

# KNU 4471.043 컴파일러 설계

---

고상기

3주차

2022 Spring

강원대학교 컴퓨터공학과

## 2주차 요약

- 컴파일러의 논리적 구조
- 어휘 분석의 예
- 구문 분석의 예
- 의미 분석의 예
- 컴파일러의 물리적 구조

### 3주차 개요

- 형식 언어란?
- 형식 문법이란?
- 정규 표현식
- 유한 오토마타
- 정규 문법과 정규 표현식 그리고 오토마타

# 형식 언어의 정의

- 언어<sub>language</sub>: 알파벳으로부터 생성되는 모든 문자열들의 부분집합
- 문법<sub>grammar</sub>: 언어를 정의하고 생성하는 도구
- 인식기<sub>recognizer</sub>: 언어를 인식하는 가상의 기계

# 형식 언어의 정의

## 알파벳<sub>alphabet</sub>

- 언어의 문장을 이루는 기본적인 기호<sub>symbol</sub>의 집합
  - 알파벳은 공집합이 아닌 기호들의 유한 집합으로  $\Sigma$ 로 표기한다.
    - $\Sigma_{kor} = \{\text{ㄱ, ㅋ, ㆁ, ㄷ, ㄹ, \dots, ㅎ}\}$
    - $\Sigma_{eng} = \{a, b, c, d, \dots, z\}$
    - $\Sigma_{binary} = \{0, 1\}$
  - 일반 프로그래밍 언어에서는 사용 가능한 문자나 기호들의 집합을 알파벳이라 한다.

## 문자열<sub>string</sub>

- 알파벳  $\Sigma$ 에 대한 문자열은 알파벳에서 정의된 기호들을 나열한 유한 수열<sub>finite sequence</sub>이다.
  - 알파벳  $\Sigma = \{a, b, c\}$ 가 주어졌을 때  $a, ca, ccba$  등을 문자열이라 한다.
  - 언어 이론에서 문자열은 종종 문장<sub>sentence</sub>이나 단어<sub>word</sub>와 동의어로 사용된다.
  - 보통 문자를 표시할 때는 영어의  $a, b, c, \dots$  등을 주로 사용하고, 문자열을 나타낼 때는  $u, v, w, \dots$  등을 주로 사용한다.
    - 예)  $w = ababaa, u = babbbbcc$

# 형식 언어의 정의

## 문자열의 길이<sub>length</sub>

- 문자열을 이루는 기호의 개수를 의미하며 문자열  $w$ 의 길이를  $|w|$ 로 표시한다.
  - $w_1 = abc$ 일 때,  $|w_1| = 3$
  - $w_2 = aabbab$ 일 때,  $|w_2| = 6$
  - $w = a_1 a_2 \cdots a_k$ 일 때,  $|w| = k$

## 문자열의 결합<sub>concatenation</sub>

- 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 때, 두 문자열의 결합은  $u \cdot v$  또는  $uv$ 로 표시하며  $uv = a_1 a_2 a_3 \cdots a_n b_1 b_2 \cdots b_m$ 이다.
  - 두 문자열  $u = Kangwon, v = University$ 가 있을 때,

$$uv = KangwonUniversity, vu = UniversityKangwon$$

# 형식 언어의 정의

## 빈 문자열<sub>empty string</sub>

- 문자열의 길이가 0인 문자열
  - $\varepsilon$ 으로 표기
  - 어떤 문자열  $u, v$ 에 대하여 다음과 같은 속성을 갖는다.
    - $u\varepsilon = u = \varepsilon u$
    - $u\varepsilon v = uv$
  - $\lambda$ 로도 표시하며, 공 문자열 또는 널 문자열<sub>null string</sub>이라고도 한다. 결합 연산의 경우  $\varepsilon$ 을 항등원으로 취한다.
  - $a^n$ 은 기호  $a$ 가  $n$ 개 연결된 문자열을 나타내는데, 이 때  $a^0 = \varepsilon$ 으로 표시할 수 있다.

## 문자열의 역<sub>reverse</sub>

- 문자열을 이루는 요소들을 역순으로 뒤집은 것이다.
  - 문자열  $w = kangwon$ 에 대해 문자열 역인  $w^R$ 은,

$$w^R = nowgnak.$$

# 형식 언어의 정의

## 접두사<sub>prefix</sub>와 접미사<sub>suffix</sub>

- 문자열  $w = uv$ 일 때,  $u$ 를  $w$ 의 접두사라 한다. 이 때,  $u \neq w$ 이면 이를 진접두사<sub>proper prefix</sub>라 한다.
- 문자열  $w = uv$ 일 때,  $v$ 를  $w$ 의 접미사<sub>suffix</sub>라 한다.
- 문자열  $w = \text{computer}$ 에 대해서,
  - 접두사:  $\varepsilon, \text{co}, \text{com}, \text{comp}, \text{compu}, \text{comput}, \text{compute}, \text{computer}$
  - 접미사:  $\varepsilon, \text{r}, \text{er}, \text{ter}, \text{uter}, \text{puter}, \text{mputer}, \text{omputer}, \text{computer}$

## $\Sigma$ -클리니-스타<sub>Kleene-Star, reflexive transitive closure</sub>

- 빈 문자열을 포함하여 알파벳  $\Sigma$ 의 결합 연산에 의해 만들 수 있는 모든 문자열들의 집합을  $\Sigma^*$ 라고 쓴다. ( $\Sigma$ 에 클리니-스타 연산을 적용한 것)
  - $\Sigma$ -클리니-스타에서 빈 문자열을 제외한  $\Sigma$ -클리니-플러스<sub>Kleene-Plus, transitive closure</sub>  $\Sigma^+$ 도 존재한다. 이 때,  $\Sigma^* - \{\varepsilon\} = \Sigma^+$ 가 성립한다.
  - $\Sigma = \{0, 1\}$  일 때,  $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$ .



# 형식 언어의 정의

## 언어<sub>language</sub>

- 알파벳  $\Sigma$ 에 대해  $\Sigma^*$ 의 부분 집합을 의미한다.
  - 유한 언어<sub>finite language</sub>: 언어에 속하는 문자열의 수가 유한한 경우
  - 무한 언어<sub>infinite language</sub>: 언어에 속하는 문자열의 수가 무한한 경우
    - $\Sigma = \{a, b\}$ 일 때,  $L_1 = \Sigma^*$ 는 무한 언어
    - $L_2 = \{a, ba, aaa\}$ 는 유한 언어
    - $L_3 = \{a^p \mid p \text{는 소수 prime number}\}$ 는 무한 언어
    - $L_4 = \{a^n b^n \mid n \geq 1\}$ 는 무한 언어
- 여기서 말하는 언어는 의미<sub>semantic</sub>의 개념을 포함하지 않는다.
- 언어는 단지 문자열들의 집합으로 정의되고 형식 언어 이론은 문자열 집합을 생성하는 형식 문법과 이를 인식하는 가상 기계에 관한 이론이다.

