CS419 Compilers Construction

A Simple One-Pass Compiler [Chapter 2]

Lecture 5

Edited by Dr. Ismail Hababeh

Adapted from slides by Dr. Mohammad Daoud

German Jordanian University

Originally from slides by Dr. Robert A. Engelen

Syntax Definition (Language Grammar)

- A grammar describes the hierarchical structure of programming languages constructions

- Example: in Java, the if-else structure has the form

  if (expression) statement **else** statement

  Let *expr* denotes an expression

  stmt denotes a statement

  $\rightarrow$ means "can have the form"

Syntax Definition (Language Grammar)

The if-else rule structure can be expressed as:

stmt → **if** (*expr*) *stmt* **else** *stmt*

This rule is called a **production**

Lexical elements like **if, else** and **()** are called **terminals**

Variables like expr and stmt are called **nonterminals**

Language Grammar

- A context-free grammar (CFG): a set of recursive rules used to generate patterns of strings.
- CFG consists of 4 components, or 4-tuple
  - A set of *non-terminals* (each nonterminal represents a set of strings of terminals)
  - A set of (*terminal* symbols)
  - A set of *productions* based on terminals and non-terminals.
  - A designated *start symbol*

Language Grammar

- CFG = *nonterminals* , *terminals*, *productions*, *start symbol*
- A context-free grammar *G* is defined as:

  G = (V, M, P, S)

- V : nonterminals (variables) // *written in italic font*
- M : terminals // **written in bold**
- P : productions or rules
- S : start variable

Context-free Grammar - Example

Context-free grammar for simple expressions such as: 9 – 5 + 2, 3 - 1, or 7 (or list of digits separated by plus or minus signs) is defined as the 4-tuple:

$$G = <\{list,digit\},\ \{+,-,\mathbf{0},\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5},\mathbf{6},\mathbf{7},\mathbf{8},\mathbf{9}\},\ P,\ list>$$

with productions P =

$list \rightarrow list + digit$ *// first production*

$list \rightarrow list - digit$ *// second production*

$list \rightarrow digit$ *// Third production*

$digit \rightarrow \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}$

Alternative way to write the Context-free grammar - Example

$G = <\{\textit{list},\textit{digit}\}, \{+,\textbf{-},\mathbf{0},\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5},\mathbf{6},\mathbf{7},\mathbf{8},\mathbf{9}\}, P, \textit{list}>$

with productions P =

$$\textit{list} \to \textit{list} + \textit{digit} \mid \textit{list} - \textit{digit} \mid \textit{digit}$$

$$\textit{digit} \to \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}$$

Note: | means OR

Grammar Derivation

- Given a CFG, we can determine the set of all *strings* (sequences of tokens) generated by the grammar using *derivation*
  - We begin with the start symbol
  - In each step, we replace one nonterminal in the current *sentential form* (production) with one of the right-hand sides of a production for that nonterminal
- The derivation can be used to prove that a string belongs to the grammar's language.

Grammar Derivation - Example

Derivation of the string 9 – 5 + 2 from the *list* productions in the pervious example

list
⇒ *list* + *digit* *// using the first production*
sentential form ⇒ *list* **-** *digit* + *digit* *// using the second production*
⇒ *digit* **-** *digit* + *digit* *// using the third production*
⇒ **9 -** *digit* + *digit*
⇒ **9 - 5** + *digit*
⇒ **9 - 5** + **2**

This example is *leftmost derivation*, because we replaced the leftmost nonterminal (underlined) in each step

Parse Trees

- **Parsing** = *process of determining if a string of tokens can be generated by a certain grammar.*
- A parse tree shows how the start symbol of a grammar derives a string in the language
- Given a context-free grammar, the parse tree is a tree with the following properties:
  - The root of the tree is labeled by the start symbol
  - Each leaf of the tree is labeled by a terminal (token) or ε (ε denotes the *empty string*)
  - Each interior node is labeled by a nonterminal
  - If $A \rightarrow X_1\ X_2 \ldots X_n$ is a production, then node A has children $X_1, X_2, \ldots, X_n$ where X_i is a terminal, nonterminal, or ε

