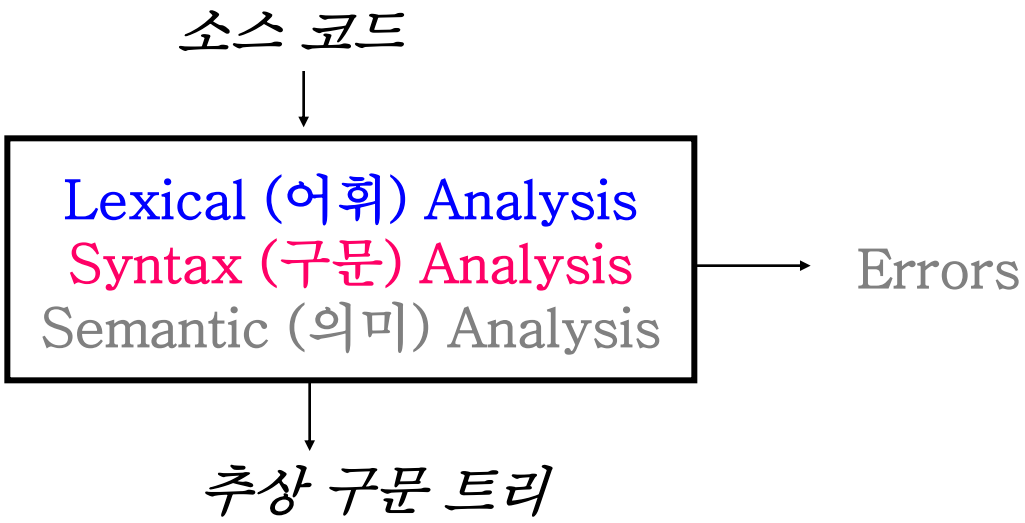
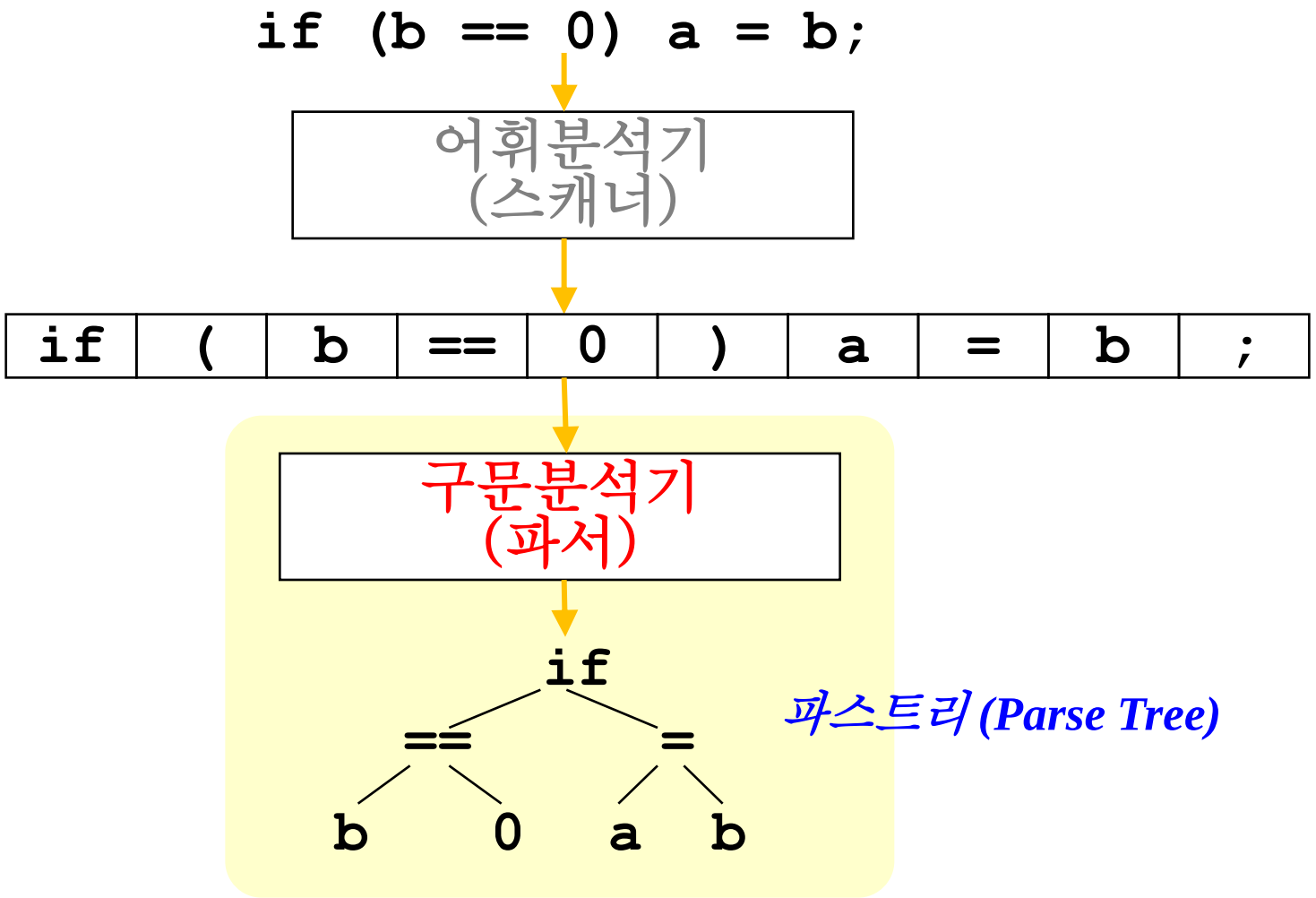
Compiler (컴파일러)
구문(syntax) 과 유도 (derivation)

