CS 4300: Compiler Theory

Chapter 2
A Simple Syntax-Directed Translator

Dr. Xuejun Liang

Outline

- This chapter is an introduction to the compiling techniques in Chapters 3 to 6 of the Dragon book
- It illustrates the techniques by developing a working Java program that translates representative programming language statements into three-address code
- The major topics are
    2. Syntax Definition
    3. Syntax-Directed Translation
    4. Parsing
    5. A Translator for Simple Expressions
    6. Lexical Analysis
    7. Symbol Tables
    8. Intermediate Code Generation

An Example Source Code

```
{
    int i; int j; float[100] a; float v; float x;
    while ( true ) {
        do i = i+1; while ( a[i] < v );
        do j = j-1; while ( a[j] > v );
        if ( i >= j ) break;
        x = a[i]; a[i] = a[j]; a[j] = x;
    }
}
```

Figure 2.1: A code fragment to be translated

The Generated Intermediate Code

```
do i = i+1; while ( a[i] < v );

do j = j-1; while ( a[j] > v );

if ( i >= j ) break;

x = a[i];

a[i] = a[j];

a[j] = x;
```

```
 1:  i = i + 1
 2:  t1 = a [ i ]
 3:  if t1 < v goto 1
 4:  j = j - 1
 5:  t2 = a [ j ]
 6:  if t2 > v goto 4
 7:  ifFalse i >= j goto 9
 8:  goto 14
 9:  x = a [ i ]
10:  t3 = a [ j ]
11:  a [ i ] = t3
12:  a [ j ] = x
13:  goto 1
14:
```

Figure 2.2: Simplified intermediate code for the program fragment in Fig. 2.1

Compiler Front End

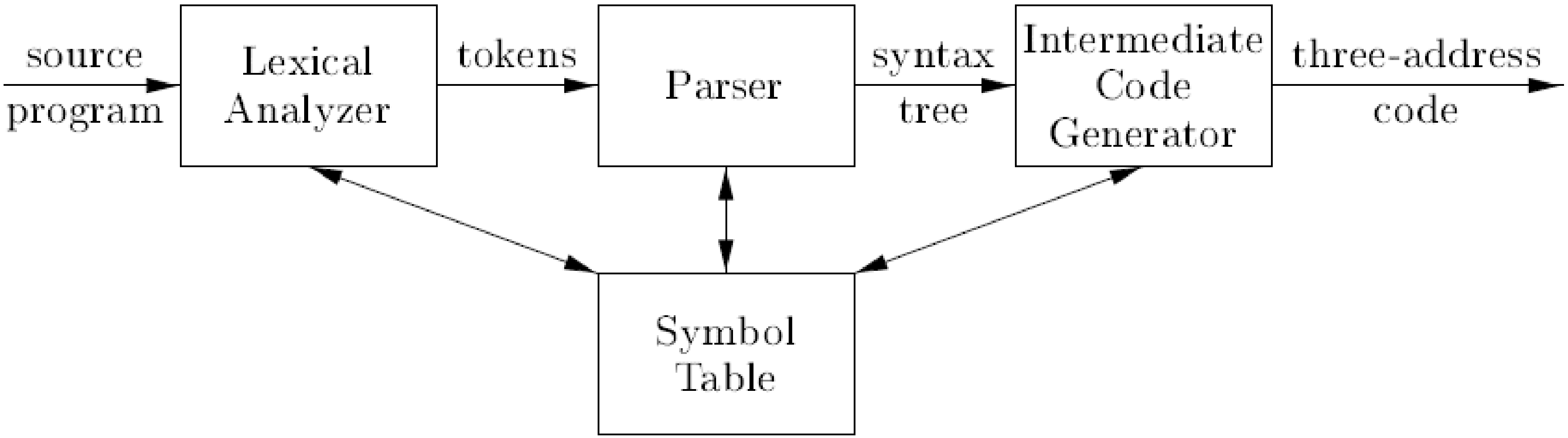


Figure 2.3: A model of a compiler front end

- For simplicity, the parser will use the syntax-directed translation of infix expressions to postfix form.
- For example, the postfix form of the expression 9-5+2 is 95-2+

