

(개요) Regular expression(정규식)은 **regular 언어**를 표현하는(denote) 식(expression)이다.

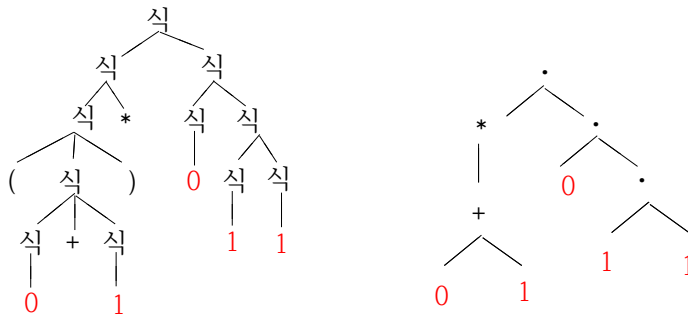
2장에서 DFA를 이용하여 Regular 언어를 정의하고, DFA를 확장하여 **확장된 오토마타**까지 정의하였다. 3장에서는 regular expression(정규식)을 정의하고 regular expression이 regular 언어를 표현하는 또 다른 방법이라는 것을 **증명**한다.

오토마타가 상태 변환을 통해 언어를 **간접적**으로 정의함에 비해, regular expression은 정의하는 언어를 **직접** 식으로 표현하므로 직관적으로 그 식이 정의하는 언어를 이해하기 쉬운 장점이 있다.

(정의 1) Regular expression은 기본문자  $\Sigma$ 에서(over) 정의한다.

- Basis
- (1) 기본문자( $a \in \Sigma$ )는 식(regular expression)이다.
  - (2)  $\epsilon$ 은 식(regular expression)이다.
  - (3)  $\emptyset$ 은 식(regular expression)이다.
- Recursion
- (1) 식 + 식은 식(regular expression)이다.
  - (2) 식 식은 식(regular expression)이다.
  - (3) 식\*는 식(regular expression)이다.
  - (4) ( 식 )은 식(regular expression)이다.

(예 1) Regular Expression  $(0+1)^*011$  over  $\{0, 1\}$ 에 parser tree<sup>1)</sup>와 AST<sup>2)</sup>



(정의 2) 언어와 언어의 연산(binary operation), 합연산(union: +), 곱연산(concatenation:  $\cdot$ ), 반복곱(exponent:  $^n$ ), 반복곱의 합(closure:  $^*$ )을 아래와 같이 정의한다.

$$+, \cdot : 2^{\Sigma^*} \times 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}, \quad n, * : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}.$$

$L_1, L_2, L \in 2^{\Sigma^*}$  라 하자.

$$L_1 + L_2 \stackrel{\text{def}}{=} L_1 \cup L_2.$$

$$L_1 \cdot L_2 \stackrel{\text{def}}{=} \{xy \in \Sigma^* \mid x \in L_1, y \in L_2\}.$$

$$L^* \stackrel{\text{def}}{=} \bigcup_{i \in \mathbb{N}_0} L^i \quad \text{단}$$

$$L^n \stackrel{\text{def}}{=}_{\text{B}} \{\epsilon\} \quad \text{for } n = 0,$$

1) Parse tree는 제 5장 context-free grammar에서 다룬다.

2) Abstract Syntax Tree(AST)는 Parse tree에서 일부 sub-tree와 leaf들을 제거한 부분나무이다.

$$L^n \stackrel{\text{R}}{=} L \cdot L^{n-1} \quad \text{for } n \geq 1.$$

(정의 3)  $\Sigma$ 에서 정의된 regular expression  $E$ 가 표현하는(denote) 하는 언어,  $L(E) \subseteq \Sigma^*$ .

