Principles of Programming Languages

http://www.di.unipi.it/~andrea/Didattica/PLP-16/

Prof. Andrea Corradini
Department of Computer Science, Pisa

Lesson 3

- Structure of compilers
- Overview of a syntax-directed compiler front-end

Compilers and the Analysis-Synthesis Model of Compilation

- Compilers are **language processors**: they translate programs written in a language into equivalent programs in another language
- There are two parts to compilation:
  - **Analysis:** determines the operations implied by the source program which are recorded in a tree structure
  - **Synthesis:** takes the tree structure and translates the operations therein into the target program

Impact of Programming Language evolution on compilers

- Compilers depend on source and target language
  - Have to integrate algorithms to support new programming constructs
  - Have to make high-performance computer architecture effective
  - Optimality of translation for all input programs not decidable. Heuristics for best tradeoff necessary
- Compilers are complex and huge pieces of software. Need support for development

Building compilers

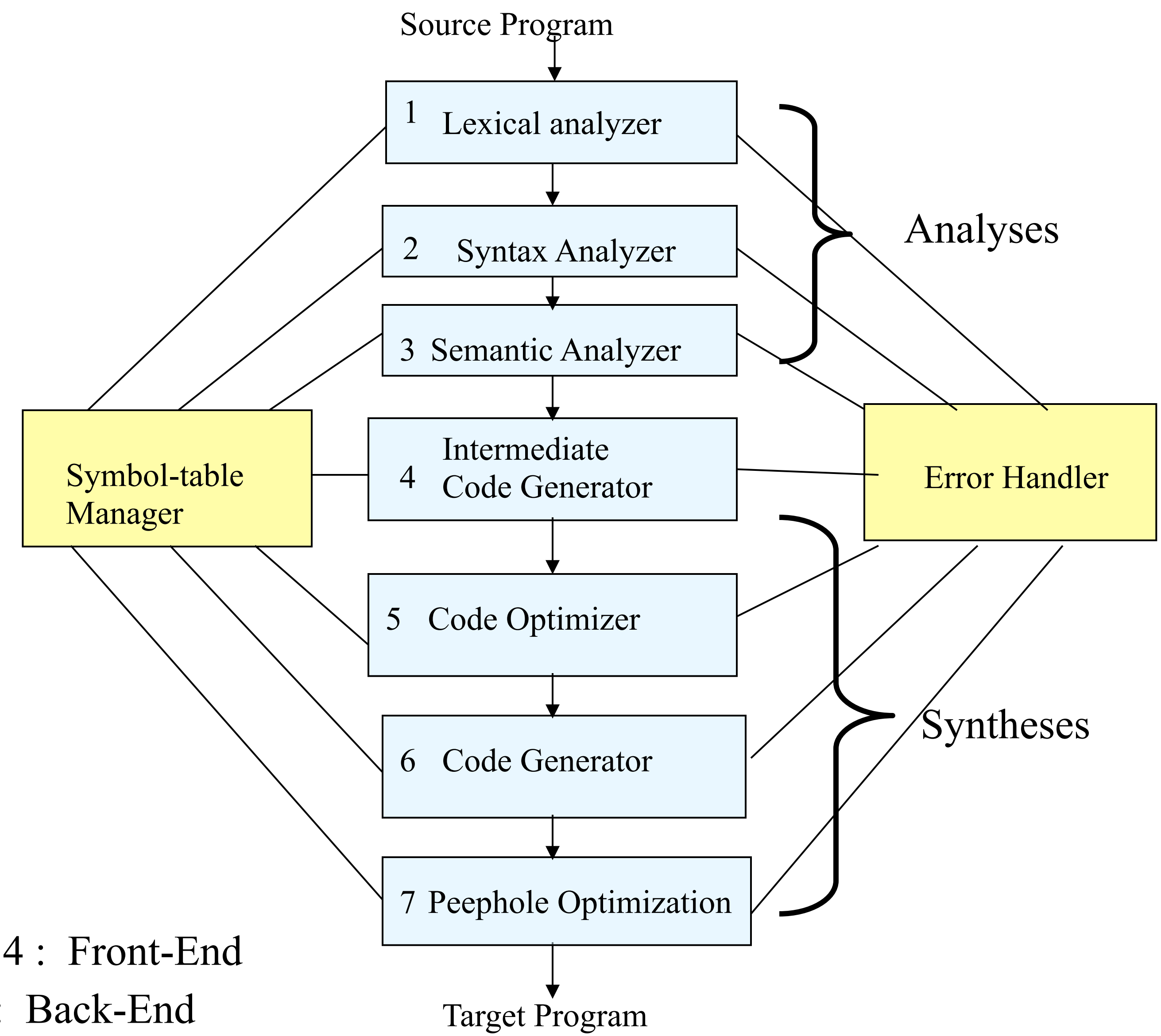
- Compiler design provide examples of real problems solved by abstracting it and applying mathematical techniques
- Is very challenging: design involves not only the compiler, but any (infinite) programs that will be translated.
- Right mathematical models and right algorithms
- Balancing generality and power vs. efficiency and simplicity

