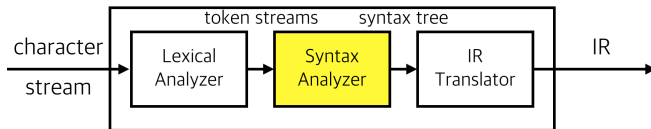


# COSE312: Compilers

## Lecture 3 — Syntax Analysis (1)

Hakjoo Oh  
2023 Spring

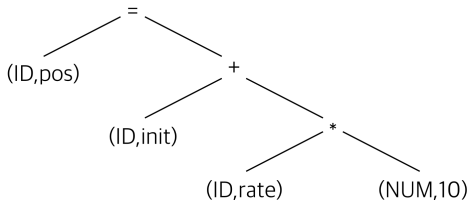
# Syntax Analysis (Parsing)



Determine whether or not the input program is syntactically valid. If so, transform the stream

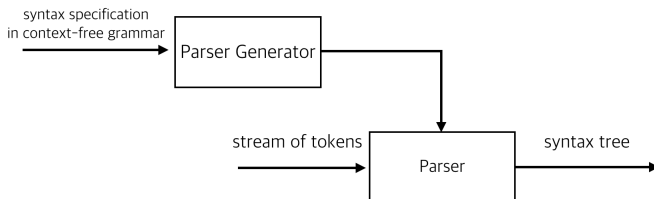
(ID, pos), =, (ID, init), +, (ID, rate), \*, (NUM,10)

into the syntax tree (or parse tree):



# Contents

- **Specification:** context-free grammars.
- **Algorithms:** top-down and bottom-up parsing algorithms
- **Tools:** automatic parser generator



# Context-Free Grammar

## Example: Palindrome

- A string is a palindrome if it reads the same forward and backward.
- $L = \{w \in \{0, 1\}^* \mid w = w^R\}$
- $L$  is not regular, but context-free.
- Every context-free language is defined by a recursive definition.
  - ▶ Basis:  $\epsilon$ ,  $0$ , and  $1$  are palindromes.
  - ▶ Induction: If  $w$  is a palindrome, so are  $0w0$  and  $1w1$ .
- The recursive definition is expressed by a context-free grammar.

$$P \rightarrow \epsilon$$

$$P \rightarrow 0$$

$$P \rightarrow 1$$

$$P \rightarrow 0P0$$

$$P \rightarrow 1P1$$

# Context-Free Grammar

## Definition (Context-Free Grammar)

A context-free grammar  $G$  is defined as a quadruple:

$$G = (V, T, S, P)$$

- $V$ : a finite set of *variables* (*nonterminals*)
- $T$ : a finite set of *terminal symbols* (tokens)
- $S \in V$ : the start variable
- $P$ : a finite set of *productions*. A production has the form

$$x \rightarrow y$$

where  $x \in V$  and  $y \in (V \cup T)^*$ .

## Example: Expressions

$$G = (\{E\}, \{(\,,\,), \text{id}\}, E, P)$$

where  $P$ :

$$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid \text{id}$$

The language includes  $\text{id} * (\text{id} + \text{id})$  because it is “derived” from  $E$  as follows:

$$\begin{aligned} E &\Rightarrow E * E \Rightarrow \text{id} * E \Rightarrow \text{id} * (E) \Rightarrow \text{id} * (E + E) \\ &\Rightarrow \text{id} * (\text{id} + E) \Rightarrow \text{id} * (\text{id} + \text{id}) \end{aligned}$$

# Derivation

## Definition (Derivation Relation, $\Rightarrow$ )

Let  $G = (V, T, S, P)$  be a context-free grammar. Let  $\alpha A \beta$  be a string of terminals and variables, where  $A \in V$  and  $\alpha, \beta \in (V \cup T)^*$ . Let  $A \rightarrow \gamma$  is a production in  $G$ . Then, we say  $\alpha A \beta$  derives  $\alpha \gamma \beta$ , and write

$$\alpha A \beta \Rightarrow \alpha \gamma \beta.$$

## Definition ( $\Rightarrow^*$ , Closure of $\Rightarrow$ )

$\Rightarrow^*$  is a relation that represents zero, or more steps of derivations:

- Basis: For any string  $\alpha$  of terminals and variables,  $\alpha \Rightarrow^* \alpha$ .
- Induction: If  $\alpha \Rightarrow^* \beta$  and  $\beta \Rightarrow \gamma$ , then  $\alpha \Rightarrow^* \gamma$ .

# Language of Grammar

## Definition (Sentential Forms)

If  $G = (V, T, S, P)$  is a context-free grammar, then any string  $\alpha \in (V \cup T)^*$  such that  $S \Rightarrow^* \alpha$  is a *sentential form*.

## Definition (Sentence)

A sentence of  $G$  is a sentential form with no non-terminals.

