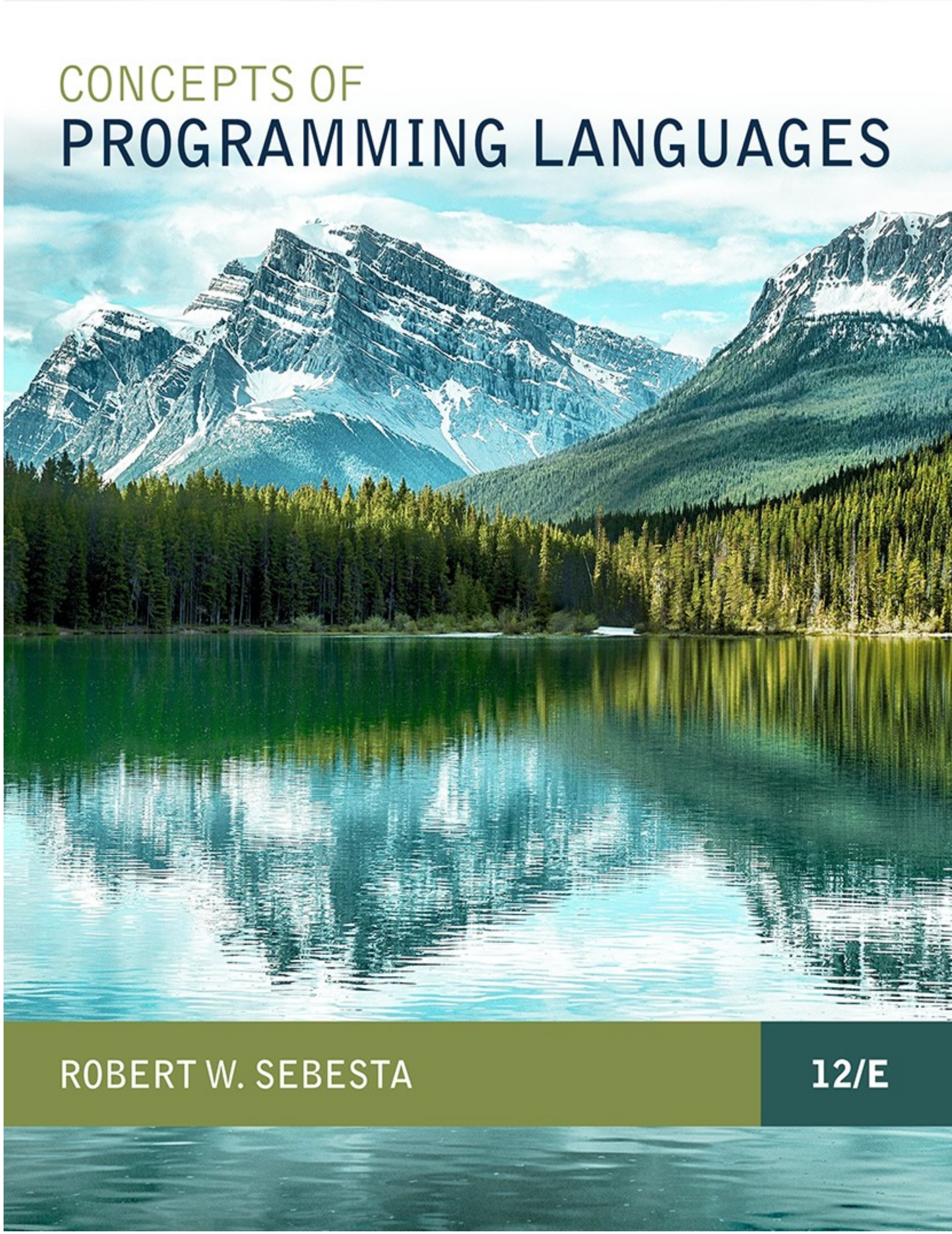
Chapter 3

Describing Syntax and Semantics



ISBN 0-321-49362-1

Chapter 3 Topics

- Introduction
- The General Problem of Describing Syntax
- Formal Methods of Describing Syntax
- Attribute Grammars
- Describing the Meanings of Programs: Dynamic Semantics

Introduction

- **Syntax:** the form or structure of the expressions, statements, and program units
- **Semantics:** the meaning of the expressions, statements, and program units
- Syntax and semantics provide a language's definition
  - Users of a language definition
    - Other language designers
    - Implementers
    - Programmers (the users of the language)

The General Problem of Describing Syntax: Terminology

- A *sentence* is a string of characters over some alphabet
- A *language* is a set of sentences
- A *lexeme* is the lowest level syntactic unit of a language (e.g., `*`, `sum`, `begin`)
- A *token* is a category of lexemes (e.g., identifier)

Parsing out an English Sentence

Mark ran very fast to get the ball.