2. Syntax Definition

An **if-else** statement in Java can have the form

if (expression) statement else statement

This structuring rule can be expressed as

$$stmt \rightarrow \textbf{if} \ (\ expr \ ) \ stmt \ \textbf{else} \ stmt$$

The rule called production, left side called head, and right side called body

- Context-free grammar is a 4-tuple with
  - A set of tokens (*terminal* symbols)
  - A set of *nonterminals*
  - A set of *productions*
  - A designated *start symbol*

Example Grammar

Context-free grammar for simple expressions:

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

with productions P =

$$list \rightarrow list \; \mathbf{+} \; digit$$

$$list \rightarrow list \; \mathbf{-} \; digit$$

$$list \rightarrow digit$$

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

Derivation and Parsing

- A grammar derives strings (called derivation) by
  - beginning with the start symbol and repeatedly
  - replacing a nonterminal by the body of a production for that nonterminal.
- The terminal strings that can be derived from the start symbol form the language defined by the grammar
- Parsing is the problem of taking a string of terminals and figuring out how to derive it from the start symbol of the grammar, and if it cannot be derived from the start symbol of the grammar, then reporting syntax errors within the string.

Derivation Example

$\underline{list}$
$\Rightarrow \underline{list} + digit$
$\Rightarrow \underline{list}$ - $digit + digit$
$\Rightarrow \underline{digit}$ - $digit + digit$
$\Rightarrow$ **9 -** $\underline{digit} + digit$
$\Rightarrow$ **9 - 5** + $\underline{digit}$
$\Rightarrow$ **9 - 5 + 2**

$list \rightarrow list + digit$

$list \rightarrow list$ - $digit$

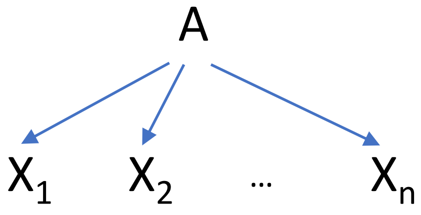
$list \rightarrow digit$

$digit \rightarrow$ **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

- This is an example *leftmost derivation*, because we replaced the leftmost nonterminal (underlined) in each step.
- Likewise, a *rightmost derivation* replaces the rightmost nonterminal in each step

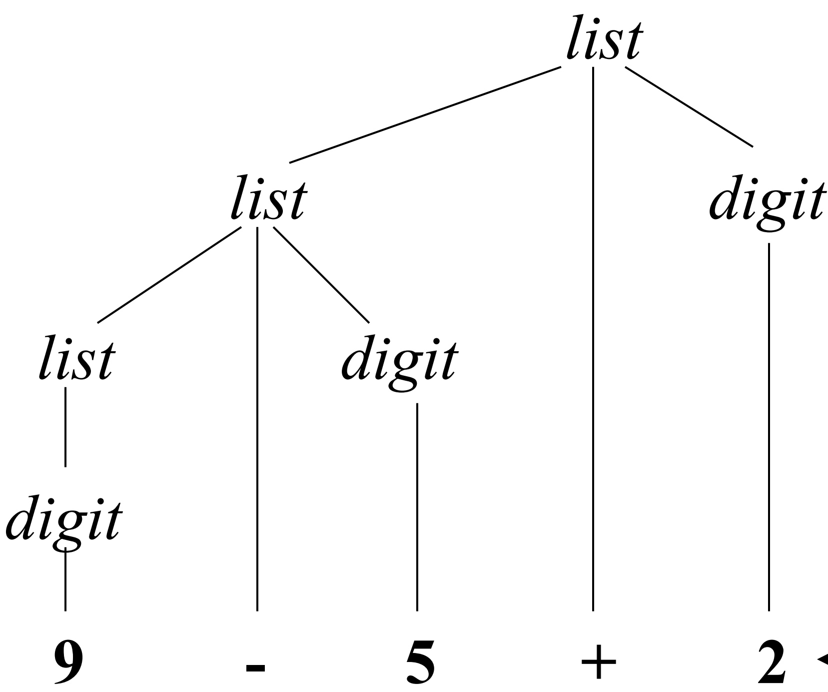
Parse Trees

- 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 *interior 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 Example

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



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

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

Ambiguity

A grammar can have more than one parse tree generating a given string of terminals. Such a grammar is said to be ambiguous.