- Basis
- (1) 정규식 기본문자( $a \in \Sigma$ )는 언어  $\{a\}$ 를,
  - (2) 정규식  $\epsilon$ 은 언어  $\{\epsilon\}$ 를,
  - (3) 정규식  $\emptyset$ 은 언어  $\{\}$ 를 각각 나타낸다.
- Recursion
- (1) 정규식  $E_1 + E_2$ 는 언어  $L(E_1) \cup L(E_2)$ 를,
  - (2) 정규식  $E_1 E_2$ 는 언어  $L(E_1) \cdot L(E_2)$ 를,
  - (3) 정규식  $E^*$ 는 언어  $L(E)^*$ 를,
  - (4) 정규식  $(E)$ 는 언어  $L(E)$ 를, 각각 나타낸다.

(예 2) Regular Expression  $(0+1)^*011$  over  $\{0, 1\}$

$$\begin{aligned} L((0+1)^*011) &= L((0+1)^*) \cdot L(011) \\ &= L((0+1)^*) \cdot L(0)L(11) \\ &= (L(0) + L(1))^* \cdot \{0\} \cdot L(1) \cdot L(1) \\ &= (L(0) \cup L(1))^* \cdot \{0\} \cdot \{1\} \cdot \{1\} \\ &= (\{0\} \cup \{1\})^* \cdot \{011\} \\ &= \{0, 1\}^* \cdot \{011\} \end{aligned}$$

(부분정리 1) 임의의 DFA가 받아들이는(accept) 언어와 같은 언어를 표현하는(denote) regular expression이 존재한다. (DFA  $\Rightarrow$  RE)

(증명) 특정 명제를 증명하기 위하여, 명제를 더 일반화된 형태로 확장하면 더 쉽게 증명할 수 있다. DFA 상태가  $n$ 개라면 각 상태에 자연수로 번호를 붙인다. 즉  $Q = \{q_1, q_2, \dots, q_n\}$ . 이때 상태  $q_i$ 에서 시작하여 상태  $q_j$ 에서 끝나는 언어  $L_{ij}$ 를 정의한다.

$$L_{ij} = \{x \in \Sigma^* \mid \delta^*(q_i, x) = q_j\}$$

1과  $n$ 사이의 모든  $i$ 와  $j$ 에 관하여( $1 \leq \forall i, j \leq n$ ),  $n^2$ 개의 언어  $L_{ij}$ 를 표현하는 regular expression  $E_{ij}$ ,  $n^2$ 개를 모두 찾을 수 있다면, 우리가 원하는 regular expression은  $E_{\text{시작}, \text{최종}_1} + E_{\text{시작}, \text{최종}_2} + \dots + E_{\text{시작}, \text{최종}_k}$ 로 표현할 수 있으므로 우리는 이 문제를 풀었다고 할 수 있다.

상태  $q_i$ 에서 시작하여 상태  $q_j$ 에서 끝나는 언어  $L_{ij}$ 를 표현하는 regular expression  $E_{ij}$ 를 찾는 방법(algorithm, proof)을 생각하여 보자. 상태  $q_i$ 에서 상태  $q_j$ 사이에 방문하는 중간 상태에 아무 제한이 없으면  $L_{ij}$  혹은  $E_{ij}$ 를 직접 구하기는 어렵다. 그래서 상태  $q_i$ 에서 상태  $q_j$ 사이에 방문하는 중간상태에 제한을 준다. 즉 특정 수  $k$ 보다 작거나 같은 수의 번호를 가진 상태만 중간상태로 허용하는 제한을 준다.

$$L_{ij}^k = \{x \in \Sigma^* \mid \delta^*(q_i, x) = q_j, x = a_1 a_2 \dots a_m\}$$

3)  $\cdot$ 은 associative하므로  $L^n \stackrel{\text{R}}{=} L^{n-1} \cdot L$ 으로 정의하여도 무관하다. 이를 실제로 확인하여 보라.

$$1 \leq \forall l < m : \delta^*(q_i, a_1 \cdots a_l) = q_p \Rightarrow p \leq k \}$$

즉  $L_{ij}^0$ 는 중간상태를 허용하지 않고 상태  $q_i$ 에서 상태  $q_j$ 로 직접 가는 경우이므로 상태변화함수  $\delta(q_i, a) = q_j$ 를 만족하는  $a$ 를  $L_{ij}^0$ 가 가진다.

$$L_{ij}^0 \stackrel{\text{B}}{=} \{a \mid \delta(q_i, a) = q_j\}$$

우리가 마지막에 구하고 싶은  $L_{ij}$ 는 DFA 상태가  $n$ 개라면  $L_{ij}^n$ 이다.