# 형식 언어 간의 연산

## 두 언어의 합집합<sub>union</sub>

- 두 언어  $L$ 과  $M$ 의 합집합  $L \cup M$ 은 언어  $L$ 에 속하는 문자열이거나  $M$ 에 속하는 문자열의 집합이다.
- $L \cup M = \{w \mid w \in L \text{ 또는 } w \in M\}$ .
- $L = \{aaa, bbb\}$ ,  $M = \{aaa, ccc\}$ 일 때,  $L \cup M = \{aaa, bbb, ccc\}$ .

## 두 언어의 결합<sub>concatenation</sub>

- 두 언어  $L$ 과  $M$ 의 결합  $LM$ 은  $L$ 에 속하는 문자열과  $M$ 에 속하는 문자열을 결합한 것으로 교환 법칙은 성립하지 않는다.
- $LM = \{uv \mid u \in L \text{ 이고 } v \in M\}$ .
- $L = \{aaa, \varepsilon\}$ ,  $M = \{bbb, c, \varepsilon\}$ 일 때,  $LM = \{\varepsilon, c, aaa, bbb, aaabbb, aaac\}$ .

# 형식 언어 간의 연산

## 거듭제곱<sub>square</sub>

- 언어  $L$ 의 거듭 제곱은 재귀적으로<sub>recursively</sub> 다음과 같이 정의한다.
  - 먼저,  $L^0 = \{\varepsilon\}$ 이라 하자.
  - 그렇다면  $n \geq 1$ 에 대해,  $L^n = LL^{n-1}$ 이 성립한다.
  - $L = \{a, ba\}$ 일 때,  $L^3 = \{aaa, aaba, abaa, ababa, baaa, baaba, babaa, bababa\}$ .

## 클리니-스타<sub>Kleene-Star, reflexive transitive closure</sub>

- 언어  $L$ 에 대한 클리니-스타  $L^*$ 는 다음과 같이 정의한다.
  - $L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \dots \cup L^n \cup \dots = \bigcup_{i=0}^{\infty} L^i$
  - 언어  $L$ 에 대한 클리니-플러스  $L^+$ 는 다음과 같이 정의한다.
  - $L^+ = L^1 \cup L^2 \cup L^3 \cup \dots \cup L^n \cup \dots = \bigcup_{i=1}^{\infty} L^i = L^* - L^0$

## 형식 문법의 정의

형식 문법  $G = (V_N, V_T, P, S)$ 는 다음과 같이 네 가지 항목으로 정의한다.

1.  $V_N$ : 논터미널 기호<sub>nonterminal symbol</sub>의 유한 집합
2.  $V_T$ : 터미널 기호<sub>terminal symbol</sub>의 유한 집합

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

3.  $P$ : 생성 규칙<sub>production rule</sub>의 유한 집합

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

4.  $S$ :  $V_N$ 에 속하는 기호로써, 다른 논터미널 기호와 구별하여 시작 기호<sub>start nonterminal</sub>라 한다.

- $S, A, B, C$ 와 같이 영문 대문자로 구성된 기호는 논터미널 기호이다. ( $S$ 는 시작 기호)
- ‘ $\langle$ ’와 ‘ $\rangle$ ’로 묶어서 나타낸 기호도 논터미널 기호이다.
- $a, b, c$ 와 같은 영문 소문자로 구성된 기호와  $+, -, *, /$  등의 연산자 기호, 괄호나 쉼표와 같은 구분자<sub>delimiter</sub>,  $0, 1, 2$ 와 같은 아라비아 숫자들은 터미널 기호이다.
- $X, Y, Z$ 와 같은 영문 알파벳 끝부분의 대문자는 임의의 터미널 기호 또는 논터미널 기호를 의미한다.
- $u, v, w, x, y, z$ 와 같은 영문 알파벳 끝부분의 소문자는 터미널 기호로 이루어진 문자열을 나타낸다.
- $\alpha, \beta, \gamma$ 와 같은 그리스어 소문자는 논터미널 기호 또는 터미널 기호로 구성된 문자열을 나타낸다.

- 형식 문법이 어떤 언어를 생성하는지 어떻게 확인하지?
- 시작 기호로부터 생성 규칙을 반복 적용하여 터미널 기호로 이루어진 문자열들을 만들어낸다.
- 이것을 유도라고 하고 ‘ $\Rightarrow$ ’ 기호를 사용하여 과정을 기술한다.
- 생성 규칙  $\alpha \rightarrow \beta$ 가 존재하고,  $\gamma, \delta \in V^*$ 일 때, 다음과 같이 생성 규칙을 적용한다.

$$\gamma\alpha\delta \Rightarrow \gamma\beta\delta$$

- 생성 규칙을 적용하면 규칙의 왼편에 있는 부분( $\alpha$ )이 오른편에 있는 부분( $\beta$ )으로 대체된다.
- ‘ $\xRightarrow{*}$ ’는 영 번 이상의 유도zero or more derivations를 의미하며, ‘ $\xRightarrow{+}$ ’는 한 번 이상의 유도one or more derivations를 의미한다.
- 만약  $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{n+1} \in V^*$ 이고,  $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \dots \Rightarrow \alpha_{n+1}$ 이 존재한다면,  $\alpha \xRightarrow{n} \alpha_{n+1}$ 으로 쓰고,  $\alpha \xRightarrow{*} \alpha_{n+1}$ 로도 쓸 수 있다.

## 유도 예

다음의 문법  $G$ 가 문자열 0, 0000, 001100 등을 생성하는지 확인해보자.

$$G = (\{S, A\}, \{0, 1\}, P, S),$$

$$P = \{S \rightarrow 0AS, S \rightarrow 0, A \rightarrow S1A, A \rightarrow 10, A \rightarrow SS\}.$$

1. 0의 경우 ' $S \Rightarrow 0$ ' 규칙을 통해 바로 확인 가능
2. 0000은?

$$S \Rightarrow 0AS$$

$$\Rightarrow 0SSS$$

$$\Rightarrow 00SS$$

$$\Rightarrow 000S$$

$$\Rightarrow 0000$$

3. 001100은?

