

Syntax

2022년 3월 8일 화요일 오후 3:33

syntax

프로그램 구문의 정형적 정의 -> Grammar로 정의

Kleen star : L에서 만들수있는 모든길이의 string들을 합친것이다 empty포함 $L^* = L^0 \cup L^1 \cup L^2 \dots$

Dager: kleen star - empty

-> non - empty $L^t = L^0 \cup L^1 \cup \dots$

Grammer

- 언어의 syntax를 정의하는 언어
- 프로그래밍 언어 정의에는 context free grammar 사용

G = (Vn, Vt, P, S)

Grammer는 4가지 요소로 표현 가능.

4가지 요소로 이루어져 있다

Vn : nonterminal symbols의 유한집합

문자열 생성시 사용되는 중간과정의 기호, 보통 대문자
non terminal은 언어에 대한 계층 구조 부여 가능

대문자는 대표적인 non-terminal이다. (ex A,B, C....)

Vt: terminal symbols의 유한집합

언어의 정의된 기호로, 영어 소문자, 아라비아 숫자, 연산자, 기호등을 포함

소문자는 대표적인 Terminal Symbols이다 (ex a,b,c..)

$Vn \cap Vt = \text{공집합}$

Non-terminal과 terminal의 교집합은 공집합

P : productions -> rule set

S : start symbol non terminal symbol이다

R, S은 텔레스 있다

Greek 문자들은 kleen star of V에 속하는 string

$$G = \{\{V_N\}, \{V_T\}, P, S\}$$

Derivation

$\alpha \rightarrow \beta$ 규칙이 있고 $\gamma, \delta \in V^*$
즉 α, β 가 모든 step에서 포함된다면
 $\alpha \& \delta$ 는 $\beta \delta$ 를 생성할 수 있다

⇒ 직접 유도 : 생성 규칙을 적용한다는 의미

if $\alpha \rightarrow \beta \in P$ and $\gamma, \delta \in V^*$

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

→ 은 Rule 정의 \Rightarrow 은 생성 규칙 적용 (derivation rule)을 의미



Kleen Star 간접 유도

: 직접 유도를 0번 또는 그 이상의 step으로 하는 것을 의미한다

↳ Dagest 간접 유도는 적어도 한번 이상 직접 유도를 한다

Leftmost derivation : 가장 왼쪽 symbol 치환하면서 진행

Rightmost derivation : 가장 오른쪽 symbol 치환하면서 진행

밑에서 Example로 확인 가능

Generation

Grammar

generation
← derive

Language

만약 a_1 이 a_2 를 간접 유도 했다면 a_1 은 a_2 를 생성했다고 한다 If $a_1 \xrightarrow{*} a_2$ 면 a_1 은 a_2 를 생성한 것

| Grammar는 Language를 생성하고

| Language는 Grammar를 degin한다

ex) $G_1 = (V_n, V_T, P, S)$ $V_n = \{B\}$, $V_T = \{1, 0\}$ $P : B \rightarrow 0 \quad | \quad B \rightarrow 1 \quad | \quad B \rightarrow B0 \quad | \quad B \rightarrow B1$

문법 G_1 은 String 0, 1을 NGTung 즉 $B \rightarrow 0, B \rightarrow 1$

그럼 G_1 은 어떤 String NGTung? $G_1 = \{0, 1, 00, 01, \dots\} =$ 모든 binary 집합

ex) $G_1 \xrightarrow{*} 010$ 을 증명하라

010 = 0 + 1 + 0

$B \rightarrow B0$

$\Rightarrow B10$

$\Rightarrow 010$

적용

Equivalent Grammars 동치

생성하는 결과물 같으면 동치라고 본다

$$L(G_1) = L(G_2) \Leftrightarrow G_1, G_2$$

문법 G 가 생성하는 언어 $L(G)$

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

$L(G)$ 는 G 가 생성하는 모든

Terminal String 집합이다.

Grammar 내용 7(1속...)

ex) G_1 이 $G_1 : S \rightarrow a | aS$ 일 때 $L(G_1)$ 은 무슨 언어?

S 가 생성하는 String? $S \rightarrow a$

$$S \Rightarrow aS \Rightarrow aa$$

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaa \quad \Rightarrow L(G_1) = \{a^n \mid n > 0\}$$

ex) $G_2 = (S, I, D, \{0, 1, \dots, 9\}, P, I)$

$$P : I \rightarrow D \cup ID$$

$$D \rightarrow 0 \cup 1 \cup 2 \cup \dots \cup 9$$

$L(G_2)$ 는 무슨 언어?

$$I \Rightarrow D$$

$$I \Rightarrow ID \Rightarrow IID \Rightarrow IIDD \dots$$

$$I \Rightarrow DD$$

$$L(G_2) = \{0^1 1^1 2^1 3^1 \dots 9^1\}^n \mid n > 0\}$$

$$265 \in L(G_2)$$

rightmost

$$I \Rightarrow ID \Rightarrow IS \Rightarrow ID5 \Rightarrow IIS \Rightarrow D65 \Rightarrow \underline{265}$$

leftmost

$$I \Rightarrow ID \Rightarrow IDID \Rightarrow IDID \Rightarrow 2ID \Rightarrow 25D \Rightarrow \underline{256}$$

Type of Grammars

문법 생성 구조의 형태에 따라 문법

1. Type 0: No restrictions

2. Type 1 : Context sensitive grammar (CSG)

a. 특정 문맥에서만 변환 가능 $\alpha \underline{A} \beta \rightarrow \alpha \underline{\gamma} \beta$

특정 문맥 이 경우 K와 B지만

A를 γ로 변환 가능

3. Type 2 : Context free grammar

a. 문맥 관계 없이 언제나 변환 가능

4. Type 3 : Regular grammar

a. String이 한쪽 방향으로만 자랄수 있다

Type 1,2,3 문법으로 만든 language가 따로 있다

Classifications of Language Example

- Simple matching language CFL

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

- Double matching lanuage CSL

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

- Mirror image language CFL

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

- Palindrome language CFL

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

- Parenthesis language CFL

$$L_p = \{ w \mid w : \text{balanced parentheses} \}$$

