KAIST 전산학과 최광무

유한상태 오토마타(Finite State Automata[1])는 Chomsky의 4가지 언어 **계급**에서 **가장 낮은 계급**인 type 3, **정규언어**(regular languages)를 이해[2]하는 기계(automata)[3]다.

결정적 유한상태 오토마타(Deterministic Finite State Automata; DFA) M은 요소 다섯 개로 이루어져 있다. 즉 $M = (Q, \Sigma, \delta, q_0, F)$로 쓰고,

(1) Q은 **상태**(state)들의 **유한** 집합이다.

(2) Σ는 **입력문자**(input symbol)들의 **유한** 집합(input vocabulary)이다.

(3) δ는 **상태변화함수**(state transition function)로서 상태집합 Q와 입력문자 Σ의 **순서쌍**을 정의역으로, 상태집합 Q를 치역으로 가진다. 즉

$\delta:\ Q \times \Sigma \rightarrow Q$로 정의하고,

현(current) **상태** $q \in Q$에서 **입력문자** $a \in \Sigma$를 **보고**[4] **상태**가 $p \in Q$로 바뀔 때, $(\delta(q, a) = p) \in \delta$ 혹은 $\delta(q, a) = p$라 짧게 쓴다.

(4) $q_0 \in Q$는 **처음상태**(initial state)라고 불리는 특별한(distinguished) 상태이다.

(5) $F \subseteq Q$는 **끝나는 상태**(final state)라고 불리는 특별한 상태들의 집합이다.

DFA $M = (Q, \Sigma, \delta, q_0, F)$은 주어진 입력문자열 $x = a_1 a_2 \cdots a_n (\forall i:\ 1 \leq i \leq n:\ a_i \in \Sigma)$에 대하여 다음과 같이 동작한다.

(A) 현 상태(current state)는 **시작상태** $q_0 \in Q$에서 **시작**하고, 현 입력문자는 입력문자열 $x = a_1 a_2 \cdots a_n$에 가장 왼쪽 문자 a_1에서 **시작**한다.

(B) 현 상태가 $q \in Q$이고 현 입력문자가 입력문자열 $x = a_1 a_2 \cdots a_n$에서 i번째 문자 $a_i \in \Sigma$이었다면, **새로운** 현 상태는 상태변화함수 $\delta(q, a_i) \in Q$가 되고 **새로운** 현 입력문자는 a_i 다음에 있는 문자 a_{i+1}이 된다.

(C) 작업 (B)를 **반복**적으로 계속 수행하여 새로운 현재 입력문자가 문자열의 **오른쪽 끝**(a_n)을 넘어서면 반복 작업 (B)를 마친다.

(D) 입력문자열 x의 문자를 모두 보아서(오른쪽 끝까지 가서) 반복 작업이 **끝났을 때**, 최종상태 q가 **끝나는 상태**($q \in F$)이면 주어진 문자열 x는 오토마타 M이 **받아들이는**(accept) 문자열이 되어 **true**를 return하고, 최종상태 q가 **끝나는 상태**가 **아니면**($q \notin F$) 받아들이는 문자열이 아니므로 **false**를 return한다.

---

1) Automata는 복수이고 단수는 automaton이다.

2) 오토마타가 언어를 이해한다는 말은 언어의 원소판별문제(membership problem)를 오토마타가 해결해준다는 뜻이다. 즉 어휘 Σ에서 정의한 언어 $L \subseteq \Sigma^*$에 대하여, 특정 오토마타 M이 모든(임의의) 문자열 $x \in \Sigma^*$에 대하여 $x \in L$이면 $M(x)$는 **true**를 $x \notin L$이면 $M(x)$는 **false**를 **항상** 대답하면 오토마타 M이 언어 $L \subseteq \Sigma^*$를 **이해**한다고 정의하고, 언어 L과 M는 **같다**(equivalent)고 정의한다.

3) 기계(machine) 대신 오토마타라는 새로운 용어를 쓴 이유는 기존의 기계와는 달리 생각할 수 있는(?) 기계라는 포함한 자동기계(automatic machine)라는 뜻을 오토마타라는 용어가 포괄한다고 본 것이다.