# 형식 문법이 생성하는 언어

## 문장과 문장 형태

- 앞서 설명한 대로  $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots w_n \Rightarrow w$ 와 같은 유도 과정이 있으면,  $S \xRightarrow{*} w$ 라고 쓸 수 있다.
- 이 때,  $w$ 가  $V^*$ 에 속하면  $w_1, w_2, \dots, w_n, w$  등을 문장 형태<sub>sentential form</sub>라 하고  $V_T^*$ 에 속하면 문장<sub>sentence</sub>라고 한다.
- 다시 말해, 터미널 기호와 논터미널 기호의 조합으로 구성되어 있는 문자열을 문장 형태라 하고, 터미널 기호로만 구성된 문자열을 문장이라고 한다.

## 형식 문법이 생성하는 언어

- 형식 문법  $G$ 가 생성하는 언어는  $G$ 에 의해 생성되는 문장의 집합이며  $L(G)$ 로 표기한다.

$$L(G) = \{w \mid S \xRightarrow{*} w, w \in V_T^*\}$$



## 형식 문법의 예 1

다음의 문법  $G_1$ 는 어떤 언어를 생성할까?

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

$$P = \{S \rightarrow A, A \rightarrow abC \mid aABC, bB \rightarrow bbb, bC \rightarrow bb\}.$$

## 형식 문법의 예 2

다음의 문법  $G_2$ 는 어떤 언어를 생성할까?

$$G_2 = (\{S, A\}, \{a, b\}, P, S),$$
$$P = \{S \rightarrow aAa, A \rightarrow aAa \mid b\}.$$

## 형식 문법의 예 3

다음의 문법  $G_3$ 는 어떤 언어를 생성할까?

$$G_3 = (\{S, A, B\}, \{a, b\}, P, S),$$

$$P = \{S \rightarrow aS \mid aA, A \rightarrow aB, B \rightarrow aB \mid a\}.$$

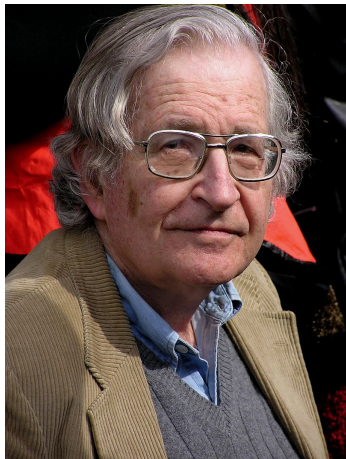
## 형식 문법의 예 4

다음의 문법  $G_4$ 는 어떤 언어를 생성할까?

$$G_4 = (\{S, A\}, \{a\}, P, S),$$

$$P = \{S \rightarrow a \mid aA, A \rightarrow aS\}.$$

## 촘스키 위계 Chomsky hierarchy



노암 촘스키 Noam Chomsky (1928-현재)

- 미국의 언어학자, 철학자, 정치 활동가, 저술가. 현재는 MIT의 언어학과 교수
- 수많은 자연어의 문법으로부터 보편적인 문법을 찾아내고자 시도 — “언어는 유한한 수단의 무한한 활용을 수반한다.”
- 변형생성문법 transformational generative grammar 이론을 개발
- 1956년, 형식 언어를 생성하는 형식 문법들 사이의 위계인 촘스키 위계 Chomsky hierarchy 를 제안

## 췁스키 위계 Chomsky hierarchy

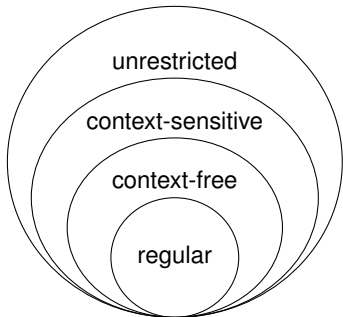
췁스키 위계는 아래의 네 유형으로 이루어진다.

- Type-0 문법: 무제약 문법 unrestricted grammar: 생성 규칙에 어떠한 제한도 없음
  - 생성 규칙:  $\alpha \rightarrow \beta$ . 단  $\alpha$ 는 빈 문자열이 될 수 없다.
- Type-1 문법: 문맥-인식 문법 context-sensitive grammar: 생성 규칙의 왼쪽 부분이 오른쪽 부분보다 길 수 없음
  - 생성 규칙:  $\alpha \rightarrow \beta$ . 단,  $|\alpha| \leq |\beta|$ ,  $\alpha \in V^+$ ,  $\beta \in V^*$ .
- Type-2 문법: 문맥-자유 문법 context-free grammar: 생성 규칙의 왼쪽 부분은 하나의 논터미널 기호여야 함
  - 생성 규칙:  $A \rightarrow \alpha$ . 단,  $A \in V_N, \alpha \in V^*$ .
- Type-3 문법: 정규 문법 regular grammar: 생성 규칙의 오른쪽 부분에 최대 하나의 논터미널이 있으며 항상 가장 왼쪽 (또는 항상 가장 오른쪽)에 위치해야 함
  - 생성 규칙:  $A \rightarrow a$ 와  $A \rightarrow aB$  (또는  $A \rightarrow Ba$ ). 단,  $a \in V_T, A, B \in V_N$ .

# 형식 문법의 분류

형식 문법을 분류하는 방법

1. 먼저 정규 문법인지 확인한다.
2. 정규 문법이 아니라면, 문맥-자유 문법인지 확인한다.
3. 문맥-자유 문법이 아니라면, 문맥-인식 문법인지 확인한다.
4. 문맥-인식 문법이 아니라면, 무제한 문법으로 분류한다.



## 형식 문법의 분류 예 1

다음의 문법  $G_1$ 는 촘스키 위계 상 어디에 속하는가?

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

$$P = \{S \rightarrow A, A \rightarrow abC \mid aABC, bB \rightarrow bbb, bC \rightarrow bb\}.$$



## 형식 문법의 분류 예 2

다음의 문법  $G_2$ 는 촘스키 위계 상 어디에 속하는가?

$$G_2 = (\{S, A\}, \{a, b\}, P, S),$$
$$P = \{S \rightarrow aAa, A \rightarrow aAa \mid b\}.$$

## 형식 문법의 분류 예 3

다음의 문법  $G_3$ 는 촘스키 위계 상 어디에 속하는가?

$$G_3 = (\{S, A, B\}, \{a, b\}, P, S),$$

$$P = \{S \rightarrow aS \mid aA, A \rightarrow aB, B \rightarrow aB \mid a\}.$$

## 형식 문법의 분류 예 4

다음의 문법  $G_4$ 는 촘스키 위계 상 어디에 속하는가?

$$G_4 = (\{S, A\}, \{a\}, P, S),$$

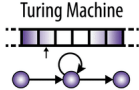

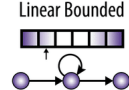

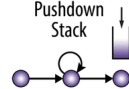

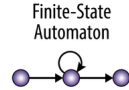

$$P = \{S \rightarrow a \mid aA, A \rightarrow aS\}.$$

## 형식 언어의 예

아래의 언어들은 형식 언어 이론에서 자주 언급되는 언어들이다.