Ball the very get. fast to ran Mark

Compile this…….

```
int myVar;

intmyVar;

int myVar = 5 + 4;

int myVar=5+4;
```

Formal Definition of Languages

- **Recognizers**
  - A recognition device reads input strings over the alphabet of the language and decides whether the input strings belong to the language
  - Example: syntax analysis part of a compiler
    - Detailed discussion of syntax analysis appears in Chapter 4
- **Generators**
  - A device that generates sentences of a language
  - One can determine if the syntax of a particular sentence is syntactically correct by comparing it to the structure of the generator

Mad Libs

- Mad Libs work because they generate sentences based on the user following the rules of the requests. The requests follow the syntax of an English sentence even though they don’t always make sense (semantics).

Mad Libs

- A Day At The Zoo! ________________(Noun) jumping up and down in its tree. He ________________(verb, past tense) ____________(adverb) through the large tunnel that led to its _________(adjective) ____________(noun).

Mad Libs in Java

_____(data type) A = new _______(class)()

+ "which is a " + Math.________(Math method)

BNF and Context-Free Grammars

- Context-Free Grammars
  - Developed by Noam Chomsky in the mid-1950s
  - Language generators, meant to describe the syntax of natural languages
  - Define a class of languages called context-free languages
- Backus-Naur Form (1959)
  - Invented by John Backus to describe the syntax of Algol 58
  - BNF is equivalent to context-free grammars

BNF Fundamentals

- In BNF, abstractions are used to represent classes of syntactic structures--they act like syntactic variables (also called *nonterminal symbols*, or just *terminals*)
- *Terminals* are lexemes or tokens
- A rule has a left-hand side (LHS), which is a nonterminal, and a right-hand side (RHS), which is a string of terminals and/or nonterminals

BNF Fundamentals (continued)

- Nonterminals are often enclosed in angle brackets
  - Examples of BNF rules:

    ```
    <if_stmt> → if <logic_expr> then <stmt>
    ```

- Grammar: a finite non-empty set of rules
- A *start symbol* is a special element of the nonterminals of a grammar

Parse the line if into a tree

`<if_stmt> → if <logic_expr> then <stmt>`

BNF Rules

- An abstraction (or nonterminal symbol) can have more than one RHS

```
<stmt> → <single_stmt>
       | begin <stmt_list> end
```

Describing Lists

- Syntactic lists are described using recursion

```
<ident_list> → ident
             | ident, <ident_list>
```

- A derivation is a repeated application of rules, starting with the start symbol and ending with a sentence (all terminal symbols)

An Example Grammar

```
<program> → <stmts>
<stmts> → <stmt> | <stmt> ; <stmts>
<stmt> → <var> = <expr>
<var> → a | b | c | d
<expr> → <term> + <term> | <term> - <term>
<term> → <var> | const
```

a = b + 5

aa

An Example Left Most Derivation

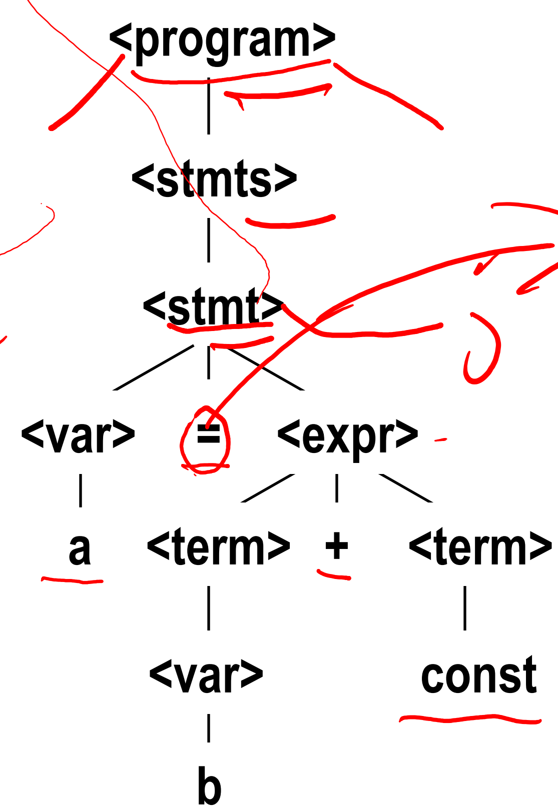
```
<program> => <stmts> => <stmt>
                    => <var> = <expr>
                    => a = <expr>
                    => a = <term> + <term>
                    => a = <var> + <term>
                    => a = b + <term>
                    => a = b + const
```