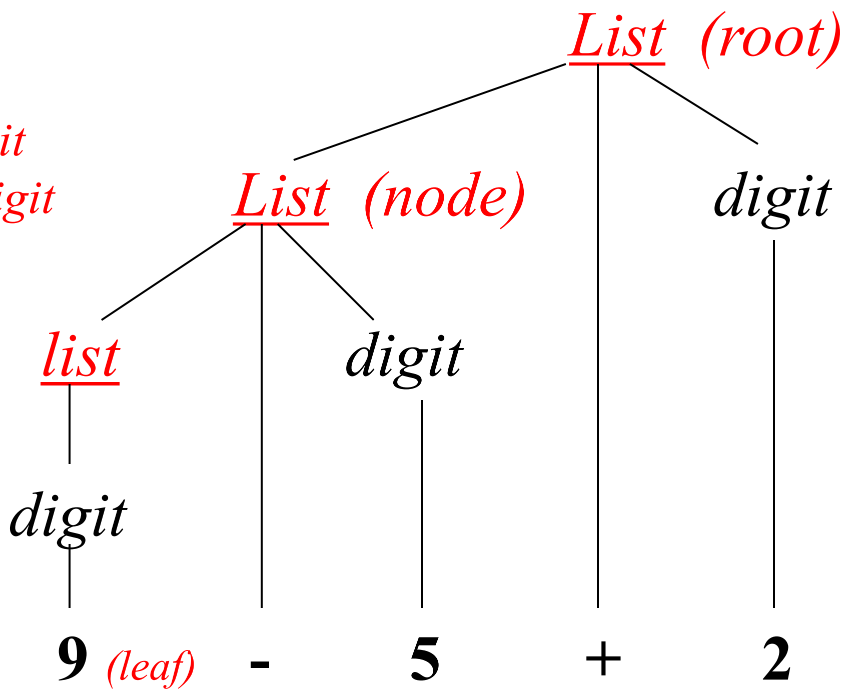
String Parsing - Example

Given the following Grammar:

G = <{*list*,*digit*}, {+,-,**0**,**1**,**2**,**3**,**4**,**5**,**6**,**7**,**8**,**9**}, *P*, *list*>

Draw the parse tree of the string **9-5+2** according to the grammar *G*

list
⇒ *list* + *digit*
⇒ *list* - *digit* + *digit*
⇒ *digit* - *digit* + *digit*
⇒ **9** - *digit* + *digit*
⇒ **9** - **5** + *digit*
⇒ **9** - **5** + **2**



The process of finding a parse tree for a given string of terminals is called *parsing that string*

The sequence of leaf's is the *yield* of the parse tree

String Parsing Ambiguity

Consider the following context-free grammar:

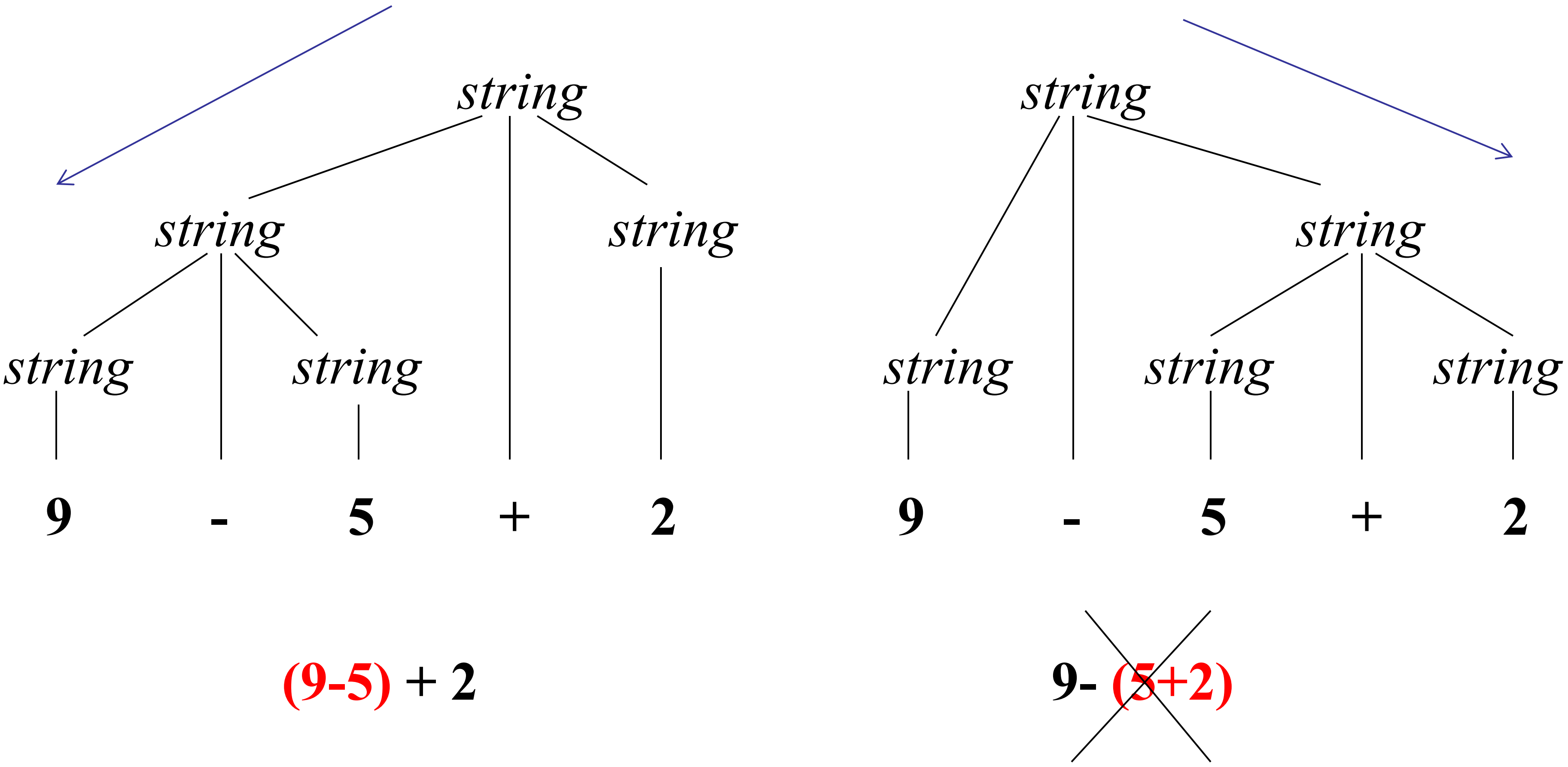
$G = \langle\{string\}, \{\mathbf{+},\mathbf{-},\mathbf{0},\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5},\mathbf{6},\mathbf{7},\mathbf{8},\mathbf{9}\}, P, string\rangle$