1. 단순 매칭 언어<sub>simple matching language</sub>:  $L_m = \{a^n b^n \mid n \geq 0\}$
2. 중복 매칭 언어<sub>double matching language</sub>:  $L_{dm} = \{a^n b^n c^n \mid n \geq 0\}$
3. 좌우 대칭 언어<sub>mirror image language</sub>:  $L_{mi} = \{ww^R \mid w \in \Sigma^*\}$
4. 회문 언어<sub>palindrome language</sub>:  $L_r = \{w \mid w = w^R\}$
5. 복사 언어<sub>copy language</sub>:  $L_c = \{ww \mid w \in \Sigma^*\}$
6. 괄호 언어<sub>parenthesis language</sub>:  $L_p = \{w \mid w \text{는 균형 잡힌 괄호 문자열}\}$

# 형식 언어와 인식기<sup>1</sup>

Language	Automaton	Grammar	Recognition
Recursively Enumerable Languages	 <p>Turing Machine</p>	Unrestricted $Baa \rightarrow A$	Undecidable 
Context-Sensitive Languages	 <p>Linear Bounded</p>	Context Sensitive $A t \rightarrow aA$	Exponential? 
Context-Free Languages	 <p>Pushdown Stack</p>	Context Free $S \rightarrow gSc$	Polynomial 
Regular Languages	 <p>Finite-State Automaton</p>	Regular $A \rightarrow cA$	Linear 

<sup>1</sup>D. B. Searls, The language of the genes. Nature 420, 211– 217, 2012

## 정규 표현식 regular expression

- 정규 언어를 표현하기 위한 일종의 표기법이다.
- 다음과 같이 재귀적으로 정의한다.
  - 알파벳  $\Sigma$ 가 주어졌을 때,  $\emptyset$ ,  $\varepsilon$  그리고  $\Sigma$ 에 속한 모든 기호  $a$ 는 정규 표현식이다.
  - 만약,  $R_1$ 과  $R_2$ 가 정규 언어  $L_1$ 과  $L_2$ 를 표현하는 정규 표현식이라면,
    - $(R_1) + (R_2)$ 는  $L_1 \cup L_2$ 를 표현하는 정규 표현식이다.
    - $(R_1) \cdot (R_2)$ 는  $L_1 L_2$ 를 표현하는 정규 표현식이다.
    - $(R_1)^*$ 는  $L_1^*$ 를 나타내는 정규 표현식이다.
- 정규 표현식을 구성하는 연산자는 클리니-스타(\*), 결합( $\cdot$ ), 합집합(+)이 있으며 연산자의 우선 순위는 클리니-스타 > 결합 > 합집합이다.
- 연산자 우선 순위가 결정되면 괄호는 생략할 수 있으며 결합 연산자 ' $\cdot$ '은 생략 가능하다.

## 정규 표현식 예

알파벳  $\Sigma = \{0, 1\}$ 이 주어졌을 때, 아래의 정규 표현식들이 나타내는 언어는?

1.  $0 + 1$
2.  $(0 + 1)0$
3.  $0^*$
4.  $(0 + 1)^*$
5.  $0^*1^*$

## 정규 표현식의 대수학적 성질

1.  $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$

(합집합에 대한 결합 법칙)

2.  $(\alpha\beta)\gamma = \alpha(\beta\gamma)$

(접속에 대한 결합 법칙)

3.  $\alpha + \beta = \beta + \alpha$

(합집합에 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.  $\varepsilon\alpha = \alpha = \alpha\varepsilon$

(결합 연산의 항등원)

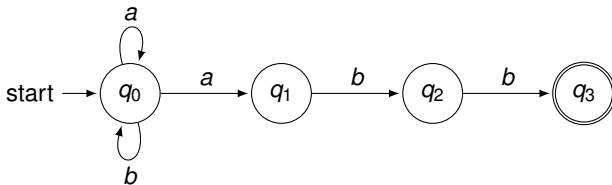
8.  $\alpha^* = \varepsilon + \alpha\alpha^*$

9.  $(\alpha^*)^* = \alpha^*$



## 유한 오토마타 finite-state automata, FA

- 가장 간단한 형태의 언어 인식기이며 간단히 FA라고 부른다.
- 형식적으로 수식을 통해 표현하는 방법과 상태 전이도 state transition diagram를 그려 표현하는 방법이 있다.
- 상태 전이도를 그릴 땐, FA의 각 상태 state를 노드로 표현하고, 간선은 전이 transition를 나타낸다.
- FA의 최종 상태 final state는 이중 원 double circle으로 나타내고 시작 상태 initial state는 시작 간선으로 표시한다.
- 아래는 정규 표현식  $(a + b)^*abb$ 를 인식하는 FA를 그린 것이다.



## FA의 형식적 정의 formal definition

FA  $M = (Q, \Sigma, \delta, q_0, F)$ 은 다음과 같이 정의된다.

1.  $Q$ : 상태의 유한 집합
2.  $\Sigma$ : 입력 기호의 유한 집합
3.  $\delta$ : 전이 함수 transition function로  $Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ ( $Q$ 의 멱집합)의 원소이다. 즉,  
 $\delta(q, a) = \{p_1, p_2, \dots, p_n\} \subseteq Q$ .
4.  $q_0$ : 시작 상태를 의미하며  $q_0 \in Q$ 를 만족한다.
5.  $F$ : 최종 상태의 집합을 의미하며  $F \subseteq Q$ 를 만족한다.

전이 함수  $\delta$ 는 아래와 같이 문자열 입력에 대해 확장될 수 있다.

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

위의 확장을 활용하여, FA  $M$ 이 인식하는 언어는 다음과 같이 정의될 수 있다.

$$L(M) = \{w \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$$

## FA를 형식적으로 표현하기 예

정규 표현식  $(a+b)^*abb$ 를 인식하는 FA  $M = (Q, \Sigma, \delta, q_0, F)$ 는 다음과 같이 표현된다.

$$\text{상태 집합 } Q = \{q_0, q_1, q_2, q_3\},$$

$$\text{입력 기호 } \Sigma = \{a, b\},$$

$$\text{전이 함수 } \delta: \quad \delta(q_0, a) = \{q_0, q_1\},$$

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

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

$$\delta(q_2, b) = \{q_3\},$$

$$\text{시작 상태 } q_0 = q_0,$$

$$\text{최종 상태 집합 } F = \{q_3\}.$$

## 결정적 FA와 비결정적 FA

- FA는 기본적으로 비결정적  $\text{nondeterministic}$ 이다.
- 비결정적 FA  $\text{nondeterministic FA}$ 는 줄여서 NFA라고 부른다.
- NFA 중에 다음 두 가지 조건을 만족하는 것들을 결정적 FA  $\text{deterministic FA}$ , 즉 DFA라고 부른다.

