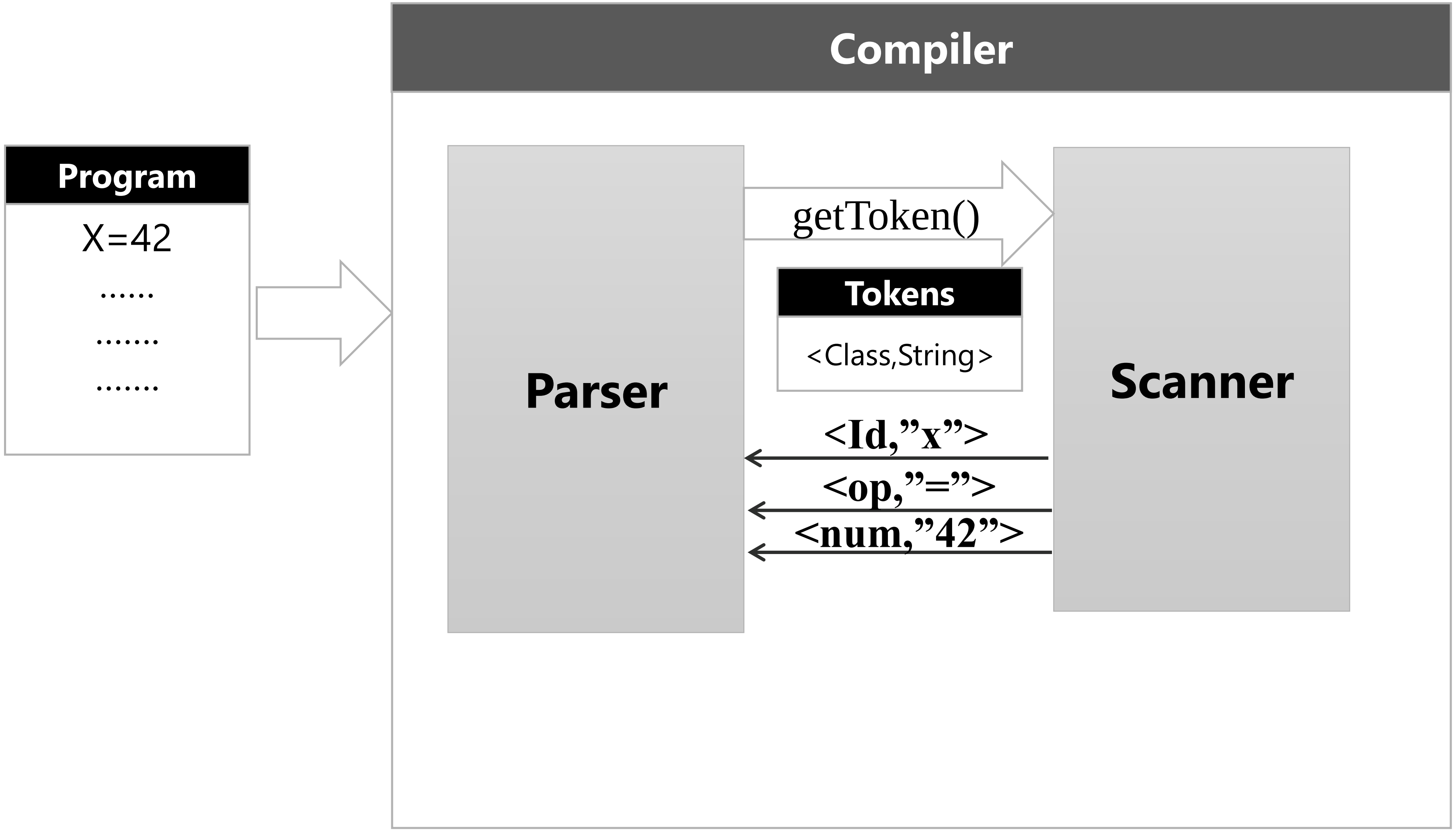
Parser

Lecture Five

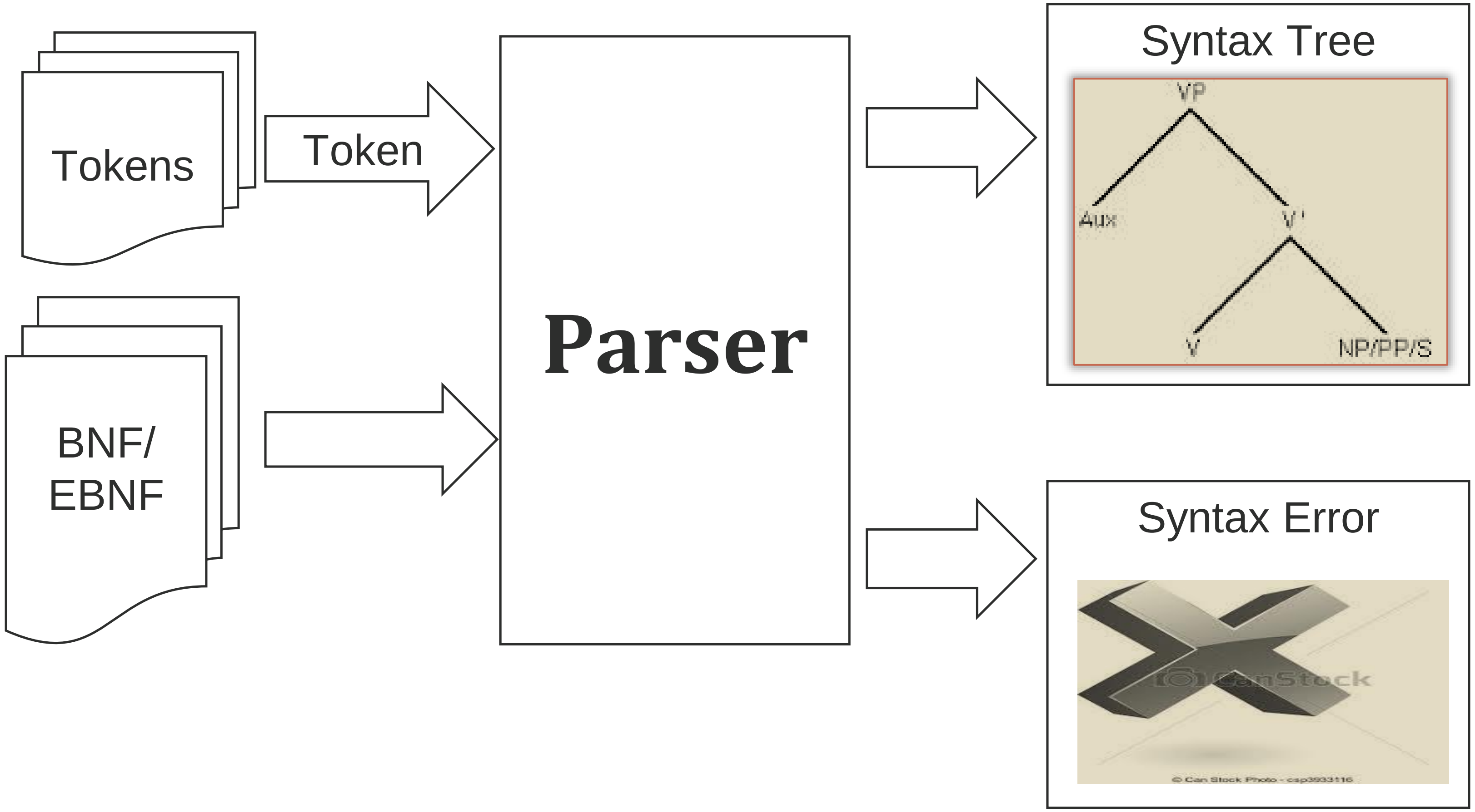

Program
X=42
……
…….
…….
Compiler
Parser
getToken()
Tokens
<Class,String>
<Id,"x">
<op,"=">
<num,"42">
Scanner

Parser (syntax Analyzer)

