



내공 있는 프로그래머로 길러주는

컴파일러의 이해

Chapter 03

형식언어와 유한 오토마타

목차

01 형식언어

02 형식문법

03 문법 표기법

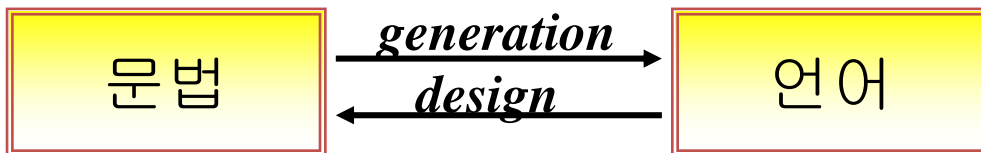
04 유한 오토마타

학습목표

- 형식언어를 이해할 수 있다.
- 형식문법을 이해할 수 있다.
- 문법의 표기법에 대해 이해할 수 있다.
- 유한 오토마타에 대해 이해할 수 있다.

3.1 형식 언어

- 언어 : 알파벳으로 부터 생성되는 모든 문자열들의 부분집합
- 문법 : 언어는 문법(grammar)에 의해서 생성되고 정의된다.



- 인식기 : 언어는 인식기에 의해 인식된다.
- [표 3-2] 언어와 인식기

표 3-2 언어와 인식기

문법	언어	인식기
type 0(무제약 문법)	재귀 열거 언어	튜링 기계(turing machine)
type 1(문맥인식 문법)	문맥인식 언어	선형한계 오토마타(linear-bounded automata)
type 2(문맥자유 문법)	문맥자유 언어	푸시다운 오토마타(push-down automata)
type 3(정규 문법)	정규 언어	유한 오토마타(finite automata)

- **[정의 3-1]알파벳(Alphabet)** : 언어의 문장을 이루는 기본적인 기호(Symbol)
 - 알파벳이란 공집합이 아닌 기호들의 유한 집합으로 Σ 로 표시한다.
 - **[ex 3-1]** : $\Sigma_1 = \{\neg, \wedge, \vee, \rightarrow, \dots, \text{ㅎ}\}$
 $\Sigma_2 = \{a, b, c, \dots, \text{ㄷ}\}$
 $\Sigma_3 = \{0, 1\}$
 - 알파벳 의미를 일반 프로그래밍 언어에서는 사용 가능한 문자나 기호들의 집합이라고 설명한다. C 언어의 경우에 알파벳은 영문 소문자, 영문자 대문자, 숫자 0~9, 특수 문자 +, *, -, _, ... 등이다
- **[정의 3-2]문자열(String)** : 알파벳 Σ 에 대한 문자열은 알파벳에서 정의된 기호들을 나열한 유한 수열(finite sequence)
 - **[ex 3-2]** : 알파벳 $\Sigma = \{a, b, c\}$ 일 때 $a, ca, ccba$ 등이 문자열
 - 문자열은 언어 이론에서는 종종 **문장(sentence)**이나 **단어(word)**와 동의어로 사용된다. 언어를 구성하는 각각의 알파벳 기호들은 하나의 문자로 표시하는데 영어의 a, b, c, \dots 을 주로 사용하고 문자열의 이름은 u, v, w, \dots 로 나타낸다. 예를 들어 $w = abaaa$ 는 $abaaa$ 라는 값을 갖는 문자열의 이름이 w 임을 나타낸다.

- **[정의 3-3] 문자열의 길이(Length)** : 문자열을 이루는 기호의 개수이며, 어떤 문자열 w 의 길이는 $|w|$ 로 표시하고, w 의 cardinality
 - **[ex 3-3]** $w_1 = abc$ $w_2 = abab$ 의 문자열에 대한 길이
 - $|w_1| = 3$ $|w_2| = 4$
 - **[ex 3-4]** : $w = v_1 v_2 v_3 \cdots v_k$ 일 때 $|w| = k$
- **[정의 3-4] 문자열의 접속(Concatenation)** : 두 개의 문자열을 연결하여 새로운 문자열을 만드는 연산
 - 문자열 u, v 가 각각 $u = a_1 a_2 a_3 \cdots a_n$, $v = b_1 b_2 b_3 \cdots b_m$ 일 경우에 두 문자열의 접속은 $u \cdot v$ 혹은 uv 로 표시하고 $uv = a_1 a_2 a_3 \cdots a_n b_1 b_2 b_3 \cdots b_m$ 이다.
 - **[예제 3-1]** 두 문자열 $u = \text{dog}$, $v = \text{house}$ 가 있을 때 $u \cdot v$ 와 $v \cdot u$
 - $u \cdot v = \text{doghouse}$ $v \cdot u = \text{housedog}$
 - **[예제 3-2]** [예제 3-1]에서 $|u \cdot v|$ 와 $|v \cdot u|$ 를 구해보자.
 - 접속 연산에 대한 길이
 - $|u \cdot v| = |u| + |v| = |\text{dog}| + |\text{house}| = 8$
 - $|v \cdot u| = |v| + |u| = |\text{house}| + |\text{dog}| = 8$

■ [정의 3-5]공문자열(Empty string) : 문자열의 길이가 0인 문자열

- ε 으로 표기

- 어떤 문자열 u, v 에 대하여 다음과 같은 속성을 가짐

- $u\varepsilon = u = \varepsilon u$

- $u\varepsilon v = uv$

- Empty string을 λ 로도 표시하며 널 문자열(null string)이라고도 한다. 문자열에서의 연산 중 접속 연산은 ε 을 **항등원**으로 취한다. a^n 은 a 가 n 개 연결된 문자열을 표시하며, a^0 은 공문자열을 나타낸다. 즉, $a^n = aaa \cdots a$ (a 가 n 개)이고 $a^0 = \varepsilon$ 이 된다.

■ w^R 은 문자열 w 의 역(reverse)로 w 를 이루는 요소들의 역순으로 얻어진다.

- [예제3-3] : 문자열 $w = ccba$ 에 대한 w^R 은 문자열 w 의 역순

- $\rightarrow w^R = abcc$

■ [정의 3-6] 접두사(Prefix) : 문자열 $w = uv$ 일 때 u 를 문자열 w 의 접두사

▪ 진접두사(Proper prefix) : $u \neq \varepsilon$ 인 접두사

▪ 접미사(Suffix) : 문자열 $w = uv$ 일 때 v 를 문자열 w 의 접미사

▪ 진접미사(Proper suffix) : $v \neq \varepsilon$ 인 접미사

▪ [ex3-5] : 문자열 $w = \text{house}$

→ 접두사 : $\varepsilon, h, ho, hou, hous, house$

진접두사 : $h, ho, hou, hous, house$

접미사 : $house, ouse, use, se, e, \varepsilon$

진접미사 : $house, ouse, use, se, e$

■ [정의 3-7] Σ^* (reflexive transitive closure, Kleene closure) : 공문자열을 포함하는 알파벳 Σ 에 대한 접속 연산에 의해 만들어 질 수 있는 모든 문자열들의 집합으로 Σ -스타(Σ star, 클리니 클로저)라고 읽음.

Σ^+ (transitive closure, positive closure) : Σ^* 에서 공문자열을 제외한 모든 문자열의 집합으로 Σ -대거(Σ dagger, 포지티브 클로저)라고 읽고 $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ 이다.

▪ [ex3-6] $\Sigma = \{0, 1\}$

→ $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$

$\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, \dots\}$

▪ [ex3-7] $\Sigma = \{a, b, c\}$

→ $\Sigma^* = \{\epsilon, a, b, c, aa, bb, cc, ab, ba, ac, ca, bc, cb, aaa, \dots\}$

$\Sigma^+ = \{a, b, c, aa, bb, cc, ab, ba, ac, ca, bc, cb, aaa, \dots\}$

■ [정의 3-8] 언어 L : 알파벳 Σ 에 대해서 Σ^* 의 부분집합

▪ 유한 언어(finite language) : 언어에 속하는 문자열의 수가 유한 개일 경우

▪ 무한 언어(infinite language) : 언어에 속하는 문자열의 수가 무한 개일 경우

▪ [ex3-8] : $\Sigma = \{a, b\}$

→ $L_1 = \Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, \dots\}$ 는 무한언어

$L_2 = \{a, ba, bbb\}$ 는 유한언어

$L_3 = \{aaa, bbb, aba, bba\}$ 는 유한언어

$L_4 = \{a^p \mid p \text{는 소수(prime number)}\}$ 는 무한언어

$L_5 = \{a^n b^n \mid n \geq 1\}$ 는 무한언어

- 여기서 언어는 의미(semantic)의 개념은 포함하지 않는다는 것
- 언어란 단지 문자열들의 집합이기 때문에 형식 언어 이론은 문자열 집합의 속성에 관한 이론

■ 언어에서의 연산 :

▪ [정의 3-9] 두 언어의 합집합

- 두 언어 L 과 M 의 합집합(union) $L \cup M$ 은 L 에 속하는 문자열이거나 M 에 속하는 문자열
- $L \cup M = \{s \mid s \in L \text{ 혹은 } s \in M\}$
- [ex 3-9] [ex 3-8]에서 $L_2 \cup L_3$

$$\rightarrow L_2 \cup L_3 = \{a, ba, bbbb, aaa, bbb, aba, bba\}$$

▪ [정의 3-10] 두 언어의 접속

- 두 언어 L 과 M 의 접속(concatenation) LM 은 L 에 속하는 문자열과 M 에 속하는 문자열을 접속한 것으로 교환 법칙은 성립하지 않음.
- $LM = \{st \mid s \in L \text{ 혹은 } t \in M\}$
- [ex 3-10] [ex 3-8]에서 $L_4 L_5$

$$\rightarrow L_4 L_5 = \{a^{p+n} b^n \mid p \text{는 소수}, n \geq 1\}$$

▪ [정의 3-11] L의 거듭제곱

- 언어 L의 거듭제곱은 재귀적으로 다음과 같이 정의한다
- $L^0 = \{\epsilon\}$
- $L^n = LL^{n-1}$ 단, $n \geq 1$
- [ex 3-11] $L = \{a, ba\}$ 라 하면

→ $L^3 = \{aaa, aaba, abaa, ababa, baaa, baaba, babaa, bababa\}$

▪ [정의 3-12] 클리니 클로저, 포지티브 클로저

- 언어 L의 클리니 클로저 L^* 는 다음과 같이 정의한다.
- $L^* = L^0 \cup L^1 \cup L^2 \cup \dots \cup L^n \cup \dots = \bigcup_{i=0}^{\infty} L^i$
- 또한 언어 L의 포지티브 클로저 L^+ 는 다음과 같이 정의한다.
- $L^+ = L^1 \cup L^2 \cup \dots \cup L^n \cup \dots = \bigcup_{i=1}^{\infty} L^i = L^* - L^0$
- 예 : $L = \{0, 1\}$ 이라면

→ L^* 는 ϵ 을 포함하여 0과 1로 만들어지는 모든 문자열

L^+ 는 0과 1로 만들어지는 모든 문자열

- 형식 문법은 크게 두 가지 방법으로 정의할 수 있다. 생성 규칙 production rule만을 가지고 표현 하거나 [정의 3-13]과 같이 네 가지 항목으로 정의하는 것이다

- [정의 3-13] 문법 $G=(V_N, V_T, P, S)$ 정의

1. V_N : 논터미널 기호들의 유한 집합
2. V_T : 터미널 기호들의 유한 집합

$$V_N \cap V_T = \phi, \quad V_N \cup V_T = V$$

- 터미널 기호와 논터미널 기호를 문법 기호(*grammar symbol*)라 하며 보통 V (vocabulary)로 표시

3. P : 생성 규칙(production rule)의 집합으로 다음과 같다.

$$\alpha \rightarrow \beta, \quad \alpha \in V^+, \quad \beta \in V^*$$

α 를 왼쪽 부분(left-hand side), β 를 오른쪽 부분(right-hand side)

\rightarrow 는 왼쪽 부분에 있는 기호가 오른쪽 부분에 있는 기호로 단순히 대체

4. S : V_N 에 속하는 기호로서 다른 논터미널 기호들과 구별하여 시작 기호(start symbol)

■ 앞으로 사용될 기호들의 일반적인 표기법

- 1) A, B, C와 같은 영문자 대문자로 구성된 기호와 시작기호를 나타내는 S 는 논터미널 기호이다.
- 2) $<$ 와 $>$ 로 묶어서 나타낸 기호도 논터미널 기호이다.
- 3) a, b, c와 같은 영문자 소문자로 구성된 기호와 $+$, $-$ 와 같은 연산자 기호, 괄호나 쉼표와 같은 구분자, 0, 1, 2와 같은 아라비아 숫자 등은 터미널 기호이다.
- 4) X, Y, Z와 같은 영문자 끝부분의 대문자는 터미널 기호와 논터미널 기호를 나타내는 문법 기호이다.
- 5) 영문자 끝부분의 소문자인 u, v, w, x, y, z 등은 터미널 기호들로 이루어진 문자열을 나타낸다.
- 6) α , β , γ 와 같은 그리스어 소문자는 문법기호로 구성된 문자열을 나타낸다.
- 7) 만약 아무런 언급이 없으면 첫 번째 생성규칙의 왼쪽에 있는 기호가 시작기호.
- 8) 생성 규칙의 왼쪽 부분에 같은 기호로 구성된 생성 규칙이 여러 개일 때 축약해서 사용할 수 있다. 예를 들어 2개의 생성 규칙 $\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$ 가 있을 때 간단히 $\alpha \rightarrow \beta_1 \mid \beta_2$ 로 표현한다.

- [예제 3-4] 문법에 대해 몇 가지 성질 알아보기 1

- 문법 $G = (\{S, A\}, \{0, 1\}, P, S)$

생성규칙 $P: S \rightarrow 0AS \quad S \rightarrow 0$

$A \rightarrow S1A \quad A \rightarrow 10 \quad A \rightarrow SS$

- [풀이] S, A 는 논터미널 기호, 0과 1은 터미널 기호, S 는 시작 기호

생성 규칙의 개수는 5개

- [예제 3-4] 문법을 축약하면 다음과 같다.

$S \rightarrow 0AS \mid 0$

$A \rightarrow S1A \mid 10 \mid SS$

.

- [예제 3-5] 문법에 대해 몇 가지 성질 알아보기 2

- $$P : E \rightarrow E + T \mid E - T \mid T$$

- $$T \rightarrow T * F \mid T / F \mid F$$

- $$F \rightarrow (E) \mid id$$

[풀이] E, T, F는 논터미널 기호이고 +, -, *, /, (,), id는 터미널 기호, E는 시작 기호이다. 그리고 생성 규칙의 개수는 8개이다.

- 정의된 문법으로부터 어떤 언어가 생성되는지, 언어가 문법에 맞는지를 알기 위해 유도를 설명한다
- \Rightarrow : 유도(derivation)
 - 만약 $\alpha \rightarrow \beta$ 가 존재 하고, $\gamma, \delta \in V^*$ 이면
$$\gamma \alpha \delta \Rightarrow \gamma \beta \delta$$
로 표시
 - 즉, 한 문자열에서 생성 규칙을 한 번 적용해서 다른 문자열로 바꾸는 것을 나타낸다.
- \Rightarrow^* : 영 번 이상의 유도(zero or more derivation)
- \Rightarrow^+ : 한 번 이상의 유도(one or more derivation)
 - $\alpha_1 \xRightarrow{n} \alpha_n : \alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n$ 이 V^* 에 속하고 $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \dots \Rightarrow \alpha_n$ 이 존재
 - α_1 은 α_n 을 생성(produce) 혹은 유도(derive)한다고 함
 - α_n 은 α_1 으로 감축(reduce)된다고 함.
- \rightarrow 는 생성 규칙에서 사용되는 기호로 단일 화살표(single arrow)라 하고, \Rightarrow 는 유도 과정을 나타내는 기호로 이중 화살표(double arrow)라 한다.