### DFA의 조건

1.  $\epsilon$ 에 의한 상태 전이( $\epsilon$ -전이  $\text{epsilon transition}$ )가 존재하지 않는다.
2. 모든 상태에서, 하나의 기호를 읽고 이동할 수 있는 상태의 수는 한 가지이다.
3. 다시 말하면, 상태 전이 함수  $\delta$ 가  $Q \times \Sigma \rightarrow Q$ 가 된다.

다음 FA  $M = (Q, \Sigma, \delta, q_0, F)$ 가 DFA인지 판단해보자.

상태 집합  $Q = \{q_0, q_1, q_2, q_3\}$ ,

입력 기호  $\Sigma = \{a, b\}$ ,

전이 함수  $\delta$ :  $\delta(q_0, a) = \{q_2\}$ ,  $\delta(q_0, b) = \{q_1\}$ ,

$\delta(q_1, a) = \{q_3\}$ ,  $\delta(q_1, b) = \{q_0\}$ ,

$\delta(q_2, a) = \{q_0\}$ ,  $\delta(q_2, b) = \{q_3\}$ ,

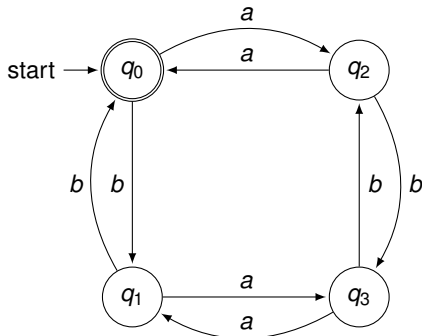
$\delta(q_3, a) = \{q_1\}$ ,  $\delta(q_3, b) = \{q_2\}$ ,

시작 상태  $q_0 = q_0$ ,

최종 상태 집합  $F = \{q_0\}$ .

## DFA에서의 문자열 인식 예

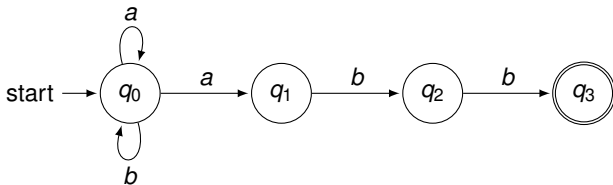
다음은 슬라이드 36페이지의 DFA를 상태 전이도로 표현한 것이다.



- 문자열 *abab*가 인식되는지 확인해보자.
- 문자열 *ababaab*가 인식되는지 확인해보자.

## NFA에서의 문자열 인식 예

다음은 슬라이드 33페이지의 NFA를 상태 전이도로 표현한 것이다.



- 문자열 *baabb*가 인식되는지 확인해보자.

## NFA와 DFA의 관계

- DFA보다 NFA가 언어의 구조를 더 쉽게 표현할 수 있다.
  - 모든 DFA는 NFA이기 때문에 당연한 사실이다. 제약 조건이 더 적으므로 표현의 가능성이 더 풍부하다.
- NFA는 DFA보다 프로그램으로 구현하기 어렵다.
  - 되추적<sup>backtracking</sup> 알고리즘을 구현하거나 하거나 기호를 읽을 때마다 상태 집합을 업데이트해야 한다.
- DFA는 프로그램으로 구현하면 문자열을 인식하는 알고리즘을 훨씬 더 효율적으로 구현할 수 있다.
  - 문자열의 길이가  $n$ 일 때,  $O(n)$  시간 알고리즘을 간단히 구현할 수 있다.
- 결론: 주어진 언어를 인식할 수 있는 NFA를 먼저 만든 후 이것을 DFA로 변환할 수는 없을까?

### Theorem

모든 NFA는 동등<sup>equivalent</sup>한 언어를 인식하는 DFA를 갖는다.

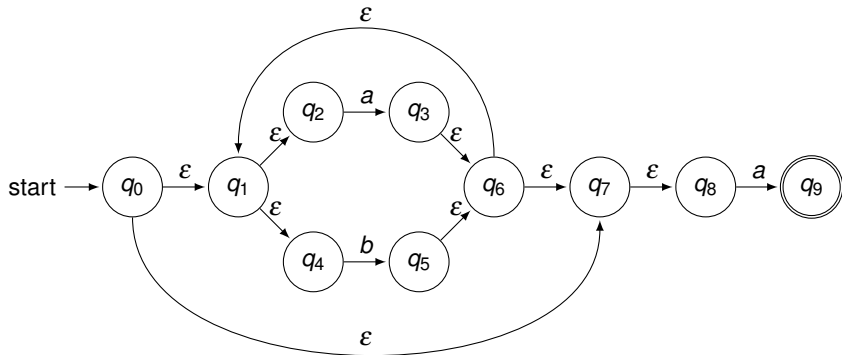


- NFA  $M = (Q, \Sigma, \delta, q_0, F)$ 가 주어지고 상태  $q \in Q$ 는  $M$ 의 상태 중 하나이고,  $P \subseteq Q$ 는  $M$ 의 상태 집합  $Q$ 의 부분 집합이다.
- 상태  $q$ 의  $\epsilon$ -클로저는  $\epsilon\text{-closure}(q)$ 로 표현하며 상태  $q$ 와 상태  $q$ 로부터  $\epsilon$ -전이에 의해 도달할 수 있는 NFA의 모든 상태의 집합을 말한다.
- 이 때, 상태 집합  $P$ 의  $\epsilon$ -클로저는  $\epsilon\text{-closure}(P)$ 로 쓰고 아래와 같이 정의한다.

$$\epsilon\text{-closure}(P) = \bigcup_{p \in P} \epsilon\text{-closure}(p).$$

- 다시 말해 상태 집합  $P$ 의  $\epsilon$ -클로저는  $P$ 에 속한 상태 각각의  $\epsilon$ -클로저를 계산한 후 합집합을 구한 것이다.

