Chapter 5

문맥 자유 문법과 파싱 알고리즘 - **Part I**

목차

Overview

- **Parsing(=syntax analysis)**
  - 문장의 구성 요소를 낱낱이 분석
- **Context-Free Grammar (CFG)**
  - 문법(Grammar) → 구문(Syntax) → 생성 규칙(production rules)
- **Parsing Algorithm**
  - top-down parsing 과 bottom-up parsing

Table of Contents

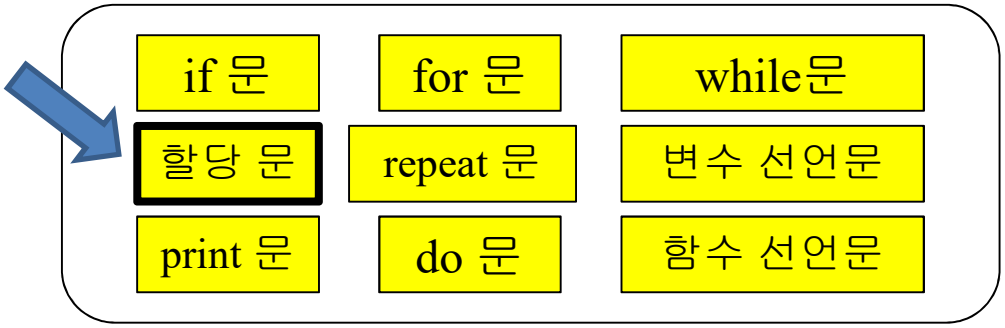
Parsing Process

입력 문장)

a = b + c; ⇒ ***token 1 = token 2 + token 3;***

어떤 문장인가?
(구문 선택)

선택된 구문과
입력 문장을 비교



구문(문장 구조)은 **grammar** 에 정의되어 있음

LHS = RHS ;

Parsing
(구문 트리 생성)

9-5+2 : Is it *right* or *wrong* ?

- **What are the *tokens*?**

  9, 5, 2

  +, -

- **We need <u>specific rules</u>.**

  list ➔ list + digit

  list ➔ list – digit

  list ➔ digit

  digit ➔ 0 | 1 | 2 | … | 9

- **What does the above *rules* mean?**

A set of specific rules = Grammar

- **Specific Rules (called grammar)**

  list → list + digit

  list → list – digit

  list → digit

  digit → 0 | 1 | 2 | … | 9

list → list + digit | list – digit | digit
digit → 0 | 1 | 2 | … | 9

Single rule = one possible structure of a Nonterminal

- **Rules**

  list → list + digit

  list → list – digit

  list → digit

  digit → 0 | 1 | 2 | … | 9

  list → list + digit

  list can have the form of list + digit

  That is, list can be replaced with list + digit

Choose the correct rule for the input.

- **Do check its syntactical correctness.**

  If 9-5+2 were a *list*, then

  9-5 ⇒ *list* and **2** ⇒ *digit*

  It's working. Why?

list ➔ list + digit | list – digit | digit
digit ➔ 0 | 1 | 2 | … | 9

Choose the incorrect rule for the input.

- **Do check its *syntactical correctness*.**

  If 9-5+2 were a *list*, then

  9 ⇒ *list* and **5+2** ⇒ *digit*

  It's not possible. Why?

list ➔ list + digit | list – digit | digit
digit ➔ 0 | 1 | 2 | … | 9

Parsing = Syntax analysis

- **Thus, we can check its *syntactical correctness*.**

  If 9-5+2 were a *list*, then

  9-5 ⇒ *list* and 2 ⇒ *digit*

  If 9-5 were a *list*, then

  9 ⇒ *list* and 5 ⇒ *digit*

  If 9 is a *list*, then 9 ⇒ *digit*

  list ➔ list + digit
  list ➔ list – digit
  list ➔ digit
  digit ➔ 0 | 1 | 2 | … | 9

- **This process is called *parsing* (*syntax analysis*)**

Table of Contents

자연 언어 vs 인공언어

- **자연 언어(*natural language*) : *informal method***
  - **"넘어져서 다리를 다쳤어"**
    - **정보가 정확하지 않음 ➔ 어쩌다가? 많이 다쳤니? 언제 그랬어? . . .**
    - **Ambiguous** (애매/모호) → **Double Meaning** (이중 의미)