2015년 2학기
충남대학교 컴퓨터공학과
조은선

컴파일러 전반부 Structure

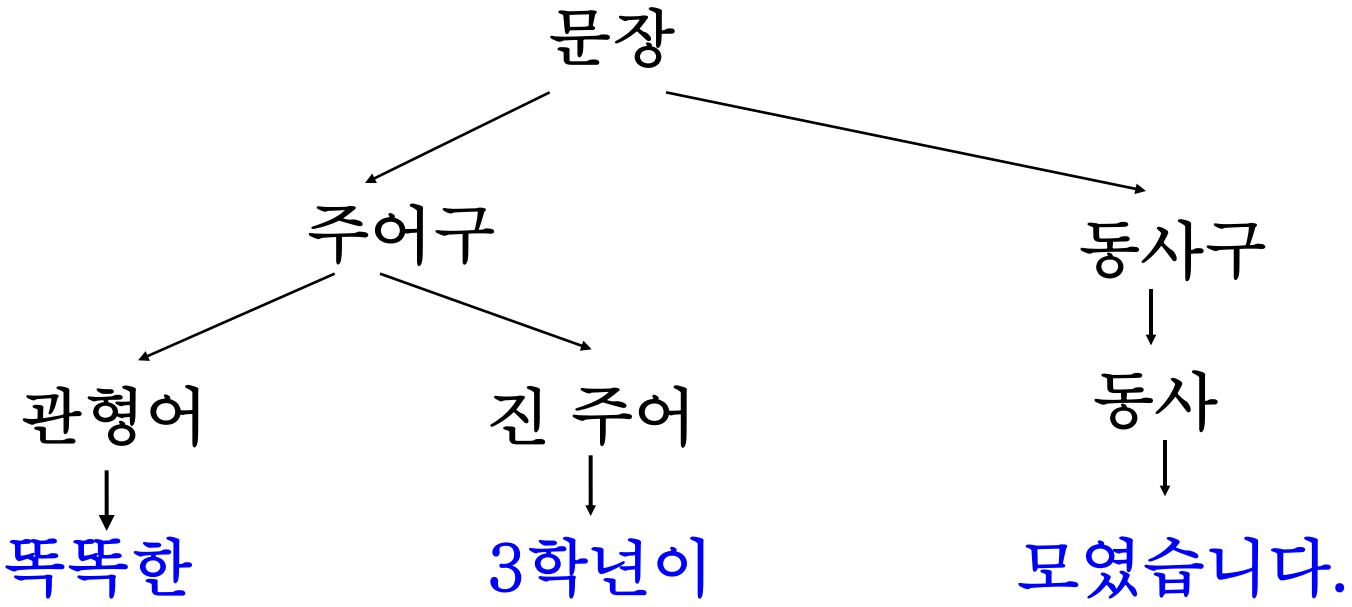


Syntax Analysis (구문 분석)



우리말 구문,문법 분석

- 문법이 맞는지 확인하고
- 각 단어의 역할을 확인한다.



"똑똑한 3학년이 모였습니다."

Syntax Analysis 와 관련된 질문들

1) How to **describe the syntax**?
   - 프로그래밍 언어 제작자가 문법을 기술(설명)하는 방법

2) How to **determine** if the input token stream **satisfies the syntax** description?
   - 주어진 input token stream이 기술된 문법에 맞는지 판별하는 방법

참고) 어휘분석 (lexical analysis) 때는..
   - 정규표현식으로 기술하고
   - 상태전이도로 따라가며 판별했다.

(1) How to Describe The Syntax

- CFG (Contex Free Grammar)
  - 언어의 문법을 정의하는 일반적인 방법
  - 간단하고 이해하기 쉽다.
  - 표현된 문법으로부터 자동적으로 인식기를 구현가능
- G = (V, T, P, S)
  - V : non terminal 심벌 집합 (중간과정 심벌)
  - T : terminal 심벌 집합
  - P : 생성 규칙 집합 : $N \rightarrow \alpha$

    where $N \in V$, $\alpha \in (V \cup T)^*$
  - S : 시작 심벌
- L(G) : 이 문법으로 생성되는 language

- **문법 심벌들의 일반적인 표기법**
  - Terminal 심벌 (**V**)
    - a, b, c와 같은 알파벳 시작 부분의 소문자와 숫자 0,1,2,…,9;
    - +, -와 같은 연산자 기호와 세미콜론, 콤마, 괄호와 같은 구분자
    - ‘if’ 또는 “then”과 같이 따옴표 사이에 표기된 문법 심벌
  - Nonterminal 심벌 (**N**)
    - A, B, C와 같은 알파벳 시작 부분의 대문자로 나타내게 됨
    - S는 보통 시작 심벌(start symbol)을 나타낸다.
    - <stmt>나 <expr>과 같이 < >로 묶어서 나타내기도 함

– 생성규칙 (**P**)

- 예)

  $S \rightarrow T + T$

  $T \rightarrow$ '0'

  $T \rightarrow$ '1'