## $\epsilon$ -클로저 구하기 예



- $\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2, q_4, q_7, q_8\}$
- $\epsilon\text{-closure}(q_1) = \{q_1, q_2, q_4\}$
- $\epsilon\text{-closure}(q_2) = \{q_2\}$

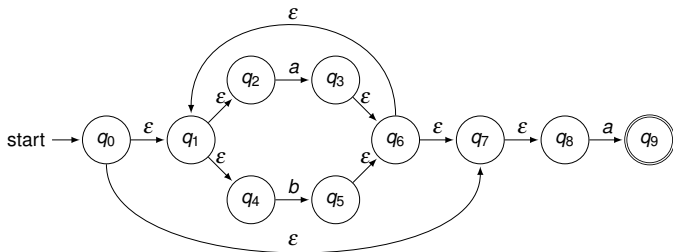
## 부분집합 구성 알고리즘 subset construction algorithm

부분집합 구성 알고리즘은 NFA로부터 동등한 언어를 인식하는 DFA를 만드는 과정이다. NFA  $M = (Q, \Sigma, \delta, q_0, F)$ 이 입력으로 주어졌다고 가정하자.

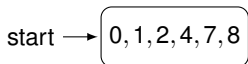
- 알고리즘의 순서는 다음과 같다.
  1. 시작 상태  $q_0$ 에 대해  $\varepsilon$ -클로저를 구하고 그 상태 집합  $\varepsilon\text{-closure}(q_0)$ 를 새로운 DFA의 시작 상태로 둔다.
  2. 방금 계산한  $\varepsilon\text{-closure}(q_0)$ 를 집합  $D_s$ 에 넣는다.
  3. 집합  $D_s$ 에서 하나의 상태  $T \subseteq Q$ 를 꺼낸 후, 그 상태에서 가능한 모든 입력 기호  $a \in \Sigma$ 에 대해 각자 도달할 수 있는 상태 집합  $T_a$ 를 만든다.
  4. 각각의 상태 집합  $T_a$ 로부터  $\varepsilon\text{-closure}(T_a)$ 를 계산한다.
  5. 모든 입력 기호  $a \in \Sigma$ 에 대해 상태  $\varepsilon\text{-closure}(T_a)$ 를 DFA의 새로운 상태로 추가하고 (집합  $D_s$ 에 추가) 상태  $T$ 에서 상태  $\varepsilon\text{-closure}(T_a)$ 로 입력 기호  $a$ 를 받아 이동하는 전이를 생성한다. ( $\varepsilon\text{-closure}(T_a)$ 가 DFA에 이미 존재한다면 전이만 생성한다.)
  6. 집합  $D_s$ 가 공집합이 될 때까지 3. 과정을 반복한다.
  7. 새롭게 만들어진 DFA의 상태  $T'$ 이 아래 조건을 만족하면 DFA의 최종 상태가 된다.

$$\varepsilon\text{-closure}(T') \cap F \neq \emptyset$$

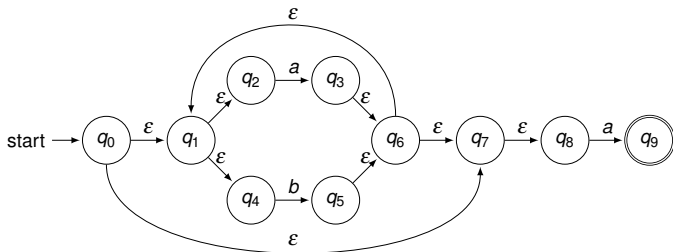
## 부분집합 구성 알고리즘 예



$$\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2, q_4, q_7, q_8\}$$

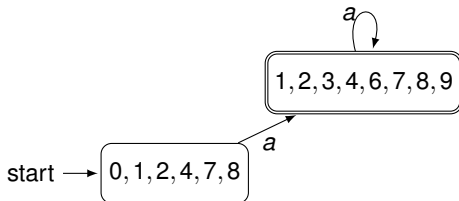


## 부분집합 구성 알고리즘 예

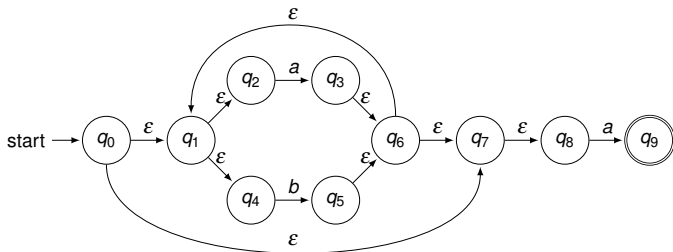


$$\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2, q_4, q_7, q_8\}$$

$$\rightarrow \varepsilon\text{-closure}(\delta(\{q_0, q_1, q_2, q_4, q_7, q_8\}, a)) = \{q_1, q_2, q_3, q_4, q_6, q_7, q_8, q_9\}$$

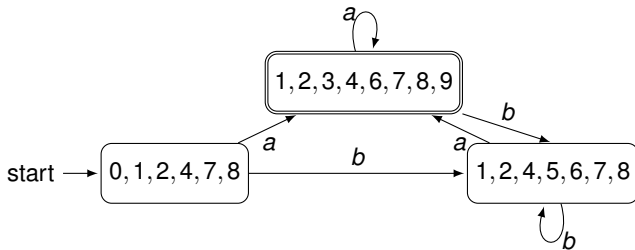


## 부분집합 구성 알고리즘 예



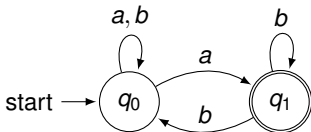
$$\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2, q_4, q_7, q_8\}$$

$$\rightarrow \varepsilon\text{-closure}(\delta(\{q_0, q_1, q_2, q_4, q_7, q_8\}, b)) = \{q_1, q_2, q_4, q_5, q_6, q_7, q_8\}$$

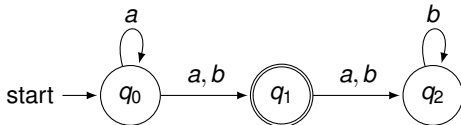


## 부분집합 구성 알고리즘 예 2

- 아래의 NFA를 DFA로 변환해보자.



- 아래의 NFA를 DFA로 변환해보자.



# DFA의 크기를 줄이는 방법

- DFA의 상태수를 최소화하기
  - DFA의 상태수를 줄여 DFA를 이용하는 어휘 분석기의 크기를 줄일 수 있다.
  - DFA의 상태 간의 동치 관계<sub>equivalence relation</sub>를 계산하여 동치인 상태들을 합친다.

