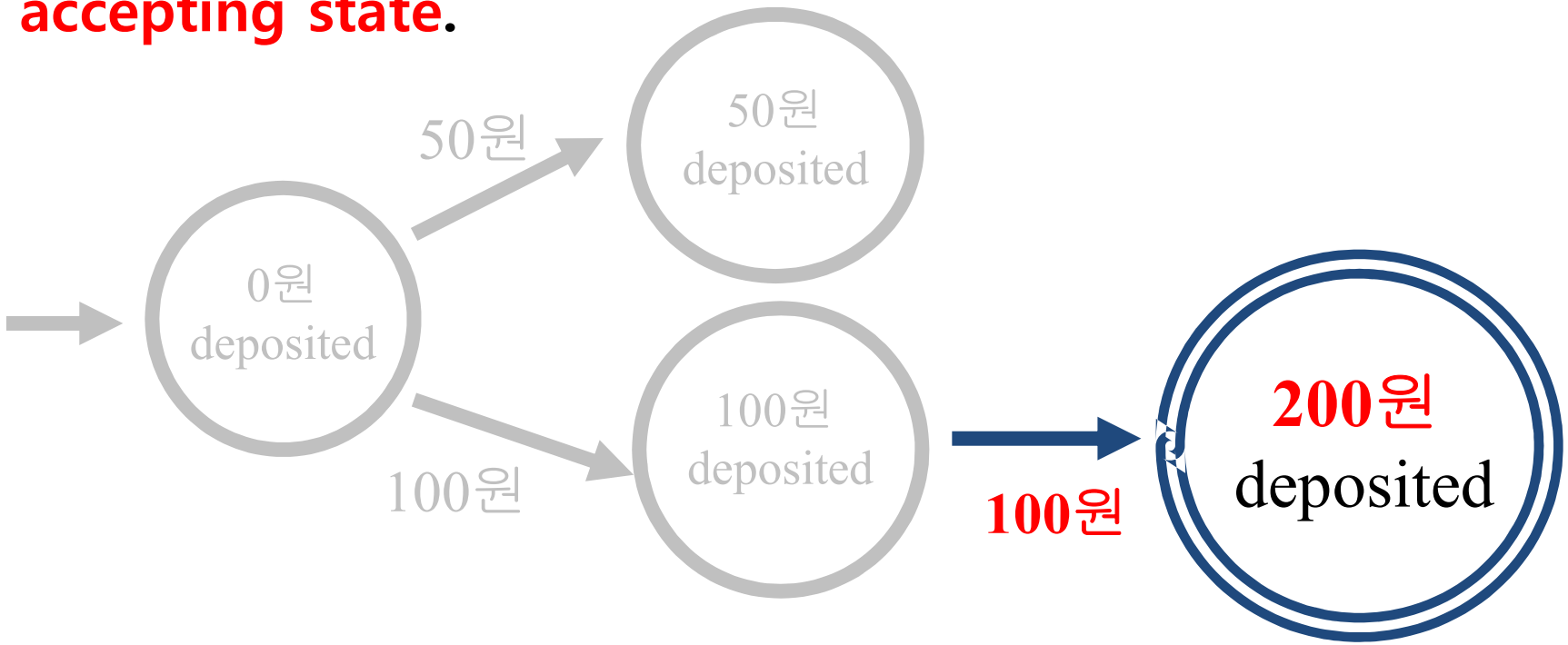
Depositing a 100원…

- Similarly…



Accepting states (or Final States)

- Add new state with double circle to denote that it's an **accepting state**.

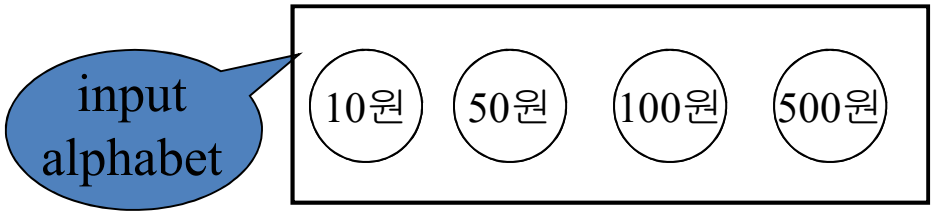


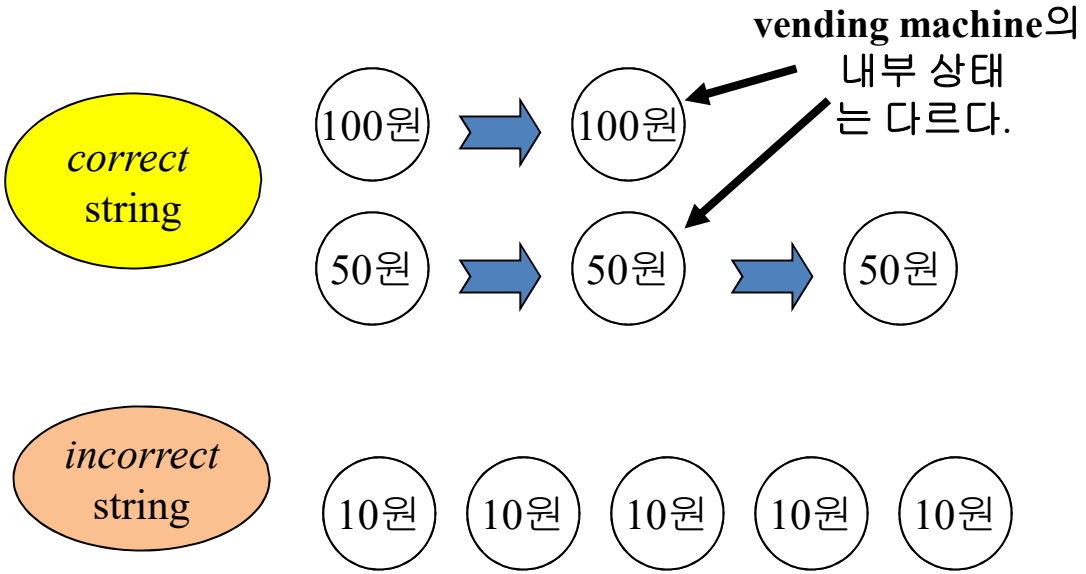
How to are tokens recognized?

- **유한 오토마타(FA, Finite Automata)를 사용**
  - 유한(finite) 개의 상태(state)를 갖고 있다.
  - 입력에 따라 상태가 바뀐다.
    - 윷판의 말은 윷을 던진 결과에 따라 움직인다.
  - 시작 상태와 종결 상태가 있다.

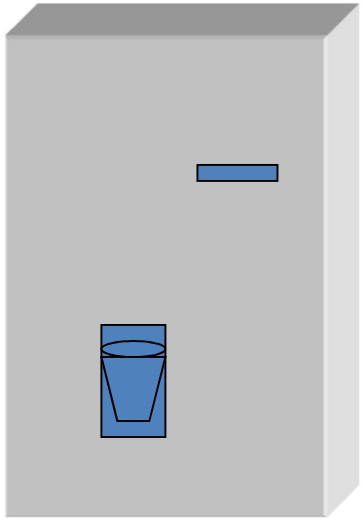
- **정규 표현은 유한 오토마타(recognizer)로 변환할 수 있다.**

유한 오토마타 동작 예





coffee vending machine



밀크 커피 **1**잔=**150**원

Finite Automata (1/2)

가상 기계(Hypothetical machine)

- 입력 알파벳
  - {10원, 50원, 100원, 500원}

- 상태 수가 유한 → Finite-*State* Automata
  - 상태 : 누적 금액
  - 모두 16개의 상태가 필요 $\{S_0, S_1, S_2, \cdots, S_{15}\}$
    - S_0(0원), S_1(10원), S_2(20원), ..., S_{15}(150원 이상) → 총 16개

Finite Automata (2/2)

가상 기계(Hypothetical machine)

- 유한 상태 집합 $\{S_0, S_1, S_2, \cdots, S_{15}\}$

- ***An*** initial state and final state***s***
  - 초기 상태(S_0 , 0원), 종결 상태(S_{15} , 150원 이상)

- 외부 입력에 따라 상태(state)가 바뀐다.
  - 현재 상태(S_1 , 10원) → 100원 투입 → 다음 상태(S_{11} , 110원)