- **인공 언어(*artificial language*) : *formal method***
  - **5W1H principles** (누가, 언제, 어디서, 무엇을,어떻게, 왜)
    - **나는 어제 집 앞에서 자전거를 타다가 한 눈을 팔다 넘어져서 다리를 다쳤어**
  - **Well-Formed Formula**
    - *Programming language = formal language*

Context Free Grammar (CFG) (1/2)

CFG는 4가지 요소를 갖고 있다.

$G = \{T, N, S, P\}$

1. T : **A set of *terminals*** (or **tokens**)
   atomic symbols of the language
   English : *a*, *b*, *c*, ..., *z*
   CFG : **( ) + - * number**

2. N : **A set of *nonterminals***
   variables denoting language constructs
   English : *Noun*, *Verb*, *Adjective*, *Adverb*, . . .
   CFG : **exp, op**

```
exp → exp op exp | ( exp ) | number
 op → + | - | *
```

Context Free Grammar (2/2)

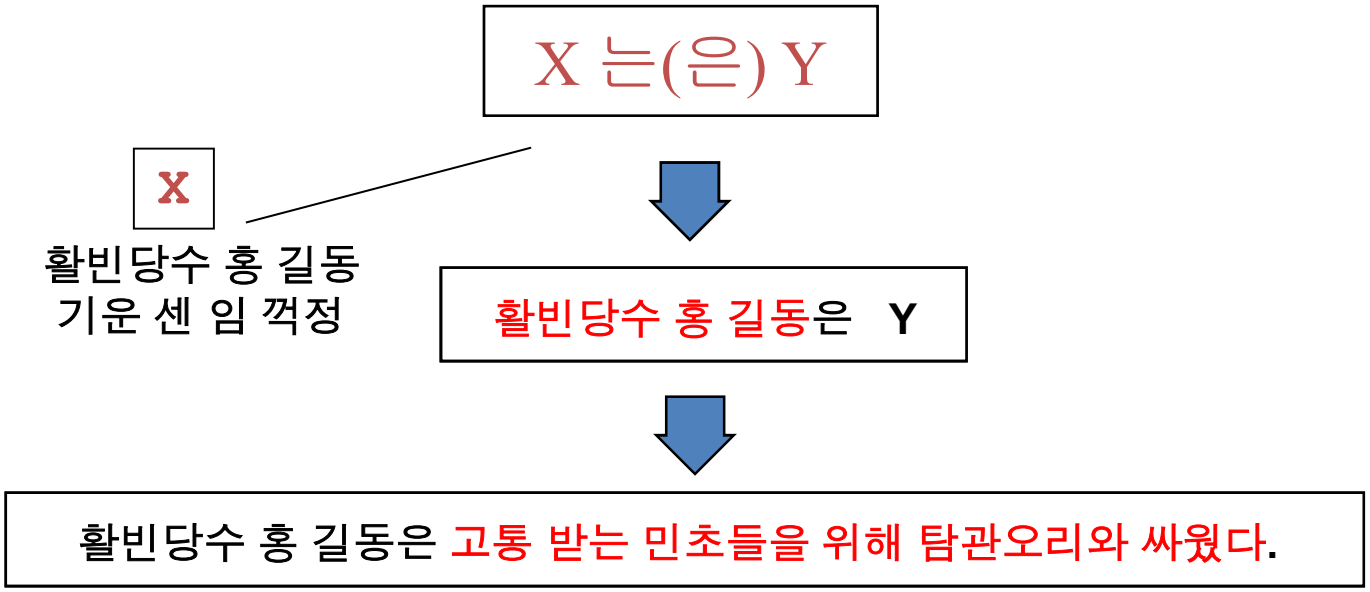
3. P : **A set of rules called *productions***
   for generating expressions of the language.
   nonterminal ::= a *string* of terminals and nonterminals

   English : *Sentence* ::= *Noun* *Verb* *Noun*
   CFG : `exp → exp op exp | ( exp ) | number`

4. $S \in N$: **A nonterminal chosen as *the start symbol***
   represents the main construct of the language.
   English : *Sentence*
   CFG : ***exp***

```
exp → exp op exp | ( exp ) | number
 op → + | - | *
```

비단말 기호(Nonterminal)란?