Other Tools that Use the Analysis-Synthesis Model

- Editors (syntax highlighting)
- Pretty printers (e.g. Doxygen)
- Static checkers (e.g. Lint and Splint)
- Interpreters
- Text formatters (e.g. TeX and LaTeX)
- Silicon compilers (e.g. VHDL)
- Query interpreters/compilers (Databases)

Several compilation techniques are used in other kinds of systems

Compilation goes through a set of phases



1, 2, 3, 4 : Front-End

5, 6, 7 : Back-End

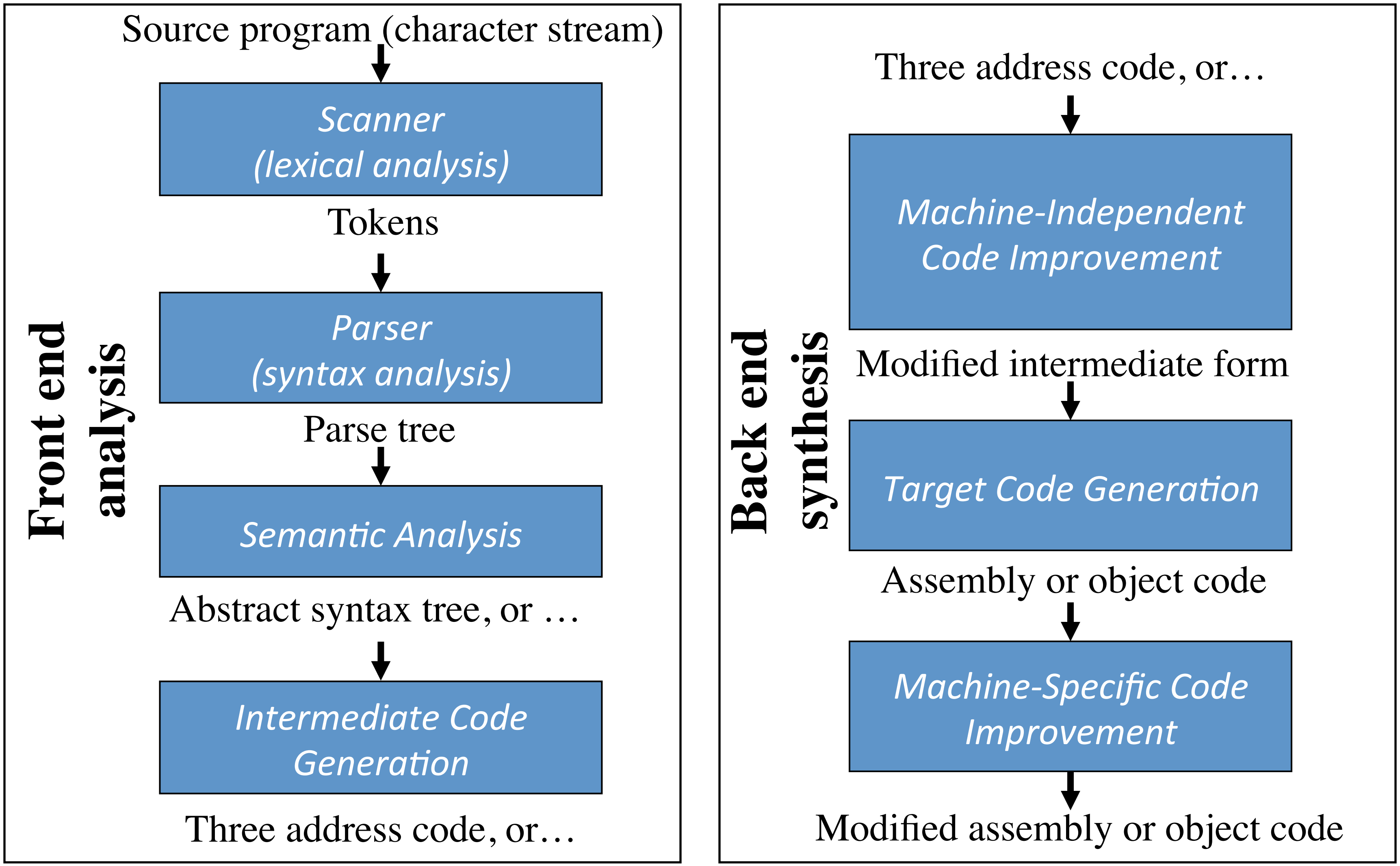
Single-pass vs. Multi-pass Compilers

- A collection of compilation phases is done only once (*single pass*) or multiple times (*multi pass*)