초기조건과 최종조건을 구하였으므로  $L_{ij}^{k-1}$ 를 이용하여  $L_{ij}^k$ 를 구하는 recursive 과정만 해결하면 된다.

$$L_{ij}^k \stackrel{\text{R}}{=} L_{ij}^{k-1} \cup L_{ik}^{k-1} \cdot (L_{kk}^{k-1})^* \cdot L_{kj}^{k-1} \quad \text{단 } 1 \leq k \leq n.$$

상태  $q_i$ 에서 상태  $q_j$ 로 가면서 중간상태로  $k$ 보다 작거나 같은 수의 번호를 가진 상태만 허용하는 언어  $L_{ij}^k$ 는 (1) 상태  $q_i$ 에서 상태  $q_j$ 로 가면서 중간에  $k-1$ 보다 작거나 같은 수의 번호를 가진 상태만 허용하거나( $L_{ij}^{k-1}$ ) 또는(∪) (2) i) 상태  $q_i$ 에서 새로 허용된 중간 상태  $q_k$ 로 가면서 중간에  $k-1$ 보다 작거나 같은 수의 번호를 가진 상태만 허용하고(·), ii) 상태  $q_k$ 에서 상태  $q_k$ 로 다시 돌아오면서 중간에  $k-1$ 보다 작거나 같은 수의 번호를 가진 상태만 허용하는 일을( $L_{kk}^{k-1}$ )를 여러 번( $*$ )허용( $(L_{kk}^{k-1})^*$ )하고(·), iii) 상태  $q_k$ 에서 상태  $q_j$ 로 가면서 중간에  $k-1$ 보다 작거나 같은 수의 번호를 가진 상태만 허용한다.

(부분정리 2) 임의의 regular expression이 표현하는(denote) 언어와 같은 언어를 받아들이는(accept) 오토마타가 존재한다. (RE  $\Rightarrow$   $\epsilon$ -NFA)

(증명) Regular expression을 (정의 1)에서 recursive하게 정의하였으므로, 그 정의에 그대로 오토마타를 정의하면 된다.

Basis 1) 정규식  $a \in \Sigma$ 가 표현하는 언어  $\{a\}$ 를 받아들이는 오토마타  
 2) 정규식  $\epsilon$ 가 표현하는 언어  $\{\epsilon\}$ 를 받아들이는 오토마타  
 3) 정규식  $\emptyset$ 가 표현하는 언어  $\{\}$ 를 받아들이는 오토마타

Recursion 1) 정규식 식 + 식이 표현하는 언어  $L(\text{식}) \cup L(\text{식})$ 을 받아들이는 오토마타  
 2) 정규식 식 식이 표현하는 언어  $L(\text{식}) \cdot L(\text{식})$ 을 받아들이는 오토마타  
 3) 정규식 식\*이 표현하는 언어  $L(\text{식})^*$ 을 받아들이는 오토마타  
 4) 정규식 ( 식 )이 표현하는 언어  $L(\text{식})$ 을 받아들이는 오토마타

구체적인 내용은 교과서 혹은 강의 TP 참조.

(중요정리 1) 다음 명제는 모두 논리적으로 같다.

- (1) 언어  $L$ 은 regular이다.
- (2) 언어  $L$ 을 받아들이는(accept) 오토마타가 존재한다.
- (3) 언어  $L$ 을 표현하는(denote) regular expression이 존재한다.