■ [예제 3-6] 유도하기

■ [예제 3-4]의 문법에서 문자열 0, 0000, 001100 등이 허용되는지 유도

■ [풀이] 괄호 안은 적용된 생성 규칙이다.

- ① $S \Rightarrow 0$ ($S \rightarrow 0$)
 - 문자열 0은 유도에 의해 생성되므로 문법에 맞는 문장이다.
- ② $S \Rightarrow 0AS$ ($S \rightarrow 0AS$)
 - $\Rightarrow 0SSS$ ($A \rightarrow SS$)
 - $\Rightarrow 00SS$ ($S \rightarrow 0$)
 - $\Rightarrow 000S$ ($S \rightarrow 0$)
 - $\Rightarrow 0000$ ($S \rightarrow 0$)
 - 이 과정은 $S \xRightarrow{*} 0000$ 과 같이 나타낼 수도 있다.
 - 문자열 0000은 유도에 의해 생성되므로 문법에 맞는 문장이다.

- ③ $S \Rightarrow 0AS$ ($S \rightarrow 0AS$)
- $\Rightarrow 0S1AS$ ($A \rightarrow S1A$)
- $\Rightarrow 001AS$ ($S \rightarrow 0$)
- $\Rightarrow 00110S$ ($A \rightarrow 10$)
- $\Rightarrow 001100$ ($S \rightarrow 0$)
- 이 과정은 $S \xRightarrow{*} 001100$ 과 같이 나타낼 수도 있다.
- 문자열 001100은 유도에 의해 생성되므로 문법에 맞는 문장이다

■ [예제 3-7] 유도하기2

■ $id + (id * id)$ 가 [예제 3-5]의 문법에 의해 생성되는 문자열인지 확인

■ [풀이] 괄호 안은 적용된 생성 규칙이다.

- $E \Rightarrow E + T (E \rightarrow E+T)$
- $\Rightarrow T + T (E \rightarrow T)$
- $\Rightarrow F + T (T \rightarrow F)$
- $\Rightarrow id + T (F \rightarrow id)$
- $\Rightarrow id + F (T \rightarrow F)$
- $\Rightarrow id + (E) (F \rightarrow (E))$
- $\Rightarrow id + (T) (E \rightarrow T)$
- $\Rightarrow id + (T * F) (T \rightarrow T * F)$
- $\Rightarrow id + (F * F) (T \rightarrow F)$
- $\Rightarrow id + (id * F) (F \rightarrow id)$
- $\Rightarrow id + (id * id) (F \rightarrow id)$

▪ 이 과정은 $E \xRightarrow{*} id + (id * id)$ 와 같이 나타낼 수도 있다.

▪ 문자열 $id + (id * id)$ 는 유도에 의해 생성되므로 문법에 맞는 문장이다.

■ [정의 3-15] 문장과 문장형태

- $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \dots \Rightarrow w_n \Rightarrow w$ 와 같은 유도과정이 있을 때 $S \xRightarrow{*} w$ 이고 w 가 V^* 에 속하면 w 를 문장 형태(sentential form)라 하고, w 가 V_T^* 에 속할 경우, w 를 문장(sentence)이라 한다.
- 즉, 문장 형태는 터미널 기호와 논터미널 기호들의 조합으로 구성되고 문장은 터미널 기호들로만 구성
- 우리가 프로그램을 작성할 때 사용하는 터미널 기호로만 구성된 문장을 언어라 한다.

■ [예제 3-8] 문장과 문장형태 구하기

- [예제 3-6]의 ②에서 이미 유도를 했는데, 이때 문장과 문장 형태를 구해보자
- [풀이]
 - 유도 과정 중에 있는 문자열 $S, 0AS, 0SSS, 00SS, 000S, 0000$ 등은 문장 형태이고, 터미널 기호로만 구성된 0000 은 문장이다. 문장 0000 은 시작 기호 S 로부터 생성된다고 한다.

■ [예제 3-9] 문장과 문장형태 구하기

- [예제 3-7]에서 이미 유도를 했는데, 이때 문장과 문장 형태를 구해보자
- [풀이]
 - 유도 과정 중에 있는 문자열 E , $E + T$, $T + T$, $F + T$, $id + T$, $id + F$, $id + (E)$, $id + (T)$, $id + (T * F)$, $id + (F * F)$, $id + (id * F)$, $id + (id * id)$ 등은 문장 형태이고, 터미널 기호로만 구성된 $id + (id * id)$ 는 문장이다. 문장 $id + (id * id)$ 는 시작 기호 E 로부터 생성된다고 한다.
- 이렇게 문법 G 로부터 생성되는 문장의 집합을 언어라 하고 $L(G)$ 로 표기한다. 문법 G 로부터 생성되는 언어는 시작 기호에서 유도될 수 있는 모든 문장의 집합이다

■ [정의 3-16] 문법 G 에 의해 생성된 언어 $L(G)$

- 문법 G 에 의해 생성되는 언어는 G 에 의해 생성되는 문장의 집합이며 $L(G)$ 로 표기한다. $L(G) = \{w \mid S \xRightarrow{*} w, w \in V_T^*\}$

■ [예제 3-10] 문법으로부터 언어 생성하기 1

- 다음과 같은 문법으로부터 생성되는 언어를 살펴보자.

- $G_1 = (\{S, A, B, C\}, \{a, b\}, P, S)$

- $P : S \rightarrow A$

- $A \rightarrow abC \mid aABC$

- $bB \rightarrow bbb$

- $bC \rightarrow bb$

- [풀이]

- 시작 기호로부터 생성되는 문장은 다음과 같다.

- ① $S \Rightarrow A (S \rightarrow A)$

- $\Rightarrow abC (A \rightarrow abC)$

- $\Rightarrow abb (bC \rightarrow bb)$

- ② $S \Rightarrow A$ ($S \rightarrow A$)
 - $\Rightarrow aABC$ ($A \rightarrow aABC$)
 - $\Rightarrow aabCBC$ ($A \rightarrow abC$)
 - $\Rightarrow aabbBC$ ($bC \rightarrow bb$)
 - $\Rightarrow aabbbbC$ ($bB \rightarrow bbb$)
 - $\Rightarrow aabbbbb$ ($bC \rightarrow bb$)
- ③ $S \Rightarrow A$ ($S \rightarrow A$)
 - $\Rightarrow aABC$ ($A \rightarrow aABC$)
 - $\Rightarrow aaABCBC$ ($A \rightarrow aABC$)
 - $\Rightarrow aaabCBCBC$ ($A \rightarrow abC$)
 - $\Rightarrow aaabbCBCBC$ ($bC \rightarrow bb$)
 - $\Rightarrow aaabbbbCBC$ ($bB \rightarrow bbb$)
 - $\Rightarrow aaabbbbbBC$ ($bC \rightarrow bb$)
 - $\Rightarrow aaabbbbbbbC$ ($bB \rightarrow bbb$)
 - $\Rightarrow aaabbbbbbbb$ ($bC \rightarrow bb$)
 - \vdots
- $L(G_1) = \{a^n b^m \mid n, m \geq 1\}$

■ [예제 3-11] 문법으로부터 언어 생성하기 2

- 다음과 같은 문법으로부터 생성되는 언어를 살펴보자.

- $G_2 = (\{S, A\}, \{b, d\}, P, S)$

- $P : S \rightarrow dAd$

- $A \rightarrow dAd$

- $A \rightarrow b$

▪ [풀이]

- 시작 기호로부터 생성되는 문장은 다음과 같다.
- ① $S \Rightarrow dAd$ ($S \rightarrow dAd$)
 - $\Rightarrow dbd$ ($A \rightarrow b$)
- ② $S \Rightarrow dAd$ ($S \rightarrow dAd$)
 - $\Rightarrow ddAdd$ ($A \rightarrow dAd$)
 - $\Rightarrow ddbdd$ ($A \rightarrow b$)

- ③ $S \Rightarrow dAd$ ($S \rightarrow dAd$)
- $\Rightarrow ddAdd$ ($A \rightarrow dAd$)
- $\Rightarrow dddAddd$ ($A \rightarrow dAd$)
- $\Rightarrow dddbddd$ ($A \rightarrow b$)
- \vdots
- $L(G_2) = \{d^nbd^n \mid n \geq 1\}$

3.2 형식 문법

- 앞에서 정의한 문법의 개념은 촘스키(Chomsky, N.)에 의해서 처음으로 소개되었고, 그는 문법을 생성 규칙에 가해지는 제한에 따라 네 종류로 나누었으며, 이를 **촘스키 계층 구조**라고 부른다.

표 3-1 촘스키 계층 구조

유형	생성 규칙의 형태	설명
0	$\phi A \psi \rightarrow \phi W \psi$	무제한(unrestricted) 문법이라고 한다. 생성 규칙에 어떠한 제한도 두지 않는 경우이다.
1	$\alpha \rightarrow \beta$ 단, $ \alpha \leq \beta , \alpha \in V^+, \beta \in V^*$	문맥인식(context-sensitive) 문법이라고 한다. 생성 규칙에 $ \alpha \leq \beta $ 의 제한을 가하는 것으로 α 는 공문자열이 될 수 없다.
2	$A \rightarrow \beta$ 단, $A \in V_N, \beta \in V^*$	문맥자유(context-free) 문법이라고 한다. 생성 규칙의 왼쪽 부분은 하나의 논터미널 기호이고 오른쪽 부분은 문자열이다.
3	$A \rightarrow tB, A \rightarrow t$ 또는 $A \rightarrow Bt, A \rightarrow t$ 단, $t \in V_T^*, A, B \in V_N$	정규(regular) 문법이라고 한다. 생성 규칙에 따라 두 가지 종류가 있는데, $A \rightarrow tB, A \rightarrow t$ 와 같은 생성 규칙을 갖는 것을 우선형(right-linear) 문법이라 하고 $A \rightarrow Bt, A \rightarrow t$ 와 같은 생성 규칙을 갖는 것을 좌선형(left-linear) 문법이라 한다.

■ 어떤 문법이 주어졌을 때, 그 문법이 어디에 속하는지 판별해 보고 그 언어가 생성하는 언어를 살펴보자.

- 문법을 판별하는 방법
 - 우선 문맥 인식 문법인지를 판별
 - 문맥 인식 문법이 아니라면 무제약 문법
 - 문맥 인식 문법이라면 문맥 자유 문법인지 판별
 - 문맥 자유 문법이 아니라면 문맥 인식 문법
 - 문맥 자유 문법이라면 정규 문법인지 판별
 - 정규 문법이 아니라면 문맥 자유 문법

■ [예제 3-14] 문법의 종류 판별하기 1

- [예제 3-4]의 문법이 네 가지 문법 중 어디에 속하는지 판별해보자.
- $S \rightarrow 0AS \mid 0$
- $A \rightarrow S1A \mid 10 \mid SS$
- **[풀이]** 우선 문맥인식 문법인지 판별하려면 각 생성 규칙의 길이를 검사해본다. 모든 생성 규칙이 $\alpha \rightarrow \beta$ 에서 $|\alpha| \leq |\beta|$ 를 만족하므로 문맥인식 문법이다. 다음으로 문맥자유 문법인지는 생성 규칙의 왼쪽 부분이 논터미널 기호 하나로 구성되어 있는지 확인하면 되는데 모두 맞으므로 문맥자유 문법이다. 마지막으로 정규 문법인지 알아보면, 생성 규칙 $S \rightarrow 0AS$ 에서 생성 규칙의 오른쪽 부분이 정규 문법에 맞지 않기 때문에 정규 문법이 아니다. 그러므로 이 문법은 문맥자유 문법이다.

■ [예제 3-15] 문법의 종류 판별하기 2

- [예제 3-5]의 문법이 네 가지 문법 중 어디에 속하는지 판별해보자.
- $E \rightarrow E + T \mid E - T \mid T$
- $T \rightarrow T * F \mid T / F \mid F$
- $F \rightarrow (E) \mid id$
- **[풀이]** 우선 문맥인식 문법인지 알아보면 모든 생성 규칙이 $\alpha \rightarrow \beta$ 에서 $|\alpha| \leq |\beta|$ 를 만족하므로 문맥인식 문법이다. 그리고 생성 규칙의 왼쪽 부분이 논터미널 기호 하나로 구성되어 있으므로 문맥자유 문법이다. 하지만 생성 규칙 $E \rightarrow E + T$ 에서 생성 규칙의 오른쪽 부분이 정규 문법에 맞지 않기 때문에 정규 문법이 아니다. 그러므로 이 문법은 문맥자유 문법이다.

■ [예제 3-16] 문법의 종류 판별하기 3

- [예제 3-10]의 문법이 네 가지 문법 중 어디에 속하는지 판별해보자.
- $P : S \rightarrow A$
- $A \rightarrow abC \mid aABC$
- $bB \rightarrow bbb$
- $bC \rightarrow bb$
- **[풀이]** 우선 문맥인식 문법인지 알아보면 모든 생성 규칙이 $\alpha \rightarrow \beta$ 에서 $|\alpha| \leq |\beta|$ 를 만족하므로 문맥인식 문법이다. 그리고 생성 규칙 $bB \rightarrow bbb$ 에서 생성 규칙의 왼쪽 부분이 논터미널 기호 하나로 이뤄져 있지 않기 때문에 문맥자유 문법이 아니다. 그러므로 이 문법은 문맥인식 문법이다

■ [예제 3-18] 문법의 종류 판별하기 5

- [예제 3-12]의 문법이 네 가지 문법 중 어디에 속하는지 판별해보자.
- $P : S \rightarrow 0S \mid 0B$
- $B \rightarrow 1C$
- $C \rightarrow 0C \mid 0$
- **[풀이]** 우선 문맥인식 문법인지 알아보면 모든 생성 규칙이 $\alpha \rightarrow \beta$ 에서 $|\alpha| \leq |\beta|$ 를 만족하므로 문맥인식 문법이다. 그리고 생성 규칙의 왼쪽 부분이 논터미널 기호 하나로 구성되어 있으므로 문맥자유 문법이다. 또한 생성 규칙 모두 정규 문법에 맞으며 우선형 문법이다

■ 네 가지 문법의 포함 관계

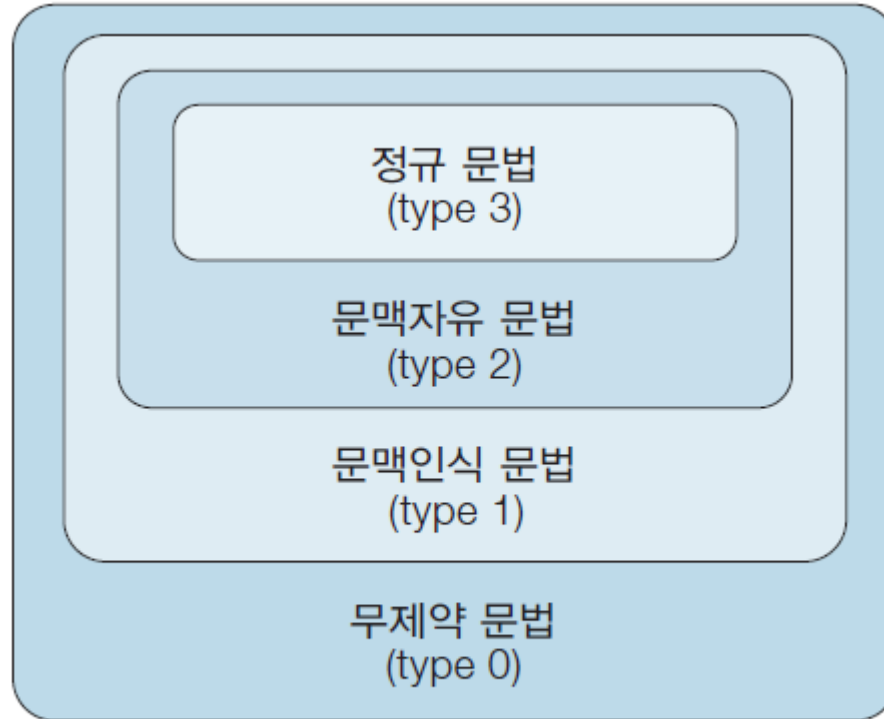


그림 3-1 네 가지 문법의 부분집합 관계

■ 일반적으로 형식 언어 이론에서 자주 인용되는 언어들

1) 단순 매칭 언어(simple matching language) : CFL

$$L_m = \{a^n b^n \mid n \geq 0\}$$

2) 중복 매칭 언어(double matching language) : CSL

$$L_{dm} = \{a^n b^n c^n \mid n \geq 0\}$$

3) 좌우 대칭 언어(mirror image language) : CFL

$$L_{mi} = \{ww^R \mid w \in V_T^*\}$$

4) 회문 언어(palindrome language) : CFL

$$L_r = \{w \mid w = w^R\}$$

5) 괄호 언어(parenthesis language) : CFL

$$L_p = \{w \mid w \text{는 balanced parenthesis}\}$$

■ [표 3-2] 언어와 인식기

표 3-2 언어와 인식기

문법	언어	인식기
type 0(무제한 문법)	재귀 열거 언어	튜링 기계(turing machine)
type 1(문맥인식 문법)	문맥인식 언어	선형한계 오토마타(linear-bounded automata)
type 2(문맥자유 문법)	문맥자유 언어	푸시다운 오토마타(push-down automata)
type 3(정규 문법)	정규 언어	유한 오토마타(finite automata)

- 정규표현(Regular Expression)
- 문법도표(Syntax Diagram)
- BNF
- EBNF

■ 정규표현 : 정규문법을 가장 잘 표현할 수 있는 방법

■ [정의 3-17] 다음과 같이 재귀적으로 정의

■ 기본 단계

- 알파벳 Σ 에 대해서

- 1. $\phi, \varepsilon, a \in V_T$ 는 모두 정규 표현이다.