- **Single pass**: more efficient and uses less memory
  – requires everything to be defined before being used
  – standard for languages like Pascal, FORTRAN, C
  – Influenced the design of early programming languages
- **Multi pass**: needs more memory (to keep entire program), usually slower
  – needed for languages where declarations e.g. of variables may follow their use (Java, ADA, …)
  – allows better optimization of target code

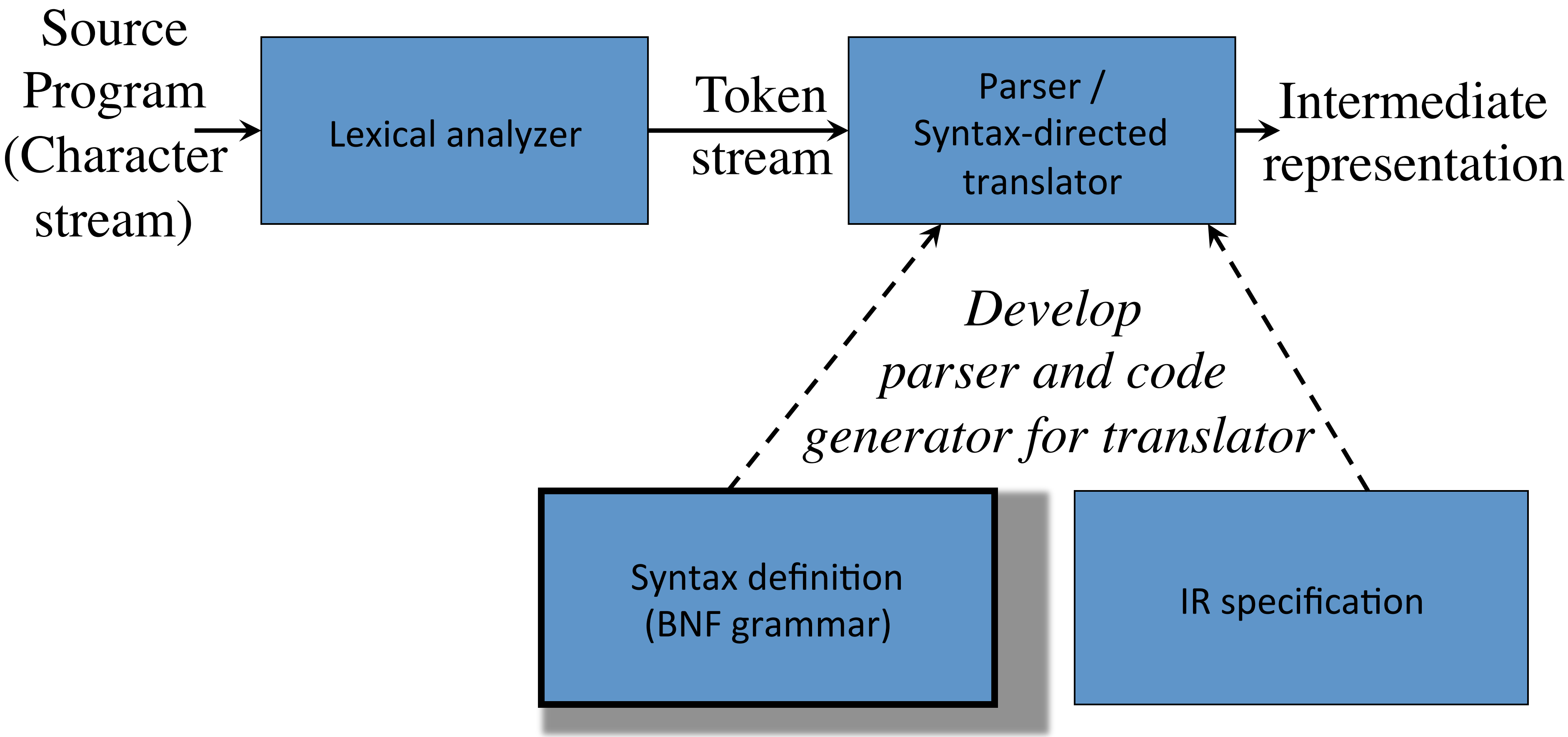
Overview of a simple syntax-directed compiler front-end