Derivations

- Every string of symbols in a derivation is a *sentential form* A **sentential form** is any string derivable from the start symbol.
- A *sentence* is a sentential form that has only terminal symbols
- A *leftmost derivation* is one in which the leftmost nonterminal in each sentential form is the one that is expanded
- A derivation may be neither leftmost nor rightmost

Parse Tree

- A hierarchical representation of a derivation



a = b + const

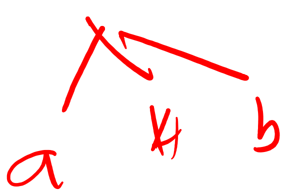
Syntax

Ambiguity in Grammars

- A grammar is *ambiguous* if and only if it generates a sentential form that has two or more distinct parse trees

<expr> → <expr> <op> <expr> | const

<op> → / | -



$\boxed{7 * 2 + 3}$

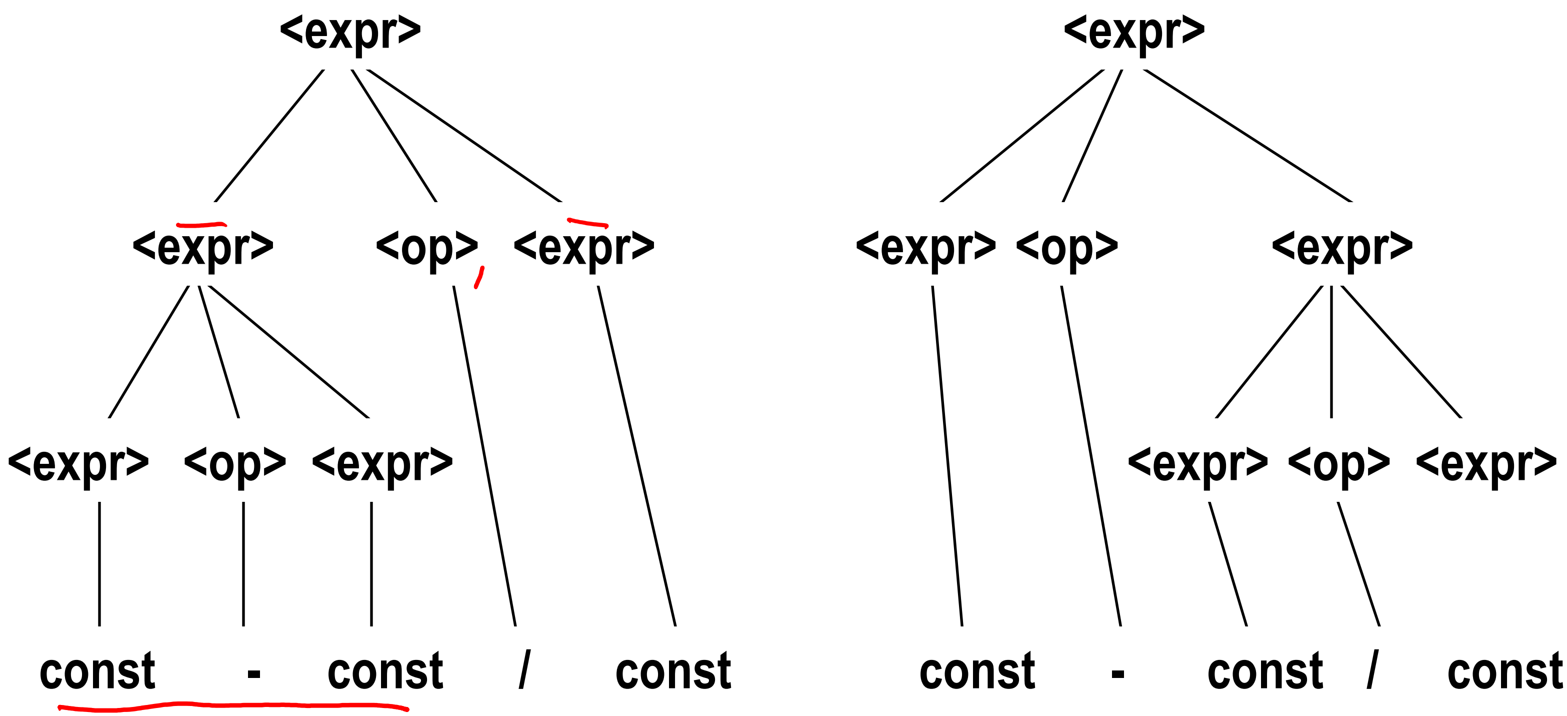
17

35

An Ambiguous Expression Grammar

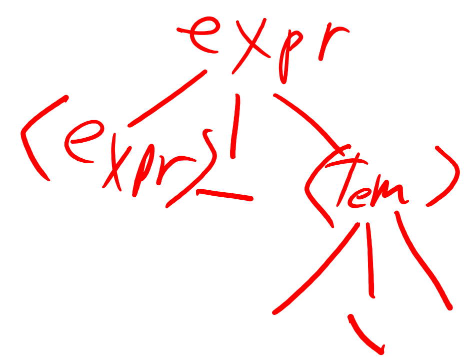
<expr> → <expr> <op> <expr> | const

<op> → / | -



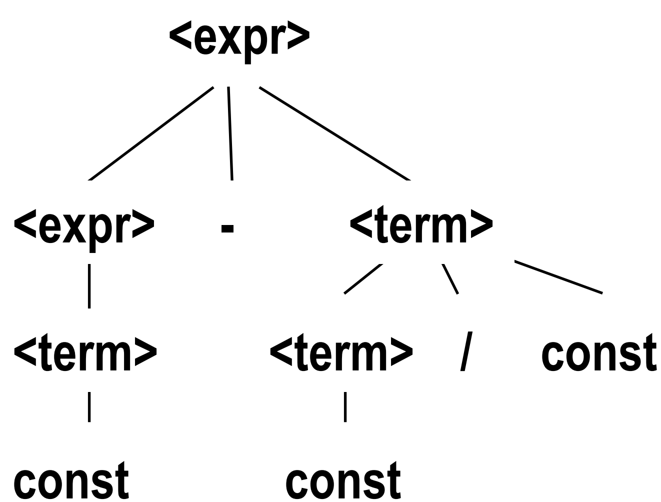
An Unambiguous Expression Grammar

- If we use the parse tree to indicate precedence levels of the operators, we cannot have ambiguity




```
