# 3. 구문론

## QnA

- 구문법(Syntax)
  - 문장 혹은 프로그램을 작성하는 방법
  - 자연어(연어, 한국어)의 문법처럼 프로그래밍 언어의 구문법이 있다.
  - 프로그래밍 언어의 이론적 기초
- 질문
  - 어떤 언어의 가능한 문장 혹은 프로그램의 개수가 무한하지 않나요?
  - 무한한 것들을 어떻게 유한하게 정의할 수 있나요?

## 재귀적 정의: 이진수의 구문법

- 숫자(D)는 0, 1 중 하나이다.
- 이진수 구성 방법
  1. 숫자(D)는 이진수(N)이다.
  2. 이진수(N) 다음에 숫자(D)가 오면 이진수(N)이다.
- 논리 규칙 형태

$$\frac{D\text{는 숫자이다}}{D\text{는 이진수}N\text{이다}} \qquad \frac{N\text{이 이진수이고 }D\text{가숫자이다}}{ND\text{는 이진수이다}}$$

- 문법 형태

```
// N → D , N → ND
N → D
  | ND
```

- 이진수: 구문법과 의미론

```
D → 0      // V('0') = 0
  | 1      // V('1') = 1

N → D      // V(D)
  | ND     // V(ND) = V(N)*2 + V(D)

101        // V('101') = V('10')*2 + V('1') = 2*2 + 1 = 5
           // V('10') = V('1')*2 + V('0') = 2
```

- 십진수: 구문법과 의미론

```
D → 0      // V('0') = 0
  | 1      // V('1') = 1
  | 2      // V('2') = 2
  ...
  | 9      // V('9') = 9

D → D      // V(D)
  | ND     // V(ND) = V(N)*10 + V(D)
```

## 수식의 구문법

- 수식
  - 5, 5 + 13, 5 + 13 + 4, 5 *13 + 4, (5 + 13)* 12, ...
- 구문법: 쓰는 방법

```
E → E * E
  | E + E
```

```
   | (E)
   | N
N → ND | D
D → 0|1|2|3|4|5|6|7|8|9
```

## 프로그래밍 언어의 구문 구조

- 프로그래밍 언어의 구문 구조를 어떻게 표현할 수 있을까?
  - **재귀를 이용한 구문법으로 정의**
- 문장 `S` 의 구문법
  - `id = E`
  - `if E then S else S`
  - `while E do S`
- 문맥-자유 문법(CFG: Context-free grammar)
  : 이러한 재귀 구조를 자연스럽게 표현할 수 있다.

```
S → id = E
   | if E then S else S
   | while E do S
```

## 문맥-자유 문법(CFG: Context-free grammar)

- 문맥-자유 문법 CFG는 다음과 같이 구성된다.
  - 터미널 심볼의 집합 $T$
  - 넌터미널심볼의 집합 $N$
  - 시작 심볼 $S$ (넌터미널 심볼 중에 하나)
  - 다음과 같은 형태의 생성(문법) 규칙들의 집합
    $X \to Y_1 Y_2 \dots Y_n$ 여기서 $X \in N$ 그리고 $Y_i \in T \cup N$
    $X \to \varepsilon$ (오른쪽이 빈 스트링인 경우)
  - 보통 **넌터미널 심볼은 대문자**로, **터미널 심볼은 소문자**로 표기한다.

> ✎ **예제**
>
> CFG에서 다음과 같은 문법이 있다고 가정합시다.
>
> ```
> S → aSb  |  ε
> ```
>
> 여기서:
>
> - **터미널 심볼**: a,ba, b
> - **넌터미널 심볼**: SS
> - **생성되는 문자열 예시**: ε,ab,aabb,aaabbb

## 유도(Derivation)

- 핵심 아이디어
  1. 시작 심볼 $S$부터 시작한다.
  2. 넌터미널 심볼 $X$를 생성규칙을 적용하여 $Y_1Y_2 \dots Y_n$ 으로 대치한다 .
  3. 이 과정을 넌터미널 심볼이 없을 때까지 반복한다.
- 생성 규칙 $X \to Y_1Y_2 \dots Y_n$ 적용
  - $X$ 를 $Y_1Y_2 \dots Y_n$ 으로 대치한다. 혹은
  - $X$ 가 $Y_1Y_2 \dots Y_n$ 을 생성한다.
- 터미널 심볼
  - 대치할 규칙이 없으므로 일단 생성되면 끝
  - 터미널 심볼은 그 언어의 토큰이다.
- 예
  - `S → aS | b`
  - `S => aS => aaS => aaaS => aaab`