- Definition of the context-free syntax of a programming language with (Context-Free) Grammars, Chomsky hierarchy
- Parse trees and top-down predictive parsing
- Ambiguity, associativity and precedence

Compiler Front- and Back-end



The Structure of the Front-End



Syntax Definition: Grammars

- A **grammar** is a 4-tuple $G = (N, T, P, S)$ where
  - T is a finite set of tokens (*terminal* symbols)
  - N is a finite set of *nonterminals*
  - P is a finite set of *productions* of the form

    $$\alpha \rightarrow \beta$$

    where $\alpha \in (N \cup T)^* \, N \, (N \cup T)^*$ and $\beta \in (N \cup T)^*$
  - $S \in N$ is a designated *start symbol*
- **A*** is the set of finite sequences of elements of **A**. If **A** = {a,b}, **A*** = {ε, a, b, aa, ab, ba, bb, aaa, …}
- **AB** = {ab | a ∈ **A**, b ∈ **B**}

Notational Conventions Used

- Terminals
  $a,b,c,\ldots \in T$
  specific terminals: **0**, **1**, **id**, **+**
- Nonterminals
  $A,B,C,\ldots \in N$
  specific nonterminals: *expr*, *term*, *stmt*
- Grammar symbols
  $X,Y,Z \in (N \cup T)$
- Strings of terminals
  $u,v,w,x,y,z \in T^*$
- Strings of grammar symbols
  $\alpha,\beta,\gamma \in (N \cup T)^*$

Derivations

- A *one-step derivation* is defined by
  $$\gamma\ \alpha\ \delta \Rightarrow \gamma\ \beta\ \delta$$
  where $\alpha \rightarrow \beta$ is a production in the grammar
- In addition, we define
  - $\Rightarrow$ is *leftmost* $\Rightarrow_{lm}$ if γ does not contain a nonterminal
  - $\Rightarrow$ is *rightmost* $\Rightarrow_{rm}$ if δ does not contain a nonterminal
  - Transitive closure $\Rightarrow^*$ (zero or more steps)
  - Positive closure $\Rightarrow^+$ (one or more steps)
- α is a *sentential form* if $S \Rightarrow^* \alpha$
- The *language generated by G* is defined by