4) 현재입력문자 $a \in \Sigma$는 이미 보았으므로, a의 다음(오른쪽) 문자를 본다.

오토마타 동작을 **프로그램**(함수)으로 표시하면 다음과 같다.

```
function M(x: 문자열) boolean5);
variable q ∈ 상태(Q); a ∈ 기본문자(Σ); i ∈ int;
(* assume x = a[1]a[2] … a[n] *)
   q := q0; i := 1; a := a[i];                        (A)
   while i <= n do         (* n ≧ 06) *)              (C)
         q := δ(q, a); i := i + 1; a := a[i]           (B)
         od
   if q ∈ F → return true                              (D)
   | q ∉ F → return false                              (D)
   fi
(* return q ∈ F *)
```

상태변화함수의 확장

오토마타 $M = (Q, \Sigma, \delta, q_0, F)$ 에서 주어진 입력문자열 $x = a_1 a_2 \cdots a_n (\forall i: 1 \leqq i \leqq n: a_i \in \Sigma)$에 대한 동작을 좀 더 자세히 살펴보자. 처음상태 q_0에서 첫 번째 입력문자 a_1을 보고 바뀐(transition) 상태를 q_1, 다음 상태 q_1에서 다음 문자 a_2을 보고 바뀐 상태를 q_2라 하면,

$\delta(q_0, a_1) = q_1$, $\delta(q_1, a_2) = q_2$, $\cdots$, $\delta(q_{n-1}, a_n) = q_n$.

즉 $0 \leq \forall i < n : \delta(q_i, a_{i+1}) = q_{i+1}$이고 $q_0, q_1, \cdots, q_n \in Q$, $a_1, \cdots, a_n \in \Sigma$.

마지막 상태 $q_n \in F$이면 true를 아니면 false를 return.

이 때 중간 상태 $q_1, q_2, \cdots, q_{n-1}$을 제거하고 연속하여 표현하면

$\delta(\delta(\cdots \delta(\delta(q_0, a_1), a_2), \cdots a_{n-1}), a_n) = q_n \in F$인지 아닌지를 return.

이를 좀 더 확장(일반화)하여 쓰면

$\delta^n(q_0, a_1 a_2 \cdots a_{n-1} a_n) \in F$인지 아닌지를 return.

δ^n: $Q \times \Sigma^n \rightarrow Q$ 을 recursive하게 정의하면

$\delta^0(q, \epsilon) \stackrel{\text{def}}{=}_{\mathrm{B}} q$ 단 $q \in Q$,

$\delta^n(q, xa) \stackrel{\text{def}}{=}_{\mathrm{R}} \delta(\delta^{n-1}(q, x), a)$ 단 $q \in Q$, $x \in \Sigma^*$, $a \in \Sigma$[7].

우리는 임의의 자연수 $n \geq 0$에 대하여

---

5) **boolean** = {**true**, **false**}이다.

6) $n = 0$이면 $x = \epsilon$이라고 하자.

7) δ의 두 번째 파라미터가 Σ에서 Σ^n으로 확장되는 δ^n의 정의에서 $\stackrel{\text{def}}{=}_{\mathrm{B}}$는 ϵ으로, $\stackrel{\text{def}}{=}_{\mathrm{R}}$는 $xa \in \Sigma^+$으로 서로 소(disjoint)임에 주의하라.

$\delta^n: Q \times \Sigma^n \to Q$ 을 정의하였다. 이를 Σ^n을 Σ^*로 확장할 때와 같이

$\delta^* \overset{\text{def}}{=} \bigcup_{i \in N_0} \delta^i$로 정의하면

$\delta^*: Q \times \Sigma^* \to Q$ 8)으로 확장하여 정의할 수 있다.