Deterministic Finite Automata (DFA)

- ***DFA*** $M = (\Sigma, S, T, s_0, A)$
  - Input Alphabet : Σ
  - A set of *finite* states : S
  - A starting(initial) state : $s_0 \in S$
  - A set of accepting(final) states : $A \subset S$
  - state transition functions (상태 전이 함수) : T (또는 δ)

δ : 델타로 읽음

$T : S \times \Sigma \rightarrow S$

T (현재 상태, 입력 기호) = 다음 상태

$$T(S_i, a) = S_j \quad \text{단, } S_i, S_j \in S,\ a \in \Sigma$$

Deterministic Finite Automata

- **DFA : 결정적 유한 오토마타**
  - DFA는 다음 2 가지 조건을 모두 만족해야 한다.
    - **ε–transition이 없다**.
    - 어떤 상태에서 하나의 입력 기호에 대해 다음 상태는 **하나**
      - $\delta : S \times \Sigma \rightarrow S$
  - 위 조건 중 어느 하나도 만족하지 않으면
    - **NFA** (비결정적 유한 오토마타)
  - **ε – transition** : ε 에 의한 상태 전이

DFA (*Deterministic* Finite Automata)
NFA (*Non-deterministic* FA)

DFA 예(1/3)

$M = (\Sigma, S, T, s_0, A) = (\{a, b\}, \{q_0, q_1, q_2\}, \delta, q_0, \{q_2\})$

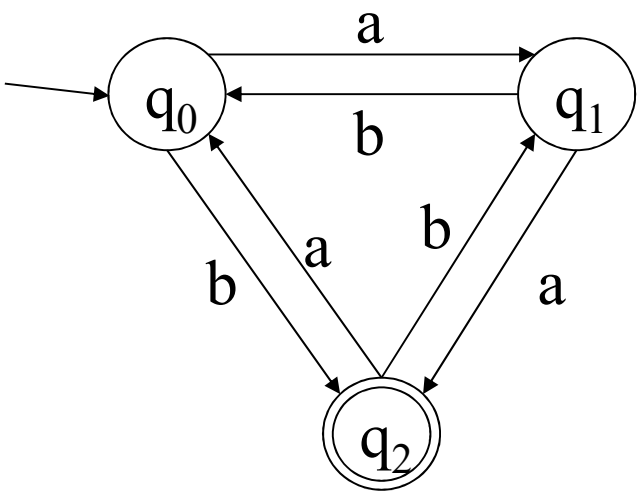
$\delta(q_0, a) = q_1$ $\quad\quad$ $\delta(q_0, b) = q_2$

$\delta(q_1, a) = q_2$ $\quad\quad$ $\delta(q_1, b) = q_0$

$\delta(q_2, a) = q_0$ $\quad\quad$ $\delta(q_2, b) = q_1$

DFA 예(2/3)

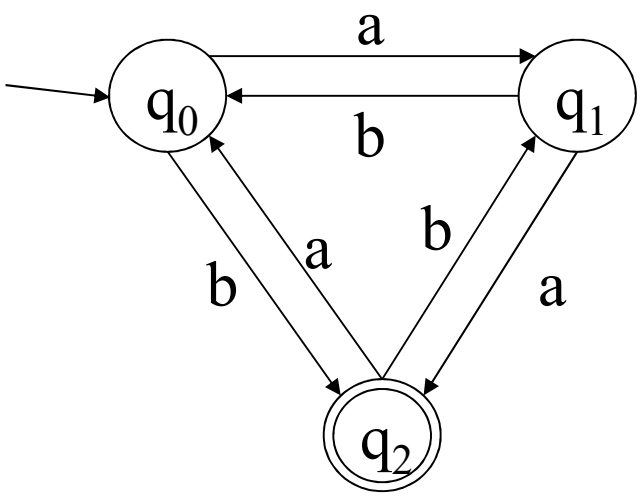
$M = (\Sigma, S, T, s_0, A) = (\{a, b\}, \{q_0, q_1, q_2\}, \delta, q_0, \{q_2\})$



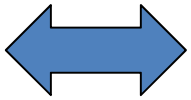
상태 전이 다이어그램
(state transition diagram)

DFA 예(3/3)

$M = (\Sigma, S, T, s_0, A) = (\{a, b\}, \{q_0, q_1, q_2\}, \delta, q_0, \{q_2\})$



상태 전이 다이어그램



	a	b
q_0	q_1	q_2
q_1	q_2	q_0
q_{2*}	q_0	q_1

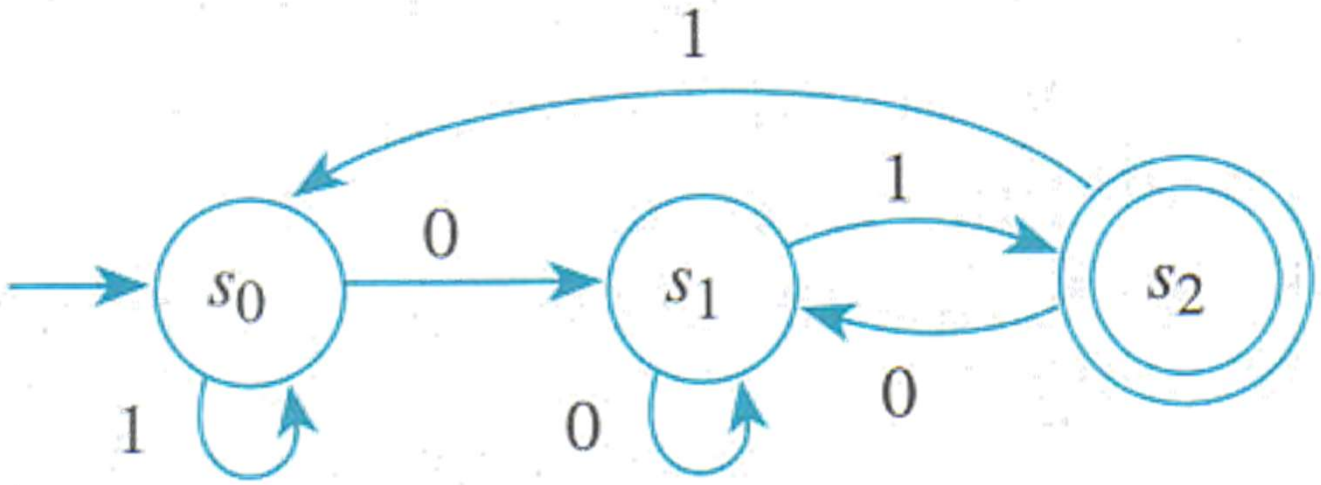
상태 전이 테이블

L(*M*) : DFA *M* 이 인식하는 언어

- ***M*** **이 인식하는 언어 *L*(*M*)은 아래 조건을 만족하는 문자열 $c_1 c_2 \ldots c_n$ 의 집합 ($c_i \in \Sigma$)**
  - $s_1 = \pi(s_0, c_1)$, $s_2 = \pi(s_1, c_2)$, … , $s_n = \pi(s_{n-1}, c_n)$인 상태들이 존재
  - 단, **s_n은 A의 원소**

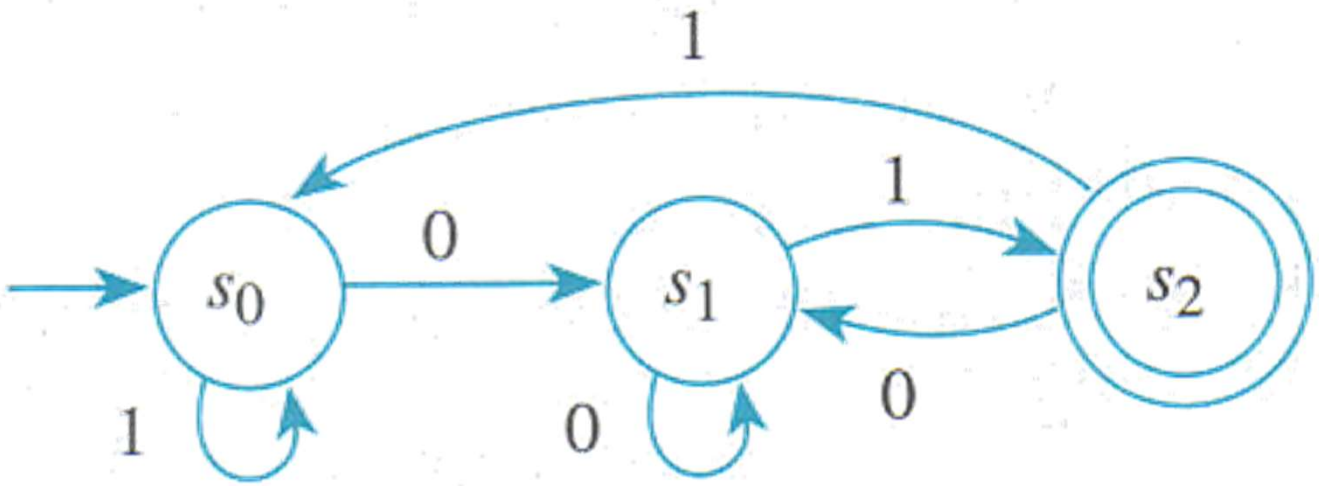
Practice #1 (1/2)

- Initial state?
- Accepting state?