## Definition

FA의 두 상태  $p$ 와  $q$ 는 모든 문자열  $w \in \Sigma^*$ 에 대해 아래의 조건을 만족할 때 구분 불가능<sub>indistinguishable</sub>하다고 말한다:

$$\delta^*(p, w) \in F \text{ 일 때 } \delta^*(q, w) \in F \text{이고, } \delta^*(p, w) \notin F \text{ 일 때 } \delta^*(q, w) \notin F.$$

만약 아래의 조건을 만족하는 만족하는 문자열  $w \in \Sigma^*$ 가 존재하면, 두 상태는 구분 가능<sub>distinguishable</sub>하다고 말한다:

$$\delta^*(p, w) \in F \text{ 이고 } \delta^*(q, w) \notin F \text{이거나, } \delta^*(p, w) \notin F \text{ 이고 } \delta^*(q, w) \in F.$$



## DFA 최소화 알고리즘 minimization algorithm

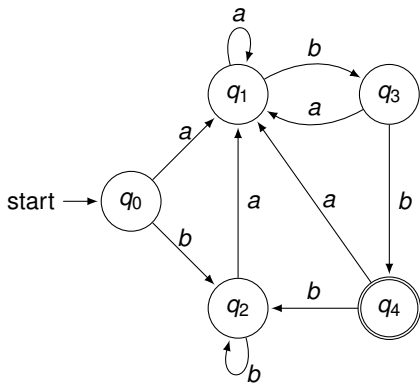
DFA  $M = (Q, \Sigma, \delta, q_0, F)$ 이 입력으로 주어졌다고 가정하자.

- 알고리즘의 순서는 다음과 같다.

1. 시작 상태  $q_0$ 에서 도달 불가능한 unreachable 상태들을 모두 제거한다.
2. 전체 상태를 최종 상태와 최종 상태가 아닌 두 동치류 equivalence class로 나눈다.
3. 하나의 동치류 안에서 같은 입력 기호에 대해 서로 다른 동치류로 가는 전이가 존재하면 그 동치류를 분할하여 나눈다.
4. 더 이상 새로운 분할이 일어나지 않을 때까지 3. 의 과정을 반복한다.
5. 최종적으로 만들어지는 최소화된 DFA  $M' = (Q', \Sigma, \delta, q'_0, F')$ 는 다음과 같이 정의된다.

- $Q'$ 의 한 상태를  $[q]$ 로 표시하며, 이는 상태  $q$ 를 포함하는 동치류를 나타낸다.
- 두 동치류  $[p]$ 와  $[q]$ 에 대해  $\delta(p, a) = q$ 면,  $\delta'([p], a) = [q]$ 이다.
- $q'_0 = [q_0]$
- $F' = \{[q] \mid F\}$

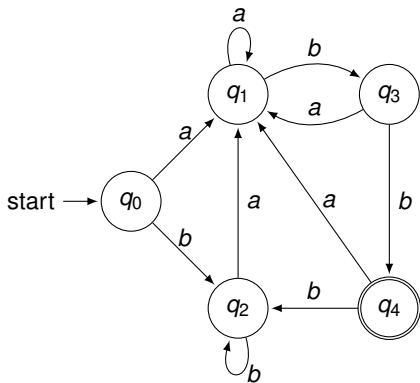
## DFA 최소화 알고리즘 예



입력	1				2
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$
$a$	1	1	1	1	1
$b$	1	1	1	2	1

- $q_3$ 이  $b$ 를 읽고 다른 동치류로 이동하므로,  $\{q_0, q_1, q_2\}$ 와  $\{q_3\}$ 을 분할한다.

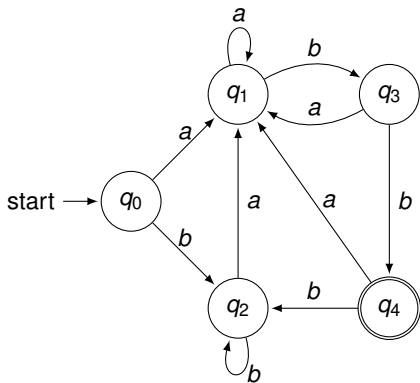
## DFA 최소화 알고리즘 예



입력	1			2	3
	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$
$a$	1	1	1	1	1
$b$	1	2	1	3	1

- $q_1$ 이  $b$ 를 읽고 다른 동치류로 이동하므로,  $\{q_0, q_2\}$ 와  $\{q_1\}$ 을 분할한다.

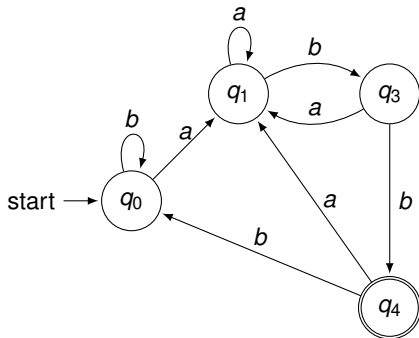
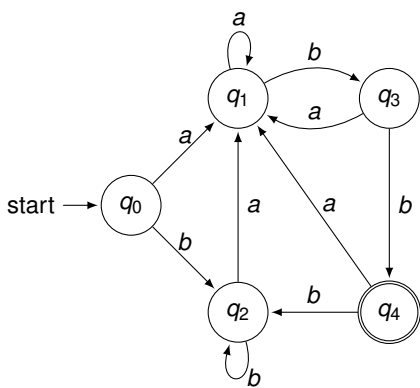
## DFA 최소화 알고리즘 예



입력	1		2	3	4
	$q_0$	$q_2$	$q_1$	$q_3$	$q_4$
$a$	2	2	2	2	2
$b$	1	1	3	4	1

- 더 이상 구분 가능한 동치류가 없음

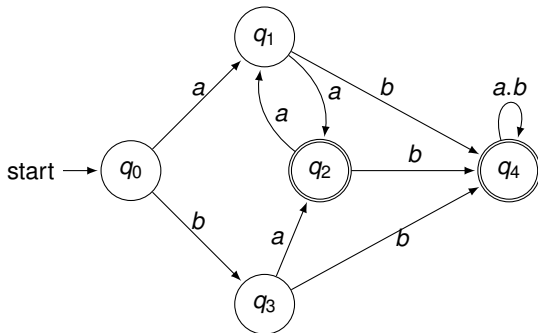
## DFA 최소화 알고리즘 예



최소화된 DFA

## DFA 최소화 알고리즘 예

- 다음 DFA를 최소화해보자.



## 정규 문법, 정규 표현식, FA의 동치 관계

아래의 방법을 통해 정규 문법, 정규 표현식, FA가 동치 관계임을 증명한다.

- 정규 문법을 정규 표현식으로 변환한다.
- 정규 표현식을 FA로 변환한다.
- FA를 정규 문법으로 변환한다.

즉 임의의 정규 언어가 주어졌을 때, 이것을 셋 중 어떤 것으로도 표현할 수 있다!

## 정규 문법을 정규 표현식으로 변환하기

- 정규 표현 방정식<sub>regular expression equation</sub>: 계수가 정규 표현으로 구성된 방정식
- 주어진 정규 문법을 정규 표현 방정식으로 바꾼 후, 방정식의 해를 구한다.