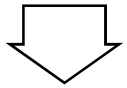


Nonterminals : 가능한 내용을 대표하는 상징적 기호. 예: **사람**
Terminals : 구체적 사실. 바뀔 수 없음. 예: **홍 길동**
Sentential form : 완전한 문장이 아님 ➔ Nonterminal 기호를 갖고 있음.
Sentence : 완전한 문장 ➔ Nonterminal 기호가 없음.

CFG에서 문장 구조 정의 방식

- **Hierarchical structure (계층 구조)**
- **Recursion (순환 정의)**

if (expression) statement *else* statement



$Stmt \rightarrow$ if ($Expr$) $Stmt$ else $Stmt$

```
if (a > b)
      if (a > c) max = a; else max = c;
else
      if (b > c) max = b; else max = c;
```

Backus-Naur Form (BNF)

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **-** | *****

<exp> ::= <exp> <op> <exp> | (<exp>) | NUMBER
<op> ::= + | - | *

예 1 : 다양한 문법 표현

- exp → exp op exp | (exp) | *number*
  op → + | - | *

- E → E O E | (E) | n
  O → + | - | *

- exp → exp ("+"|"-"|"*") exp
  | "(" exp ")"
  | *number*

```
exp → exp op exp
exp → ( exp )
exp → number
op  → +
op  → -
op  → *
```

Table of Contents

Derivation (1/2)

- (34 - 3) * 42 가 문법에 맞는가?
  - ➔ (number – number) * number

```
exp → exp op exp | ( exp ) | number
op → + | - | *
```

Derivation (2/2)

(number – number)* number 가 문법에 맞는가?

- Derivation for the above expression

  exp ⇒ **exp op exp** [exp -> exp op exp]

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **-** | *****

Derivation (2/2)

(number - number)* number 가 문법에 맞는가?

- Derivation for the above expression

exp ⇒ **exp op exp** [exp -> exp op exp]

⇒ **exp op *number*** [exp -> ***number***]

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **-** | *****

Derivation (2/2)

(number – number)* number 가 문법에 맞는가?

- Derivation for the above expression

exp ⇒ **exp op exp** [exp -> exp op exp]

⇒ **exp op** ***number*** [exp -> ***number***]

⇒ **exp * *number*** [op -> ***]

exp → *exp op exp* | **(** *exp* **)** | ***number***

op → **+** | **–** | *****

Derivation (2/2)

(number – number)* number 가 문법에 맞는가?

- Derivation for the above expression

exp ⇒ **exp op exp** [exp -> exp op exp]

⇒ **exp op** ***number*** [exp -> ***number***]

⇒ ***exp*** ***** ***number*** [op -> *****]

⇒ **(** ***exp*** **)** ***** ***number*** [exp -> (exp)]

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **–** | *****

Derivation (2/2)