Practice #1 (2/2)

- What happens if we input 011?
- What about 0111?



DFA 인식 과정 예(1/2)

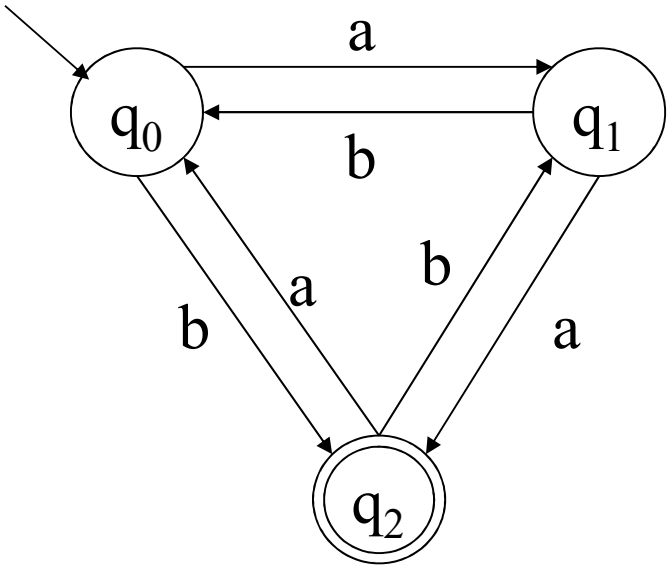
입력 문자열 = $c_1\ c_2\ c_3$ = *aba*

$\delta(q_0, c_1) = \delta(q_0, a) = q_1$,

$\delta(q_1, c_2) = \delta(q_1, b) = q_0$,

$\delta(q_0, c_3) = \delta(q_0, a) = q_1 \neq q_2$

∴ 인식하지 못함



DFA 인식 과정 예(2/2)

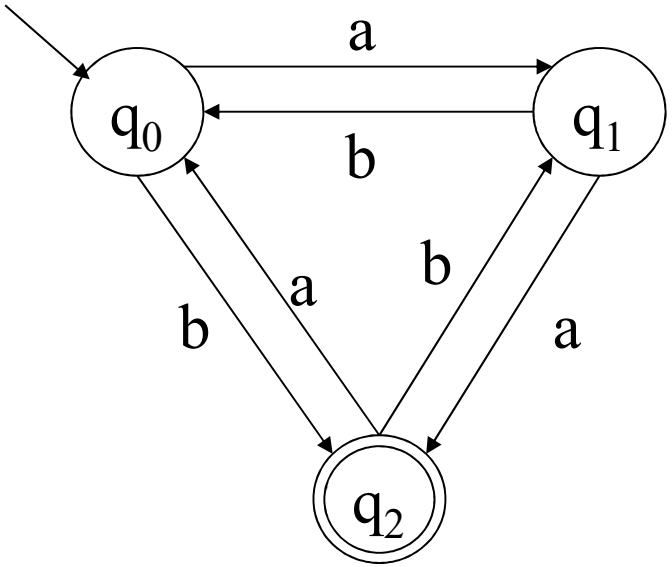
입력 = *ababb*

$\delta(q_0, a) = q_1$, $\delta(q_1, b) = q_0$

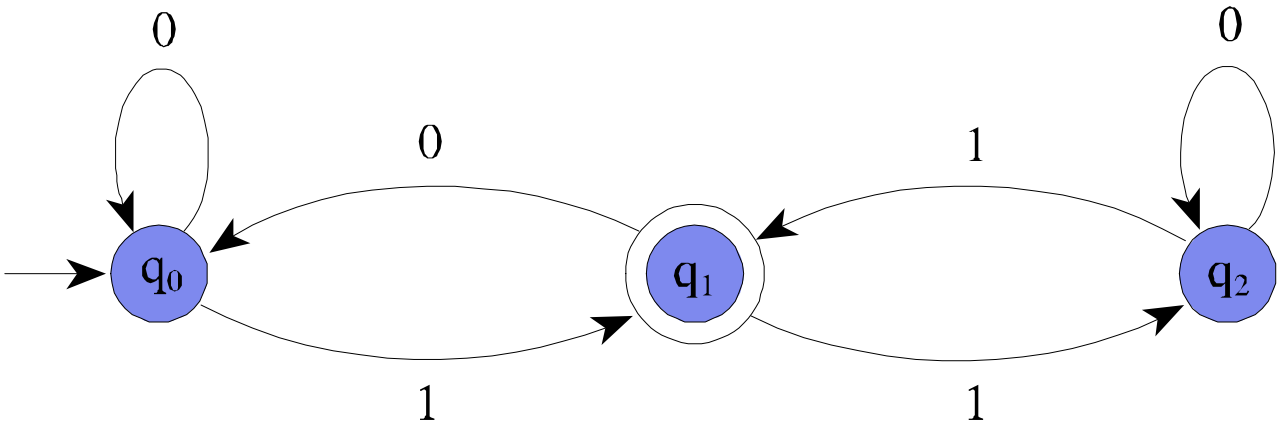
$\delta(q_0, a) = q_1$, $\delta(q_1, b) = q_0$

$\delta(q_0, b) = q_2$

∴ 인식



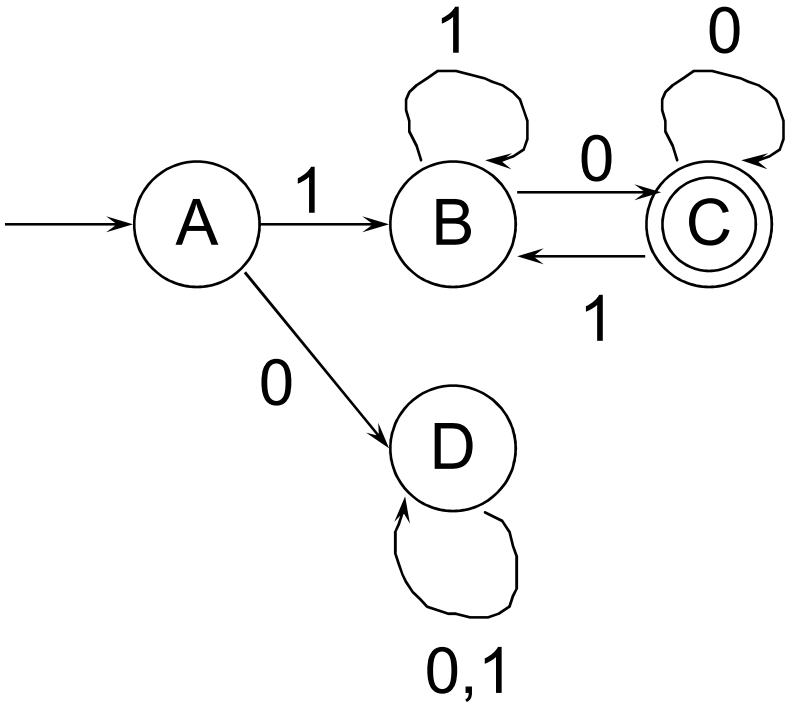
Practice #3



Q1. Identify 5 tuple of DFA M.
Q2. Which strings will be accepted?