- 2. 만일 r, s 가 정규 언어 L_r, L_s 를 표현하는 정규 표현이라 하면,

- (1) $(r) + (s)$ 는 $L_r \cup L_s$ 를 나타내는 정규 표현이다.

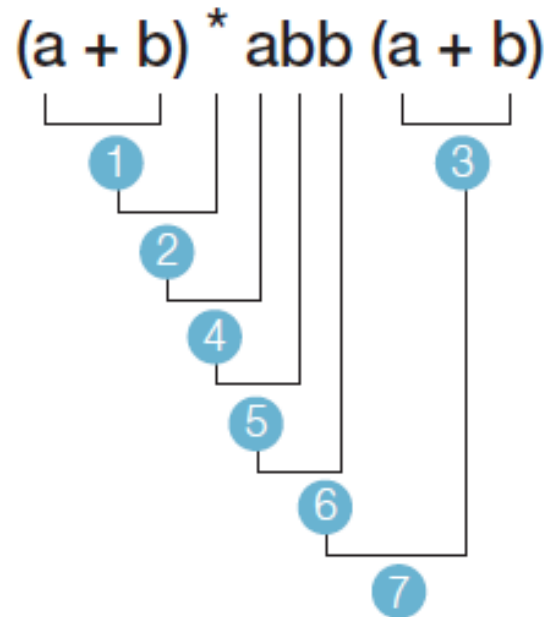
- (2) $(r) \cdot (s)$ 는 $L_r \cdot L_s$ 를 나타내는 정규 표현이다.

- (3) $(r)^*$ 는 $(L_r)^*$ 를 나타내는 정규 표현이다.

- 위의 정의에서 연산자의 연산자 우선 순위(precedence)는 $* > \cdot > +$ 이며 괄호는 연산 순위 때문에 사용하였다. 만약 연산자 $*$ (클리니 클로저)가 우선 순위가 가장 높고 왼쪽 결합 법칙을 택하며, 연산자 \cdot (접속)이 두 번째 우선 순위를 갖고 왼쪽 결합 법칙을 가지며, 연산자 $+$ (합집합)가 가장 낮은 우선 순위를 갖고 왼쪽 결합 법칙을 갖는다. 만약 연산자 우선 순위와 결합 법칙이 결정되면 괄호를 사용하지 않아도 될 것이다.

3.3 문법의 표기법

- 다음 정규 표현식의 연산 순서는?



■ [예제 3-20] 정규표현에 의해 생성되는 언어

■ 알파벳 $\Sigma = \{0, 1\}$ 에 대해 정규 표현이 나타내는 언어는?

■ ① $0 + 1$ ② $(0 + 1)0$ ③ 0^* ④ $(0 + 1)^*$

■ [풀이]

- ① $0 + 1$ 은 언어 $\{0, 1\}$ 을 나타낸다.
- ② $(0 + 1)0$ 은 언어 $\{00, 10\}$ 을 나타낸다.
- ③ 0^* 는 언어 $\{\epsilon, 0, 00, 000, \dots\}$ 을 나타낸다.
- ④ $(0 + 1)^*$ 는 언어 $\{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$ 을 나타낸다. 즉 ϵ 을 포함하여 0과 1로 만들 수 있는 모든 문자열의 집합이다.

■ [ex 3-12]

■ $(a + b)^*abb$ 는 a 와 b 로 만들어지는 모든 문자열 중에서 abb 로 끝나는 문자열의 집합이다.

■ [예제 3-21] 정규표현인지 판단하고 생성되는 언어 구하기

- $\Sigma = \{0, 1, a, b\}$ 에 대해 정규 표현인지 아닌지를 판단하고 생성되는 언어를 설명해보자.
- ① $0^*(0 + 1)^*$
- **[풀이]** ① [정의 3-17]의 기본 단계 ③번에 의해 0과 1은 정규 표현이고, 0이 정규 표현이므로 귀납 단계 ③번에 의해 0^* 는 정규 표현이다. 같은 방법으로 0과 1이 정규 표현이므로 귀납 단계 ①번에 의해 $(0 + 1)$ 도 정규 표현이다. $(0 + 1)$ 이 정규 표현이므로 귀납 단계 ③번에 의해 $(0 + 1)^*$ 도 정규 표현이다. 또한 0^* 와 $(0 + 1)^*$ 가 정규 표현이므로 귀납 단계 ②번에 의해 $0^*(0 + 1)^*$ 도 정규 표현이다. 이 언어는 ε 을 포함하여 0과 1로 만들 수 있는 모든 문자열의 집합이다.

■ [예제 3-22] 정규표현으로 나타내기

- 다음과 같은 문법에서 `ident`를 정규 표현으로 나타내보자.
- $\langle \text{ident} \rangle ::= (\langle \text{letter} \rangle | _)\{\langle \text{letter} \rangle | \langle \text{digit} \rangle | _ \}$
- $\langle \text{letter} \rangle ::= a | \dots | z$
- $\langle \text{digit} \rangle ::= 0 | 1 | \dots | 9$
- **[풀이]** $(l + _)(l + d + _)^*$
- 단, $l \rightarrow a | \dots | z$
- $d \rightarrow 0 | 1 | \dots | 9$
- `ident`는 첫 자가 영문자 소문자 a, \dots, z , 언더바로 시작하고 두 번째 자부터는 영문자 소문자 a, \dots, z , 숫자 $0, 1, \dots, 9$ 그리고 언더바가 되며 길이는 제한이 없는 문자열이다.

■ [정리 3-1] 정규표현의 대수학적 성질

- 다음과 같은 문법에서 `ident`를 정규 표현으로 나타내보자.
- $\langle \text{ident} \rangle ::= (\langle \text{letter} \rangle | _)\{\langle \text{letter} \rangle | \langle \text{digit} \rangle | _ \}$
- $\langle \text{letter} \rangle ::= a | \dots | z$
- $\langle \text{digit} \rangle ::= 0 | 1 | \dots | 9$
- [풀이] $(l + _)(l + d + _)^*$
- 단, $l \rightarrow a | \dots | z$
- $d \rightarrow 0 | 1 | \dots | 9$
- `ident`는 첫 자가 영문자 소문자 a, \dots, z , 언더바로 시작하고 두 번째 자부터는 영문자 소문자 a, \dots, z , 숫자 $0, 1, \dots, 9$ 그리고 언더바가 되며 길이는 제한이 없는 문자열이다.

■ [정리 3-1] 정규표현의 대수학적 성질

- ① $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$ (+에 대한 결합 법칙)
- ② $(\alpha\beta)\gamma = \alpha(\beta\gamma)$ (접속에 대한 결합 법칙)
- ③ $\alpha + \beta = \beta + \alpha$ (+에 대한 교환 법칙)
- ④ $\alpha + \alpha = \alpha$
- ⑤ $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$ (분배 법칙)
- ⑥ $(\beta + \gamma)\alpha = \beta\alpha + \gamma\alpha$ (분배 법칙)
- ⑦ $\varepsilon\alpha = \alpha = \alpha\varepsilon$ (접속 연산의 항등원)
- ⑧ $\alpha^* = \varepsilon + \alpha\alpha^*$
- ⑨ $(\alpha^*)^* = \alpha^*$

■ 문법 도표

- 쉽게 이해할 수 있도록 문법을 도식화하는 방법
- 문법도표는 사각형과 타원 그리고 이들 사이를 연결한 간선(edge)으로 구성

■ 문법도표를 그리는 방법

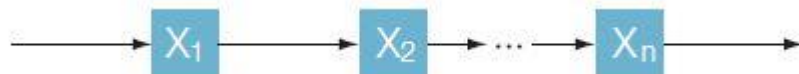
1. 터미널 기호 b 는 원 안에 b 로 표기하고, 다음 기호를 보기 위해 나가는 간선을 그린다.



2. 논터미널 기호 B 는 사각형 안에 B 를 표기하고, 터미널 기호의 표기법과 같이 간선을 그린다



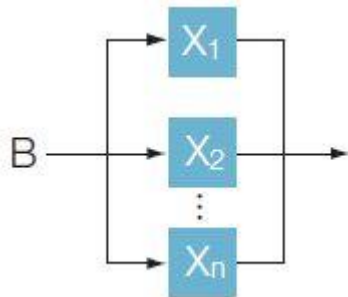
3. 생성 규칙 $A \rightarrow X_1 X_2 \dots X_n$ 은 다음과 같이 문법 도표로 표시한다. X_i 가 논터미널 기호인 경우



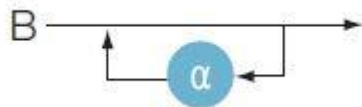
만약 X_i 가 터미널 기호인 경우에는 사각형 대신에 원을 사용

3.3 문법의 표기법

4. 생성 규칙 $A \rightarrow X_1 | X_2 | \dots | X_n$ 은 다음과 같이 문법 도표로 표시
 X_i 가 논터미널 기호인 경우



- 만약 X_i 가 터미널 기호인 경우에는 사각형 대신에 원을 사용
5. 정규표현 $A \rightarrow \alpha^*$ 는 다음과 같은 문법도표로 표현



3.3 문법의 표기법

■ [예 3-23] 문법 도표로 표현하기

- 다음 문법을 문법 도표로 표현하라.

$$G = (V_N, V_T, P, S)$$

$$V_N = \{A, B, C\}$$

$$V_T = \{a, (,), b, c\}$$

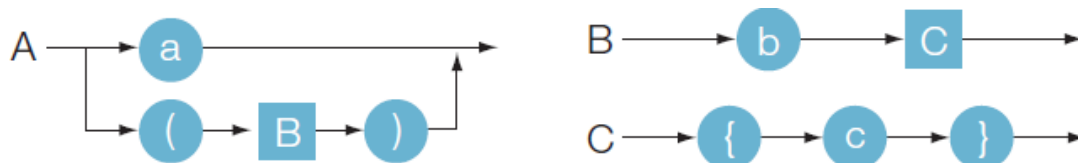
$$P: A \rightarrow a \mid (B)$$

$$B \rightarrow bC$$

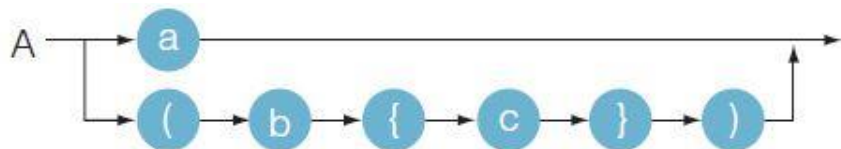
$$C \rightarrow \{c\}$$

$$S = A$$

→ 각 논터미널 기호에 대한 문법 도표를 그리면



이 문법 도표는 시작 기호에 대해 한꺼번에 표현



■ BNF(Backus - Naur Form) 표기법

- 프로그래밍 언어의 형식적 정의(formal definition)을 위해 가장 널리 사용되는 방법
- 이 표기법은 ALGOL 60의 문법을 표현하기 위해 가장 먼저 사용
- 현재에는 대부분의 언어들을 표현하는데 가장 많이 사용
- 이 표기법은 메타 기호(meta-symbol; 메타기호는 표현하려는 언어의 일부분이 아니라, 그 언어를 표현하려고 사용된 특수기호)로서 세 가지 기호를 사용
 - 논터미널 기호는 <와 >로 묶어 표현
 - 대체(replacement)를 나타내기 위해서 ::=를 사용
 - 양자 택일을 나타내기 위해서 | 를 사용

■ [예제 3-24] BNF로 표현하기1

- [예제 3-5]의 문법을 BNF 표기법으로 표현해보자.
- 논터미널 기호인 E, T, F는 각각 $\langle E \rangle$, $\langle T \rangle$, $\langle F \rangle$ 로 나타내고, 대체인 \rightarrow 는 $::=$ 로 표시하면 된다.
- $P : \langle E \rangle ::= \langle E \rangle + \langle T \rangle \mid \langle E \rangle - \langle T \rangle \mid \langle T \rangle$
- $\langle T \rangle ::= \langle T \rangle * \langle F \rangle \mid \langle T \rangle / \langle F \rangle \mid \langle F \rangle$
- $\langle F \rangle ::= (\langle E \rangle) \mid \text{id}$

■ [예제 3-25] BNF로 표현하기2

- 첫 번째 기호가 영문 소문자로 시작하고, 두 번째 기호부터는 영문 소문자나 숫자이며, 길이에 제한이 없는 식별자를 BNF 표기법으로 표현해보자.
- $\langle \text{식별자} \rangle ::= \langle \text{영문} \rangle \mid \langle \text{식별자} \rangle \langle \text{영문} \rangle \mid \langle \text{식별자} \rangle \langle \text{숫자} \rangle$
- $\langle \text{영문} \rangle ::= a \mid b \mid c \mid \dots \mid z$
- $\langle \text{숫자} \rangle ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$

■ [예제 3-25]에서 정의한 식별자의 길이가 길어야 8이라고 하면

- BNF를 이용해서 표현하기에는 다음과 같은 어려움이 따름

$$\begin{aligned} \langle \text{식별자} \rangle ::= & \langle \text{영문자} \rangle \mid \langle \text{영문자} \rangle \langle \text{영문자} \rangle \mid \langle \text{영문자} \rangle \langle \text{숫자} \rangle \\ & \mid \langle \text{영문자} \rangle \langle \text{영문자} \rangle \langle \text{영문자} \rangle \mid \langle \text{영문자} \rangle \langle \text{영문자} \rangle \langle \text{숫자} \rangle \\ & \mid \langle \text{영문자} \rangle \langle \text{숫자} \rangle \langle \text{영문자} \rangle \mid \langle \text{영문자} \rangle \langle \text{숫자} \rangle \langle \text{숫자} \rangle \mid \\ & \dots \\ & \mid \langle \text{영문자} \rangle \langle \text{숫자} \rangle \langle \text{숫자} \rangle \langle \text{숫자} \rangle \langle \text{숫자} \rangle \langle \text{숫자} \rangle \langle \text{숫자} \rangle \langle \text{숫자} \rangle \end{aligned}$$
$$\langle \text{영문자} \rangle ::= a \mid b \mid c \mid \dots \mid z$$
$$\langle \text{숫자} \rangle ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$$

- 위와 같이 나열해야 되는데 중간에 엄청난 가지 수의 생성 규칙이 존재
- 이와 같이 BNF는 반복되는 부분을 표시하는데 어려움을 가짐
- 반복되는 부분을 쉽게 표시하면서 BNF로 표시하는 방법이 EBNF이다.