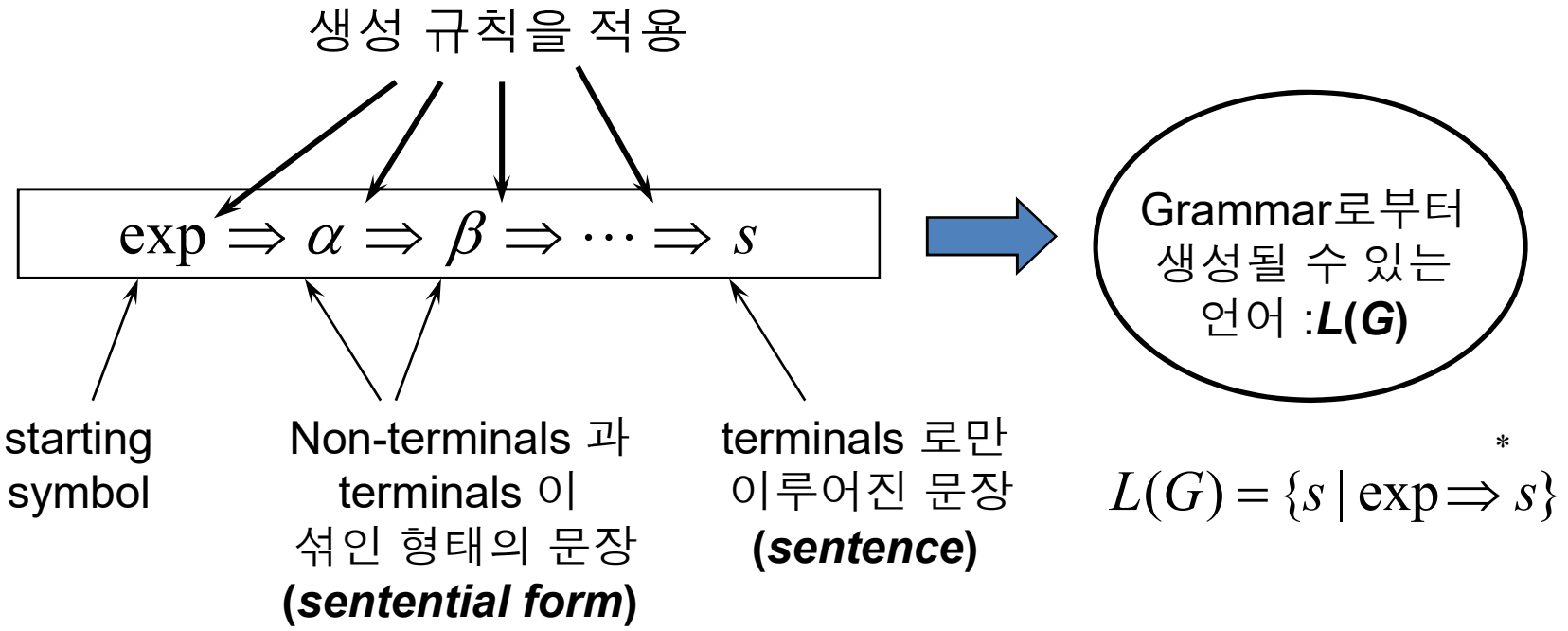
(number – number)* number 가 문법에 맞는가?

- Derivation for the above expression

exp ⇒ **exp op exp** [exp -> exp op exp]
⇒ **exp op** ***number*** [exp -> ***number***]
⇒ **exp * *number*** [op -> *]
⇒ **(*exp*) * *number*** [exp -> (exp)]
⇒ **(exp op *exp*) * *number***
⇒ **(exp *op* number) * *number***
⇒ **(*exp* – *number*) * *number***
⇒ **(*number* – *number*) * *number***

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **-** | *****

Derivation : 유도(誘導) 또는 파생



예 2: ((a))는 문법에 맞는가?

- **E → (E) | a**

$G = \{T, N, S, P\}$

- A set of nonterminals, $N = \{E\}$
- A set of terminals, $T = \{(,), a\}$
- The start symbol, $S = E$

- **((a))**의 유도 과정
  - $E \Rightarrow$ **(E)** [E -> (E)]
    $\Rightarrow$ (**(E)**) [E -> (E)]
    $\Rightarrow$ ((**a**)) [E -> a]

예 3: 이 문법이 정의하는 언어는?

- **E → (E) | a**
  - 문법이 정의하는 언어 $L(G)$
    - $L(G) = \{a, (a), ((a)), (((a))), \ldots\}$
      $= \{(^n\ \mathbf{a}\ )^n \mid n \geq 0\}$

예 4: 다음 문법이 생성하는 언어는?

- **E → (E)**
  - $L(G) = \{\}$

예 5: 다음 문법이 생성하는 언어는?