0 0 1 0 1 0 1 1 1 0 1
0 0 1 1 0 0 1 1

Practice #4



state \ input	0	1
A	D	B
B	C	B
C*	C	B
D	D	D

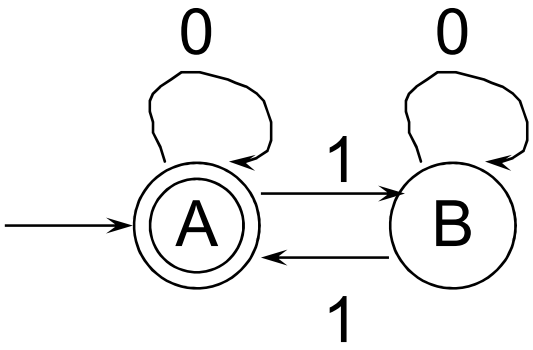
Which strings will be accepted?

1 1 0 0

1 0 0 0 0 1

0 0 0 1

Practice 5: Even Parity Checker



state \ input	0	1
A*	A	B
B	B	A

Which strings will be accepted?

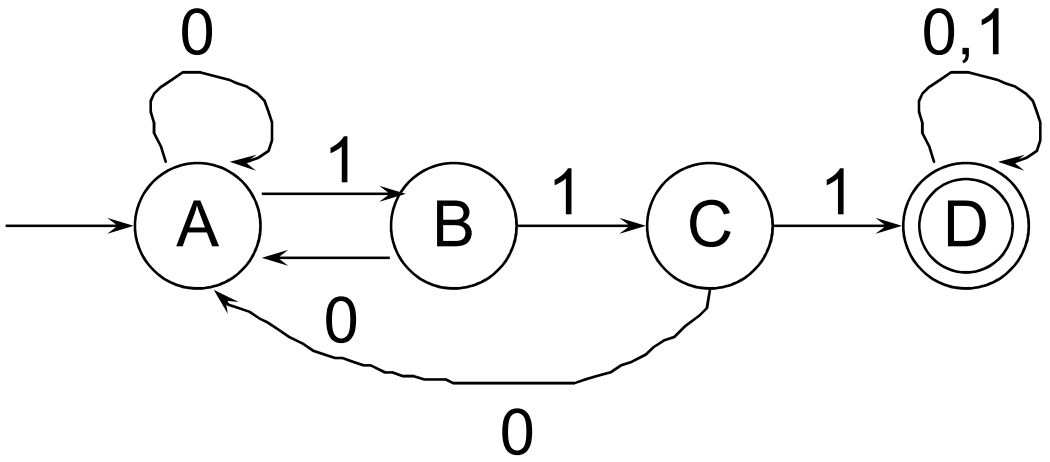
1 0 0 1
1 0 1 0 0 1
0 0 1 1

Q1. Is the DFA can recognize an *empty-string* ?

Q2. Draw a DFA of an *odd-parity* checker.

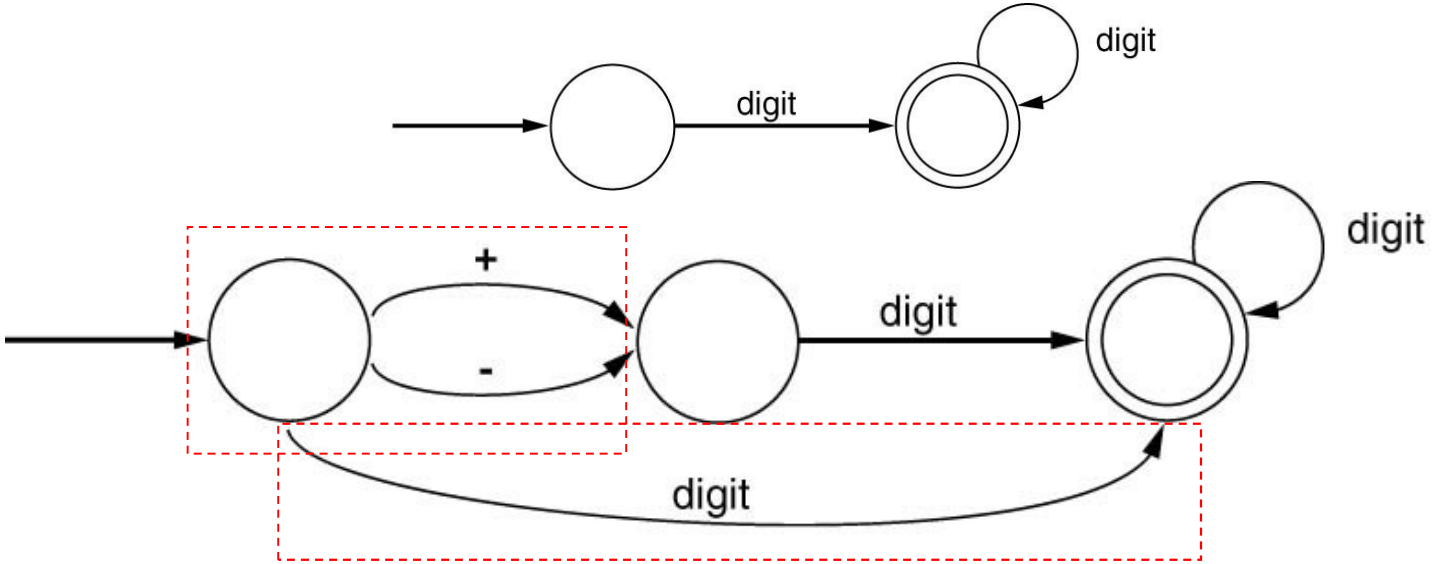
Quiz

Q: 아래 **DFA**는 어떤 특징을 갖는 문자열을 인식하는가?



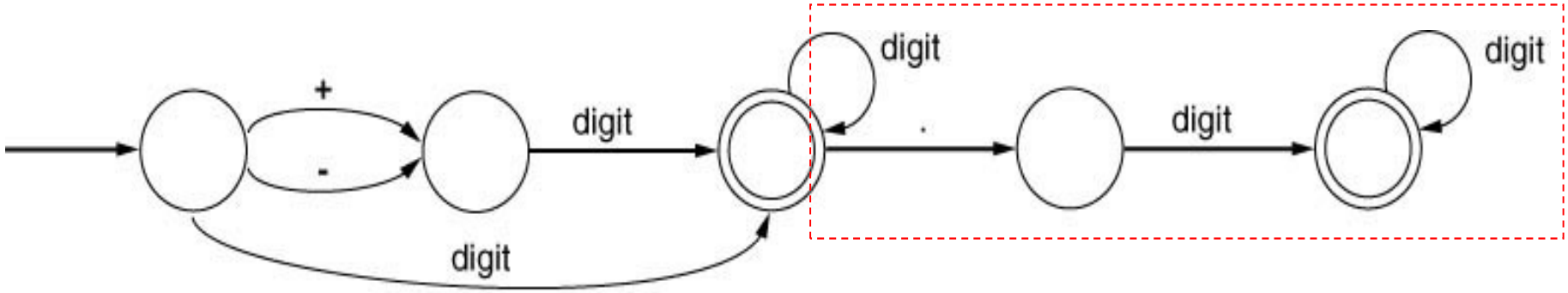
DFA for recognizing a signedNat

```
digit = [0-9]
nat = digit+
signedNat = (+|-)? nat
```



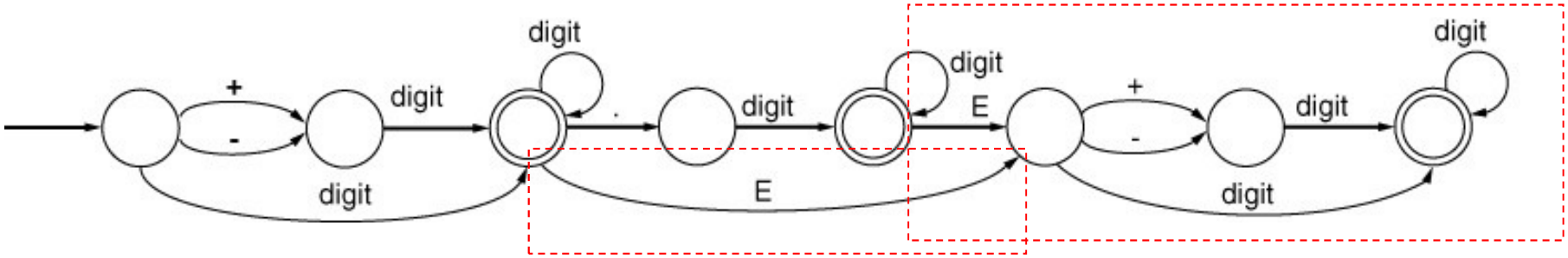
DFA for recognizing a number

```
digit = [0-9]
nat = digit+
signedNat = (+|-)? nat
number = signedNat ("." nat)?
```