  $T \rightarrow$ '2'

- $A \rightarrow a_1, A \rightarrow a_2, \ldots, A \rightarrow a_k$와 같이 생성 규칙의 왼쪽이 모두 A인 경우에, $A \rightarrow a_1 | a_2 | \ldots | a_k$로 간단히 표기

  예) $T \rightarrow$ '0'|'1'|'2'

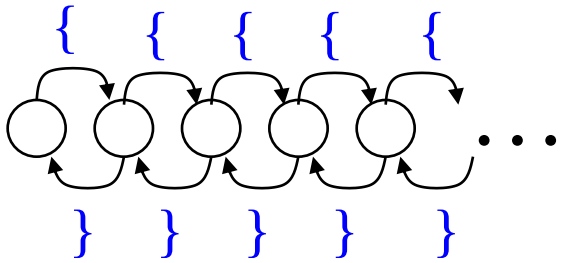
– 시작 심벌 (**S**)

- 만약 아무런 언급이 없으면 첫번째 생성 규칙의 왼쪽에 있는 것이 시작 심벌

정규표현식을 문법 기술에 사용한다면?

- 정규표현식
  - 토큰을 기술하는데에 좋음
  - DFA 로 변환하여 구현하기에도 좋음
- 그러나..
  - Block, expressions, statements … 등의 Nested 구조의 구문을 표현하기에 power 가 떨어짐.
  - 예) block의 괄호 맞추기

{{} {} {{} { }}}



Class Problem

- 다음 정규표현식을 CFG로 바꾸어적으시오.
  1. (a|b) c
  2. a+
  3. (1(0|1)*) | 0

- {{{{}}} } 등 정상적인 괄호 매치에 대한 CFG를 작성하시오.

여러가지 CFG 표기법

- **BNF(Backus-Naur Form)**
  - nonterminal 심벌 : <와 >로 묶어서 표기
  - terminal 심벌 : 문자 스트링

    <id> ::= <letter> | <id><letter> | <id><digit>

    <letter> ::= a | b | c | … | y | z

    <digit> ::= 0 | 1 | 2 | … | 8 | 9

- **EBNF(Extended BNF)**
  - 메타 심벌 도입 : 반복되는 부분이나 선택적인 부분을 간결하게 표현하기 위한 특수 심벌

    <id> ::= <letter>{<letter>|<digit>}07

    <letter> ::= a | b | c | … | y | z

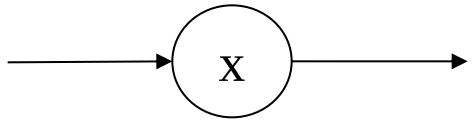
    <digit> ::= 0 | 1 | 2 | … | 9

예 ANTLR 문법 일종의 EBNF

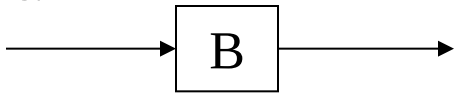
```
grammar MiniC;

program         : decl+ ;
decl            : var_decl
                | fun_decl              ;
var_decl        : type_spec IDENT  ';'
                | type_spec IDENT '[' ']' ';' ;
type_spec       : VOID
                | INT                       ;
fun_decl        : type_spec IDENT '(' params ')' compound_stmt;
params          : param ('+' param)*
                | VOID                      ;
param           : type_spec IDENT
                | type_spec IDENT '[' ']'     ;
...
```

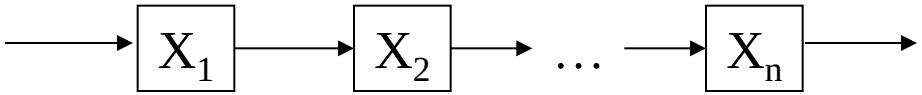
- 문법흐름도 (syntax diagram)
  - terminal x



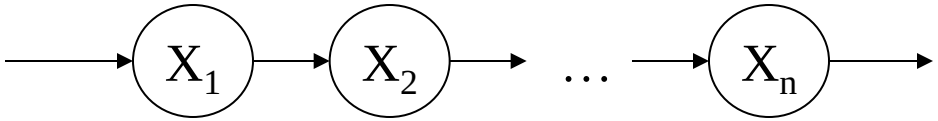
  - nonterminal B



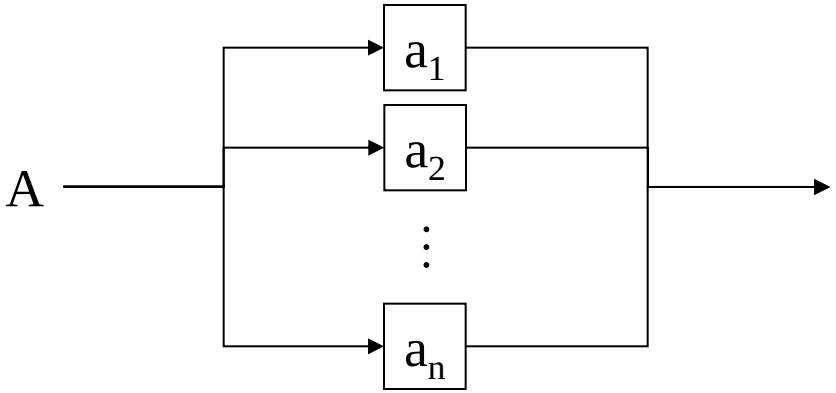
  - 생성규칙 $A::=X_1X_2 \ldots X_n$
    - X_i가 nonterminal인 경우