■ EBNF(Extended BNF) 표기법

- 반복되는 부분을 BNF 표기법보다 읽기 쉽고 간결하게 표현
- 반복되는 부분을 나타내기 위해서 메타 기호로 { }와 < >를 사용
 - {a}는 a가 0번 이상 반복될 수 있다는 것을 의미
 - 정규표현 a^* 와 같은 의미로 생각
 - 또한 선택적인 부분을 표시할 때는 []로 표현
 - [x]는 x가 나타나지 않거나 한 번 나타날 수 있음을 의미
 - [x]는 {x} 과 같은 의미
- 메타 기호를 terminal 기호로 사용하는 경우에는 그 기호를 작은 따옴표(' 와 ')로 묶어 표현
 - {}, [], |, < >, ::= 와 같이 EBNF에서 사용되어지는 메타기호를 터미널 기호로 사용하는 경우 발생하는 혼돈을 피하기 위해서 사용

■ [예제 3-27] EBNF로 표현하기 1

- 첫 번째 기호가 영문 소문자로 시작하고, 두 번째 기호부터는 영문 소문자나 숫자이며, 최대 여덟 자인 식별자를 EBNF 표기법으로 표현해보자.

▪ [풀이]

- $\langle \text{식별자} \rangle ::= \langle \text{영문자} \rangle \{ \langle \text{영숫자} \rangle \}_0^7$
- $\langle \text{영숫자} \rangle ::= \langle \text{영문자} \rangle \mid \langle \text{숫자} \rangle$
- $\langle \text{영문자} \rangle ::= a \mid b \mid c \mid \dots \mid z$
- $\langle \text{숫자} \rangle ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$

■ [예제 3-28] EBNF로 표현하기 2

- if 문장에서 else 부분이 선택적으로 나타날 수 있다면 다음과 같이 표현

→ $\langle \text{if_st} \rangle ::= \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement} \rangle \text{ [else } \langle \text{statement} \rangle \text{]}$

■ [예제 3-29] 메타 기호를 터미널 기호로 사용하기

- 메타 기호 ::=과 |를 터미널 기호로 사용

→ $\langle \text{BNF_rule} \rangle ::= \langle \text{left_part} \rangle ' ::= ' \langle \text{right_part} \rangle$

$\langle \text{right_part} \rangle ::= \langle \text{right_part_element} \rangle \{ | \langle \text{right_part_element} \rangle \}$

■ 오토마타

- 디지털 컴퓨터의 추상적 모델
- 입력 자료를 읽는 기능, 출력 기능, 무한개의 기억 소자(cell)로 이루어진 일시 기억 장치, 유한 개의 내부 상태 중 하나의 상태를 항상 유지하는 제어(control) 장치로 구성

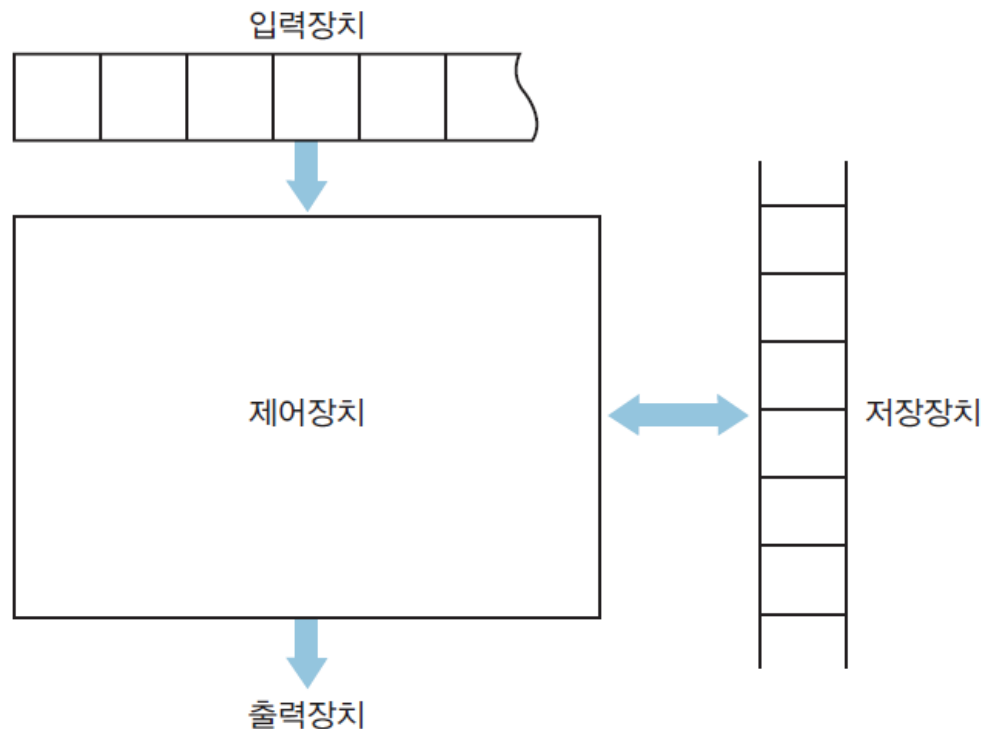


그림 3-2 일반적인 오토마타의 구성

- **기능적인 측면에서 인식기(accepter)와 변환기(transducer)로 구분**
 - 인식기의 경우 입력된 결과에 대해 오토마타는 인식/기각(accept/reject) 등을 표시하는 이진 기호를 출력
 - 언어 변환기는 주어진 입력에 대응하는 새로운 문자열을 출력
 - 변환기에는 상태에 따른 출력을 내는 Moore 기계와 전이에 따른 출력을 내는 Mealy 기계 등이 있다.
- **오토마타는 결정적 오토마타(deterministic automata)와 비결정적 오토마타(non-deterministic automata)로 구분**
 - 결정적 오토마타는 한 상태에서 하나의 입력을 받았을 때 다음 상태가 유일하지만 비결정적 오토마타는 한 상태에서 하나의 입력을 받았을 때 다음 상태가 두 개 이상인 것을 말한다.

■ 유한 오토마타

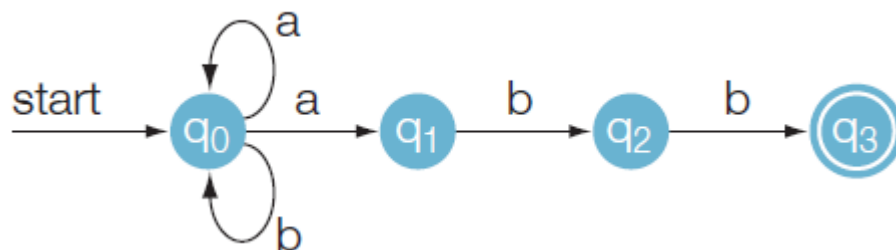
- 어떤 알파벳 Σ 로 부터 만들어지는 문자열의 특별한 것들을 받아들이는 시스템의 수학적 모델로서, 그 시스템에서 변화할 수 있는 상태가 유한개인 것.
- 컴퓨터의 여러 분야에서 널리 사용되고 있는데 특히 플립플롭(flip-flop)을 비롯한 여러 컴퓨터 관련 고안물들, 형식언어의 연구, 그리고 컴파일러 등에 유용하게 쓰임
- 일반적으로 컴파일러 중에서 어휘분석기는 유한 오토마타의 대표적인 것
- **표현 방법**
 - 상태 전이도(state transition diagram)와 같이 비형식적(informal)으로 표현하는 방법 - 상태 전이도는 그래프를 통하여 쉽게 개념이 들어오기 때문에 일반적으로 유한 오토마타를 설명할 때 상태전이도를 많이 사용
 - 형식적(formal)으로 표현하는 방법 - 5가지의 구성 원소들로 표현하는 방법

■ 상태 전이도

- 오토마타의 각 상태(state)를 노드(node)로 표현
- 이동함수 $\delta(q, a) = p$ 에 대해서는 상태 q 에서 p 로 가는 라벨(label)이 a 인 간선(edge)으로 표기
- 최종 상태들은 이중원(double circle)으로 나타내고, 시작 상태는 시작 간선으로 표시하는 유향그래프(directed graph)

■ [예제 3-30] 상태전이도로 표현하기 1

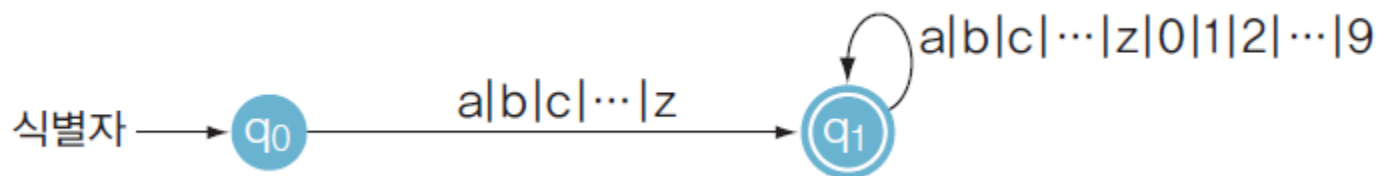
- [ex 3-12]의 정규 표현 $(a + b)^*abb$ 를 인식하는 유한 오토마타를 상태 전이도로 표현해 보자.
- [풀이]



- [예제 3-30]의 유한 오토마타가 문자열 aabb를 인식할 수 있는지 살펴보자. 시작 상태 q0에서 입력 a를 만나면 q0 상태를 유지하고, 다음 문자인 a를 입력받으면 q1 상태로 전이한다. 그리고 q1 상태에서 다음 입력인 b를 받으면 q2 상태로 전이하고, 다시 b를 입력받으면 q3 상태로 전이한다. 입력을 다 읽은 후의 상태가 q3이므로 문자열 aabb는 주어진 유한 오토마타에 의해 인식된다. 하지만 문자열 aabb가 이런 식으로만 되는 것은 아니다.

■ [예제 3-31] 상태전이도로 표현하기 2

- 첫 번째 기호가 영문 소문자로 시작하고, 두 번째 기호부터는 영문 소문자나 숫자이며, 길이에 제한이 없는 식별자를 인식하는 유한 오토마타를 상태 전이도로 표현해보자
- [풀이]



- 시작 상태인 q_0 에서 시작하여 a, b, \dots, z 등의 영문자가 나오면 다음 상태이자 최종 상태인 q_1 로 가서 종료될 수 있으며, 계속해서 영문자 a, b, \dots, z 나 숫자 $1, 2, \dots, 9$ 등이 나오면 다시 반복할 수 있다는 의미이다.

■ [정의 3-18] 유한 오토마타의 형식적 정의

- 유한 오토마타를 형식적으로 표현하면 5-튜플(tuple)로 구성
- 유한 오토마타를 M 이라 하면,

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

단, Q : 상태들의 유한 집합

Σ : 입력 기호들의 유한 집합

q_0 : 시작 상태(start state, initial state)로 $q_0 \in Q$

F : 종결 상태들의 유한 집합 $F \subseteq Q$

δ : 전이함수(transition function)로서 $Q \times \Sigma \rightarrow 2^Q(Q$ 의 멱집합)

즉, $\delta(q, a) = \{P_1, P_2, \dots, P_n\}$ 단, $\{P_1, P_2, \dots, P_n\} \subseteq Q$

■ [예제 3-32] 형식적으로 표현하기 1

- [예제 3-30]의 정규 표현 $(a + b)^*abb$ 를 인식하는 유한 오토마타를 형식적으로 표현
- [풀이] 유한 오토마타를 M 이라 하면,
 - $M = (Q, \Sigma, \delta, q_0, F)$
 - 단, $Q = \{q_0, q_1, q_2, q_3\}$
 - $\Sigma = \{a, b\}$
 - $\delta : \delta(q_0, a) = \{q_0, q_1\}$
 - $\delta(q_0, b) = \{q_0\}$
 - $\delta(q_1, a) = \{q_2\}$
 - $\delta(q_2, b) = \{q_3\}$
 - $q_0 = q_0$
 - $F = \{q_3\}$

■ [예제 3-33] 형식적으로 표현하기 2

- 첫 번째 기호가 영문 소문자로 시작하고, 두 번째 기호부터는 영문 소문자나 숫자이며, 길이에 제한이 없는 식별자를 인식하는 유한 오토마타를 형식적으로 표현
- **[풀이]** 유한 오토마타를 M 이라 하면,
 - $M = (Q, \Sigma, \delta, q_0, F)$
 - 단, $Q = \{q_0, q_1\}$
 - $\Sigma = \{a, b, c, \dots, z, 0, 1, 2, \dots, 9\}$
 - $\delta : \delta(q_0, a) = \{q_1\} \quad \delta(q_0, b) = \{q_1\} \quad \delta(q_0, c) = \{q_1\} \quad \delta(q_0, d) = \{q_1\}$
 - $\delta(q_0, e) = \{q_1\} \quad \delta(q_0, f) = \{q_1\} \quad \delta(q_0, g) = \{q_1\} \quad \delta(q_0, h) = \{q_1\}$
 - $\delta(q_0, i) = \{q_1\} \quad \delta(q_0, j) = \{q_1\} \quad \delta(q_0, k) = \{q_1\} \quad \delta(q_0, l) = \{q_1\}$
 - $\delta(q_0, m) = \{q_1\} \quad \delta(q_0, n) = \{q_1\} \quad \delta(q_0, o) = \{q_1\} \quad \delta(q_0, p) = \{q_1\}$
 - $\delta(q_0, q) = \{q_1\} \quad \delta(q_0, r) = \{q_1\} \quad \delta(q_0, s) = \{q_1\} \quad \delta(q_0, t) = \{q_1\}$
 - $\delta(q_0, u) = \{q_1\} \quad \delta(q_0, v) = \{q_1\} \quad \delta(q_0, w) = \{q_1\} \quad \delta(q_0, x) = \{q_1\}$
 - $\delta(q_0, y) = \{q_1\} \quad \delta(q_0, z) = \{q_1\} \quad \delta(q_1, a) = \{q_1\} \quad \delta(q_1, b) = \{q_1\}$
 - $\delta(q_1, c) = \{q_1\} \quad \delta(q_1, d) = \{q_1\} \quad \delta(q_1, e) = \{q_1\} \quad \delta(q_1, f) = \{q_1\}$
 -

- $\delta(q1, g) = \{q1\}$ $\delta(q1, h) = \{q1\}$ $\delta(q1, i) = \{q1\}$ $\delta(q1, j) = \{q1\}$
 - $\delta(q1, k) = \{q1\}$ $\delta(q1, l) = \{q1\}$ $\delta(q1, m) = \{q1\}$ $\delta(q1, n) = \{q1\}$
 - $\delta(q1, o) = \{q1\}$ $\delta(q1, p) = \{q1\}$ $\delta(q1, q) = \{q1\}$ $\delta(q1, r) = \{q1\}$
 - $\delta(q1, s) = \{q1\}$ $\delta(q1, t) = \{q1\}$ $\delta(q1, u) = \{q1\}$ $\delta(q1, v) = \{q1\}$
 - $\delta(q1, w) = \{q1\}$ $\delta(q1, x) = \{q1\}$ $\delta(q1, y) = \{q1\}$ $\delta(q1, z) = \{q1\}$
 - $\delta(q1, 0) = \{q1\}$ $\delta(q1, 1) = \{q1\}$ $\delta(q1, 2) = \{q1\}$ $\delta(q1, 3) = \{q1\}$
 - $\delta(q1, 4) = \{q1\}$ $\delta(q1, 5) = \{q1\}$ $\delta(q1, 6) = \{q1\}$ $\delta(q1, 7) = \{q1\}$
 - $\delta(q1, 8) = \{q1\}$ $\delta(q1, 9) = \{q1\}$
 - $q0 = q0$
 - $F = \{q1\}$
- [예제 3-33]에서 보듯이 전이 함수는 매우 길기 때문에 이를 간단하게 나타내기 위해 상태 전이표(state transition table)를 도입한다. 상태 전이표는 유한 오토마타의 상태 전이를 행렬(matrix)로 나타낸 것으로, 행렬의 행과 열은 각각 상태 집합과 입력 기호를 나타내고 행과 열이 교차하는 위치에 다음 상태를 기록한다. 만약 전이 함수가 상태와 입력에 해당하는 위치에 값이 없다면 상태 전이표에서 그 위치에 \emptyset 을 넣는다.