- **E → E + a | a**
  - *L(G)* = {*a*, *a+a*, *a+a+a*, *a+a+a+a*, … }

    = {strings consisting of **a**'s separated by **+**'s}

  - **a+a+a** 의 유도 과정

    $E \Rightarrow \mathbf{E + a}$

    $\Rightarrow \mathbf{E + a} + a$

    $\Rightarrow \mathbf{a} + a + a$

예 6: 다음 문법이 생성하는 언어는?

statement → *if-stmt* | `other`

if-stmt → `if` (*exp*) *statement*

| `if` (*exp*) *statement* `else` *statement*

exp → `0` | `1`

```
other
if (0) other
if (1) other
if (0) other else other
if (1) other else other
if (0) if (0) other
if (0) if (1) other else other
if (1) other else if (0) other else other
 . . . . . .
```

Left *versus* Right Recursive Rules

Left Recursive rule

- 일반형: $A \rightarrow A\ \alpha \mid \beta$ (= $\beta\ \alpha^*$)

  예: $A \rightarrow A\ a \mid b$

  $A \Rightarrow A\ a \Rightarrow A\ aa \Rightarrow A\ aaa \Rightarrow baaa$

Right Recursive rule

- 일반형: $A \rightarrow \alpha\ A \mid \beta$ (= $\alpha^*\ \beta$)

  예: $A \rightarrow a\ A \mid b$

  $A \Rightarrow a\ A \Rightarrow aa\ A \Rightarrow aaa\ A \Rightarrow aaab$

ε-production

- **A grammar rule has an <u>empty right-hand side</u>**
  - $A \rightarrow A\ a \mid \varepsilon$ or $A \rightarrow a\ A \mid \varepsilon$

예: $A \rightarrow A\ a \mid \varepsilon$

$A \Rightarrow \mathbf{A}\ a \Rightarrow \mathbf{A}\ aa \Rightarrow \varepsilon\ aa \Rightarrow aa$ $\qquad \beta\alpha^* = \varepsilon a^* = a^*$

예: $A \rightarrow a\ A \mid \varepsilon$

$A \Rightarrow a\ \mathbf{A} \Rightarrow aa\ \mathbf{A} \Rightarrow aa\ \varepsilon \Rightarrow aa$ $\qquad \alpha^*\beta = a^*\varepsilon = a^*$

예 7: 다음 문법이 생성하는 언어는?

■ $A \rightarrow (A) A \mid \varepsilon$

- the strings of all "*balanced parentheses*"

- (( ))( ) 의 유도 과정
  - $A \Rightarrow (A)\ A \Rightarrow (A)\ (A)\ A \Rightarrow (A)\ (A) \Rightarrow (A)\ (\ )$
    $\Rightarrow ((A)\ A\ )\ (\ )\ \Rightarrow (\ (A)\ )\ (\ ) \Rightarrow ((\ ))\ (\ )$

예 8: 다음 문법이 생성하는 언어는?

stmt-sequence → *stmt* ; *stmt-sequence* | *stmt*

stmt → s

$A \rightarrow \alpha\ A \mid \beta$ ➔ $\alpha^*\ \beta$ ➔ (*stmt ;*)* *stmt* ➔ (*s ;*)* *s*

$L(G)$ = {**s, s;s, s;s;s,** ...} ➔ ; is a *separator*

예 9: 다음 문법이 생성하는 언어는?

stmt-sequence → *stmt* ; *stmt-sequence* | *stmt*

stmt → s

stmt-sequence → *stmt* ; *stmt-sequence* | ε

stmt → s

$A \rightarrow \alpha\ A \mid \beta$ ➔ $\alpha^* \beta$ ➔ (*stmt* ;)* ε ➔ (*s* ;)*

$L(G) = \{\varepsilon,$ **s;, s;s;, s;s;s;,** ...$\}$ ➔ ; is a *terminator*

어떤 Nonterminal 을 선택할 것인가?