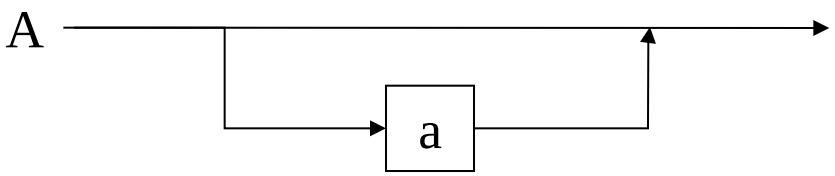
    - X_i가 terminal인 경우



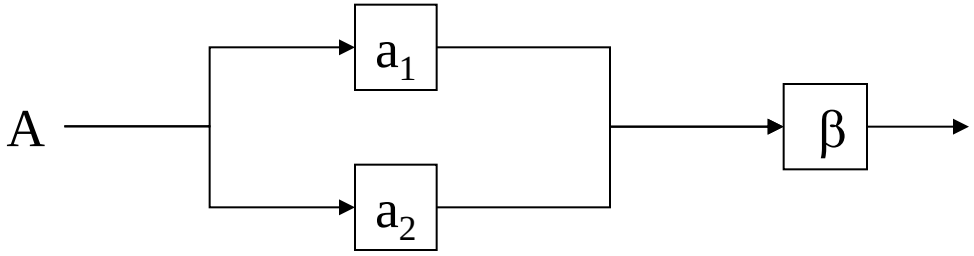
– 택일 생성 규칙 $A::=a_1|a_2|\ldots|a_n$



– EBNF A ::= [a]



– EBNF $A ::= (a_1 \mid a_2)\beta$

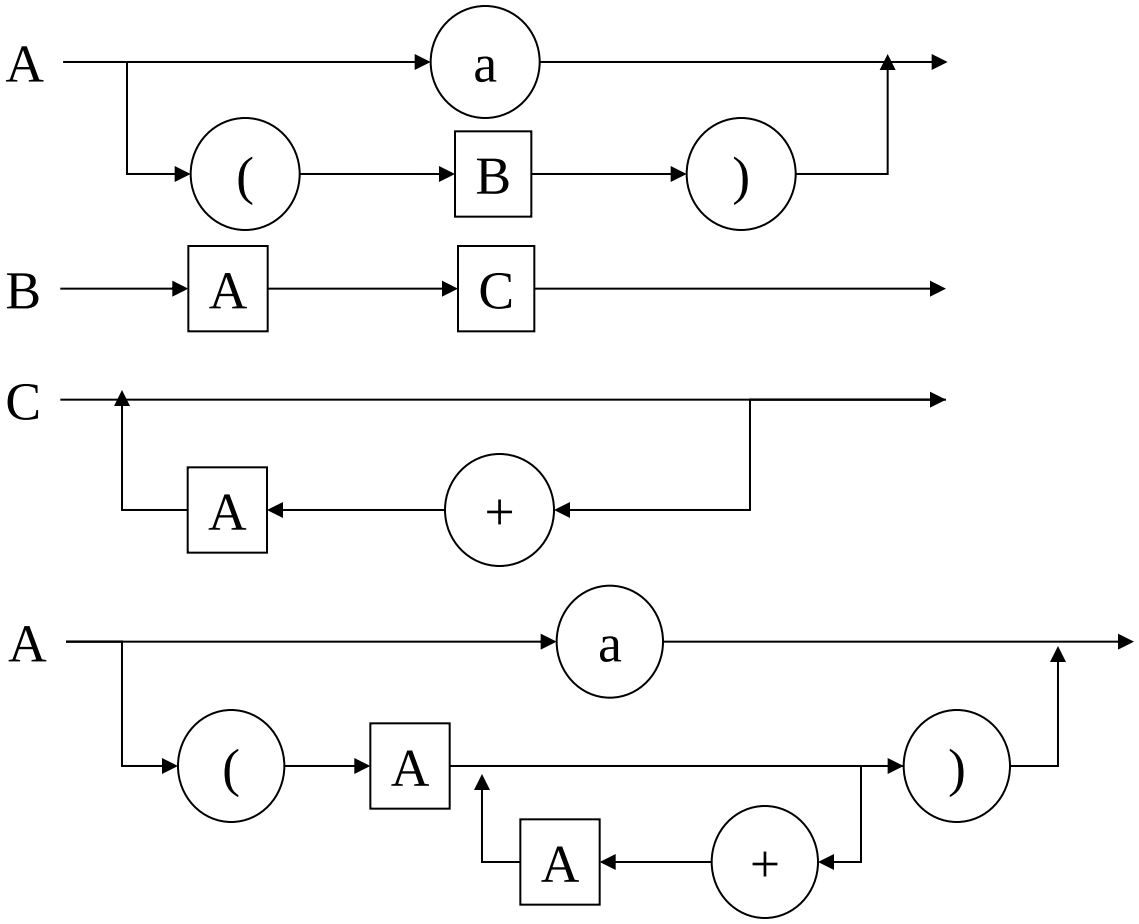


예제

A ::= a | (B)