Lecture 5 Topics

- The Parsing Process
- Context-Free Grammars
- Derivations
- Parse Trees
- Abstract Syntax Trees
- Ambiguous Grammars
- Dealing with ambiguity
- EBNF (Extended BNF)

The Parsing Process



The Parsing Process

- Input: *tokens*
- Output: A *syntax tree*
- lexical structure of tokens is specified by regular expressions, the syntax of programs is specified by the *grammar rules* of a *context-free grammar*.
- Parser must distinguish between valid and invalid programs (strings of tokens).
- (1 + + 2) + 3
- The parser verifies that the string can be generated by the grammar for the source language

Regular Languages

- The languages used till now is Regular Language
- A language is *regular* if it is accepted by some DFA.

  Examples:

  - The strings that represent floating point numbers in Java.
- Some Languages cannot be represented by DFA. They are not regular
- any language that requires **unbounded counting** cannot be regular

  Examples:

  - $L1 = \{0^n 1^n \mid n \geq 1\}$: The set of strings consisting of one or more 0's followed by the same name of 1's.
  - $L2 = \{(^n)^n \mid n>0\}$:The language of balanced parentheses. $e.g.\ (), ()(), (()), (()()), \ldots$

Grammar

sentence –> <subject> <verb-phrase> <object>

subject –> This | Computers | I

verb-phrase –> <adverb> <verb> | <verb>

adverb –> never

verb –> is | run | am | tell

object –> the <noun> | a <noun> | <noun>

noun –> university | world | cheese | lies

leftmost derivation

- sentence –> <subject> <verb-phrase> <object>
- sentence–> This <verb-phrase> <object>
- sentence–> This <verb> <object>
- sentence–> This is <object>
- sentence–> This is a <noun>
- sentence–> This is a university

sentence –> <subject> <verb-phrase> <object>

subject –> This | Computers | I

verb-phrase –> <adverb> <verb> | <verb>

adverb –> never

verb –> is | run | am | tell

object –> the <noun> | a <noun> | <noun>

noun –> university | world | cheese | lies

Context Free Grammar(CFG)

Context free grammar G = (N, T, S, P)

- N : set of Non-terminal symbols
- T : set of Terminal symbols
- S : Start symbol ($S \in N$)
- P : set of Production rules $\{\alpha \rightarrow \beta |\ \alpha \in N \wedge \beta \in (N \cup T)^*\}$
- $N \cap T = \emptyset$

Note :

- $(N \cup T)^* = \{N \cup T \cup \varepsilon\}$
- ::= may be used instead of $\rightarrow$

CFG Example

$L1 = \{ 0^n1^n \mid n \geq 1\}$

$S \rightarrow 01$

$S \rightarrow 0S1$

- Nonterminal= {S}.
- Terminals = {0, 1}.
- Start symbol = S.
- Productions =

  $S \rightarrow 01$

  $S \rightarrow 0S1$

CFG Example

L2 $=\{(^n)^n \mid n>0\}$:The language of balanced parentheses.

$S \rightarrow (S)$

$S \rightarrow \varepsilon$

- N= {S}.
- T= {(,)}.
- Start symbol = S.
- Productions =

  $S \rightarrow (S)$

  $S \rightarrow \varepsilon$

CFG Example

$$exp \;\rightarrow\; exp\ op\ exp \mid \mathbf{(}\ exp\ \mathbf{)} \mid \boldsymbol{\mathtt{number}}$$

$$op \;\rightarrow\; \mathbf{+} \mid \mathbf{-} \mid \mathbf{*}$$

- N= {exp,op}.
- T= {(,),number,*,-,+}.
- Start symbol = exp.
- Productions =

$$exp \;\rightarrow\; exp\ op\ exp \mid (\ exp\ ) \mid \mathtt{number}$$

$$op \;\rightarrow\; + \mid - \mid *$$

Derivation

- Given a production:

  A ::= X1 X2 ... Xn

  aAb => aX1 X2 ... Xnb   is called a derivation

- A derivation is a sequence of replacements of structure names by choices on the right-hand sides of grammar rules.
  - It starts with a single structure name and ends with a string of token symbols
  - At each step in a derivation, a single replacement is made using one choice from a grammar rule.

