

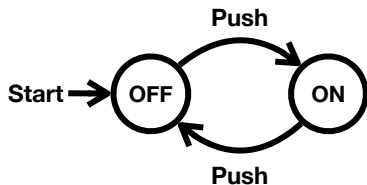
Lecture 3 – Deterministic Finite Automata (DFA)

COSE215: Theory of Computation

Jihyeok Park



2026 Spring



① Mathematical Preliminaries

- Mathematical Notations
- Inductive Proofs
- Notations in Languages

② Basic Introduction of Scala

- Basic Features
- User-Defined Data Types
- First-Class Functions
- Immutable Collections

1. Deterministic Finite Automata (DFA)

Definition

Transition Diagram and Transition Table

Extended Transition Function

Acceptance of a Word

Language of DFA (Regular Language)

Examples

Definition (Deterministic Finite Automata (DFA))

A **deterministic finite automaton** (DFA) is a 5-tuple:

$$D = (Q, \Sigma, \delta, q_0, F)$$

- Q is a finite set of **states**
- Σ is a finite set of **symbols**
- $\delta : Q \times \Sigma \rightarrow Q$ is the **transition function**
- $q_0 \in Q$ is the **initial state**
- $F \subseteq Q$ is the set of **final states**

$$D_1 = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$$

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

$$\delta(q_1, a) = q_2$$

$$\delta(q_2, a) = q_2$$

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

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

$$\delta(q_2, b) = q_0$$

```
// The type definitions of states and symbols
type State = Int
type Symbol = Char
// The definition of DFA
case class DFA(
  states: Set[State],
  symbols: Set[Symbol],
  trans: Map[(State, Symbol), State],
  initState: State,
  finalStates: Set[State],
)
```

```
// An example of DFA
val dfa1: DFA = DFA(
  states      = Set(0, 1, 2),
  symbols     = Set('a', 'b'),
  trans       = Map(
    (0, 'a') -> 1, (1, 'a') -> 2, (2, 'a') -> 2,
    (0, 'b') -> 0, (1, 'b') -> 0, (2, 'b') -> 0,
  ),
  initState   = 0,
  finalStates = Set(2),
)
```

$$D_1 = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$$

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

$$\delta(q_1, a) = q_2$$

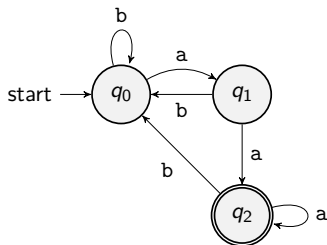
$$\delta(q_2, a) = q_2$$

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

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

$$\delta(q_2, b) = q_0$$

Transition Diagram



Transition Table

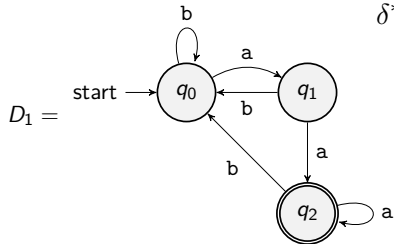
q	a	b
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_0
$*q_2$	q_2	q_0

where \rightarrow denotes the **initial state**
and $*$ denotes the **final state**.

Definition (Extended Transition Function)

For a given DFA $D = (Q, \Sigma, \delta, q_0, F)$, the **extended transition function** $\delta^* : Q \times \Sigma^* \rightarrow Q$ is defined as follows:

- **(Basis Case)** $\delta^*(q, \epsilon) = q$
- **(Induction Case)** $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$ where $a \in \Sigma, w \in \Sigma^*$

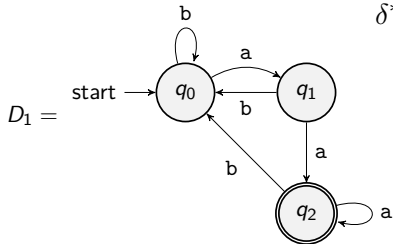


$$\begin{aligned}
 \delta^*(q_0, baa) &= \delta^*(\delta(q_0, b), aa) = \delta^*(q_0, aa) \\
 &= \delta^*(\delta(q_0, a), a) = \delta^*(q_1, a) \\
 &= \delta^*(\delta(q_1, a), \epsilon) = \delta^*(q_2, \epsilon) \\
 &= q_2
 \end{aligned}$$

Definition (Extended Transition Function)

For a given DFA $D = (Q, \Sigma, \delta, q_0, F)$, the **extended transition function** $\delta^* : Q \times \Sigma^* \rightarrow Q$ is defined as follows:

- **(Basis Case)** $\delta^*(q, \epsilon) = q$
- **(Induction Case)** $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$ where $a \in \Sigma, w \in \Sigma^*$



$$\begin{aligned}
 \delta^*(q_0, aba) &= \delta^*(\delta(q_0, a), ba) = \delta^*(q_1, ba) \\
 &= \delta^*(\delta(q_1, b), a) = \delta^*(q_0, a) \\
 &= \delta^*(\delta(q_0, a), \epsilon) = \delta^*(q_1, \epsilon) \\
 &= q_1
 \end{aligned}$$

```
// The type definition of words
type Word = String
case class DFA(...):
  ...
  // The extended transition function of DFA
  def extTrans(q: State, w: Word): State = w match
    case ""      => q
    case x <| w => extTrans(trans(q, x), w)

// An example transition for a word "baa"
dfa1.extTrans(0, "baa") // 2
// An example transition for a word "aba"
dfa1.extTrans(0, "aba") // 1
```