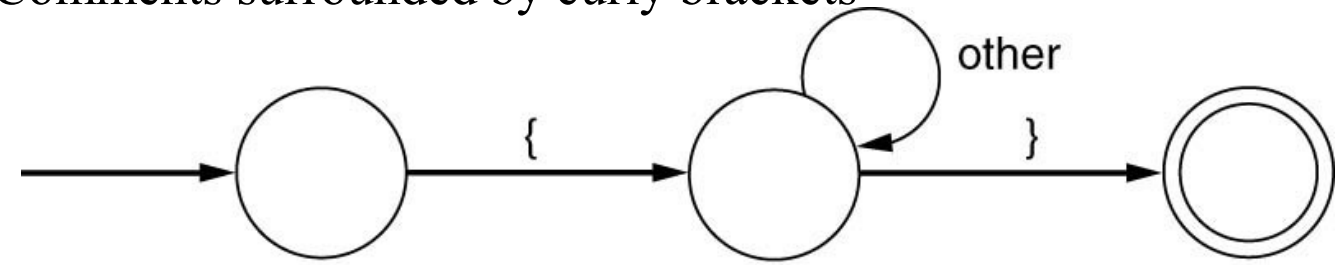
DFA *for* a floating-point number

```
digit = [0-9]
nat = digit+
signedNat = (+|-)? nat
number = signedNat ("." nat)? (E signedNat)?
```