### Theorem (아덴 정리<sub>Arden's theorem</sub>)

$\alpha, \beta$ 가 정규 표현식일 때,  $\alpha$ 가  $\varepsilon$ 을 포함하지 않는다면  $X = \alpha X + \beta$ 의 유일한 해는  $X = \alpha^* \beta$ 이다.<sup>2</sup>

### 증명

$$\begin{aligned} X &= \alpha X + \beta \\ &= \alpha(\alpha^* \beta) + \beta \\ &= \alpha^+ \beta + \beta \\ &= (\alpha^+ + \varepsilon) \beta \\ &= \alpha^* \beta \text{ (해가 유일하다는 증명은 생략)} \end{aligned}$$

<sup>2</sup><https://web.archive.org/web/20110708171054/https://www.cs.cmu.edu/cdm/pdf/KleeneAlg.pdf>



## 정규 문법을 정규 표현식으로 변환하기 2

정규 문법  $G = (V_N, V_T, P, S)$ 이 입력으로 주어졌다고 가정하자.

- 알고리즘의 순서는 다음과 같다.
  - 정규 문법으로부터 정규 표현 방정식을 만든다.
  - 정규 표현 방정식 중  $X = \alpha X + \beta$  꼴의 식을 찾아 모두  $X = \alpha^* \beta$ 로 변환한다.
  - 다른 방정식에 새롭게 구해진 정규 표현식을 대입하여 시작 기호에 해당하는 정규 표현식을 찾는다.
  - 시작 기호에 대해 정규 표현식을 찾게 되면, 해당 정규 표현식이 기존 정규 문법이 생성하던 언어를 표현하게 된다.

## 정규 문법을 정규 표현식으로 변환하는 예

다음 정규 문법  $G$ 를 정규 표현식으로 변환해보자.

$$G = (\{S, A, B\}, \{a, b\}, P, S),$$

$$P = \{S \rightarrow aA \mid bS, A \rightarrow aS \mid bB, B \rightarrow aB \mid bB \mid \varepsilon\}.$$

### 1. 생성 규칙을 정규 표현 방정식으로 변환

$$S = aA + bS, A = aS + bB, B = aB + bB + \varepsilon = (a + b)B + \varepsilon$$

### 2. 아덴 정리에 따라, $B = (a + b)^* \varepsilon = (a + b)^* 0$ 이 된다.

### 3. 구해진 정규 표현식을 다른 방정식에 대입한다.

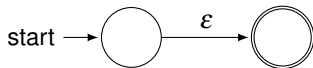
$$\begin{aligned} S &= aA + bS \\ &= a(aS + bB) + bS \\ &= aaS + abB + bS \\ &= (aa + b)S + ab(a + b)^* \end{aligned}$$

## 정규 표현식을 FA로 변환하기

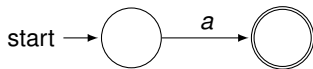
- 정규 표현식이 재귀적<sub>recursively</sub>으로 정의되었음을 기억하자.
- 정규 표현식을 더 큰 정규 표현식으로 조립할 때의 경우의 수만큼만 고려하면 된다.
- 공집합  $\emptyset$ 을 인식하는 FA:



- 빈 문자열  $\epsilon$ 을 인식하는 FA:

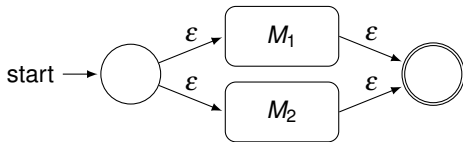


- 기호  $a \in \Sigma$ 를 인식하는 FA:

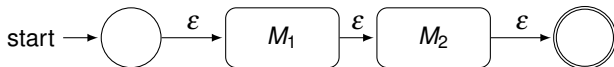


## 정규 표현식을 FA로 변환하기 2

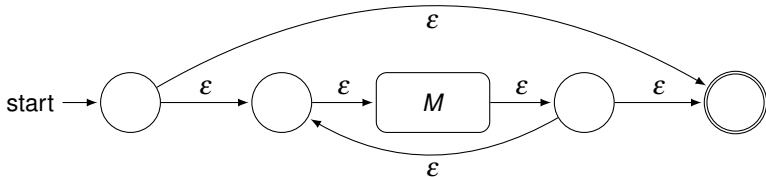
- FA  $M_1$ 과  $M_2$ 가 주어졌을 때,  $L(M_1) + L(M_2)$ 를 인식하는 FA:



- FA  $M_1$ 과  $M_2$ 가 주어졌을 때,  $L(M_1)L(M_2)$ 를 인식하는 FA:



- FA  $M$ 이 주어졌을 때,  $L(M)^*$ 를 인식하는 FA:

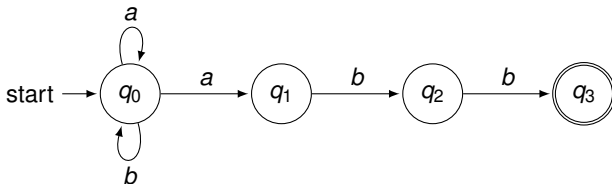


## 정규 표현식을 FA로 변환하기 예

- 정규 표현식  $(a+b)^*$ 를 인식하는 FA를 구성해보자.
- 정규 표현식  $(a+b)^*a$ 를 인식하는 FA를 구성해보자.

## FA를 정규 문법으로 변환하기

- FA를 정규 문법으로 변환하는 것은 매우 간단하다.
- FA  $M = (Q, \Sigma, \delta, q_0, F)$ 가 주어졌다고 가정하자.
  1.  $V_N = Q$ ,  $V_T = \Sigma$ ,  $S = q_0$ 로 둔다.
  2.  $\delta(q, a) = p$  꼴의 상태 전이마다  $q \rightarrow ap$  꼴의 생성 규칙을  $P$ 에 추가한다.
  3. 최종 상태 집합에 속하는  $q \in F$ 에 대해,  $q \rightarrow \varepsilon$  생성 규칙을 추가한다.
- 아래의 예를 통해 연습해보자.



Any questions?

- A. Aho, J. Ullman, R. Sethi, M. S. Lam, Compilers: Principles, Techniques, and Tools (2nd Edition), Addison Wesley, 2006
- R. Sebesta, Concepts of Programming Languages, 5th Edition, Addison-Wesley, 2001
- K. C. Loudon, Compiler Construction: Principles and Practice, Cengage Learning, 1997
- K. C. Loudon and K. A. Lambert, Programming languages : Principles and Practice, 3rd Edition, Cengage Learning, 2012
- 박두순, 컴파일러의 이해, 한빛아카데미, 2016
- 김종훈, 김종진, 프로그래밍 언어론 : 쉽게 배우는 언어의 원리와 구조, 한빛미디어, 2013