Consider the following context-free grammar:

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

with production $P =$

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

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

Two parse trees for 9-5+2

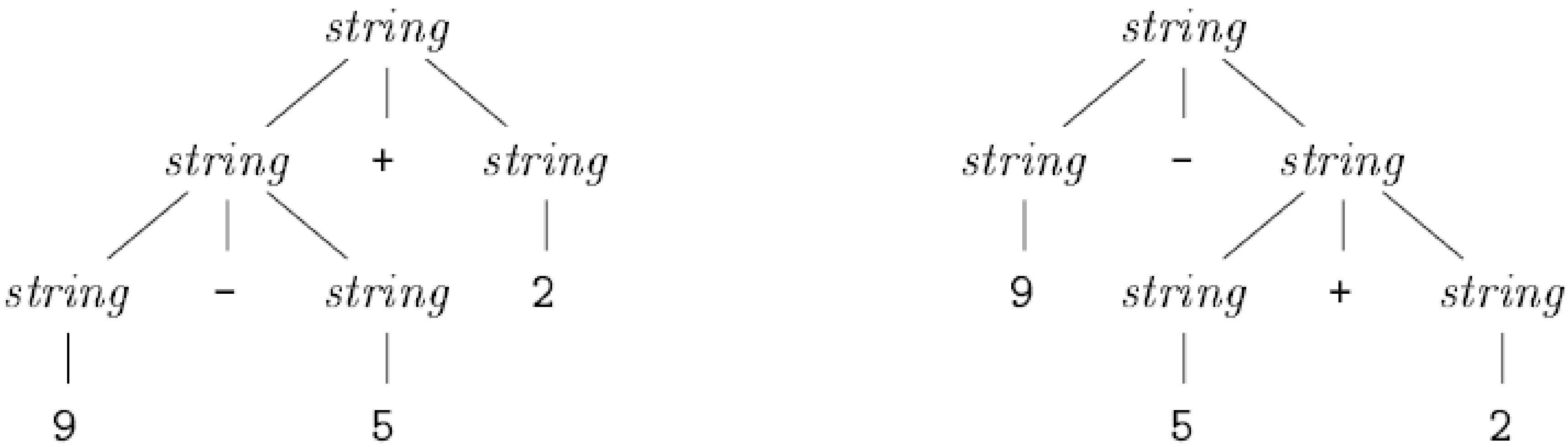


Figure 2.6: Two parse trees for 9−5+2

9-5+2 = (9-5)+2

9-5+2 = 9-(5+2)

Associativity of Operators

Left-associative operators have *left-recursive* productions

$$left \rightarrow left + digit \mid digit$$

String **9+5+2** has the same meaning as **(9+5)+2**

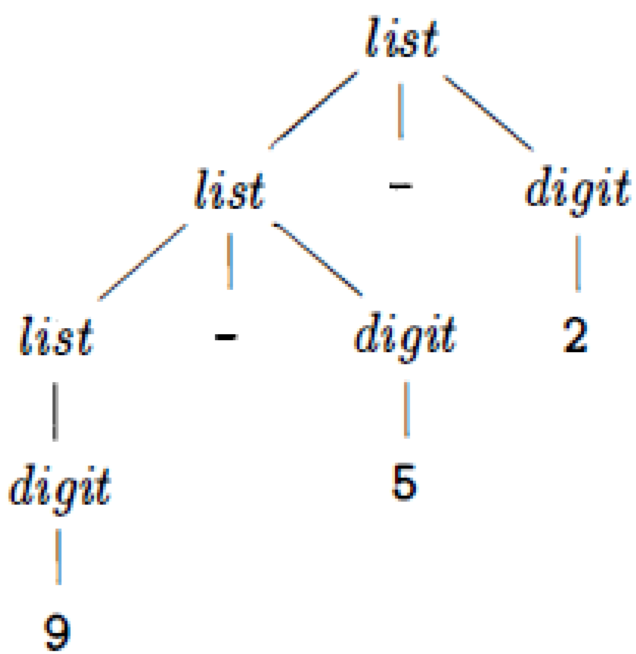
Right-associative operators have *right-recursive* productions