## Definition (Language of Grammar)

The language of a grammar  $G$  is the set of all sentences:

$$L(G) = \{w \in T^* \mid S \Rightarrow^* w\}.$$



## Derivation is not unique

At each step in a derivation, there are multiple choices to be made, e.g., a sentence  $\neg(\text{id} + \text{id})$  can be derived by

$$E \Rightarrow \neg E \Rightarrow \neg(E) \Rightarrow \neg(E + E) \Rightarrow \neg(\text{id} + E) \Rightarrow \neg(\text{id} + \text{id})$$

or alternatively by

$$E \Rightarrow \neg E \Rightarrow \neg(E) \Rightarrow \neg(E + E) \Rightarrow \neg(E + \text{id}) \Rightarrow \neg(\text{id} + \text{id})$$

# Leftmost and Rightmost Derivations

- Leftmost derivation: the leftmost non-terminal in each sentential is always chosen. If  $\alpha \Rightarrow \beta$  is a step in which the leftmost non-terminal in  $\alpha$  is replaced, we write  $\alpha \Rightarrow_l \beta$ .

$$E \Rightarrow_l -E \Rightarrow_l -(E) \Rightarrow_l -(E + E) \Rightarrow_l -(\text{id} + E) \Rightarrow_l -(\text{id} + \text{id})$$

- Rightmost derivation (canonical derivation): the rightmost non-terminal in each sentential is always chosen. If  $\alpha \Rightarrow \beta$  is a step in which the rightmost non-terminal in  $\alpha$  is replaced, we write  $\alpha \Rightarrow_r \beta$ .

$$E \Rightarrow_r -E \Rightarrow_r -(E) \Rightarrow_r -(E + E) \Rightarrow_r -(E + \text{id}) \Rightarrow_r -(\text{id} + \text{id})$$

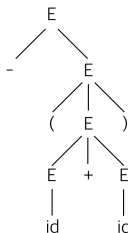
- If  $S \Rightarrow_l^* \alpha$ ,  $\alpha$  is a *left sentential form*.
- If  $S \Rightarrow_r^* \alpha$ ,  $\alpha$  is a *right sentential form*.

# Parse Tree

A graphical tree-like representation of a derivation. E.g., the derivation

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\text{id} + E) \Rightarrow -(\text{id} + \text{id})$$

is represented by the parse tree:



- Each interior node represents the application of a production.
- The interior node is labeled by the head of the production.
- Children are labeled by the symbols in the body of the production.

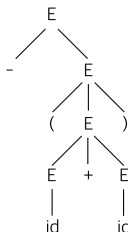
# Parse Tree

A parse tree ignores variations in the order in which symbols are replaced.  
Two derivations

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\text{id} + E) \Rightarrow -(\text{id} + \text{id})$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(E + \text{id}) \Rightarrow -(\text{id} + \text{id})$$

produce the same parse tree:



The parse trees for two derivations are identical if the derivations use the same set of rules (they apply those rules only in a different order).

# Ambiguity

A grammar is *ambiguous* if

- it produces more than one parse tree for some sentence,
- it has multiple leftmost derivations, or
- it has multiple rightmost derivations.

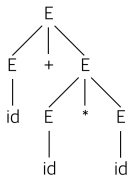
## Example

The grammar

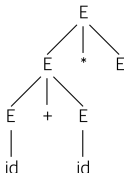
$$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid \text{id}$$

is ambiguous, because it permits two different leftmost derivations for  $\text{id} + \text{id} * \text{id}$ :

①  $E \Rightarrow E + E \Rightarrow \text{id} + E \Rightarrow \text{id} + E * E \Rightarrow \text{id} + \text{id} * E \Rightarrow \text{id} + \text{id} * \text{id}$



②  $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow \text{id} + E * E \Rightarrow \text{id} + \text{id} * E \Rightarrow \text{id} + \text{id} * \text{id}$



# Writing a Grammar

Transformations to make a grammar more suitable for parsing:

- eliminating ambiguity
- eliminating left-recursion
- left factoring

# Eliminating Ambiguity

We can usually eliminate ambiguity by transforming the grammar. E.g., an ambiguous grammar:

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$$

To eliminate the ambiguity, we express in grammar

- (precedence) bind  $*$  tighter than  $+$ 
  - ▶  $1 + 2 * 3$  is always parsed by  $1 + (2 * 3)$
- (associativity)  $*$  and  $+$  associate to the left
  - ▶  $1 + 2 + 3$  is always parsed by  $(1 + 2) + 3$

An unambiguous grammar:

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow \text{id} \mid (E) \end{aligned}$$

- parse tree for  $\text{id} + \text{id} + \text{id}$
- parse tree for  $\text{id} + \text{id} * \text{id}$



## Exercise

Transform the grammar

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow \text{id} \mid (E)$$

so that  $*$  associate to the right.

## Eliminating Left-Recursion

A grammar is left-recursive if it has a non-terminal  $A$  such that there  $A$  appears as the first right-hand-side symbol in an  $A$ -production, e.g.,

$$E \rightarrow E + T \mid T$$

To eliminate left-recursion, rewrite the grammar using right recursion:

$$E \rightarrow T E'$$

$$E' \rightarrow + T E'$$

$$E' \rightarrow \epsilon$$

# Left Factoring

The grammar

$$S \rightarrow \text{if } E \text{ then } S \text{ else } S$$

$$S \rightarrow \text{if } E \text{ then } S$$

has rules with the same prefix. We can *left factor* the grammar as follows:

$$S \rightarrow \text{if } E \text{ then } S X$$

$$X \rightarrow \epsilon$$

$$X \rightarrow \text{else } S$$

# Summary

- The syntax of a programming language is usually specified by context-free grammars.
- Basic definitions and terminologies: context-free grammar, derivation, left/rightmost derivations, parse tree, ambiguous/unambiguous grammar, grammar transformation (eliminating ambiguity, eliminating left-recursion, left factoring)