B ::= AC

C ::= {+A}



(2) How to determine if the input token stream satisfies the syntax description

- 문법과 언어
  - 문법 :
    S → <주어구> <동사구>
    <주어구>→ <관형구> <진 주어>
  - 언어 : "똑똑한 3학년이 모였습니다."
- 유도
  - 문법에서 언어를 생성해내는 것
- 구문분석
  - 언어에서 문법을 확인해 가는 것
  - 방법은 .. 유도 되는지 보면 됨

유도 (derivation)

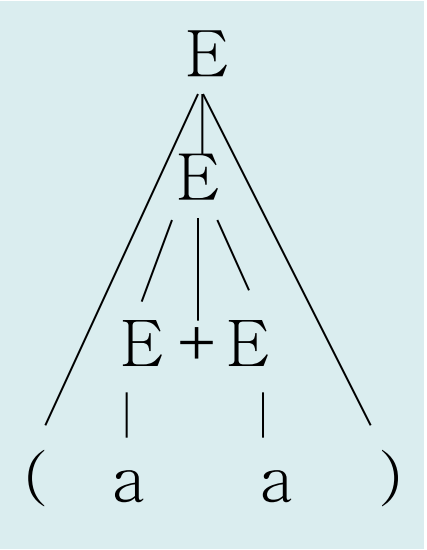
- 유도
  - 문법에서 문장(언어)을 생성하기
  - 생성 규칙을 연속적으로 적용하여 Nonterminal을 확장함으로써 얻음
  - 예:

문법: E → E + E | E * E | (E) | a

언어: (a+ a)

유도: E ⇒ (E) ⇒ (E+ E) ⇒ (a+ E) ⇒ (a+ a)

- 유도 트리
  - 유도 경로를 추상화 시켜 표현한 것



Non-terminal 심벌의 대치 순서

"생성 규칙의 '→' 다음에 하나 이상의 nonterminal 심벌이 올 수 있음"

- 좌측유도(leftmost derivation) : 가장 왼쪽에 있는 nonterminal 먼저 대치
- 우측 유도(rightmost derivation): 가장 오른쪽에 있는 nonterminal 먼저 대치

- 문법

1. E→E+E
2. E→E*E
3. E→(E)
4. E→a

- 좌측유도

E ⇒ E*E 2
⇒ (E)*E 3
⇒ (E+E)*E 1
⇒ (a+E)*E 4
⇒ (a+a)*E 4
⇒ (a+a)*a 4

- 우측유도

E ⇒ E*E 2
⇒ E*a 4
⇒ (E)*a 3
⇒ (E+E)*a 1
⇒ (E+a)*a 4
⇒ (a+a)*a 4

Q) 각 경우 유도 트리는?

Class Problem

- 다음과 같은 문법이 있을 때, a*(a+ a)에 대해 좌측유도와 우측유도를 각각 하시오
  - 문법

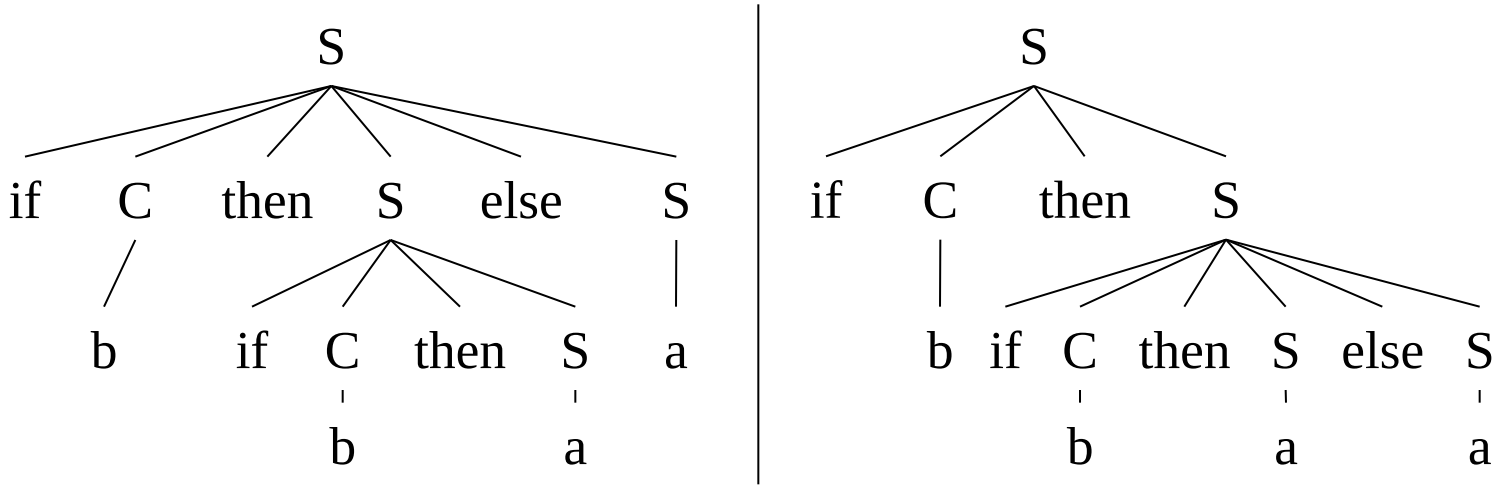
  1. E→E+ E
  2. E→E*E
  3. E→(E)
  4. E→a

- 각각의 경우에 대해 유도 트리를 그리시오.