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

Derivation (Example)

Grammar G = ({E}, {**+**,*****,**(**,**)**,**-**,**id**}, P, E) with productions

P = $E \rightarrow E \ \mathbf{+}\ E$

$E \rightarrow E \ \mathbf{*}\ E$

$E \rightarrow \mathbf{(}\ E\ \mathbf{)}$

$E \rightarrow \mathbf{-}\ E$

$E \rightarrow \mathbf{id}$

Example derivations:

$$E \Rightarrow \mathbf{-}\ E \Rightarrow \mathbf{-\ id}$$

$$E \Rightarrow_{rm} E \ \mathbf{+}\ E \Rightarrow_{rm} E \ \mathbf{+\ id} \Rightarrow_{rm} \mathbf{id + id}$$

$$E \Rightarrow^{*} E$$

$$E \Rightarrow^{*} \mathbf{id + id}$$

$$E \Rightarrow^{+} \mathbf{id * id + id}$$

Another grammar for expressions

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

Productions $P =$

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

A *leftmost derivation*:

$$\underline{list}$$
$$\Rightarrow_{lm} \underline{list} \mathbf{+} digit$$
$$\Rightarrow_{lm} \underline{list} \mathbf{-} digit \mathbf{+} digit$$
$$\Rightarrow_{lm} \underline{digit} \mathbf{-} digit \mathbf{+} digit$$
$$\Rightarrow_{lm} \mathbf{9} \mathbf{-} \underline{digit} \mathbf{+} digit$$
$$\Rightarrow_{lm} \mathbf{9} \mathbf{-} \mathbf{5} \mathbf{+} \underline{digit}$$
$$\Rightarrow_{lm} \mathbf{9} \mathbf{-} \mathbf{5} \mathbf{+} \mathbf{2}$$

Chomsky Hierarchy: Language Classification

- A grammar G is said to be
  - *Regular* if it is *right linear* where each production is of the form
    $A \rightarrow w\,B$ or $A \rightarrow w$
    or *left linear* where each production is of the form
    $A \rightarrow B\,w$ or $A \rightarrow w$ $(w \in T^*)$
  - *Context free* if each production is of the form
    $A \rightarrow \alpha$
    where $A \in N$ and $\alpha \in (N \cup T)^*$
  - *Context sensitive* if each production is of the form
    $\alpha\,A\,\beta \rightarrow \alpha\,\gamma\,\beta$
    where $A \in N$, $\alpha,\gamma,\beta \in (N \cup T)^*$, $|\gamma| > 0$
  - *Unrestricted*

Chomsky Hierarchy

$\mathcal{L}(\mathit{regular}) \subset \mathcal{L}(\mathit{context\ free}) \subset \mathcal{L}(\mathit{context\ sensitive}) \subset \mathcal{L}(\mathit{unrestricted})$

Where $\mathcal{L}(T) = \{ L(G) \mid G$ is of type $T \}$
That is: the set of all languages
generated by grammars G of type T

Examples:

Every *finite language* is regular! (construct a FSA for strings in $L(G)$)