P :	S → ASB	S → ε
	A → a	B → b

left-most derivation **(좌단 유도)**

S => ***A***SB => a***S***B => a***A***SBB => aa***S***BB => aa***B***B => aab***B*** => aabb

right-most derivation **(우단 유도)**

S => AS***B*** => A***S***b => AAS***B***b => AA***S***bb => A***A***bb => ***A***abb => aabb

*left-most derivation → Top-Down parse → L**L** parsing*
*right-most derivation → Bottom-Up parse → L**R** parsing*

Leftmost derivations

예: **(34-3)*42**

- *exp* ⇒ *exp op exp*
  ⇒ **(** *exp* **)** *op exp*
  ⇒ **(** *exp op exp* **)** *op exp*
  ⇒ **(** ***number*** *op exp* **)** *op exp*
  ⇒ **(** ***number*** **-** *exp* **)** *op exp*
  ⇒ **(** ***number*** **-** ***number*** **)** *op exp*
  ⇒ **(** ***number*** **-** ***number*** **)** ***** *exp*
  ⇒ **(** ***number*** **-** ***number*** **)** ***** ***number***

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **-** | *****

Rightmost derivations

예: **(34-3)*42**

- *exp* ⇒ *exp op exp*
  ⇒ *exp op* ***number***
  ⇒ *exp* **** number***
  ⇒ **(** *exp* **)** **** number***
  ⇒ **(** *exp op exp* **)** **** number***
  ⇒ **(** *exp op* ***number*** **)** **** number***
  ⇒ **(** *exp* ***- number*** **)** **** number***
  ⇒ **(** ***number - number*****)** **** number***

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **-** | *****

Leftmost vs. Rightmost Derivations

예: **(34-3)*42**

- *exp* ⇒ *exp op exp*
  ⇒ *exp op* ***number***
  ⇒ *exp* ***** ***number***
  ⇒ **(** *exp* **)** ***** ***number***
  ⇒ **(** *exp op exp* **)** ***** ***number***
  ⇒ **(** *exp op* ***number*** **)** ***** ***number***
  ⇒ **(** *exp* **-** ***number*** **)** ***** ***number***
  ⇒ **(** ***number*** **-** ***number*****)** ***** ***number***

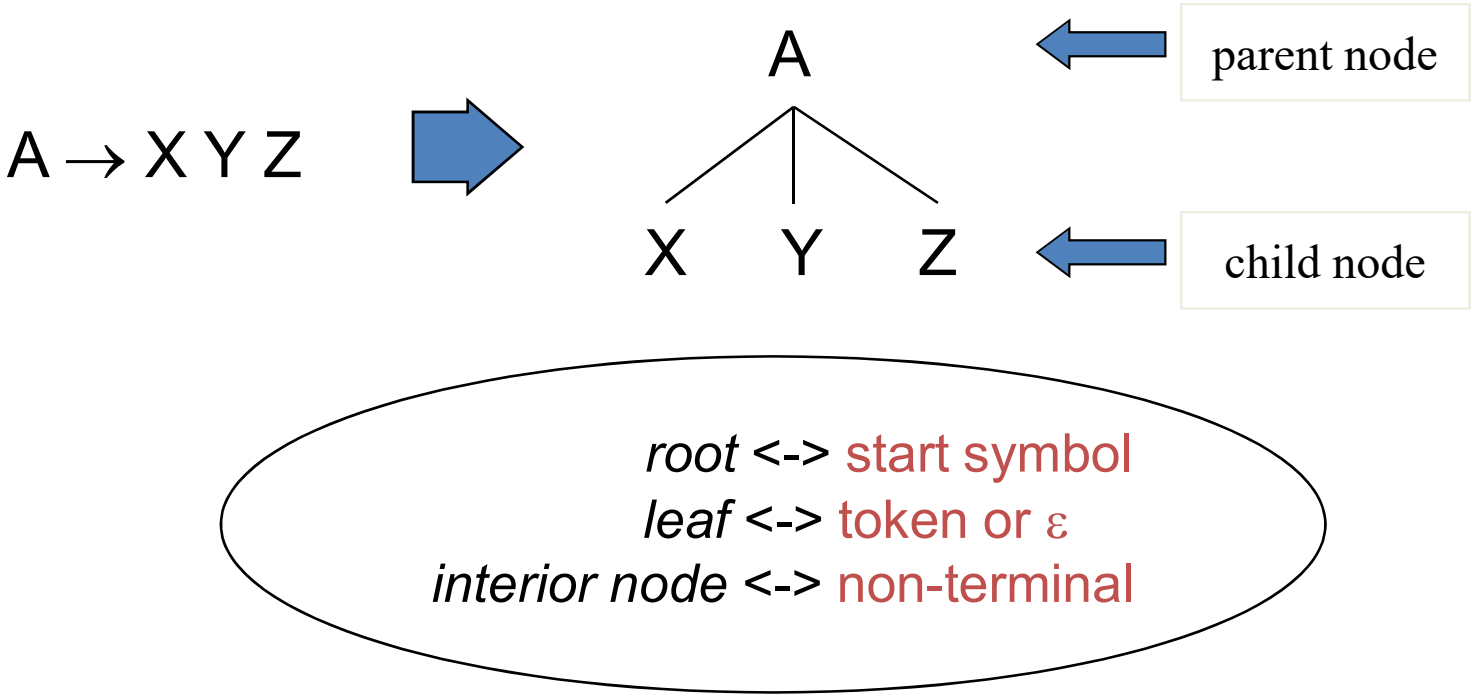
- *exp* ⇒ *exp op exp*
  ⇒ **(** *exp* **)** *op exp*
  ⇒ **(** *exp op exp* **)** *op exp*
  ⇒ **(** ***number*** *op exp* **)** *op exp*
  ⇒ **(** ***number*** **-** *exp* **)** *op exp*
  ⇒ **(** ***number*** **-** ***number*** **)** *op exp*
  ⇒ **(** ***number*** **-** ***number*** **)** ***** *exp*
  ⇒ **(** ***number*** **-** ***number*** **)** ***** ***number***