3.4 유한 오토마타

- [예제 3-33]의 전이 함수는 다음과 같이 간단하게 상태 전이표로 나타낼 수 있다.

δ	a	b	c ... z	0	1	2 ... 9
q_0	q_1	q_1	$q_1 \dots q_1$	ϕ	ϕ	$\phi \dots \phi$
q_1	q_1	q_1	$q_1 \dots q_1$	q_1	q_1	$q_1 \dots q_1$

- 전이 함수는 유한 오토마타의 상태 전이를 행렬(matrix)로 표시한 상태 전이표(state transition table)로 표시되어진다. 이 상태 전이 함수의 형태에 따라 결정적 유한 오토마타(Deterministic Finite Automata; DFA)와 비결정적 유한 오토마타(Nondeterministic Finite Automata; NFA)를 구분한다.

■ [정의 3-19] 결정적 유한 오토마타(DFA)

- DFA는 다음의 두 가지 조건을 만족해야 한다.
- 첫째, ϵ 에 의한 상태전이(state transition)가 없고
- 둘째, $\delta(q, a) = \{P_1, P_2, \dots, P_n\}$ 에서 $n = 1$ 이다.
- 즉, $\delta : Q \times \Sigma \rightarrow Q$
- 다시 말하면, 임의의 상태에서 하나의 입력 기호에 대해서 다음 상태는 오직 하나이거나 상태 전이가 없어야 한다.

■ ϵ 에 의한 상태 전이를 ϵ -전이라고도 한다.

■ [예제 3-34] DFA 확인하기 1

- [예제 3-33]의 유한 오토마타가 DFA인지 알아보자.
- **[풀이]** [예제 3-33]의 유한 오토마타는 ϵ 에 의한 상태 전이가 없고, 임의의 상태에서 하나의 입력에 대해 다음 상태가 단 하나이므로 DFA이다.

■ [예제 3-36] DFA 확인하고 상태전이도로 표현하기

- 0과 1이 짝수 개인 문자열을 받아들이는 다음과 같은 유한 오토마타가 DFA인지 알아보고 상태 전이도로 표현해보자

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

▪ 단, $Q = \{q_0, q_1, q_2, q_3\}$

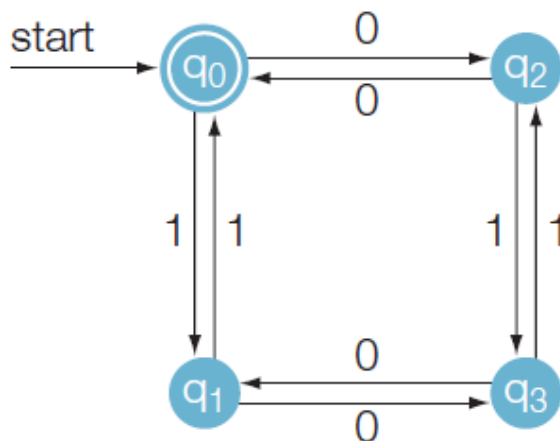
▪ $\Sigma = \{0, 1\}$

▪ $q_0 = q_0$

▪ $F = \{q_0\}$

δ	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

- [풀이] ϵ 에 의한 상태 전이가 없고, 임의의 상태에서 하나의 입력에 대해 다음 상태가 단 하나이므로 DFA이다. 이를 상태 전이도로 표현하면 다음과 같다.



■ [예제 3-37] DFA로 부터 문장 인식하기

- [예제 3-33]의 식별자를 인식하는 유한 오토마타는 DFA인데 이 DFA로부터 문장 a12를 인식해보자.
- [풀이]
 - 시작 상태가 q_0 이므로 다음과 같은 과정을 거친다.
 - $\delta(q_0, a) = q_1$
 - $\delta(q_1, 1) = q_1$
 - $\delta(q_1, 2) = q_1$
 - 모든 입력을 읽은 후 마지막에 도달한 상태가 q_1 이며, 최종 상태가 q_1 이므로 a12는 주어진 DFA에 의해 인식된다.

■ [예제 3-39] DFA로 부터 문장 인식하기 3

- [예제 3-36]의 DFA로부터 0101을 인식하는 과정을 알아보자.
- [풀이]
 - 시작 상태가 q_0 이므로 다음과 같은 과정을 거친다.
 - $\delta(q_0, 0) = q_2$
 - $\delta(q_2, 1) = q_3$
 - $\delta(q_3, 0) = q_1$
 - $\delta(q_1, 1) = q_0$
 - 모든 입력을 읽은 후 마지막에 도달한 상태가 q_0 이며, 최종 상태가 q_0 이므로 0101은 주어진 DFA에 의해 인식된다.

- 앞의 과정에서 주어진 DFA에 의해서 인식되는 문자열이 무엇인지를 알기가 어렵다. 이를 위하여 전이 함수를 다음과 같이 확장해보자.

$$Q \times \Sigma \rightarrow Q \Rightarrow Q \times \Sigma^* \rightarrow Q$$

확장된 전이 함수는 한 개의 기호를 문자열로 확장하는 것이다.

즉,

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

$$\delta(q_0, wa) = \delta(\delta(q_0, w), a)$$

와 같이 확장해야 한다.

여기서, $w \in \Sigma^*$ 이고 $a \in \Sigma$ 이다. 즉, q_0 상태에서 문자열 wa 를 본다는 것은 w 를 전부 본 상태에서 a 를 보는 것과 같다는 것을 의미한다.

■ [예제 3-40] DFA로 부터 문장 인식하기 4

- [예제 3-37]에서 확장된 함수에 의해 문장 a12를 인식하는 과정을 살펴보자.
- [풀이] $\delta^*(q_0, a12) = \delta(\delta^*(q_0, a1), 2)$
- $= \delta(\delta(\delta(q_0, a), 1), 2)$
- $= \delta(\delta(q_1, 1), 2)$
- $= \delta(q_1, 2)$
- $= q_1 \in F$
- 그러므로 a12는 인식된다. 이는 다음과 같은 방법으로도 인식할 수 있다.
- $\delta^*(q_0, a12) = \delta^*(q_1, 12)$
- $= \delta(q_1, 2)$
- $= q_1 \in F$

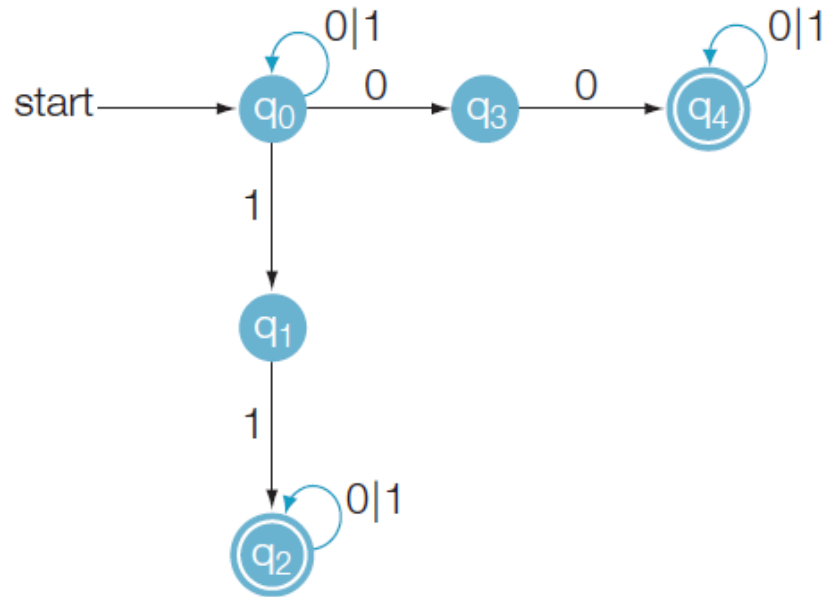
■ [정의 3-20] 비결정적 유한 오토마타(NFA)

- 첫째, ϵ 에 의한 상태 전이가 있거나
- 둘째, $\delta(q, a) = \{P_1, P_2, \dots, P_n\}$ 에서 $n \geq 2$ 인 n 이 하나라도 존재한다.
- 즉, $\delta : Q \times \Sigma \rightarrow 2^Q$
- 다시 말하면, 어떤 상태에서 하나의 입력기호에 대해서 다음 상태가 두 개 이상인 것이 하나라도 존재한다.

■ [예제 3-43] NFA인지 확인하고 상태전이도로 표현하기

- 2개의 연속적인 0이 있거나, 2개의 연속적인 1이 있는 문자열을 받아들이기 위한 유한 오토마타는 다음과 같다. NFA인지 알아보고 형식적으로 표현해보자

3.4 유한 오토마타



▪ [풀이]

- 전이 함수 중 $\delta(q_0, 0) = \{q_0, q_3\}$ 이므로 NFA이다. 이를 형식적으로 표현하면 다음과 같다.
- $M = (Q, \Sigma, \delta, q_0, F)$
- 단, $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$

3.4 유한 오토마타

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	$\{q_4\}$	$\{q_4\}$

- $q_0 = q_0$
- $F = \{q_2, q_4\}$

■ [예제 3-44] NFA부터 문장 인식하기

- [예제 3-32]의 유한 오토마타가 문장 baabb를 인식하는지 알아보자.

- [풀이]

- 인식하는 과정을 두 가지 방법으로 풀어볼 수 있다. 첫 번째는 하나하나 전이 함수를 적용하여 인식하는 방법이고, 두 번째는 갈 수 있는 상태를 모두 그림으로 표현하여 인식하는 방법이다.

- ① $\delta(q_0, b) = \{q_0\}$ 이 된다. ...(1)

- $\delta(q_0, a) = \{q_0, q_1\}$ 이 된다. ...(2)

- 상태가 2개이므로 이 중에서 현재 상태를 q_0 으로 선택하자.

- $\delta(q_0, a) = \{q_0, q_1\}$ 이 된다. ...(3)

- 상태가 2개이므로 이 중에서 현재 상태를 q_0 으로 선택하자.

- $\delta(q_0, b) = \{q_0\}$ 이 된다. ...(4)

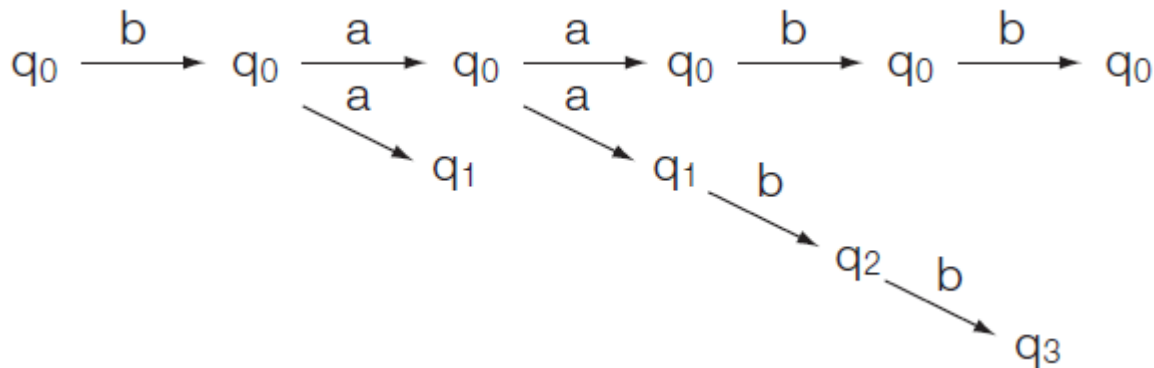
- 다시 현재 상태 q_0 에서 입력 b 를 읽으면

- $\delta(q_0, b) = \{q_0\}$ 이 된다. ...(5)

- 입력 문자열을 다 읽은 후 최종 상태는 $\{q_0\}$ 이 되었지만 최종 상태가 $\{q_3\}$ 이므로 주어진 문자열은 인식되지 않는다고 해야 한다. 그러나 NFA에서는 이런 결론에 문제가 있다. 아직도 적용하지 않은 상태가 많이 남아 있기 때문이다. 즉 (2)에서의 q_1 상태, (3)에서의 q_1 상태 등이다. 이 모든 상태를 적용해야 하는데 그렇게 하려면 어떤 순서로 해야 하는지도 고려해야 할 것이다. 여기서는 (2)에서 q_1 상태부터 적용하겠다.

3.4 유한 오토마타

- 이것 또한 최종 상태에 도달하지 못한다. 이번에는 (3)에서 q_1 상태부터 적용하겠다.
- $\delta(q_1, b) = \{q_2\}$ 가 된다.
- $\delta(q_2, b) = \{q_3\}$ 이 된다.
- 모든 입력을 읽은 후 마지막에 도달한 상태가 $\{q_3\}$ 이고 최종 상태가 $\{q_3\}$ 이므로 문장 baabb는 주어진 NFA에 의해 인식된다.
- ② 이번에는 다음과 같이 그림으로 나타내보자.



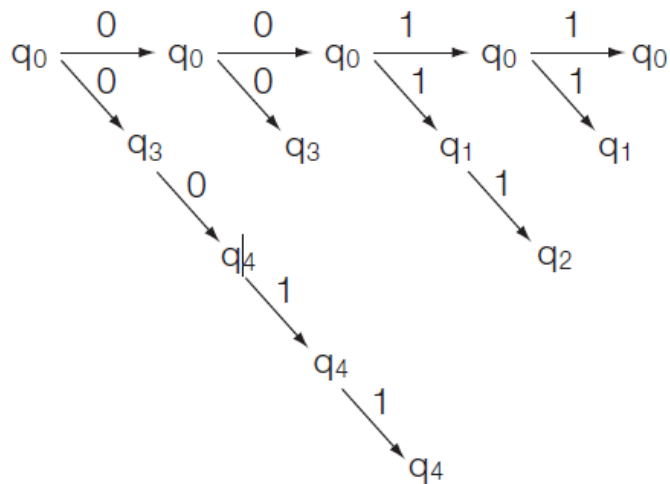
- 이 경우는 시작 상태에서 출발하여 도달 가능한 모든 상태를 나타내는 것이다. 이 중에서 입력 문자열 baabb를 모두 읽은 후 도달 가능한 상태는 $\{q_0, q_3\}$ 이다.
- $\therefore \{q_0, q_3\} \cap F = \{q_3\}$
- \therefore 문장 baabb는 주어진 NFA에 의해 인식된다.