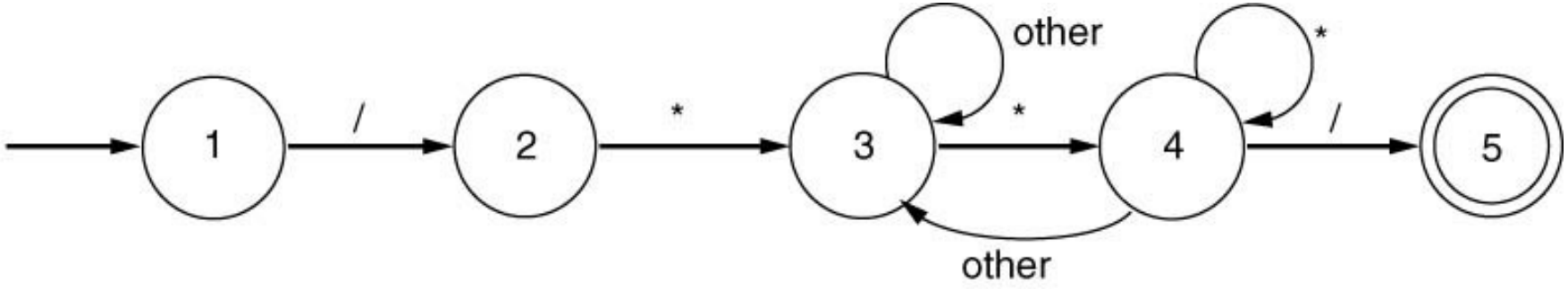


DFAs for recognizing comments

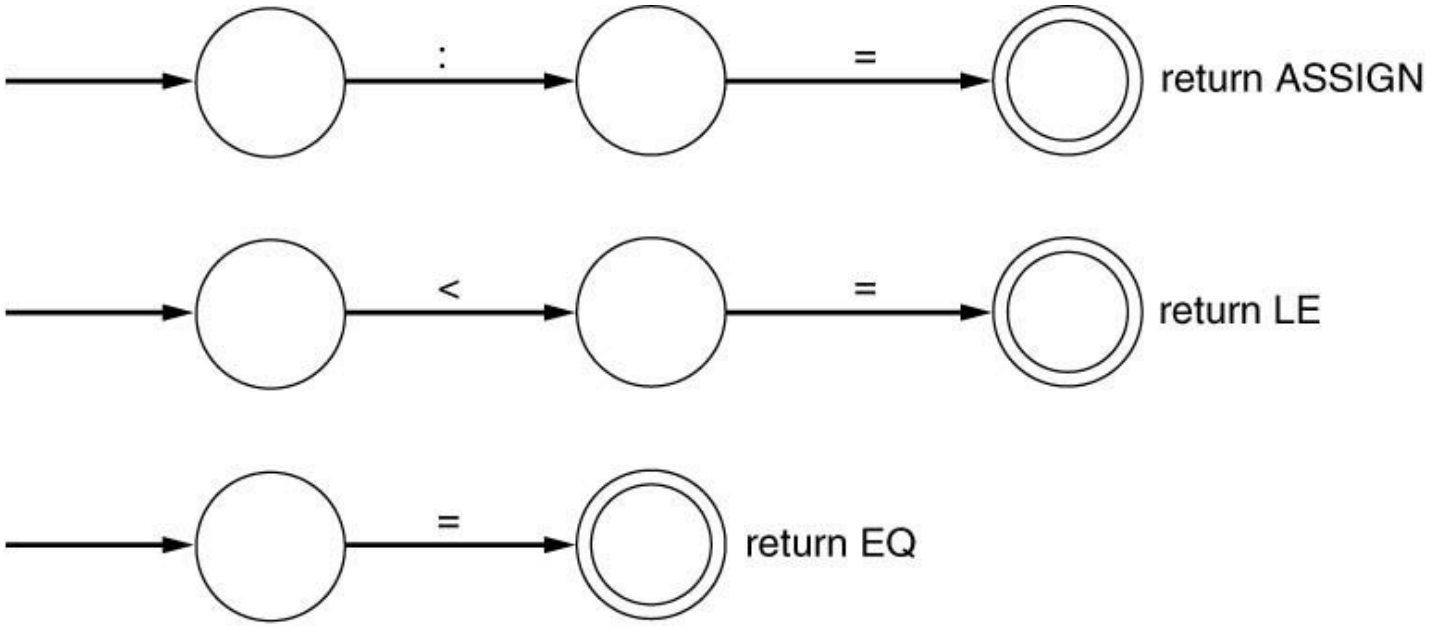
Comments surrounded by curly brackets



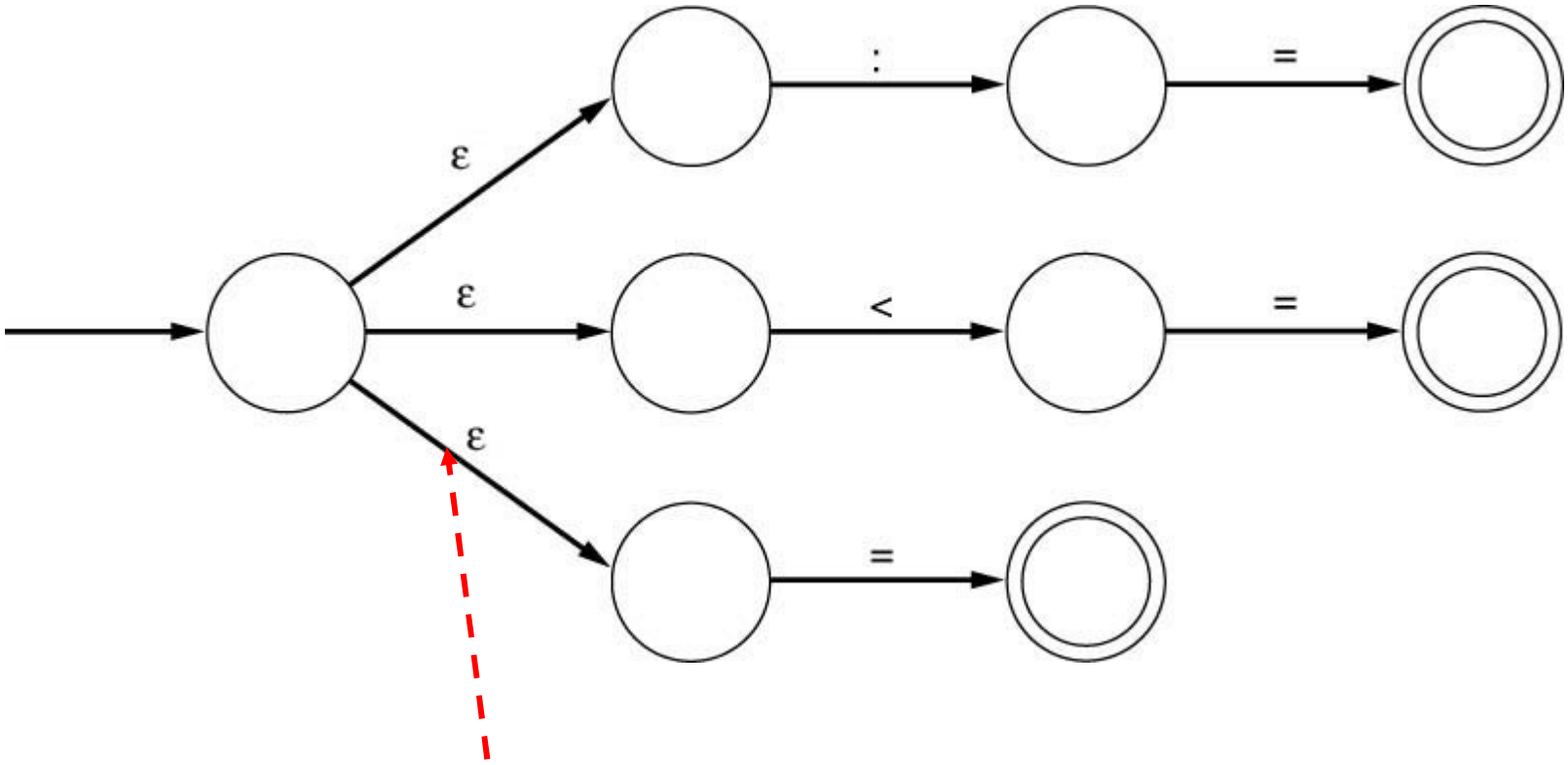
Comments that are delimited by a sequence of 2 characters



PL token에 대한 NFA 구성 예 (1/2)



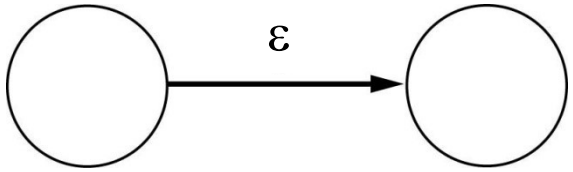
PL token에 대한 NFA 구성 예 (2/2)



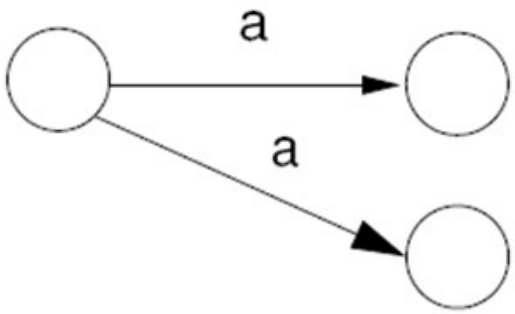
ε-transition may occur “spontaneously”

비결정적 유한 오토마타(NFA) (1/2)

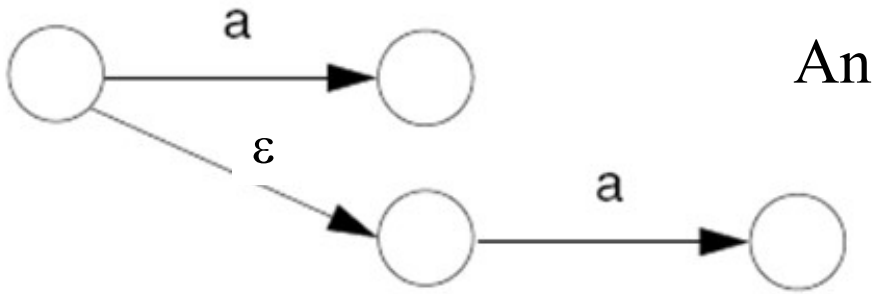
- **찾고자 하는 모든 token에 대한 DFA를 한 개의 DFA로 합쳐야 함.**
  - DFA의 경우 입력 기호에 대해 다음 상태가 *unique*하게 결정되어야 하기 때문에 하나로 합치기가 어려움
- **NFA : DFA 정의를 확장**
  - *More than one transition* from a state may exist
  - ε-transition
    - A transition may occur without consulting the input string.
    - Without lookahead and without change to the input string



비결정적 유한 오토마타(NFA) (2/2)



An NFA with two *a* transitions



An NFA with a *ε*- transitions

Why we need NFA?

정규 표현(RE)은 항상 FA로 나타낼 수 있다.

RE는 NFA로 쉽게 변환할 수 있다!

DFA는 coding이 쉽다!

NFA 정의

- **NFA** $M = (\Sigma, S, T, s_0, A)$
  - **상태 천이 함수 T** : $S \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathbf{P}(S)$
    - **P**(*S*) : the *power* set of S
      - 예: *S*={1,2} 이면 *P*(*S*) = { {}, {1}, {2}, {1,2} }

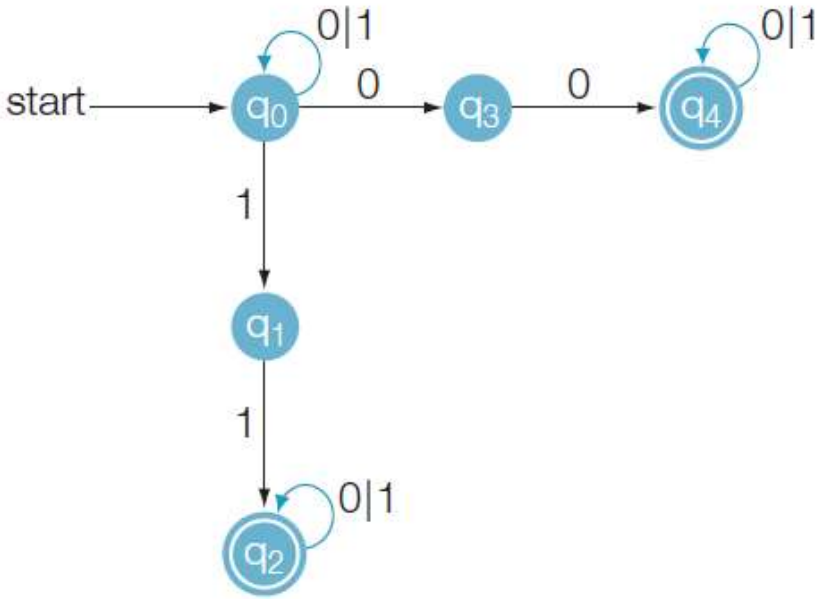
DFA

$T : S \times \Sigma \rightarrow S$

$$T(S_i, a) = S' \quad \text{단, } S' \subseteq S,\ a \in \Sigma \cup \{\varepsilon\}$$

- ***M*** **이 인식하는 언어 *L*(*M*)은 아래 조건을 만족하는 문자열 $c_1 c_2 \ldots c_n$ 의 집합 ($c_i \in \Sigma \cup \{\varepsilon\}$)**
  - $s_1 = T(s_0, c_1)$, $s_2 = T(s_1, c_2)$, … , $s_n = T(s_{n-1}, c_n)$인 상태들이 존재
  - 단, $s_n \in A$