Which one is *leftmost derivation?*

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **-** | ***

Table of Contents

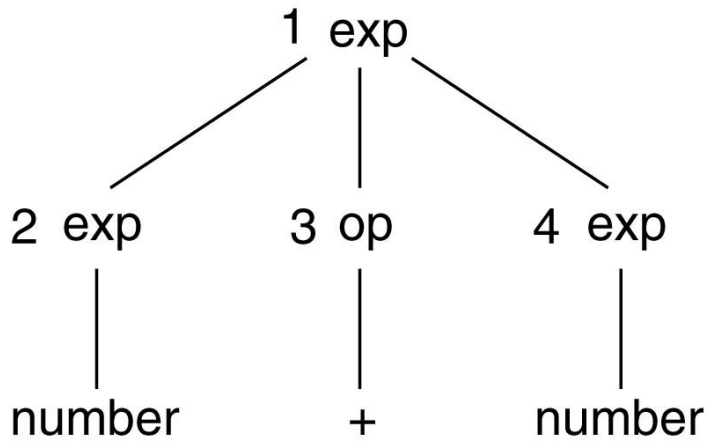
구문 트리(Parse Tree)



유도 과정(즉 parsing)을 graphic 하게 도시(圖示)

Leftmost Derivations

(1) $exp \Rightarrow \mathbf{exp}\ \ op\ \ exp$
(2) $\Rightarrow number\ \ op\ \ exp$
(3) $\Rightarrow number + exp$
(4) $\Rightarrow number + number$