with productions P =

$string \to string \mathbf{+} string \mid string \mathbf{-} string \mid \mathbf{0} \mid \mathbf{1} \mid \ldots \mid \mathbf{9}$

This grammar is *ambiguous*, because more than one parse tree generates the string **9-5+2**

String Parsing Ambiguity - Example



Associativity of Operators

- *Left-associative* operators have *left-recursive* productions

  *+, -, *, / are left-associative*

  Example: String **a+b+c** has the same meaning as **(a+b)+c**

- *Right-associative* operators have *right-recursive* productions

  = is right-associative

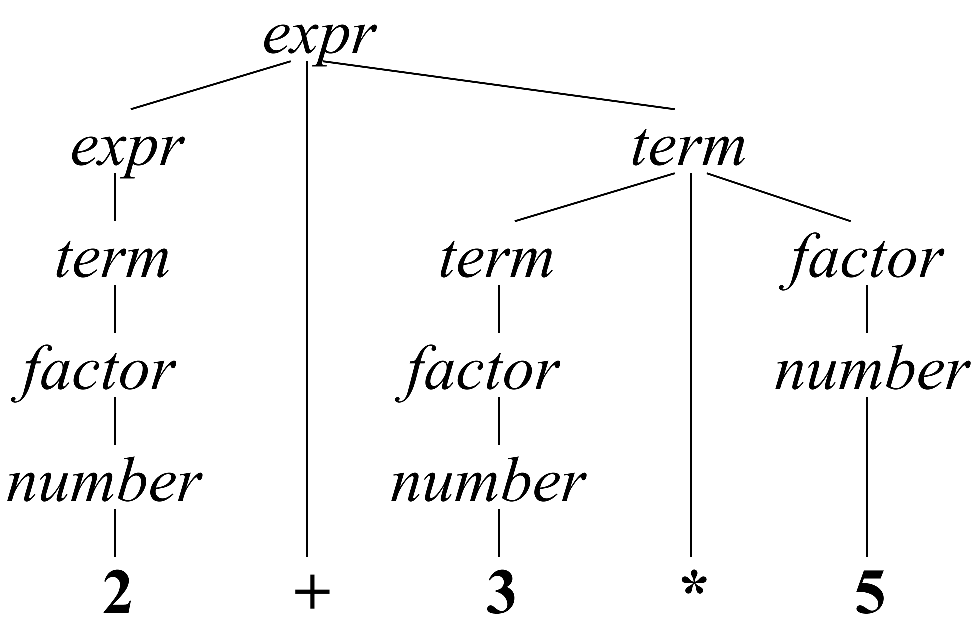
Example: String **a=b=c** has the same meaning as **a=(b=c)**

Precedence of Operators

Operators with higher precedence "bind more tightly"
Given the following productions of a grammar G:

$$expr \rightarrow expr + term \mid term$$
$$term \rightarrow term * factor \mid factor$$
$$factor \rightarrow number \mid (\ expr\ )$$
$$number \rightarrow 0 \mid \ldots \mid 9$$

String **2+3*5** has the same meaning as **2+(3*5)**



Grammar for a Subset of Java Statements

stmt → **id =** *expr;*
| **if (***expr) stmt*
| **if (***expr) stmt* **else** *stmt*
| **while** *(expr) stmt*
| **do** *stmt* **while** *(expr)***;**
| {stmts}

stmts → *stmts stmt*
| ε