- The set of all strings of token symbols obtained by derivations from the `exp` symbol is the *language defined by the grammar* of expressions. We can write this as:

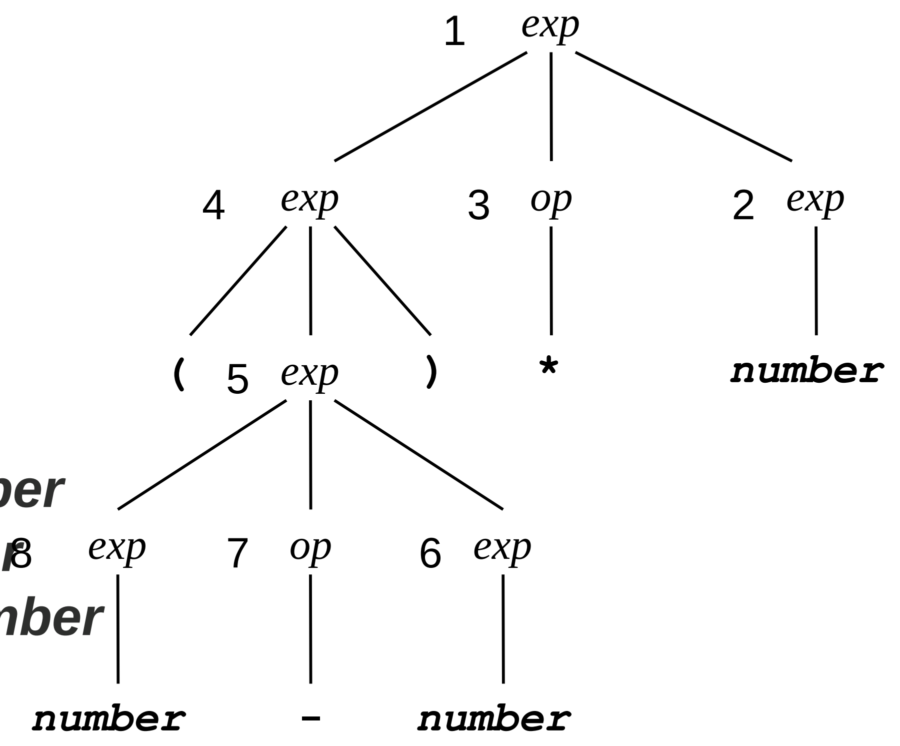
$$L(G) = \{ s: exp \Rightarrow^* s\}$$

Derivation (Example)

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

Derive `(34-3)*42`

(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***



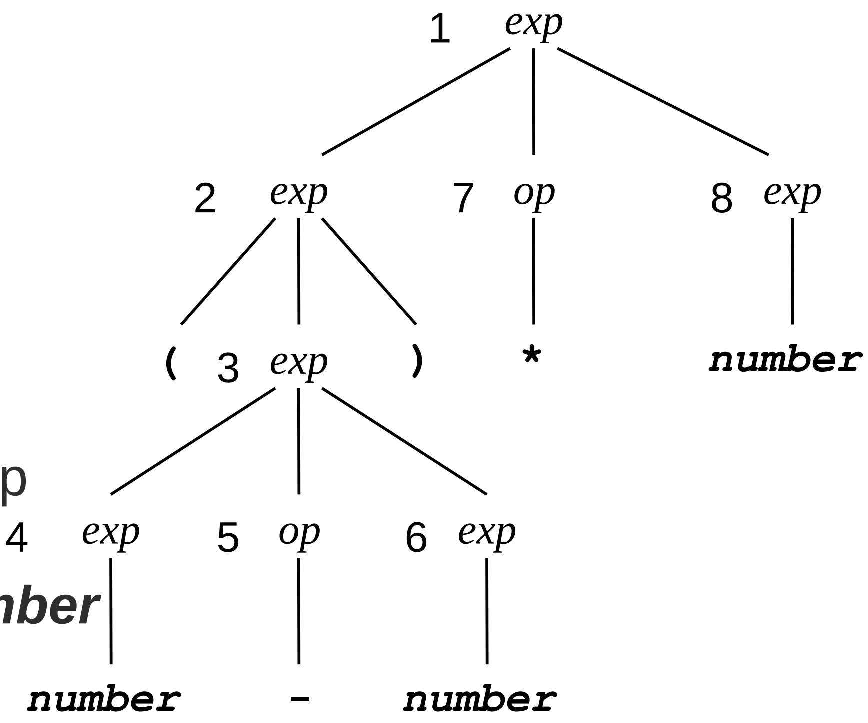
Right most derivationexpand the Rightmost nonterminal in the production