$L_1 = \{ \mathbf{a}^n\mathbf{b}^n \mid n \geq 1 \}$ is context free

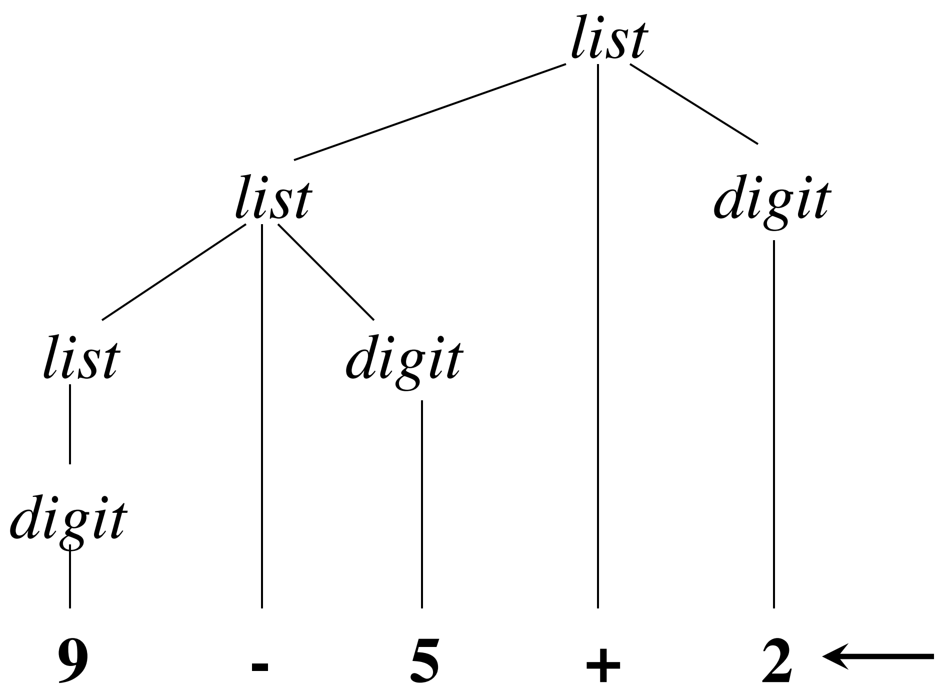
$L_2 = \{ \mathbf{a}^n\mathbf{b}^n\mathbf{c}^n \mid n \geq 1 \}$ is context sensitive

Parse Trees (context-free grammars)

- Tree-shaped representation of derivations
- The *root* of the tree is labeled by the start symbol
- Each *leaf* of the tree is labeled by a terminal (=token) or ε
- Each *internal node* is labeled by a nonterminal
- If $A \rightarrow X_1\, X_2\, \ldots\, X_n$ is a production, then node A has immediate *children* $X_1, X_2, \ldots, X_n$ where X_i is a (non)terminal or ε (ε denotes the *empty string*)

Parse Tree for the Example Grammar

Parse tree of the string **9-5+2** using grammar *G*



The sequence of leafs is called the *yield* of the parse tree

Ambiguity

Consider the following context-free grammar:

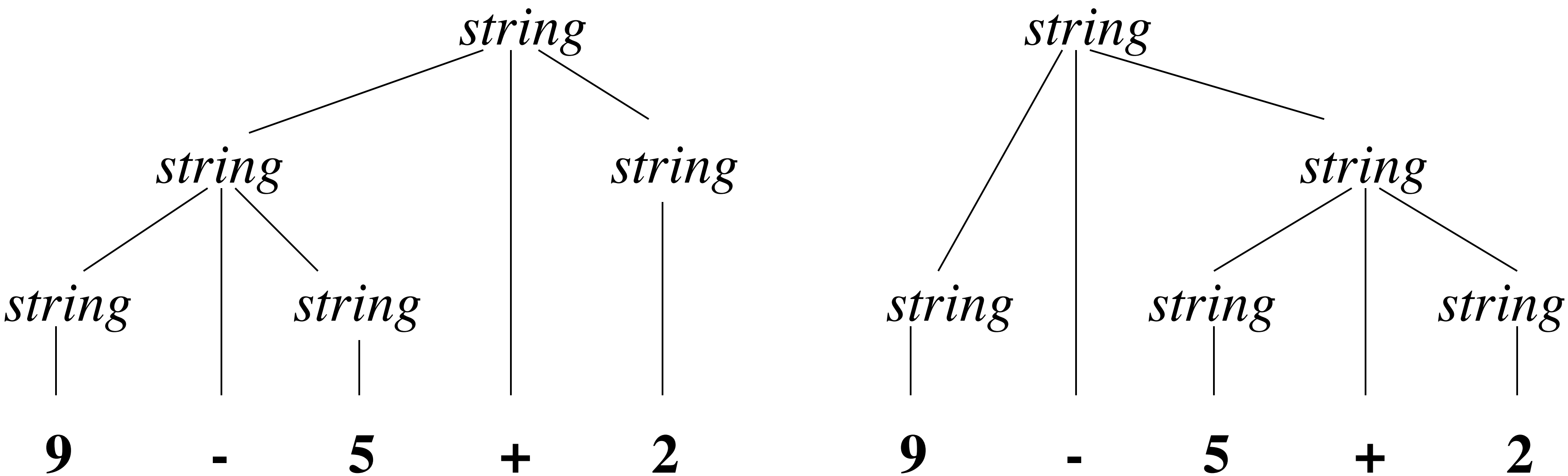
$$G = \langle \{string\}, \{\mathbf{+},\mathbf{-},\mathbf{0},\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5},\mathbf{6},\mathbf{7},\mathbf{8},\mathbf{9}\}, P, string \rangle$$