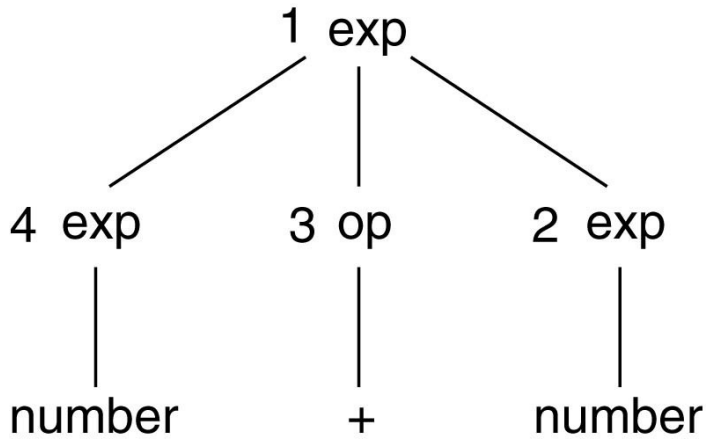


preorder numbering

```
exp → exp op exp | ( exp ) | number
op → + | - | *
```

Rightmost Derivations

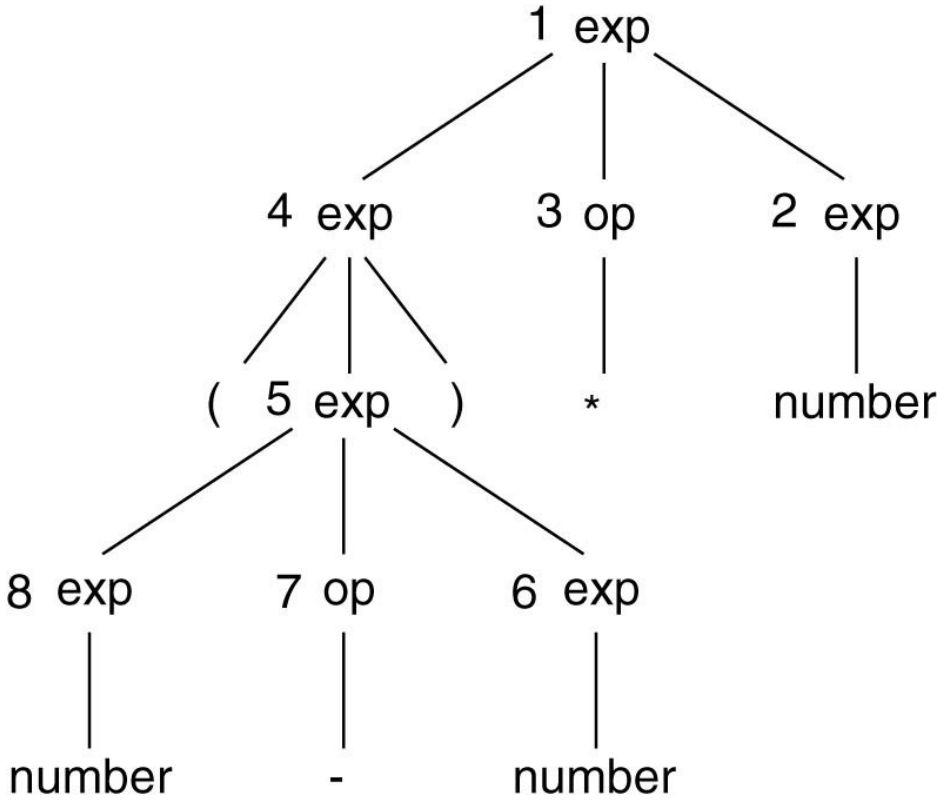
(1) $exp \Rightarrow exp\ \ op\ \ exp$
(2) $\Rightarrow exp\ op\ number$
(3) $\Rightarrow exp + number$
(4) $\Rightarrow number + number$



postorder numbering

```
exp → exp op exp | ( exp ) | number
op → + | - | *
```

Parse Tree for (34-3)*42



Right most derivation

(1)	*exp*	⇒ *exp op exp*
(2)		⇒ *exp op number*
(3)		⇒ *exp* * *number*
(4)		⇒ (*exp*) * *number*
(5)		⇒ (*exp op exp*) * *number*
(6)		⇒ (*exp op number*) * *number*
(7)		⇒ (*exp* - *number*) * *number*
(8)		⇒ (*number* - *number*) * *number*

Quiz #1

- Given

  S → 0 A S | 0

  A → S 1 A | S S | 1 0

- 입력 문장이 **0 0 1 1 0 0** 일 때
  - 좌단 유도를 통해 구문 분석을 하시오.
  - 우단 유도를 통해 구문 분석을 하시오.

Quiz #2

- **Given**

  E → E + T | T | E - T

  T → T * F | F | T / F

  F → (E) | id | -E | num

- **Draw a parse tree**

  num + num * num

- **Draw a parse tree**

  - id + num

Quiz #3

- Consider the context-free grammar

  `S → S S + | S S * | a`

  and the string `aa + a*`

a) Give a *leftmost* derivation for the string

b) Give a *rightmost* derivation for the string

c) Give a parse tree for the string

d) *Describe the language* generated by this grammar