- 모호성(Ambiguity)
  - 문법 G에 의해 생성되는 어떤 문장이 두개 이상의 유도 트리를 갖는다면 문법 G는 모호하다 (ambiguous) 한다.

예) 문장 “if b then if b then a else a”에 대해

S → if C then S else S
S → if C then S
S → a
C → b



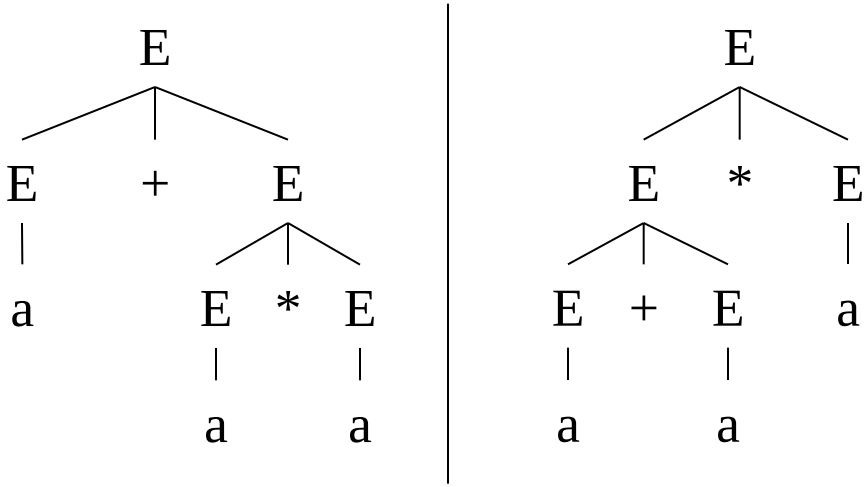
모호성 해결
같은 언어를 생성하는 다른 문법을 사용함

- 앞의 예 (참고만 할 것)

```
S→  matched_stmt
    | unmatched_stmt
matched_stmt →
      if expr then matched_stmt else matched_stmt
    | other
unmatched_stmt →
      if expr then stmt
    | if expr then matched_stmt else unmatched_stmt
```

예)

$E \rightarrow E+E \mid E*E \mid (E) \mid a$



..... 입력이 a + a * a일 경우

모호성 해결 1 – 연산자 우선순위 도입

- 연산자 우선순위를 도입한 동일한 문법 만들기
  - 연산자 마다 새로운 Non-terminal을 도입하되
  - Recursion을 left 나 right 둘중 하나만 두고,
  - 시작심벌과 가장 가까운 쪽에 연산자 우선순위가 낮은 것을 둠

예) $E \rightarrow E+E \mid E*E \mid (E) \mid a$ 에 대해 다음과 같이 변경

$E \rightarrow E+T$ $\quad\quad$ $E \rightarrow T$

$T \rightarrow T*F$ $\quad\quad$ $T \rightarrow F$

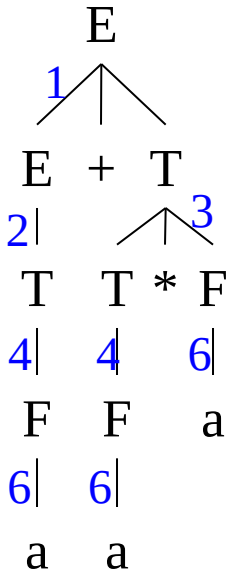
$F \rightarrow (E)$ $\quad\quad$ $F \rightarrow a$

예제

a + a * a

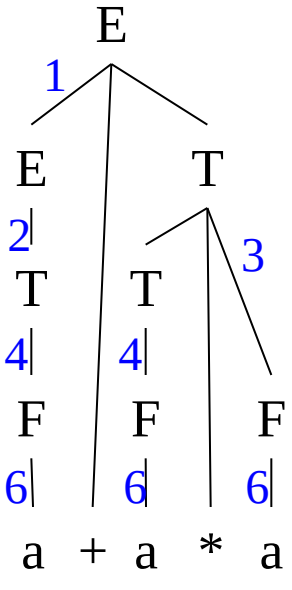
1. E → E+ T　　2. E → T
3. T → T*F　　4. T → F
5. F → (E)　　6. F → a

E ⇒ E + T　1
⇒ T + T　2
⇒ F + T　4
⇒ a + T　6
⇒ a + T*F　3
⇒ a + F*F　4
⇒ a + a*F　6
⇒ a + a*a　6



좌측유도

E ⇒ E + T　1
⇒ E + T*F　3
⇒ E + T*a　6
⇒ E + F*a　4
⇒ E + a*a　6
⇒ T + a*a　2
⇒ F + a*a　4
⇒ a + a*a　6



우측유도

Class Problem

- 다음 문법에서 !가 우선순위가 가장 높고, & 가 우선순위가 가장 낮다고 할 때, 변경된 문법을 적으시오.