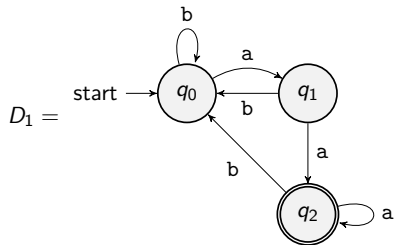
where `<|` is a helper function to extract the first symbol and the rest of the word but you do not need to understand the details of how it works.

```
// A helper function to extract first symbol and rest of word
object `<|` { def unapply(w: Word) = w.headOption.map((_, w.drop(1))) }
```



Definition (Acceptance of a Word)

For a given DFA $D = (Q, \Sigma, \delta, q_0, F)$, we say that D **accepts** a word $w \in \Sigma^*$ if and only if $\delta^*(q_0, w) \in F$

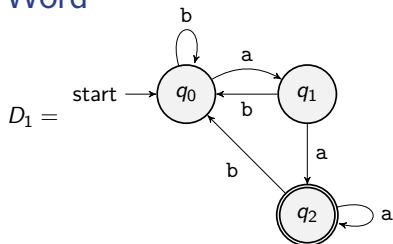


$$\delta^*(q_0, baa) = q_2 \in F$$

It means that D_1 **accepts** baa.

$$\delta^*(q_0, aba) = q_1 \notin F$$

It means that D_1 does **not accept** aba.



```
case class DFA(...):
  ...
  // The acceptance of a word by DFA
  def accept(w: Word): Boolean =
    finalStates.contains(extTrans(initState, w))

// An example acceptance of a word "baa"
dfa1.accept("baa") // true

// An example non-acceptance of a word "aba"
dfa1.accept("aba") // false
```

Definition (Language of DFA)

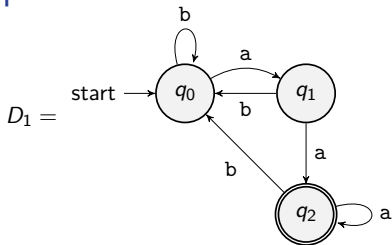
For a given DFA $D = (Q, \Sigma, \delta, q_0, F)$, the **language** of D is defined as:

$$L(D) = \{w \in \Sigma^* \mid D \text{ accepts } w\}$$

Definition (Regular Language)

A language L is **regular** if and only if there exists a DFA D such that $L(D) = L$

Example 1



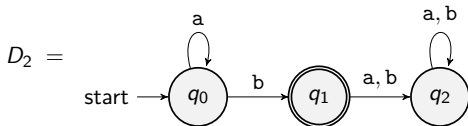
$\delta^*(q_0, baa) = q_2 \in F$
 $\Rightarrow D_1$ accepts baa
 $\Rightarrow baa \in L(D_1)$

$\epsilon, a, b, ab, ba, bb, aab, aba, abb, bab, \dots \notin L(D_1)$
 $aa, aaa, baa, aaaa, abaa, baaa, bbaa, \dots \in L(D_1)$

$$L(D_1) = \{waa \mid w \in \{a, b\}^*\}$$

- q_0 represents ϵ or any word ending with b
- q_1 represents any word ending with exactly one a
- q_2 represents any word ending with at least two a 's

Example 2



$\epsilon, a, aa, ba, bb, aaa, aba, abb, baa, bab, bba, \dots \notin L(D_2)$

$b, ab, aab, aaab, aaaab, aaaaab, aaaaaab, \dots \in L(D_2)$

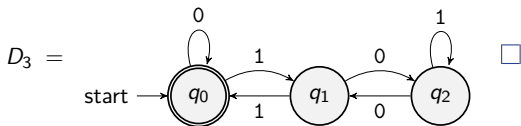
$$L(D_2) = \{a^n b \mid n \geq 0\}$$

- q_0 represents zero or more a's
- q_1 represents zero or more a's followed by b
- q_2 represents any other words

Theorem

The language $L = \{w \in \{0, 1\}^* \mid d(w) \equiv 0 \pmod{3}\}$ is regular ($d(w)$ is the natural number represented by w in binary).

Proof) You need to construct a DFA D_2 such that $L(D_2) = L$. Consider the following DFA D_2 :



- q_0 represents binary format of an integer n s.t. $n \equiv 0 \pmod{3}$
- q_1 represents binary format of an integer n s.t. $n \equiv 1 \pmod{3}$
- q_2 represents binary format of an integer n s.t. $n \equiv 2 \pmod{3}$

Theorem

The language $L = \{a^n b^n \mid n \geq 0\}$ is regular.

You need to construct a DFA D such that $L(D) = L$. However, it is **impossible** because L is actually **not regular**.

Then, is it possible to prove that L is not regular?

Yes, it is possible BUT you will learn how to prove it (using Pumping Lemma) later in this course.

1. Deterministic Finite Automata (DFA)

Definition

Transition Diagram and Transition Table

Extended Transition Function

Acceptance of a Word

Language of DFA (Regular Language)

Examples

- Nondeterministic Finite Automata (NFA)

Jihyeok Park

`jihyeok_park@korea.ac.kr`

`https://plrg.korea.ac.kr`