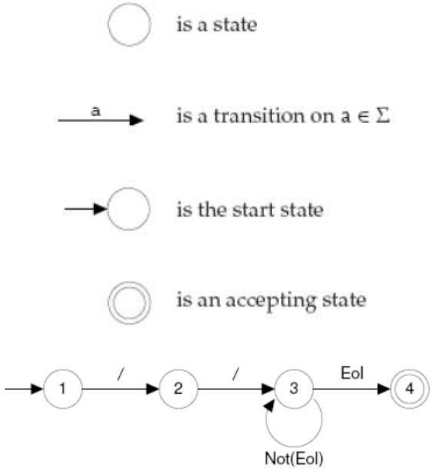
Finite Automata (sing. Automaton, abbrev FA)

- FAs have a finite number of states.
- FAs have a finite alphabet (Σ).
- Transitions **between states** are labeled with the characters from Σ or a character set $\Pi \in \Sigma$, or a **regular expression**.
- A period (**.**) is an alternative notation for Σ in FA diagrams (and a is used in many (all?) RE engine languages).
- FAs have a **single start state**, its incident edge is unlabeled. The first character of input is consumed in this state.
- FAs have a subset of states called **accepting states** (technically, this can be $\emptyset$ but we will only consider FAs with at least one accepting state).



Deterministic Finite Automata (DFA)

DFA $\equiv$ A finite automata with a **unique** transition for any character at any state.

1. Any **regular expression** can be expressed as a DFA
2. Any **DFA** can be expressed as a regular expression

$$\text{DFA} \quad \equiv \quad \text{RE}$$

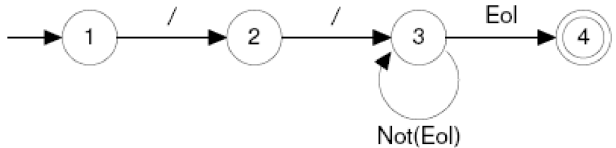
Coding DFAs — Explicit Algorithm

```
/⋆   Assume CurrentChar contains the first character to be scanned   ⋆/
if CurrentChar = '/'
then
    CurrentChar ← READ( )
    if CurrentChar = '/'
    then
        repeat
            CurrentChar ← READ( )
        until CurrentChar ∈ { Eol, Eof }
    else   /⋆ Signal a lexical error ⋆/
else   /⋆ Signal a lexical error ⋆/
if CurrentChar = Eol
then   /⋆ Finished recognizing a comment ⋆/
else   /⋆ Signal a lexical error ⋆/
```

Figure 3.4: Explicit control scanner.

Coding DFAs — Table Driven Algorithm

(a)



(b)

State	Character				
	/	Eol	a	b	...
1	2				
2	3				
3	3	4	3	3	3
4					

Figure 3.2: DFA for recognizing a single-line comment. (a) transition diagram; (b) corresponding transition table.

Coding DFAs — Table Driven Algorithm

```
/* Assume CurrentChar contains the first character to be scanned */
State ← StartState
while true do
    NextState ← T[State, CurrentChar]
    if NextState = error
    then break
    State ← NextState
    CurrentChar ← READ( )
if State ∈ AcceptingStates
then /* Return or process the valid token */
else /* Signal a lexical error */
```

Figure 3.3: Scanner driver interpreting a transition table.