S→ '!' S
S→ S '&' S
S→ S '|' S
S→ a

모호성 해결 2 – 결합 법칙 도입

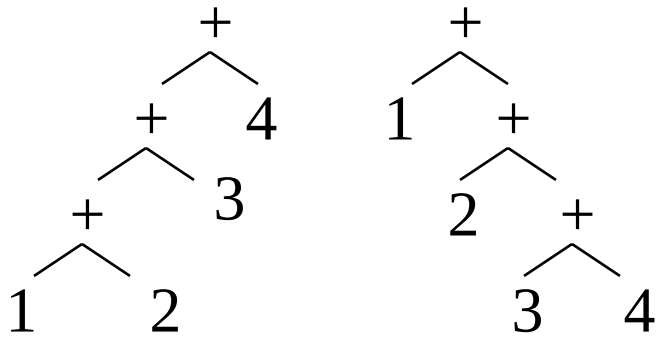
- 연산자 우선순위를 도입한 동일한 문법 만들기
  - 모든 연산자는 좌측, 또는 우측결합 이거나, 결합이 성립되지 않는다.
    - **Left: a + b + c = (a + b) + c**
    - **Right: a ^ b ^ c = a ^ (b ^ c)**
    - **Non: a < b < c 오류 (결합 불가)**
  - 원한다면, 문법에서 결합법칙을 강제함.
    - Recursion을 잘 배치
    - Left Recursion 은 좌측결합에 사용
      - A→ A+a | a
    - Right Recursion은 우측결합에 사용
      - A→ a^A | a

결합 법칙을 강제하기

- 예) 다음 두 문법의 차이는?

E → E + T
E → T
T → num

E → T + E
E → T
T → num

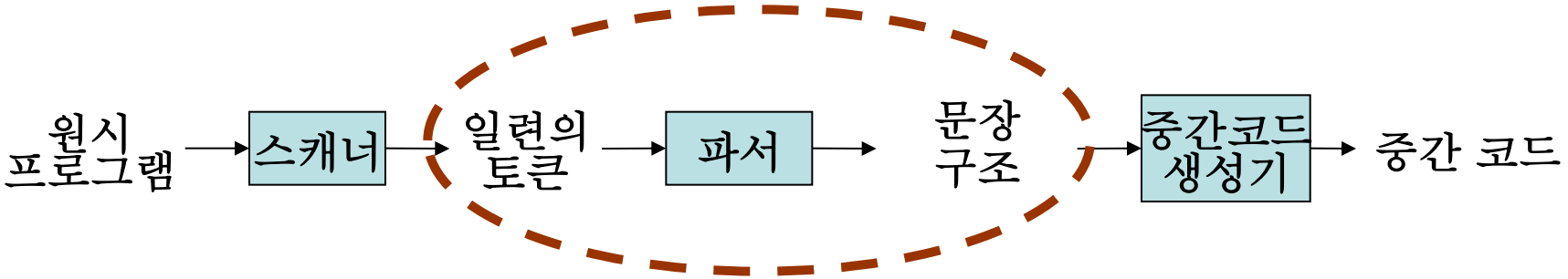


1+2+3+4

구문분석 방법

- 구문분석 (= 파스)
  - 주어진 스트링이 정의된 문법에 의해 생성될 수 있는지의 여부 (적합성) 를 결정하는 과정
  - 유도가 되는지 보면 됨
  - 올바른 문장에 대해서는 '문장 구조'를,
  - 틀린 문장에 대해서는 오류 메시지를 나타냄

- 구문분석기(=파서)



문장 구조를 나타내기 위한 자료 구조

- 파스 트리(parse tree)
  - 문장 구조를 나타내는 트리
  - 문법을 나타내는 생성 규칙을 적용한 유도 트리와 같은 모양 (!) 의 트리
    - 루트노드 : 정의된 문법의 시작 심벌
    - 중간노드 : 각 생성 규칙의 좌측 nonterminal 심벌
    - 단말노드 : 주어진 스트링을 생성하는 terminal 심벌

- 생성규칙 번호 리스트
  - 문법의 생성 규칙을 통해 유도하는 과정에서 적용되어온 일련의 생성 규칙 번호

구문분석의 두 가지 방법

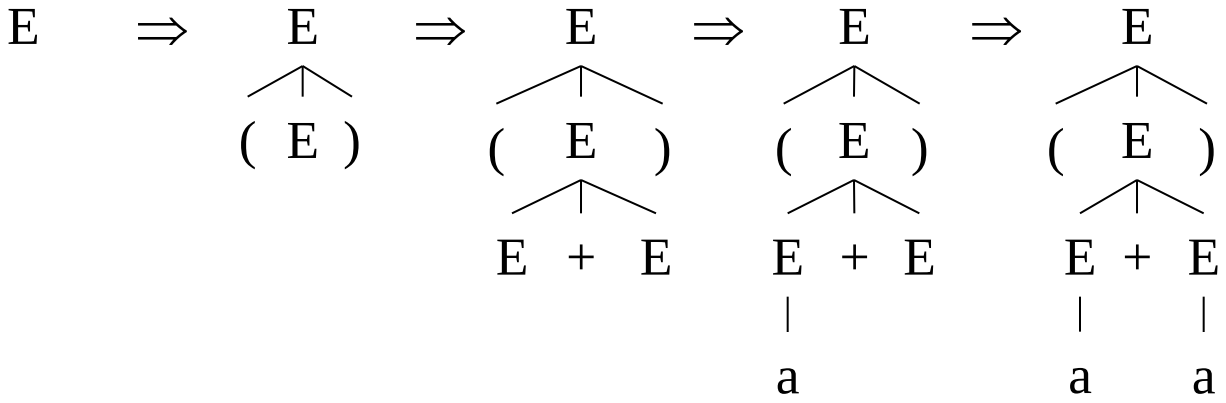
- Top-down방식
  - 루트 노드로부터 시작하여 단말노드를 생성하며 확인