NFA 예

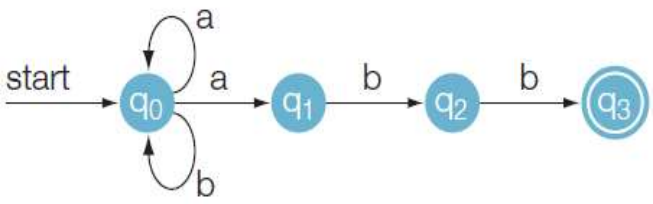


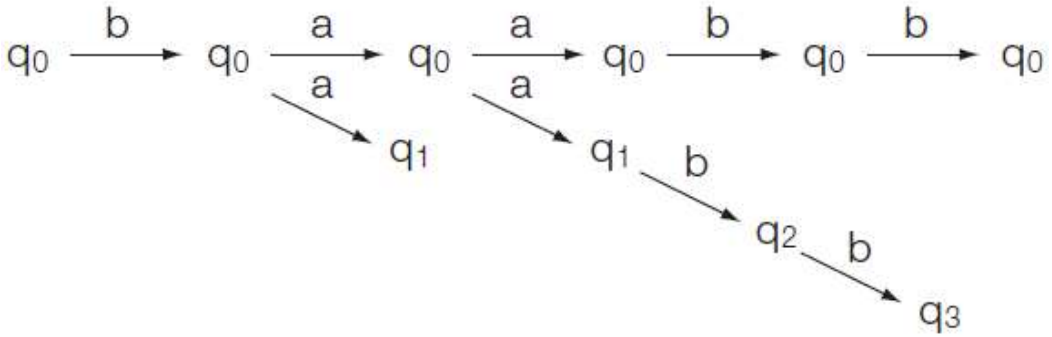
δ	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	ϕ	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	ϕ
q_4	$\{q_4\}$	$\{q_4\}$

NFA M 이 인식하는 언어 L(M)

- NFA $M = (\Sigma, S, T, s_0, A)$ 일 때 M 이 인식하는 언어 $L(M)$은
- 아래 조건을 만족하는 문자열 $c_1 c_2 \ldots c_n$ 의 집합 ($c_i \in \Sigma \cup \{\varepsilon\}$)
  - $s_1 = T(s_0, c_1)$, $s_2 = T(s_1, c_2)$, … , $s_n = T(s_{n-1}, c_n)$인 상태들이 존재
  - 단, $s_n \in A$

NFA 문자열 인식 예(1)





δ(q0, baabb) = δ(q0, aabb)

= δ(q0, abb) ∪ δ(q1, abb)

= {δ(q0, bb) ∪ δ(q1, bb)} ∪ ø

= δ(q0, b) ∪ δ(q2, b)

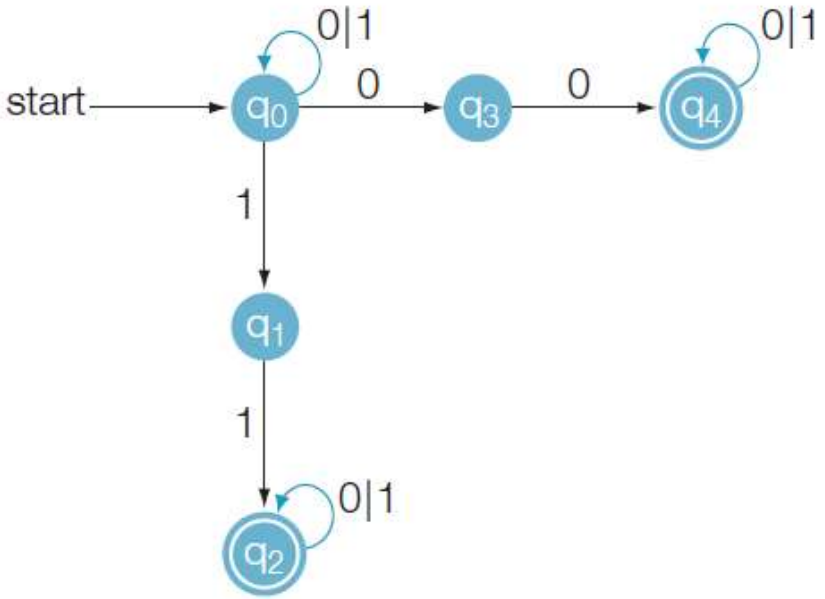
= {q0} ∪ {q3}

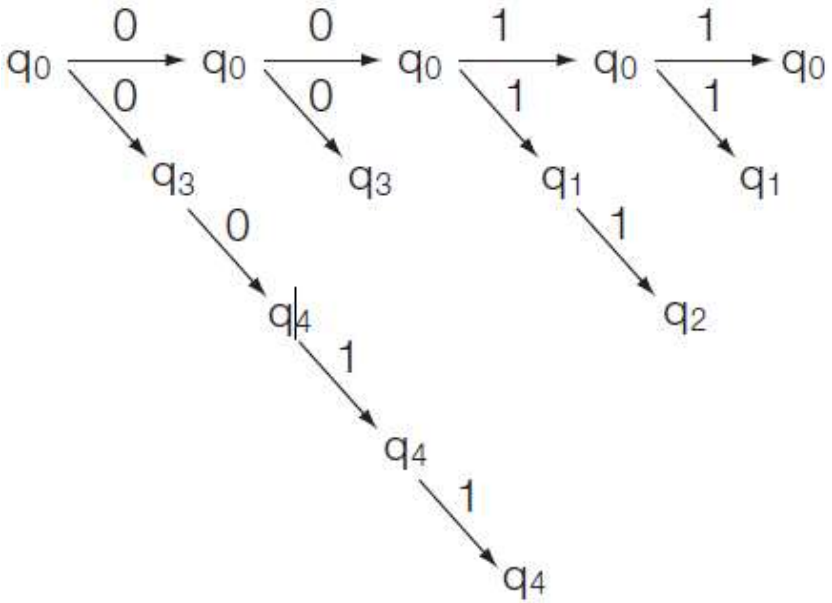
= {q0, q3}

{q0, q3} ∩ A = {q3}

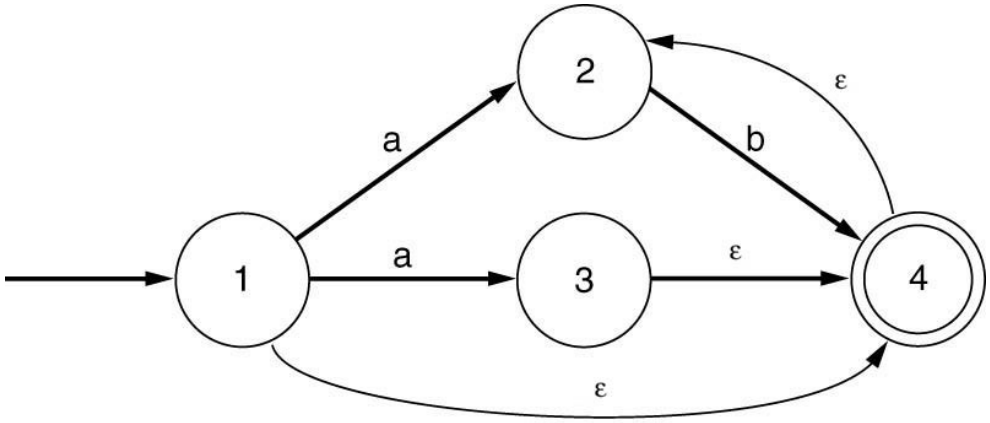
∴ 위 NFA는 문자열 **baabb**을 인식한다.