Derivation (Example)

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

Derive `(34-3)*42`

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

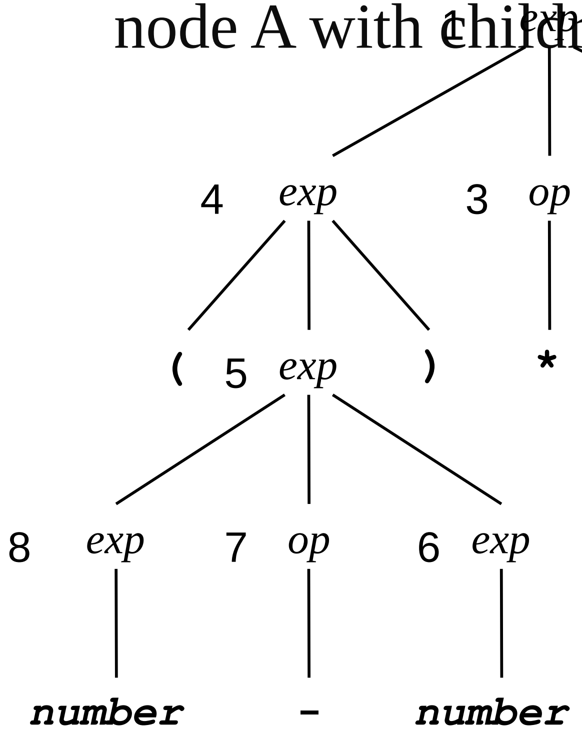


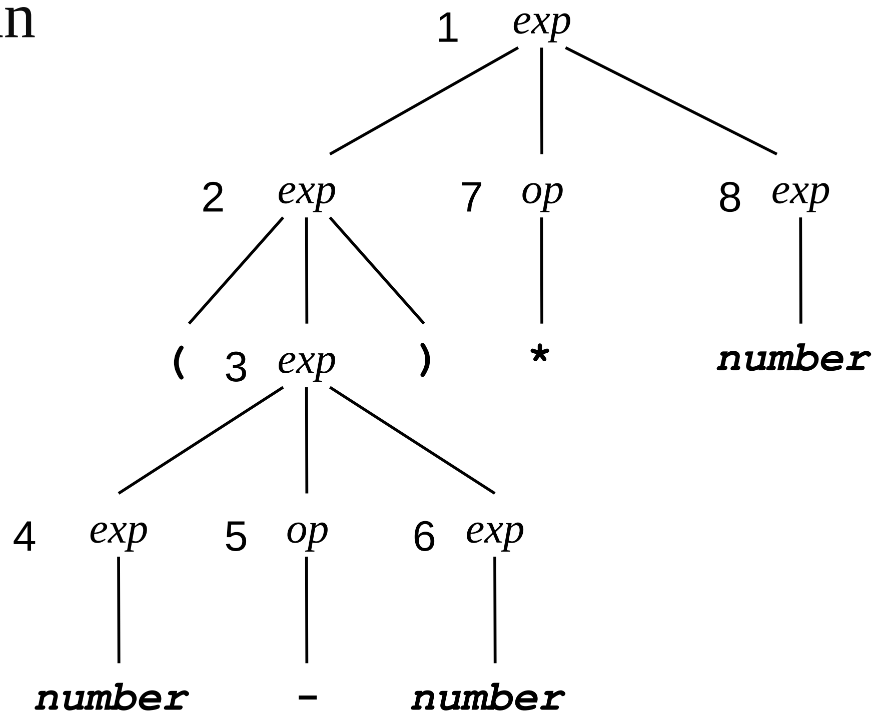
Left most derivationexpand the leftmost nonterminal in the production