$$right \rightarrow letter = right \mid letter$$

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

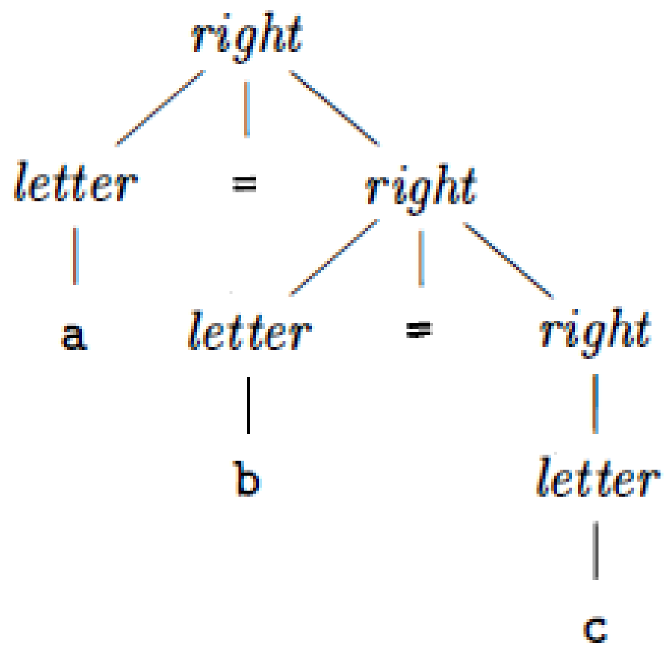
Parse trees for left- and right-associative grammars

$list \rightarrow list - digit \mid digit$



9-5-2 is (9-5)-2

$right \rightarrow letter = right \mid letter$



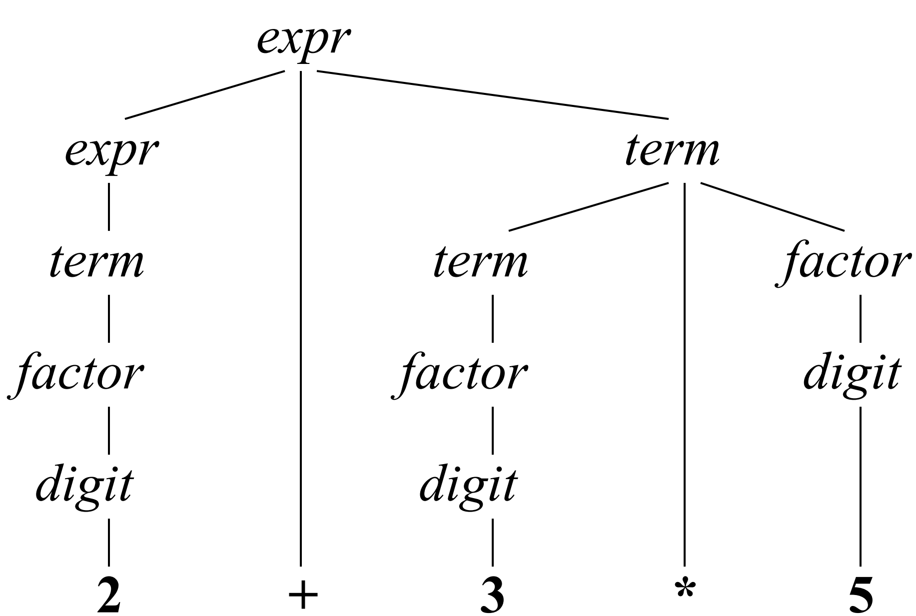
a=b=c is a=(b=c)

Precedence of Operators

Operators with higher precedence “bind more tightly”

$$
\begin{aligned}
&expr \rightarrow expr + term \mid term\\
&term \rightarrow term \;\mathbf{*}\; factor \mid factor\\
&factor \rightarrow digit \mid \mathbf{(}\; expr \;\mathbf{)}
\end{aligned}
$$

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



Syntax (Grammar)

Expressions

$$
\begin{array}{lcl}
expr & \rightarrow & expr + term \mid expr - term \mid term \\
term & \rightarrow & term * factor \mid term \; / \; factor \mid factor \\
factor & \rightarrow & \textbf{digit} \mid (\; expr \;)
\end{array}
$$

Subset of Java Statements

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{id} = expression \; ; \\
 & \mid & \textbf{if} \; (\; expression \;) \; stmt \\
 & \mid & \textbf{if} \; (\; expression \;) \; stmt \; \textbf{else} \; stmt \\
 & \mid & \textbf{while} \; (\; expression \;) \; stmt \\
 & \mid & \textbf{do} \; stmt \; \textbf{while} \; (\; expression \;) \; ; \\
 & \mid & \{ \; stmts \; \}
\end{array}
$$

$$
\begin{array}{lcl}
stmts & \rightarrow & stmts \; stmt \\
 & \mid & \epsilon
\end{array}
$$