NFA 문자열 인식 예(2)


start
q0
q1
q2
q3
q4
0|1
0
1
0
1
0|1
0|1

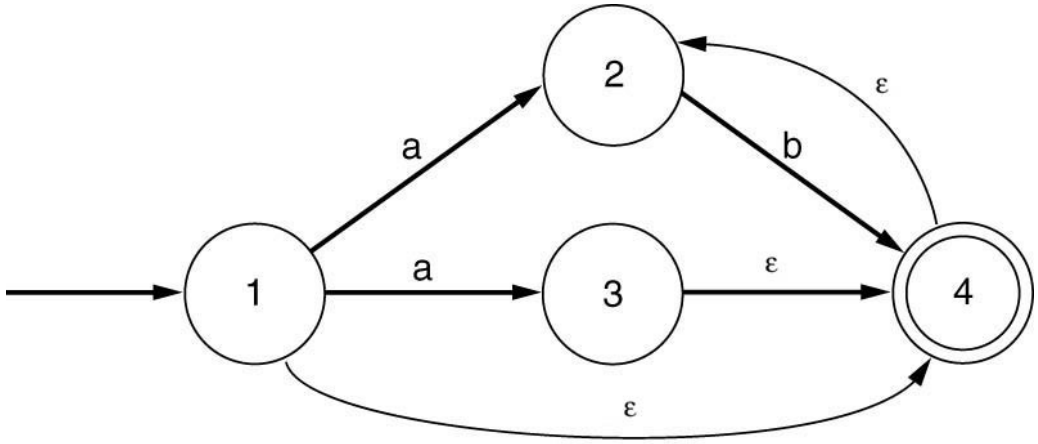

q0
q0
q0
q0
q0
q3
q3
q1
q1
q4
q2
q4
q4
0
0
0
0
1
1
1
1
0
1
1
1

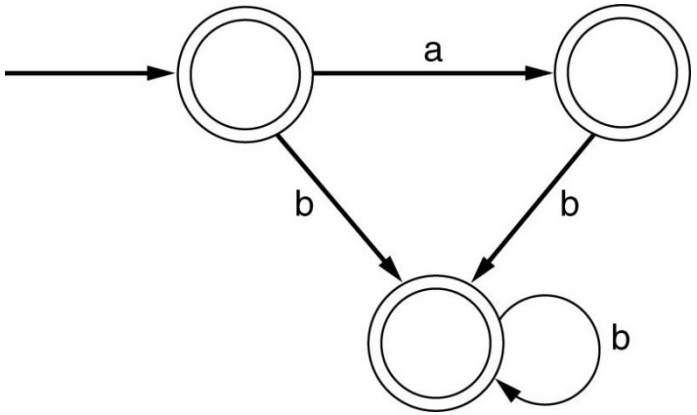
NFA 는 abb를 어떻게 인식할까?



시작상태 1 ➔ **ε-transition** ➔ { **1, 2, 4** }
첫 번째 문자 a : {1, 2, 4} 에 속한 각 상태에 대해
1 ➔ a ➔ {2, 3} ➔ **ε-transition** ➔ { **2, 3, 4** }
2 ➔ a ➔ {}
4 ➔ a ➔ {}
두 번째 문자 b : {2, 3, 4} 에 속한 각 상태에 대해
. . . . . .

NFA 를 DFA로 바꾸면?





An equivalent DFA

NFA는 똑같은 기능을 하는 **DFA**로 항상 변환할 수 있다**!**

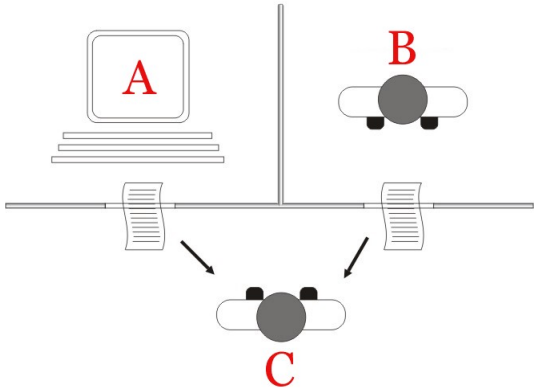
About Turing Machine

(supplementary material)

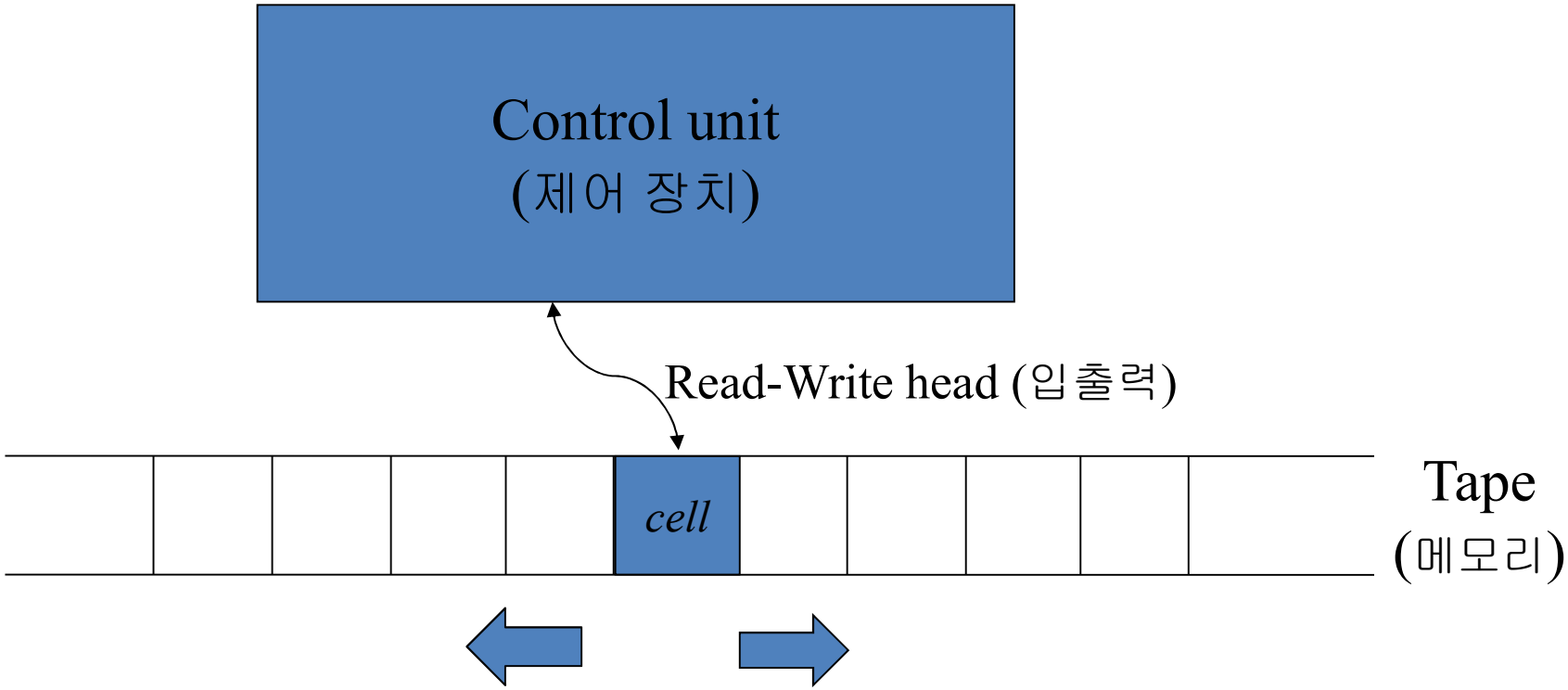
Do you know Alan Turing?

Do you know Alan Turing?

- Mathematician, logician, cryptanalyst
- *Turing* machine
- Manchester Mark 1
- Enigma machine
- Homosexual
- Apple
- Turing test



Turing Machine



Turing Machine 이란?

■ 테이프(tape) : 저장 공간

▪ 1차원 셀 배열

- 셀(cell)
  – 한 개의 symbol을 저장하고 있거나 비어있을 수 있다.
- 배열 크기는 무한대
  – 왼쪽 또는 오른쪽으로 무한대로 확장 가능

▪ Tape head

- 왼쪽 또는 오른쪽으로 셀 단위로 이동
- 셀에 저장되어 있는 symbol 을 읽어오거나 다른 symbol로 변경

Turing Machine

Turing Machine M

$$M = (Q, \Sigma, \Gamma, [\ ], \delta, q_0, F)$$

Σ : 입력 알파벳 (Γ 에서 공백 기호 제외)

Γ : 테이프 알파벳

$[\ ] \in \Gamma$: 공백 기호(*an empty symbol*)

δ : 천이 함수 $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

Q : 상태 집합

$q_0 \in Q$: control unit의 초기 상태(*start state*)

$F \subseteq Q$: 종료 상태(*final state*) 집합

Turing Machine

δ : $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

- Control unit의 현재 상태 Q
- tape head가 현재 가리키는 셀에 저장된 기호 Γ

- Control unit의 새로운 상태 Q
- 셀에 새롭게 저장되는 기호 Γ
- 이동 방향 : L 또는 R {L,R}