Parse Tree

Parse tree:

- the start symbol →the root
- the terminals of the input sequence → leafs
- for each production A →X1 X2 ... Xn used in a derivation, a node A with children X1 X2 ... Xn





Derivations

- Derivations allow us to replace any of the variables in a string. Leads to many different derivations of the same string.
- Leftmost Derivations: expand the leftmost nonterminal in the production
- Rightmost Derivations: expand the rightmost nonterminal in the production

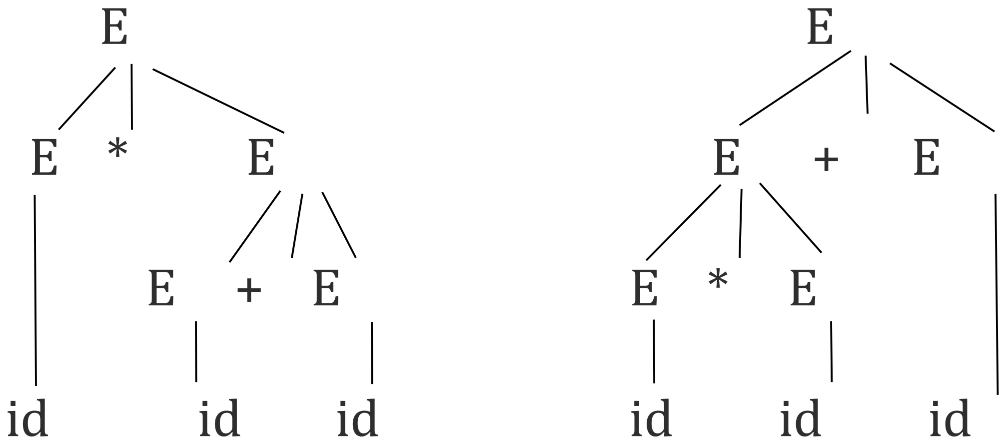
Ambiguous Grammars

E ::= E + E

| E * E

| id

String id*id+id



It is ***ambiguous***: more than one parse tree for the same input depend on derivation used.

Ambiguous Grammars

Remove Ambiguity:

1. Make the grammar left-recursive

   E →E + E’ | E’

2. Make the grammar right-recursive

   E →E’ + E | E’

3. Make the grammar non-recursive

   E ::= E’ + E’ | E’

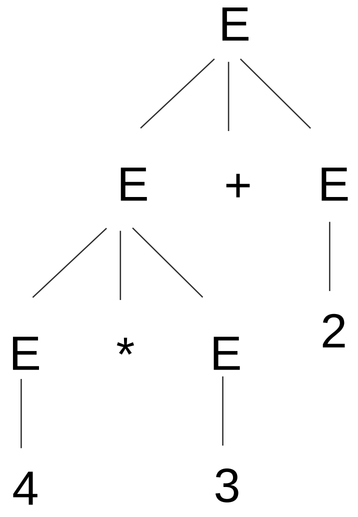
Parsing

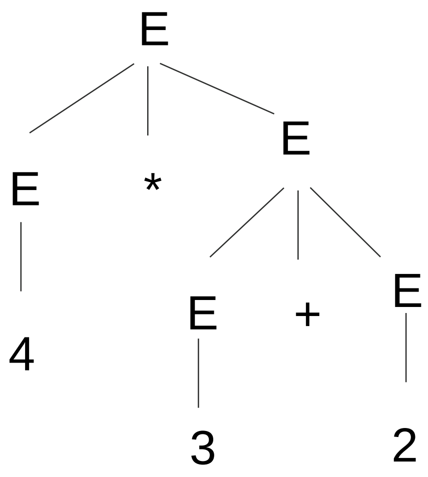
- A *leftmost* derivation corresponds to a (*top-down*) pre-order traversal of the parse tree.
- A *rightmost* derivation corresponds to a (*bottom-up*) post-order traversal, but in reverse.
- Top-down parsers construct leftmost derivations.
  - (LL = Left-to-right traversal of input, constructing a Leftmost derivation)
- Bottom-up parsers construct rightmost derivations in reverse order.
  - (LR = Left-to-right traversal of input, constructing a Rightmost derivation)