- 직접 유도(Direct derivation) ⇒
  - 생성 규칙을 한 번 적용
  - 생성규칙 $X_i \rightarrow Y_1 Y_2 \ldots Y_n$이 존재하면
    - $X_1 \ldots X_i \ldots X_n \Rightarrow X_1 \ldots X_{i-1} Y_1 Y_2 \ldots Y_n X_{i+1} \ldots X_n$
- 유도(Derivation) ⇒*
  - 생성 규칙을 여러 번 적용
  - $X_1 \ldots X_n \Rightarrow \ldots \Rightarrow Y_1 \ldots Y_m$ 이 가능하면 $X_1 \ldots X_n \Rightarrow *Y_1 \ldots Y_m$

**좌측 유도(leftmost derivation)**

> 각 직접 유도 단계에서 가장 왼쪽 넌터미널을 선택하여 이를 대상으로 생성 규칙을 적용한다.

**우측 유도(rightmost derivation)**

> 각 직접 유도 단계에서 가장 오른쪽 넌터미널을 선택하여 이를 대상으로 생성 규칙을 적용하면 된다.

**유도 트리(Derivation tree)**

- 유도 과정 혹은 구문 구조를 보여주는 트리
- 유도 트리 = 파스 트리 = 구문 트리

**유도 예제**

- CFG

```
E → E * E (1)
 | E + E (2)
 | (E)   (3)
 | N     (4)
N → N D | D
D → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

- 생성할 스트링: 3 + 4 x 5
- 유도
  : E ⇒ E + E ⇒ N + E ⇒ D + E ⇒ 3 + E ⇒ 3 + E * E ⇒ ... ⇒ 3 + 4 x 5
- 3 + 4 + 5 유도 ?
- 파스트리

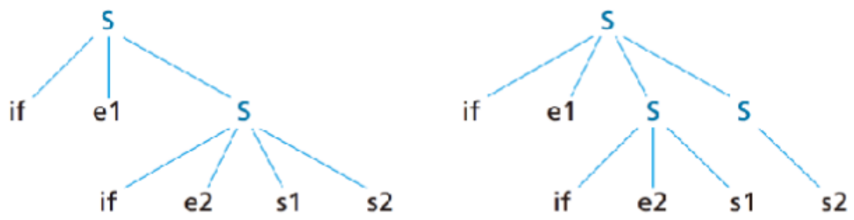
```
        E
       /|\
      E + E
     /   /|\
    3   E * E
        |   |
        4   5
```

## 모호성(Ambiguity)

- 모호한 문법 1
  - 어떤 스트링에 대해 **두 개 이상의 좌측 유도**를 갖는다.
  - 어떤 스트링에 대해 **두 개 이상의 우측 유도**를 갖는다.
  - 어떤 스트링에 대해 **두 개 이상의 파스 트리**를 갖는다.
- 모호한 문법 2

```
S → if E then
S | if E then S else S
```

- 이 문장에 대한 두 개의 파스 트리 `if e1 then if e2 then s1 else s2`




## 모호성 처리 방법 1

- 문법 재작성
    - 원래 언어와 같은 언어를 정의하면서 모호하지 않도록 문법 재작성
- 예

> 우선 순위를 적용하여 모호하지 않도록 재작성
> 수식은 여러 개의 항들을 더하는 구조이다.

```
E → E + T | T
T → T * F | F
F → N | (E)

// 3 + 4 * 5의 좌측 유도
E => E + T
  =>* N + T
  => 3 + T
  => 3 + T * F
  => 3 + F * F
  => 3 + N * F
  =>* 3 + 4 * N
  =>* 3 + 4 * 5
```

## 모호성 처리 방법 2

- 언어 구문 일부 변경

> 원래 언어와 약간 다른 언어를 정의하도록
> 언어의 구문을 일부 변경하여
> 모호하지 않은 문법 작성

```
S → if E then S end
  | if E then S else S

// 작성 예
if e1 then if e2 then s1 else s2 end
if e1 then if e2 then s1 end else s2
```