$\delta^0(q, \epsilon) \overset{\text{def}}{=}_B q$ 단 $q \in Q$,

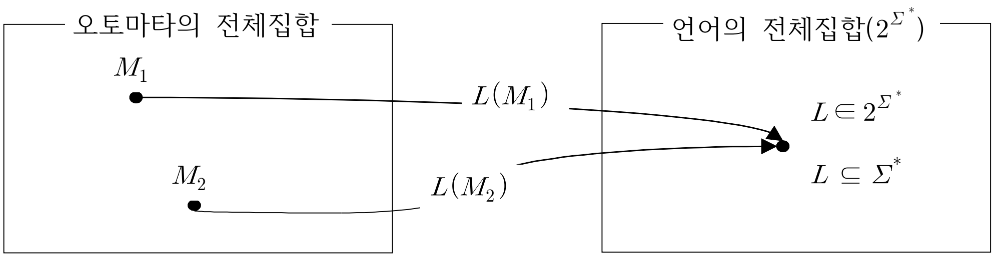
$\delta^*(q, xa) \overset{\text{def}}{=}_R \delta(\delta^*(q, x), a)$ 단 $q \in Q$, $x \in \Sigma^*$, $a \in \Sigma$.

(정의) DFA $M = (Q, \Sigma, \delta, q_0, F)$ 이 받아들이는 언어(문자열들의 집합)

$$L(M) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in F\}$$

을 정의한다. 이 때 언어 $L(M)$을 오토마타 M이 **정의**하는 언어라고도 한다.

(정의) 두 개의 오토마타 M_1과 M_2가 가 정의하는 언어가 같으면, 즉 $L(M_1) = L(M_2)$이면, $M_1 = M_2$라 쓰고, M_1과 M_2는 **같은 일을 하는**(**equivalent**; **같은**) 오토마타 부른다.

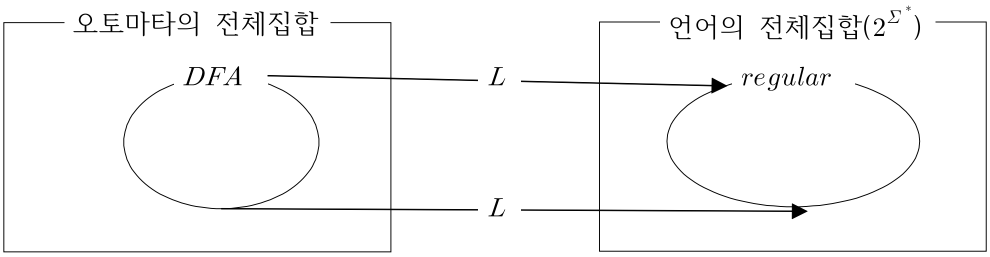


같은 언어 $L \subseteq \Sigma^*$을 **정의하는(같은 일을 하는** 그러나) 서로 **다른** DFA M은 **여러** 개이다. 그러나 그 중 **대표 오토마타**는 상태 수 $|Q|$가 가장 작은(Minimal DFA) **하나**9)로 4장에서 정의할 것이다.

<u>Regular 언어와 regular 언어 클래스의 정의</u>

(정의) 임의의 언어 $L \subseteq \Sigma^*$을 정의하는 DFA M이 존재한다면, $L = L(M)$, 그 언어 L은 **regular 언어**라고 부른다.

(정의) Regular 언어들의 집합을 **regular 언어 클래스**라 정의하고 이를 **type** 3(계급) **언어 클래스**라고도 부른다.



---

8) 교과서에서는 $\hat{\delta}$ 을 사용하나 강의노트에서는 δ^*로 쓴다.
9) 상태 이름만 바뀌면 다른 DFA이지만 구조적으로는 같은(structurally isomorphic) DFA는 같다는 관점에서 대표 오토마타는 **하나**이다.

언어 클래스[10]는 주목해야할 개념이다. 앞으로 **더 높은**[11] 언어 클래스들과 오토마타 클래스, 문법 클래스, 함수 클래스, 프로그램 클래스 등등 간의 **같음**(**equivalent**; **같은 일을 하는**)이라는 개념이 등장할 것이다.

10) 언어 클래스는 언어들의 집합인데, 언어도 집합(문자열들의 집합)이므로 집합(언어)의 집합이라는 말보다는 집합(언어) 클래스라는 말을 더 즐겨 쓴다.
11) Type 3에 상위 클래스로 type 2인 문맥자유(context free), type 1인 끝나는(recursive), type 0인 프로그램 할 수 있는 (recursively enumerable)이 있으며, type 1의 하위 클래스로 NP-complete (intractable)와 polynomial(tractable)이 있다.