- Bottom-up방식
  - 단말 노드로부터 루트 노드를 향하여 위로 생성하며 확인

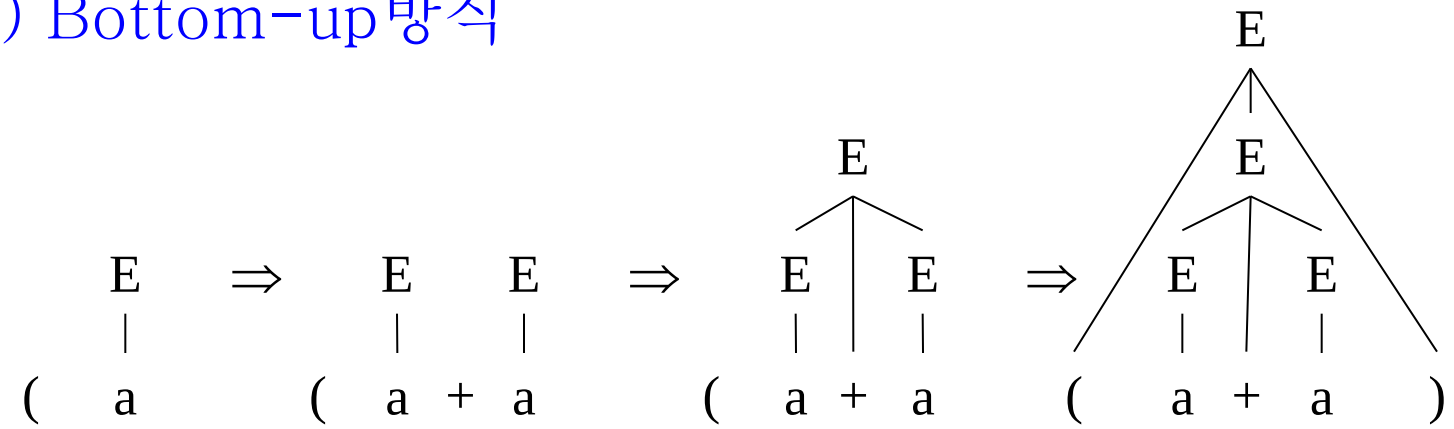
예제 (a+ a)

1. E → E + E
2. E → E * E
3. E → (E)
4. E → -E
5. E → a

1) Top-down방식



2) Bottom-up방식



- 그런데, 구문분석 방식은 (more)
  - Top-down방식은
    - 좌측유도와 같은 순서의 생성규칙 적용

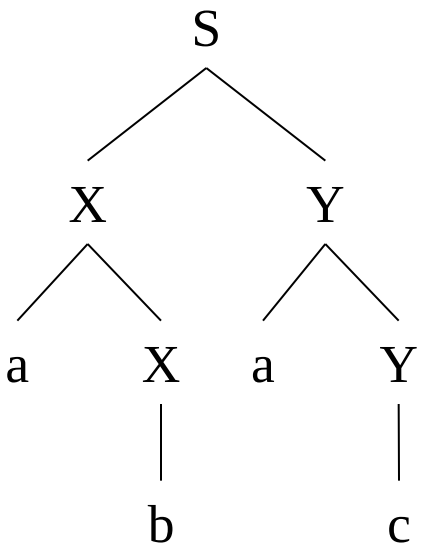
    ← "**좌파스**"란? 좌측유도 중 적용된 생성규칙들의 리스트

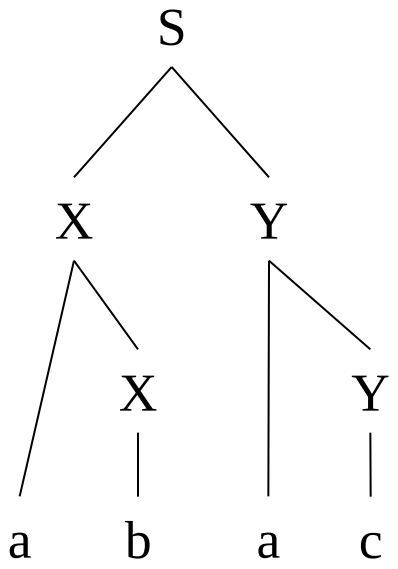
  - Bottom-up방식은
    - 우측유도의 역순의 생성규칙 적용과 같음!
      - 입력 스트링의 왼쪽에서 부터 매치하므로

    ← "**우파스**"란? : 우측유도 중 적용된 생성규칙들의 리스트의 역순

예제 1 : **abac**

1. $S \rightarrow XY$ 2. $X \rightarrow aX$ 3. $X \rightarrow b$
4. $Y \rightarrow aY$ 5. $Y \rightarrow c$





top down
좌파스 : 12345

$S \Rightarrow XY$
$\Rightarrow aXY$
$\Rightarrow abY$
$\Rightarrow abaY$
$\Rightarrow abac$

$S \Rightarrow XY$
$\Rightarrow XaY$
$\Rightarrow Xac$
$\Rightarrow aXac$
$\Rightarrow abac$

bottom up
우파스 : 32541

Class Problems

a + a * a

1. $E \rightarrow E+T$
2. $E \rightarrow T$
3. $T \rightarrow T*F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow a$

* 위의 문자열에 대해
  1. 좌측유도와 우측유도를 구하시오.
  2. 좌파스와 우파스를 구하시오.