■ [예제 3-45] NFA로 부터 문장 인식하기

- [예제 3-43]의 NFA로부터 문장 0011을 인식하는지 알아보자.
- [풀이]
- [예제 3-44]와 같이 두 가지 방법으로 풀어보자.
- ① $\delta(q_0, 0) = \{q_0, q_3\}$ 이 된다. ...(1)
- 상태가 2개이므로 이 중에서 현재 상태를 q_0 으로 선택하자.
- $\delta(q_0, 0) = \{q_0, q_3\}$ 이 된다. ...(2)
- 상태가 2개이므로 이 중에서 현재 상태를 q_0 으로 선택하자.
- $\delta(q_0, 1) = \{q_0, q_1\}$ 이 된다. ...(3)
- 상태가 2개이므로 이 중에서 현재 상태를 q_0 으로 선택하자.
- $\delta(q_0, 1) = \{q_0, q_1\}$ 이 된다. ...(4)
- 입력 문자열을 다 읽은 후 최종 상태는 $\{q_0, q_1\}$ 이 되었다. 그러나 최종 상태는 $\{q_2, q_4\}$ 이므로 주어진 문자열은 인식되지 않는다.

3.4 유한 오토마타

- 이번에는 적용하지 않은 상태 중 (1)에서 q_3 상태부터 적용하겠다.
- $\delta(q_3, 0) = \{q_4\}$ 가 된다.
- $\delta(q_4, 1) = \{q_4\}$ 가 된다.
- $\delta(q_4, 1) = \{q_4\}$ 가 된다.
- 최종 상태는 $\{q_2, q_4\}$ 이므로 문장 0011은 주어진 NFA에 의해 인식된다.
- ② 이번에는 다음과 같이 그림으로 나타내보자.



- 시작 상태에서 출발하여 입력 문자열 0011을 모두 읽은 후 도달 가능한 상태는 $\{q_0, q_1, q_2, q_4\}$ 이다.
- $\therefore \{q_0, q_1, q_2, q_4\} \cap F = \{q_2, q_4\}$
- \therefore 문장 0011은 주어진 NFA에 의해 인식된다.

- [예제 3-44], [예제 3-45]에서 알 수 있듯이 이런 형태로 전이 함수를 적용하면 NFA에 의해 인식되는 문자열이 무엇인지 명확하게 알지 못한다. 이를 알기 위해 다음과 같이 전이 함수를 하나의 입력 기호에서 입력 문자열로 확장해보자.

$$\rightarrow \delta : Q \times \Sigma \rightarrow 2^Q \Rightarrow Q \times \Sigma^* \rightarrow 2^Q$$

$$\text{단, } \delta^*(q, \epsilon) = \{q\}$$

$$\begin{aligned} \delta^*(q, wa) &= \{\gamma \mid \delta^*(q, w) \text{에 의해서 도달 할 수 있는 모든 상태 } p \text{에 대해서 } \gamma \in \delta(p, a)\} \\ &= \bigcup_{p \in \delta(q, w)} \delta^*(p, a) \end{aligned}$$

여기서 $w \in \Sigma^*$ 이고 $a \in \Sigma$ 이다. 이것은 q 상태에서 wa 를 본다는 것은 w 를 다 본 상태들에서 a 를 본 상태 집합의 합집합과 같다는 것을 의미

■ [예제 3-46] NFA로 부터 문장 인식하기 3

- [예제 3-44]에서 문장 baabb에 대해 확장된 전이 함수에 적용해보자.

▪ [풀이]

- $\delta(q_0, baabb) = \delta(q_0, aabb)$
- $= \delta(q_0, abb) \cup \delta(q_1, abb)$
- $= \{\delta(q_0, bb) \cup \delta(q_1, bb)\} \cup \emptyset$
- $= \delta(q_0, b) \cup \delta(q_2, b)$
- $= \{q_0\} \cup \{q_3\}$
- $= \{q_0, q_3\}$
- $\therefore \{q_0, q_3\} \cap F = \{q_3\}$
- \therefore 문장 baabb는 주어진 NFA에 의해 인식된다.

■ NFA와 DFA

- DFA보다 NFA가 언어의 구조를 쉽게 표현
- NFA는 DFA보다 프로그램으로 구현하기 어려움
- DFA는 NFA보다 프로그램으로 구현했을 경우에 효율면에서 훨씬 우수
- 일반적인 구현은 DFA로 해야 되는데 NFA가 언어의 구조를 쉽게 표현할 수 있기 때문에 서로가 변환이 필요

■ DFA와 NFA는 서로가 동등(equivalent)

- 증명은 생략

■ NFA를 DFA로 변환

1. ϵ -전이가 있는 NFA를 DFA로 변환
2. ϵ -전이가 없는 NFA를 DFA로 변환

■ ϵ -전이가 있는 NFA를 DFA로 변환

- 이 변환은 ϵ -closure를 이용하여 변환
- NFA에서 의미가 같은 여러 상태들이 DFA에서 하나의 상태로 변환된다는 것

■ [정의 3-21] ϵ -closure

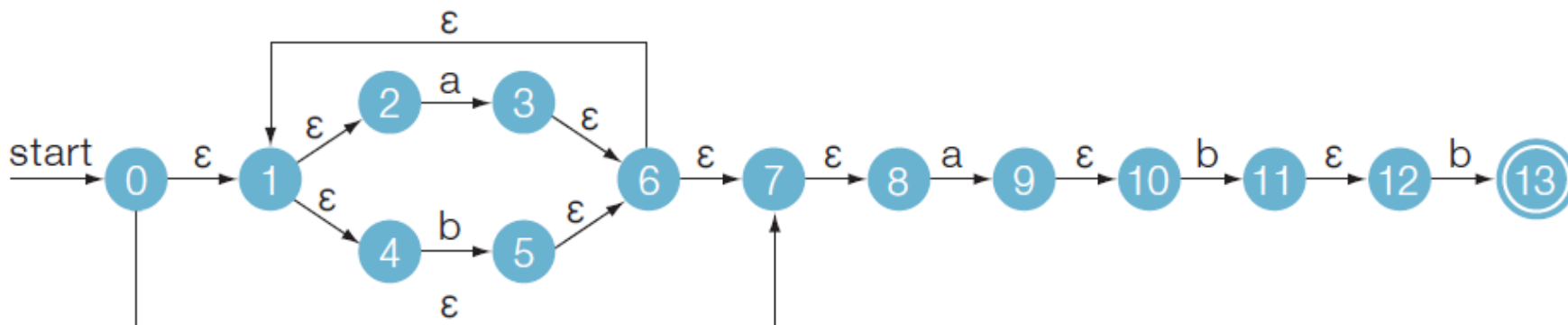
1. S 가 한 개의 상태일 경우
 - ϵ -closure(S)는 NFA의 상태 S 와 상태 S 로부터 레이블이 ϵ 인 간선으로 도달될 수 있는 NFA의 모든 상태들의 집합
 - 이 방법은 ϵ -closure(S)의 원소가 변하지 않을 때까지 반복
2. S 가 하나 이상의 상태 집합으로 되어 있는 경우
 - ϵ -closure(S)는 NFA의 상태 S 안에 있는 모든 상태에 대해서 1.과 같은 방법으로 집합들을 구하여 합집합한 것

$$\epsilon\text{-closure}(S) = \bigcup_{x \in S} \epsilon\text{-closure}(x)$$

단, x 는 S 안에 포함된 상태이다.

■ [예제 3-48] ϵ -closure 계산하기

- [예제 3-30]의 정규 표현 $(a + b)^*abb$ 를 인식하는 NFA를 상태 전이도로 나타내면 다음과 같다. 이 상태 전이도로부터 ϵ -closure를 구해보자. [예제 3-30]의 상태 전이도와 다른 이유와 정규 표현으로부터 NFA를 만드는 방법은 뒤에서 설명할 것이다.



■ [알고리즘 3-1] ε 전이가 있는 NFA를 DFA로 변환하는 방법

[입력] ε -전이가 있는 NFA

[출력] ε -전이가 있는 NFA와 동등한 DFA

[방법]

- 1) ε -NFA의 시작 상태 q_0 에 대해 ε -closure(q_0)를 구하고 ε -closure(q_0)를 DFA의 시작 상태로 놓는다.
- 2) DFA의 시작 상태를 집합 D_S 에 넣는다.
- 3) 집합 D_S 에 있는 상태에 대해서 ε -NFA에 있는 ε 를 제외한 각각의 입력 기호에 대해 도달 할 수 있는 상태 집합을 각각 T_1, T_2, \dots 라 한다. 그런 다음 ε -closure(T_1), ε -closure(T_2), \dots 를 구하여, 이를 DFA의 새로운 상태로 만들어 집합 D_S 에 추가한다. 만약 새로운 상태들이 이미 만들어진 상태와 같은 집합이면, DFA의 새로운 상태로 만들지 않고 현재 상태에서부터 이전에 만들어진 상태로 입력문자에 따른 간선만 만든다. 지금 적용한 상태를 집합 D_S 에서 제거한다.
- 4) 집합 D_S 가 공집합이 될 때까지 과정 3)을 되풀이한다.
- 5) 만들어진 상태 중에서 ε -NFA의 최종 상태를 포함하는 상태들은 모두 DFA의 최종 상태가 된다.

3.4 유한 오토마타

- 변환에 앞서 각 상태에 대해 ϵ -closure를 구한다.
- ϵ -closure(0) = {0, 1, 2, 4, 7, 8}
- ϵ -closure(1) = {1, 2, 4}
- ϵ -closure(3) = {1, 2, 3, 4, 6, 7, 8}
- ϵ -closure(5) = {1, 2, 4, 5, 6, 7, 8}
- ϵ -closure(9) = {9, 10}
- ϵ -closure(11) = {11, 12}
- ϵ -closure(0, 9) = ϵ -closure(0) \cup ϵ -closure(9)
- $\quad\quad\quad = \{0, 1, 2, 4, 7, 8, 9, 10\}$
- ϵ -closure(5, 11) = ϵ -closure(5) \cup ϵ -closure(11)
- $\quad\quad\quad = \{1, 2, 4, 5, 6, 7, 8, 11, 12\}$

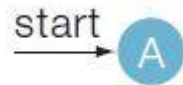
■ [예제 3-49] ϵ 전이가 있는 NFA를 DFA로 변환하기

- [예제 3-48]에 [알고리즘 3-1]을 적용하여 ϵ -전이가 있는 NFA를 DFA로 변환

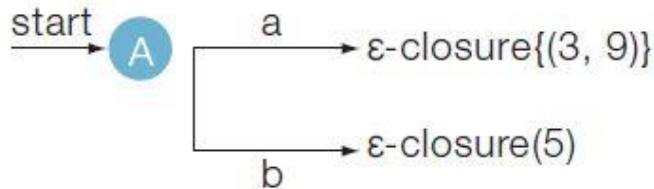
▪ [풀이]

- 1) ϵ -NFA의 시작 상태가 0이므로 ϵ -closure(0)가 DFA의 시작 상태가 된다.

ϵ -closure(0) = {0, 1, 2, 4, 7, 8} = A 라 하고, $D_S = \{A\}$ 가 되고 DFA의 시작 상태는 다음과 같다.



- 2) ϵ -NFA에 있는 ϵ 를 제외한 입력 기호를 보면 a와 b이므로 D_S 에서 A를 꺼내서 각각의 입력 기호 a, b에 의해 도달할 수 있는 상태들을 구한다.



각각의 상태에 대해서 ϵ -closure를 구한 후, 새로운 상태라면 이름을 붙이면 된다.

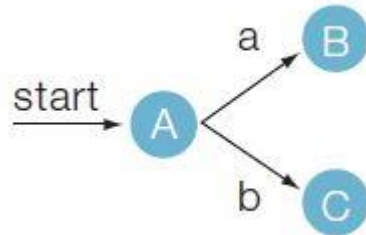
$$\epsilon\text{-closure}(3, 9) = \{3, 6, 1, 2, 4, 7, 8, 9, 10\} = B$$

$$\epsilon\text{-closure}(5) = \{5, 6, 7, 8, 1, 2, 4\} = C$$

$$D_S = \{A, B, C\}$$

DFA의 상태 전이도는 다음과 같다.

3.4 유한 오토마타



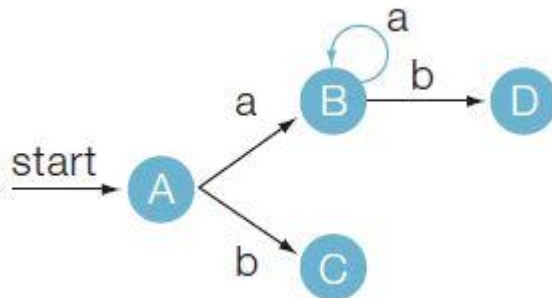
3) $D_S = \{B, C\}$ 가 공집합이 아니므로 알고리즘을 되풀이한다.

우선 B 상태에서 입력 기호 a, b 를 보고 갈수 있는 상태 집합을 구하면 $\{3, 9\}, \{5, 11\}$ 이 되므로 ϵ -closure $\{(3, 9)\}, \epsilon$ -closure $\{(5, 11)\}$ 를 구한다.

$$\epsilon\text{-closure}\{(3, 9)\} = \{3, 6, 1, 2, 4, 7, 8, 9, 10\} = B$$

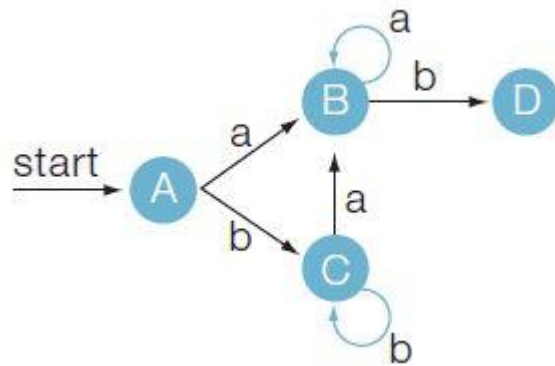
$$\epsilon\text{-closure}\{(5, 11)\} = \{5, 6, 7, 8, 1, 2, 4, 11, 12\} = D$$

여기에서 ϵ -closure $\{(3, 9)\}$ 는 이미 만들어진 상태 B 와 같으므로 새로운 상태를 만들지 않고 입력 기호에 따른 간선만 그린다.



$D_S = \{C, D\}$ 이므로, 같은 방법으로 C 상태에서 입력기호 a, b 를 보고 갈수 있는 상태 집합을 구하면 $\{3, 9\}, \{5\}$ 가 되므로 ϵ -closure $\{(3, 9)\}$ 는 B 상태이고 ϵ -closure $\{(5)\}$ 는 C 상태이다. 그러므로 DFA의 상태 전이도는 다음과 같다.

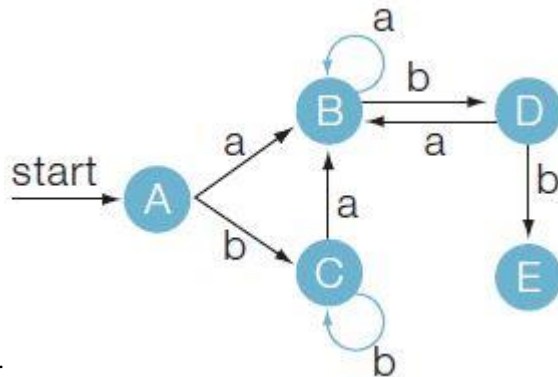
3.4 유한 오토마타



$D_S = \{D\}$ 이다.