with production P =

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

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

Ambiguity (cont'd)



Associativity of Operators

Left-associative operators have *left-recursive* productions

$$\textit{left} \rightarrow \textit{left} \; \mathbf{+} \; \textit{term} \mid \textit{term}$$

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

Right-associative operators have *right-recursive* productions

$$\textit{right} \rightarrow \textit{term} \; \mathbf{=} \; \textit{right} \mid \textit{term}$$

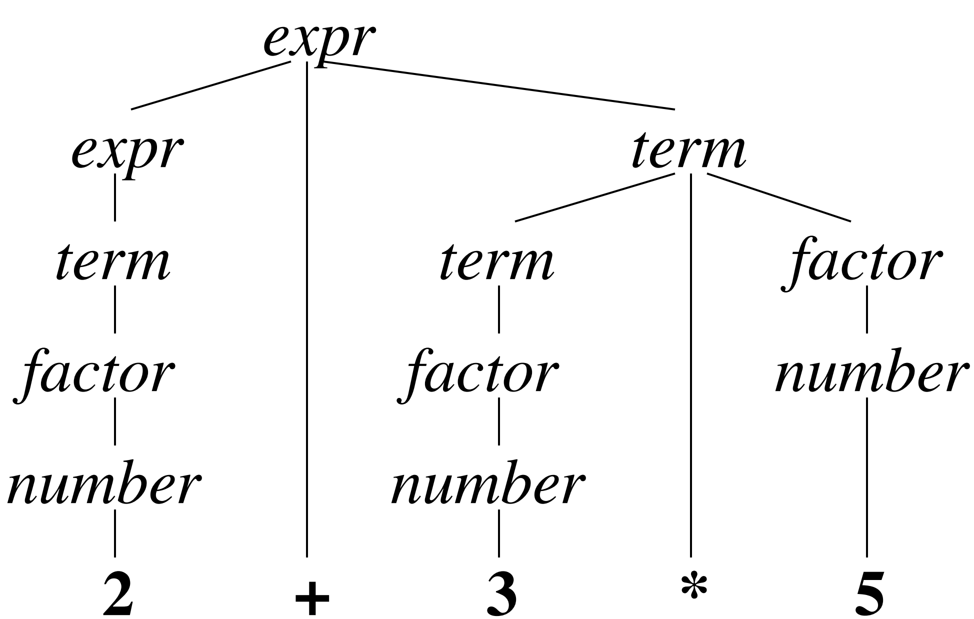
String **a=b=c** has the same meaning as **a=(b=c)**

Precedence of Operators

Operators with higher precedence "bind more tightly"

> *expr* → *expr* **+** *term* | *term*
> *term* → *term* ***** *factor* | *factor*
> *factor* → *number* | **(** *expr* **)**

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



Syntax of Statements

stmt → **id :=** *expr*
| **if** *expr* **then** *stmt*
| **if** *expr* **then** *stmt* **else** *stmt*
| **while** *expr* **do** *stmt*
| **begin** *opt_stmts* **end**

opt_stmts → *stmt* **;** *opt_stmts*
| ε