Ambiguous Grammar(priority)

E→E+E|E*E| E^E| num

String:4*3+2





<u>Remove Ambiguity</u>

E→E+T|T

T→T*F|F

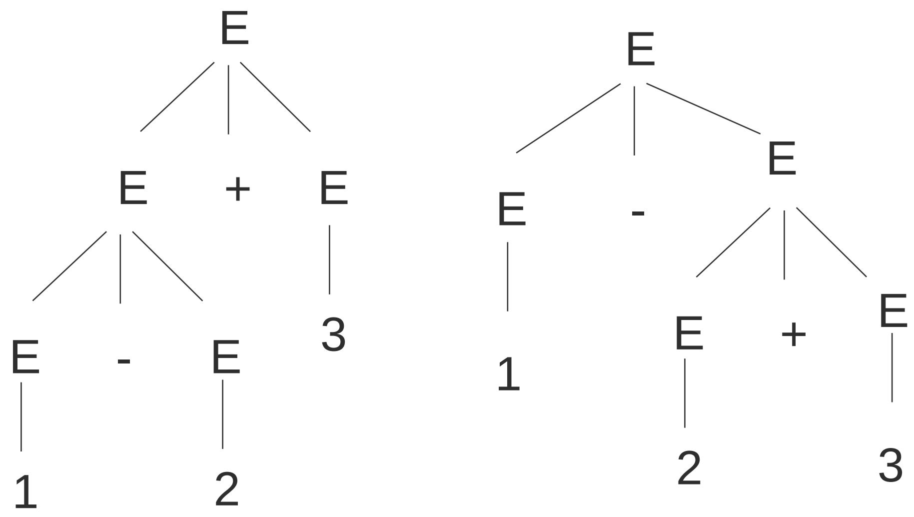
F→G^F|G

G→num

Ambiguous Grammar (associativity)

E→E+E|E-E|num

String:1-2+3



Remove Ambiguity

E→E+T|E-T|T

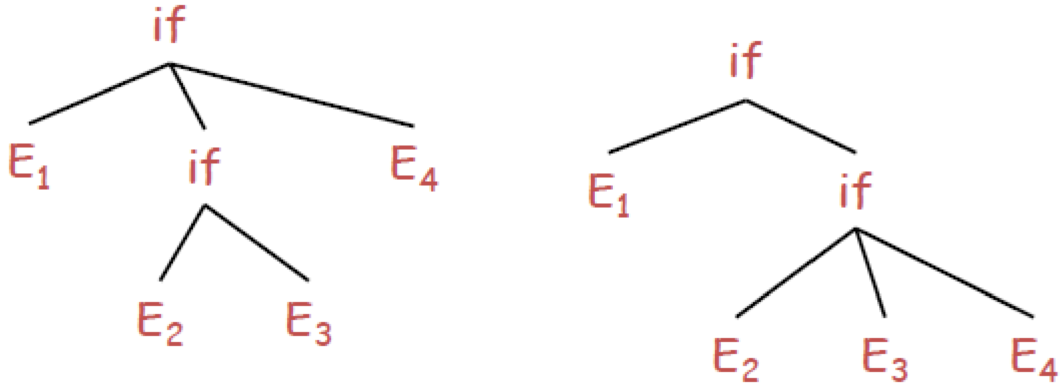
T→num

Ambiguous Grammars

E → if E then E

| if E then E else E

String : if E1 then if E2 then E3 else E4



Unambiguous Grammar

E → MIF | UIF

MIF → if E then MIF else MIF

UIF → if E then E

| if E then MIF else UIF