- 4) D_S 가 공집합이 아니므로 D 상태에서 입력기호 a, b 를 보고 갈수 있는 상태 집합을 구하면 $\{3, 9\}, \{5, 13\}$ 이 되므로, ϵ -closure($\{3, 9\}$)는 B 상태이고

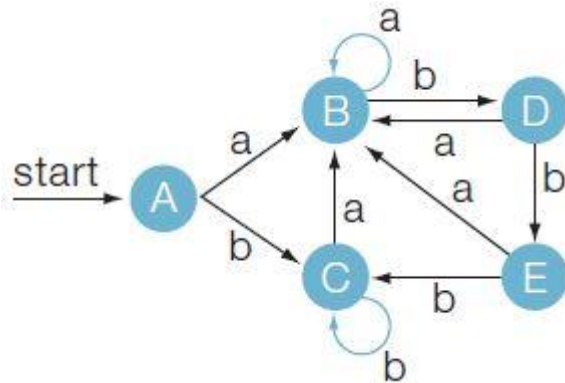
$$\epsilon\text{-closure}(\{5, 13\}) = \{5, 6, 7, 8, 1, 2, 4, 13\} = E$$



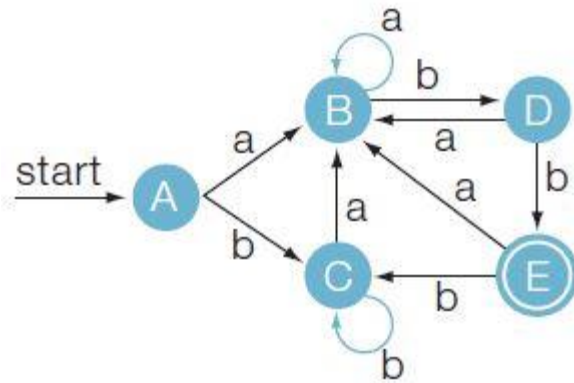
$D_S = \{E\}$ 이다.

- 5) D_S 가 E 상태에 대해서도 같은 방법으로 구하면 입력기호 a 에 대해 $\{3, 9\}$, 입력기호 b 에 대해 $\{5\}$ 를 얻으므로 ϵ -closure($\{3, 9\}$)는 B 상태이고 ϵ -closure($\{5\}$) = C 상태가 된다. 즉,

3.4 유한 오토마타



D_5 가 공집합이므로 ϵ -NFA의 최종 상태 13을 포함하는 DFA의 상태를 구해 보면 E 이므로 E 는 DFA의 최종 상태가 된다. 즉,



이와 같은 일련의 과정을 상태 전이표로 나타내면 다음과 같다.

3.4 유한 오토마타

δ		입력 기호	
		a	b
상태	A	B	C
	B	B	D
	C	B	C
	D	B	E
	E	B	C

여기서 A 가 시작 상태이고 E 가 최종 상태이다

■ [정리3-3] ε 전이가 없는 NFA를 DFA로 변환하는 방법

- NFA에 의해서 인식되는 언어를 L 이라 하면, L 을 인식하는 DFA가 존재한다.
- L 을 인식하는 NFA $M = (Q, \Sigma, \delta, q_0, F)$ 라 하면,

DFA $M' = (Q', \Sigma', \delta', q_0', F)$ 는 다음과 같이 구성

$Q' = 2^Q$ 즉, Q' 의 한 상태는 $[q_1, q_2, \dots, q_j]$ 의 형태로 표시한다.

단, $q_j \in Q$ (단, $j = 1, 2, \dots, j$)이다.

$F = \{q \in Q' \mid q \text{는 } M \text{의 최종 상태를 포함하는 } Q \text{안에 있는 모든 상태들의 집합}\}$

$q_0' = [q_0]$

$\delta(\{q_1, q_2, \dots, q_j\}, a) = \{P_1, P_2, \dots, P_j\}$ 라 하면

$\delta'([q_1, q_2, \dots, q_j], a) = [P_1, P_2, \dots, P_j]$ 이다.

- 입력 문자열의 길이에 대하여 수학적 귀납법으로 증명

■ [예제 3-51] NFA를 DFA로 변환하기 3

- [정리 3-3]을 이용하여 ϵ -전이가 없는 NFA를 DFA로 변환하시오.

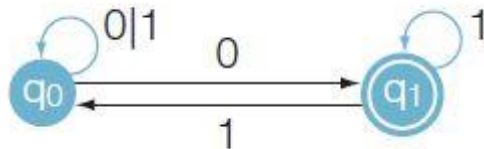
NFA $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$

단,

δ	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	\emptyset	$\{q_0, q_1\}$

이다.

→ 이를 상태 전이도로 나타내면,



3.4 유한 오토마타

- 이때 NFA를 DFA로 변환해보자. DFA $M' = (Q', \Sigma, \delta', q_0', F)$ 라 하면 [정리 3-3]에 의해 다음과 같이 된다.

$M' = (Q', \Sigma, \delta', q_0', F)$ 라 하면,

$$Q' = \{[q_0], [q_1], [q_0, q_1]\}$$

$$q_0' = [q_0]$$

$$F = \{[q_1], [q_0, q_1]\}$$

$$\delta': \delta'([q_0], 0) = \delta(\{q_0\}, 0) = \{q_0, q_1\} = [q_0, q_1]$$

$$\delta'([q_0], 1) = \delta(\{q_0\}, 1) = \{q_0\} = [q_0]$$

$$\delta'([q_1], 0) = \delta(\{q_1\}, 0) = f$$

$$\delta'([q_1], 1) = \delta(\{q_1\}, 1) = \{q_0, q_1\} = [q_0, q_1]$$

$$\delta'([q_0, q_1], 0) = \delta(\{q_0, q_1\}, 0) = \{q_0, q_1\} = [q_0, q_1]$$

$$\delta'([q_0, q_1], 1) = \delta(\{q_0, q_1\}, 1) = \{q_0, q_1\} = [q_0, q_1]$$

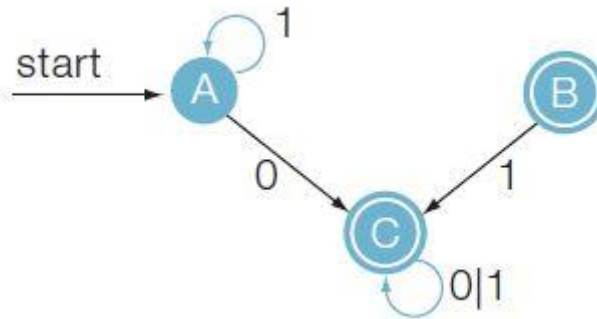
이 된다.

위의 상태 전이 함수를 상태전이표로 표시하면 다음과 같다.

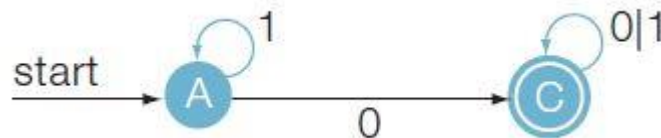
3.4 유한 오토마타

δ'	0	1
$[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_1]$	ϕ	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

$[q_0]$, $[q_1]$, $[q_0, q_1]$ 을 각각 A, B, C로 상태이름을 바꾸고 상태 전이도를 그리면 다음과 같다.



위의 상태전이도는 DFA로 바뀌어져 있지만, B상태는 시작 상태에서 **도달 불가능한 상태 (inaccessible state)**이므로, 문장을 인식하는데 불필요한 상태가 된다. 그러므로 B상태는 제거할 수 있으므로 다음과 같은 상태전이도를 얻을 수 있다.



3.4 유한 오토마타

- 지금까지 두 가지 방법으로 NFA를 DFA로 변환하는 방법을 알아보았다. 이제 새로운 시도를 해보자. ϵ -전이가 없는 NFA를 ϵ -closure를 이용하여 DFA로 변환하는 것이다
- [예제 3-52] NFA를 DFA로 변환하기 4
 - [예제 3-51]에 주어진 ϵ -전이가 없는 NFA를 DFA로 변환하되 [정리 3-3]을 이용하지 않고, ϵ -전이가 있는 NFA를 DFA로 변환하는 [알고리즘 3-1]을 적용해보자.

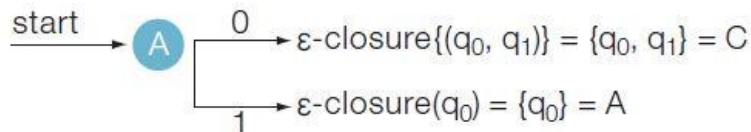
■ [풀이]

1) ϵ -NFA의 시작 상태가 q_0 이므로 ϵ -closure(q_0)가 DFA의 시작 상태가 된다.

ϵ -closure(q_0) = $\{q_0\} = A$ 라 하고, $D_S = \{A\}$ 가 된다.

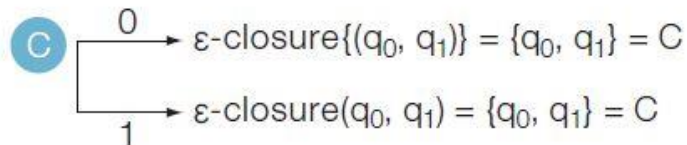
2) ϵ -NFA에 있는 ϵ 를 제외한 입력 기호를 보면 0와 1이므로 D_S 에서 A 를 꺼내서 각각의 입력 기호 0, 1에 의해 도달할 수 있는 상태들을 구한다.

각각의 상태에 대해서 ϵ -closure를 구한 후, 새로운 상태라면 이름을 붙이면 된다.



A상태에 대해서 적용했으므로 $D_S = \{C\}$ 가 된다.

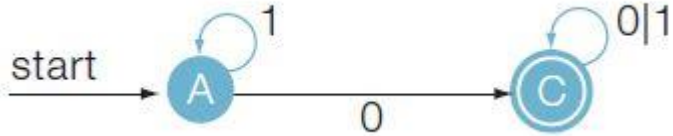
3) $D_S = \{C\}$ 가 공집합이 아니므로 알고리즘을 되풀이한다. C상태에서 입력 기호 0,1를 보고 갈수 있는 상태 집합을 구하면 된다.



3.4 유한 오토마타

- 4) D_s 가 공집합이므로 ϵ -NFA의 최종 상태 $\{q_1\}$ 을 포함하는 DFA의 상태를 구해보면 C 이므로 C 는 DFA의 최종 상태가 된다.

일련의 과정을 상태 전이도로 나타내면 ϵ -전이가 없는 NFA를 DFA로 변환하는 방법의 결과와 같아 짐을 알 수 있다.



■ DFA의 상태수 최소화(state minimization)

- DFA를 이용하는 어휘분석기의 상태 전이표의 크기를 줄임
 - 기억공간을 적게 차지하도록 하고 또한 어휘분석 프로그램을 간단히 하는데 큰 도움
- 상태수를 최소화하는 방법
 - 동치관계(equivalence relation)을 이용 상태수를 합침(state merge)

■ [정의 3.22] 구별가능(distinguishable)

- 문자열 $w \in \Sigma^*$ 대해, 만약 q_1 상태에서 w 를 모두 본 상태가 q_3 이고 q_2 에서 w 를 모두 본 상태가 q_4 일 때 q_3, q_4 중 하나만 종결 상태에 속하면 2개의 상태 q_1 과 q_2 는 구분 가능하다고 한다.

■ [정의 3.23] 구별불가능(Indistinguishable)

- 문자열 $w \in \Sigma^*$ 에 대해, 만약 q_1 상태에서 w 를 모두 본 상태가 q_3 이고 q_2 에서 w 를 모두 본 상태가 q_4 일 때 q_3, q_4 가 모두 종결 상태에 속하거나 모두 종결 상태가 아니라면 2개의 상태 q_1 과 q_2 는 구분 불가능indistinguishable하다고 한다

■ [알고리즘 3-2] DFA의 상태수 최소화(state minimization)

[입력] 하나의 DFA M

[출력] M 과 동일한 언어를 수용하고 가능한 한 적은 상태수를 갖는 하나의 DFA M'

[방법]

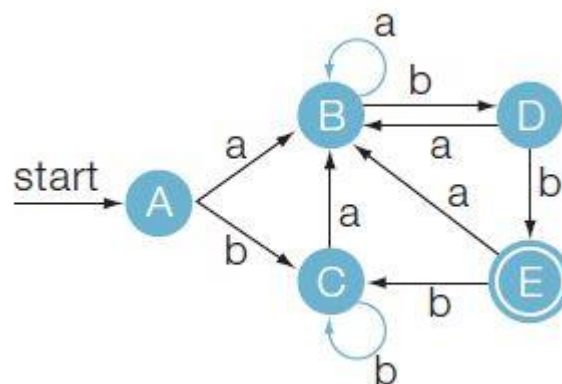
- 1) 시작상태로부터 도달 불가능한 상태를 모두 제거
 - 2) 초기의 동치관계인 최종 상태(final state)와 최종상태가 아닌 것(non-final state)의 두 동치류(equivalence class)로 분할(partition)
 - 3) 하나의 동치류 안에서 같은 입력기호에 대해 서로 다른 동치류로 가는 간선이 존재하면 또 다른 분할을 하여 새로운 동치류를 만듦
 - 4) 3)의 과정을 반복하여 더 이상 새로운 분할이 일어나지 않을 때까지 반복
 - 5) M 의 최종 상태에 속하는 상태가 동치류속에 있으면 이 동치류는 M' 의 최종 상태
- 최소화된 DFA $M' = (Q', \Sigma', \delta', q_0', F')$ 는 다음과 같은 조건을 만족
 - (1) Q' : 동치류의 집합으로서 Q 의 한 상태를 $[q]$ 로 표시
의미는 상태 q 를 포함하는 동치류
 - (2) $[P], [q]$ 를 임의의 동치류라 하고 $\delta(P, a) = q$ 이면 $\delta'([P], a) = [q]$ 이다.
 - (3) $q_0' = [q_0]$
 - (4) $F' = \{[q] \mid q \in F\}$

3.4 유한 오토마타

■ [예제 3-54] DFA의 상태수 최소화하기 1

- [예제 3-49]에서 만들어진 다음의 DFA를 최소화 해보자.

δ	입력 기호		
	a	b	
상태	A	B	C
	B	B	D
	C	B	C
	D	B	E
	E	B	C



▪ [풀이]

1. 도달 불가능한 상태를 제거, 그러나 도달 불가능한 상태는 존재하지 않음
2. 최종 상태와 최종 상태가 아닌 상태의 두 동치류 $\{E\}$ $\{A, B, C, D\}$ 로 만들
3. 각 동치류가 각 입력기호에 대해 구별되는가를 조사하여 더 이상 분할이 일어나지 않을 때까지 반복한다.

3.4 유한 오토마타

입력	동치류 이름	1				2
	동치류	A	B	C	D	E
a		1	1	1	1	1
b		1	1	1	2	1

- 1번 동치류에서 입력기호 b에 대해서 {A, B, C}와 {D}가 구별되므로 분할한다.

