Context-Free Grammar

5.1 Rewriting System(**고쳐 쓰기 틀**)

(**정의** 5.1) A가 기본문자집합(vocabulary)이고, 관계 $P \subseteq A^* \times A^*$가 집합 A^*에서 정의된 이진 관계(binary relation)일 때 순서쌍 $R = (A, P)$를 기본문자 A에 관한 **고쳐 쓰기 틀**(rewriting system)이라고 부른다. 이 때 관계 P를 고쳐 쓰기의 production rule(규칙)이라고 부르고, $\alpha, \beta \in A^*$에 대해 $(\alpha, \beta) \in P$, $\alpha \; P \; \beta$ 또는 $\alpha \to \beta \in P$로 쓰고 α를 규칙의 좌변(Lefthand side; LHS) β를 규칙의 우변(Righthand side; RHS)이라 각각 부른다.

(**정의** 5.2) **고쳐 쓰기**(유도; rewriting, derivation)
고쳐 쓰기 틀 $R = (A, P)$에서 규칙 $\alpha \to \beta \in P$에 대하여, $\gamma, \delta \in A^*$가 임의의 문자열일 때, $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$로 쓰고 문자열 $\gamma\alpha\delta$를 규칙 $\alpha \to \beta \in P$를 써서 $\gamma\beta\delta$로 **다시 썼다**(rewrite; derive)고 하고, 특별히 **다시 쓴** 규칙 $r = \alpha \to \beta \in P$를 표현하고 싶을 때는 $\gamma\alpha\delta \Rightarrow^r \gamma\beta\delta$로 표현한다.

임의의 문자열의 부분문자열(substring)이 규칙 $\alpha \to \beta \in P$의 좌변 α과 같으면 나머지 부분(prefix γ과 suffix δ)을 제외한 α만 규칙의 우변 β로 바꾸어서 **다시 쓸**($\gamma\alpha\delta \Rightarrow^{\alpha\to\beta} \gamma\beta\delta$) 수 있다.

다시 쓰는 관계가 가진 기본문자집합을 일부 분류하여 다음과 같이 **문법**을 정의한다.
(**정의** 5.3) 다시 쓰기 틀 $R = (N \cup T, P)$에서 **문법** $G = (N, T, P, S)$을 아래로 정의한다.
i) N은 넌 터미널(nonterminal; variable) 문자에 집합,
ii) T는 터미널(terminal) 문자에 집합, 단 $N \cap T = \varnothing$,
iii) 규칙 P는 $(N \cup T)^*$에서 정의된 관계이고, 특히 문법규칙이라고도 부른다.
iv) $S \in N$은 시작문자(start symbol).

문법은 디시 쓰기 틀에서 문장(sentence)에 구분을 위하여 기본문자를 **터미널**과 **넌 터미널**로 나누고, 문장 다시 쓰기의 **시작**을 위하여 시작문자 S를 별도로 정의한 것이다.

(**정의** 5.4) 문법 $G = (N, T, P, S)$에 **언어** $L(G)$를 아래로 정의한다.
$$L(G) = \{x \in T^* \mid S \Rightarrow^* x\}.$$

문법에 새로 쓰기 중간과정에서 나타나는 문자열을 **문장형태**(sentential form)라고 부르고 문장형태가 터미널 문자만으로 이루어 졌을 때 **문장**(sentence) 된다.

5.2. 문맥자유 문법

(**정의** 5.5) **문맥자유(context-free) 문법(grammar)**

(정의 5.4)에서 문법규칙에 **좌변**이 **넌 터미널 문자**로 제한된다. 즉 $A \rightarrow \alpha \in P$에서 $A \in N$이고 $\alpha \in (N \cup T)^*$이다.

(**정의** 5.6) 문맥자유(context-free) 언어(language)

임의의 언어 L을 만들어내는 **문맥자유문법** G가 있을 때, $L = L(G)$, 언어 L을 **문맥자유언어**라 부른다.

5.3 문맥자유문법과 파스트리 그리고 고쳐 쓰기순서

$S \Rightarrow^{\pi} x \in T^*$이고 $\pi \in P^*$일 때, 사용된 문법규칙으로 subtree를 만들면 전체 문장 $x \in T^*$에 대한 트리를 만들 수 있고, 이를 파스트리라 부른다.

문맥자유문법의 파싱(parsing)은 임의의 터미널 문자열 $x \in T^*$가 주어진 문법 G에 문장이면 해당하는 파스트리를 만들어주고, 문장이 아니면 이를 알려주는 과정이다.

문맥자유문법의 **문장형태**는 일반적으로 여러 개에 넌 터미널을 가지는데, 어떤 넌 터미널이 다시 쓰기에 먼저 참여할까 하는 것은 파스트리에 최종모양과는 관계가 없다. 따라서 문장형태에 가장 왼쪽에 있는 넌 터미널부터 다시 쓰기를 하는 왼쪽부터 고쳐쓰기(leftmost derivation)과 오른쪽부터 고쳐쓰기(rightmost derivation)에 두 가지 극단적인 경우를 생각할 수 있다. 이를 각각 $\Rightarrow_{lm}$와 $\Rightarrow_{\mathrm{rm}}$로 표시한다.

$$S \Rightarrow_{lm}^{\pi_1^L} xA\gamma \Rightarrow_{lm} x\beta\gamma \Rightarrow_{lm}^{\pi_2^L} xy\gamma \Rightarrow_{lm}^{\pi_3^L} xyz, \qquad \pi_1^L, \pi_2^L, \pi_3^L \in P^*,$$

단 $x \in T^*,\ \gamma \in (N \cup T)^*,\ A \rightarrow \beta \in P,\ y, z \in T^*,\ \beta \Rightarrow^* y,\ \gamma \Rightarrow^* z.$

$$S \Rightarrow_{\mathrm{rm}}^{\pi_1^R} \alpha Az \Rightarrow_{\mathrm{rm}} \alpha\beta z \Rightarrow_{\mathrm{rm}}^{\pi_2^R} \alpha yz \Rightarrow_{\mathrm{rm}}^{\pi_3^R} xyz, \qquad \pi_1^R, \pi_2^R, \pi_3^R \in P^*,$$

단 $z \in T^*,\ \alpha \in (N \cup T)^*,\ A \rightarrow \beta \in P,\ y, x \in T^*,\ \beta \Rightarrow^* y,\ \alpha \Rightarrow^* x.$

이 때 $\pi_L = \pi_1^L(A \rightarrow \beta)\pi_2^L\pi_3^L \in P^*$라 하고 $\pi_R = \pi_1^R(A \rightarrow \beta)\pi_2^R\pi_3^R \in P^*$라 하면 π_L과 π_R를 각각 문장 x에 대한 left parse와 right parse라고 부른다. 문장 x에 대한 left parse나 right parse가 정해지면 해당하는 파스트리도 정해진다.