입력	동치류 이름	1			2	3
	동치류	A	B	C	D	E
a		1	1	1	1	1
b		1	2	1	3	1

- 1번 동치류에서 입력기호 b에 대해서 {A, C}와 {B}가 구별되므로 분할한다

입력	동치류 이름	1		2	3	4
	동치류	A	C	B	D	E
a		2	2	2	2	2
b		1	1	3	4	1

3.4 유한 오토마타

- 각 동치류는 더 이상 구별되지 않는다. 마지막으로 DFA의 최종 상태가 E 이므로 E를 포함하는 모든 동치류는 {E}이므로 {E}가 M'의 최종 상태이다. 그래서 각 동치류 {A, C}, {B}, {D}, {E}를 X, Y, Z, W라 하면 최소화된 DFA $M' = (Q', \Sigma', \delta', q_0', F')$ 에서

$$Q' = \{X, Y, Z, W\}$$

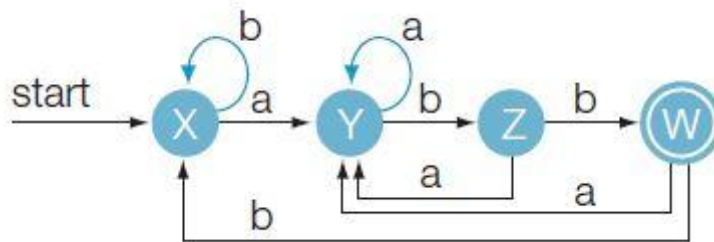
$$\Sigma' = \{a, b\}$$

$$q_0' = X$$

$$F' = \{W\}$$

δ	a	b
X	Y	X
Y	Y	Z
Z	Y	W
W	Y	X

상태수가 5개에서 4개로 줄었으며 최소화된 DFA를 상태 전이도로 표현하면



■ 정규 언어, 정규 문법, 유한 오토마타의 동치 관계

- 1) 전체 문법이 주어지면 문법으로 부터 토큰들을 분리해내고 이 토큰들을 정규 문법으로 표현함.
- 2) 토큰에 대한 정규 문법을 정규 표현으로 표시.
- 3) 이 정규 표현을 인식하는 인식기를 만들면 을 주면 언어가 주어지면, 이를 정규표현으로 변환,
- 4) 정규표현을 인식하는 NFA을 구성, NFA를 DFA로 변환, DFA를 최소화 하면 어휘 분석기를 만들 수 있음

■ 정규 문법, 정규 표현, 유한 오토마타의 관계가 서로 동치관계 임을 증명

1. 정규 문법 \Rightarrow 정규 표현
2. 정규 표현 \Rightarrow 유한 오토마타
3. 유한 오토마타 \Rightarrow 정규문법

■ 정규 문법 \Rightarrow 정규 표현으로 변환

- 정규 표현에 의해서 정의된 문법 G 에 의해 생성되는 언어 $L(G)$ 가 무엇인지를 알기 위해서는, 정규 문법을 정규 표현으로 변환
 - 정규 문법을 계수(coefficient)가 정규 표현으로 구성된 방정식인 정규 표현 방정식(regular expression equation)으로 바꾸고 정규 표현 방정식에서 해(solution)를 구함

• 정규 표현 방정식으로부터 해를 구하는 방법

» [정리 3-4] α, β 가 정규 표현이고, α 가 ε 을 포함하지 않는다면 $X = \alpha X + \beta$ 의 유일한 해(unique solution)는 $X = \alpha^* \beta$ 이다.

» [증명] 1) 정규 표현 방정식 $X = \alpha X + \beta$ 의 해가 $X = \alpha^* \beta$ 임을 증명

$$\begin{aligned} X &= \alpha X + \beta \\ &= \alpha(\alpha^* \beta) + \beta \\ &= \alpha^+ \beta + \beta \\ &= (\alpha^+ + \varepsilon) \beta \\ &= \alpha^* \beta \text{ 이므로} \end{aligned}$$

$X = \alpha X + \beta$ 의 해가 $\alpha^* \beta$ 가 됨을 알 수 있다.

2) 이 해가 유일하다는 것을 증명 \Rightarrow 생략

▪ [알고리즘 3-3] 정규 문법 \Rightarrow 정규 표현으로 변환 알고리즘

[입력] 정규 문법 $G = (V_N, V_T, P, S)$

단, V_N : non-terminal 기호들의 유한 집합

V_T : terminal 기호들의 유한 집합

P : 생성 규칙들의 집합

$A \rightarrow tB, A \rightarrow t$ 혹은 $A \rightarrow Bt, A \rightarrow t$

단, $t \in V_T^*$ $A, B \in V_N$

S : non-terminal 기호로서 시작 기호

[출력] 정규 문법 G 가 생성하는 언어 $L(G)$ 를 나타내는 정규 표현

[방법]

- (1) 정규 문법으로부터 정규 표현 방정식을 만든다.
- (2) 정규 표현 방정식 중 $X = \alpha X + \beta$ 형태를 찾아 $X = \alpha^* \beta$ 로 변환한다.
- (3) 시작 기호로부터 출발하여 다른 방정식들을 대입한다. 계산 결과에 $X = \alpha X + \beta$ 형태의 식이 나타나면 $X = \alpha^* \beta$ 로 변환하면, 정의된 정규 문법 G 로 부터 생성되는 정규 언어 $L(G)$ 를 나타내는 정규 표현이 된다.

- [예제 3-56] 정규 문법 $G = (\{S, A, B\}, \{a, b\}, P, S)$ 가 다음과 같을 때 G 로부터 생성되는 $L(G)$ 를 정규 표현으로 나타내보자.

$$S \rightarrow aA \mid bS$$

$$A \rightarrow aS \mid bB$$

$$B \rightarrow aB \mid bB \mid \varepsilon$$

[풀이] 생성 규칙을 정규 표현 방정식으로 변환하면

$$S = aA + bS \quad \text{-----(1)}$$

$$A = aS + bB \quad \text{-----(2)}$$

$$B = aB + bB + \varepsilon \quad \text{-----(3)}$$

(3)번식 $B = aB + bB + \varepsilon = (a+b)B + \varepsilon$ 이므로 $X = \alpha X + \beta$ 형태의 식이 된다.

$$B = (a+b)^* \varepsilon = (a+b)^* \quad \text{---(4)}$$

더 이상 $X = \alpha X + \beta$ 형태가 없고 시작 기호가 S 이므로 (1)번 식부터 시작

3.4 유한 오토마타

$$S = aA + bS$$

$$= a(aS + bB) + bS \text{ ((2)번 식 대입)}$$

$$= aaS + abB + bS$$

$$= aaS + ab(a+b)^* + bS \text{ ((4)번 식 대입)}$$

$$= (aa+b)S + ab(a+b)^*$$

그런데 이 식은 $X = \alpha X + \beta$ 형태이므로

$$S = (aa + b)^* ab(a+b)^*$$

- [예제 3-57] 정규 문법 $G = (\{S, A, B\}, \{a, b\}, P, S)$ 가 다음과 같을 때 G 로부터 생성되는 $L(G)$ 를 정규 표현으로 나타내보자.

- $$S \rightarrow aA \mid bS$$

- $$A \rightarrow aA \mid bA \mid b$$

- [풀이] 생성 규칙을 정규 표현 방정식으로 변환하면 다음과 같다.

- $$S = aA + bS \dots(1)$$

- $$A = aA + bA + b \dots(2)$$

- 먼저 (2)번 식에서 다음 식이 유도된다.

- $$A = aA + bA + b$$

- $$= (a + b)A + b$$

- $X = \alpha X + \beta$ 형태의 식이 된다.

- $\therefore A = (a + b)^*b \dots(3)$

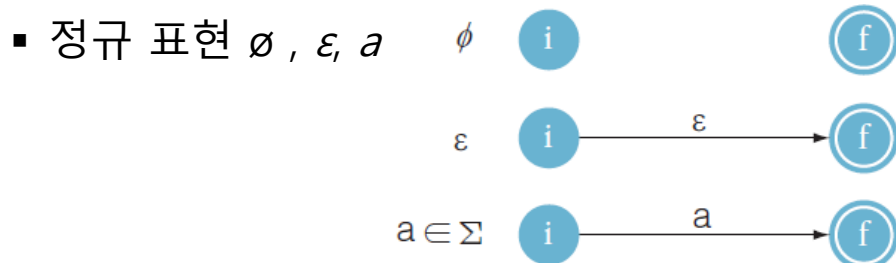
- (1)번 식에 (3)번 식을 대입해서 풀면 다음과 같다.

- $S = aA + bS$
- $= a(a + b)^*b + bS$
- $= bS + a(a + b)^*b$
- 그런데 이 식은 $X = \alpha X + \beta$ 의 형태이므로 $S = b^*a(a + b)^*b$ 이다.
- $\therefore L(G) = b^*a(a + b)^*b$

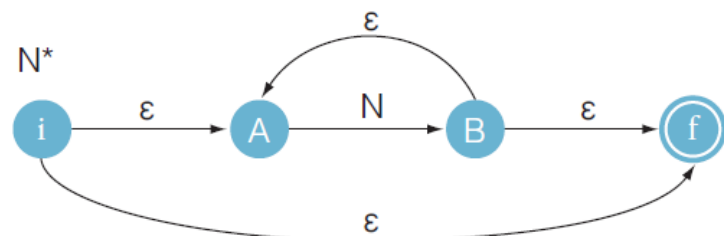
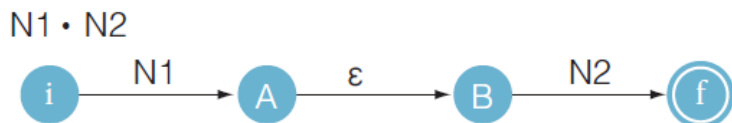
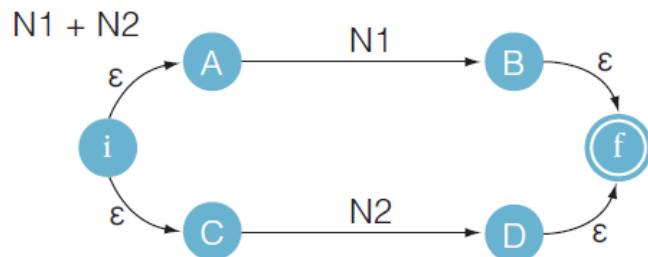
3.4 유한 오토마타

정규 표현 \Rightarrow 유한 오토마타로 변환

- 정규 표현의 정의에 있는 각각을 받아들이는 ϵ -NFA를 구성



- 정규 표현 $N1+N2, N1 \cdot N2, N^*$ 에 대해

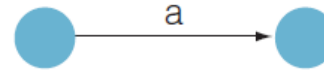


3.4 유한 오토마타

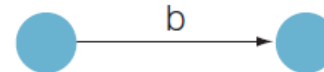
■ [예제 3-58] 정규 표현을 받아들이는 유한 오토마타 만들기 1

- 정규 표현 $(a + b)^*$ 를 인식하는 유한 오토마타를 구성해보자.

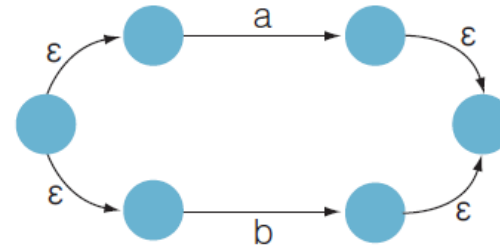
- ① a 를 인식하는 유한 오토마타를 구성



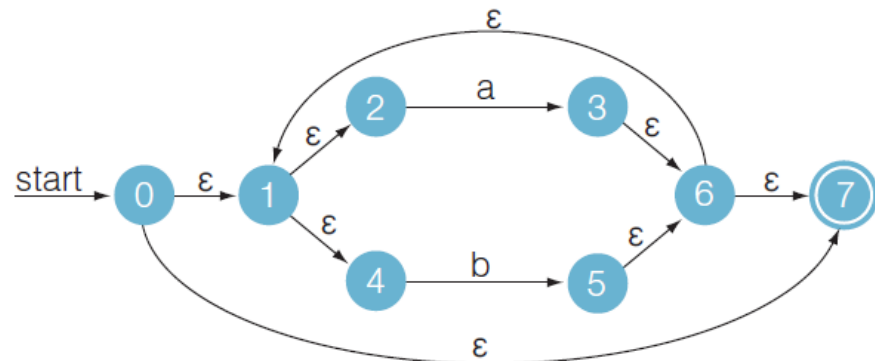
- ② b 를 인식하는 유한 오토마타를 구성



- ③ $a + b$ 를 인식하는 유한 오토마타를 구성



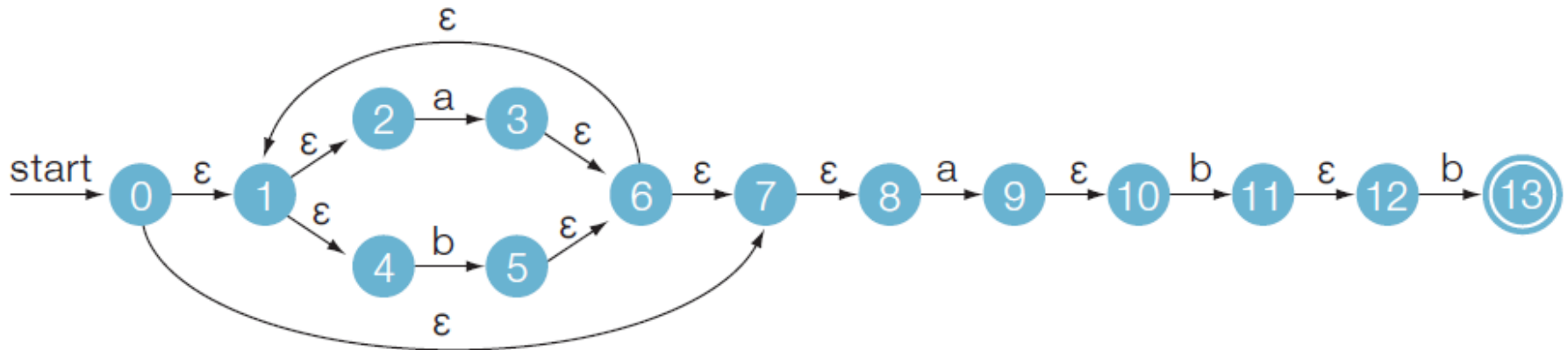
- ④ $(a + b)^*$ 를 인식하는 유한 오토마타를 구성



3.4 유한 오토마타

■ [예제 3-59] 정규 표현을 받아들이는 유한 오토마타 만들기 2

- [예제 3-48]의 정규 표현 $(a + b)^*abb$ 를 인식하는 NFA를 구성해보자.



■ [알고리즘 3-4] 유한 오토마타로 부터 정규 문법으로 변환

- 유한 오토마타 $M = (Q, \Sigma, \delta, q_0, F)$ 으로 부터 정규 문법 $G = (V_N, V_T, P, S)$ 를 만드는 것은 다음과 같이 하면 된다.

• **[입력]** 유한 오토마타 $M = (Q, \Sigma, \delta, q_0, F)$

단, Q : 상태들의 유한집합

Σ : 입력 기호들의 유한집합

δ : 상태전이 함수

q_0 : 종결기호

[출력] 정규문법 $G = (V_N, V_T, P, S)$

단, V_N : non-terminal 기호들의 집합

V_T : terminal 기호들의 집합

P : 생성규칙의 집합

S : 시작기호

[방법]

$V_N = Q$

$V_T = \Sigma$

$S = q_0$

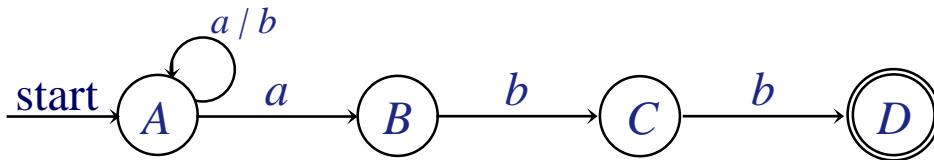
P : if $\delta(q, a) = r$, then $q \rightarrow ar$;

if $q \in F$, then $q \rightarrow \varepsilon$;

3.4 유한 오토마타

■ [예제 3-61] 유한 오토마타를 정규 문법으로 변환하기

- [예제 3-60]의 결과인 유한 오토마타를 정규 문법 G 로 변환해보자.



→ $G = (V_N, V_T, P, S)$

단, $V_N = \{A, B, C, D\}$

$V_T = \{a, b\}$

$S = A$

$P : A \rightarrow aA \mid bA \mid aB$

$B \rightarrow bC$

$C \rightarrow bD$

$D \rightarrow \varepsilon$

정규언어, 정규문법, 유한오토마타의 동치관계