<expr> → <expr> - <term>  |  <term>
<term> → <term> / const| const
```

a b

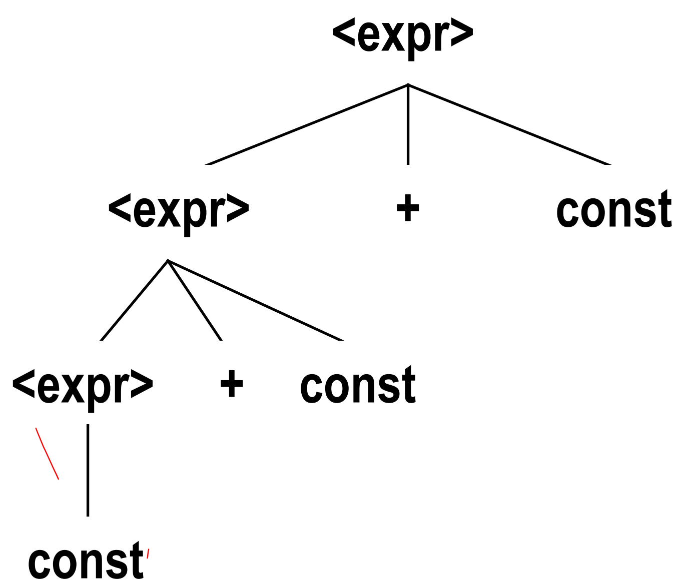


Associativity of Operators

- Operator associativity can also be indicated by a grammar

```
<expr> -> <expr> + <expr> |  const  (ambiguous)
<expr> -> <expr> + const  |  const  (unambiguous)
```

a+b+c



Extended BNF

- Optional parts are placed in brackets [ ]

```
<proc_call> -> ident [(<expr_list>)]
```

- Alternative parts of RHSs are placed inside parentheses and separated via vertical bars

```
<term> → <term> (+|-) const
```

- Repetitions (0 or more) are placed inside braces { }

```
<ident> → letter {letter|digit}
```

BNF and EBNF

- BNF

```
<expr> → <expr> + <term>
       | <expr> - <term>
       | <term>
<term> → <term> * <factor>
       | <term> / <factor>
       | <factor>
```

- EBNF

```
<expr> → <term> {(+ | -) <term>}
<term> → <factor> {(* | /) <factor>}
```

Recent Variations in EBNF

- Alternative RHSs are put on separate lines
- Use of a colon instead of `=>`
- Use of $_{\texttt{opt}}$ for optional parts
- Use of `oneof` for choices

Static Semantics

- Nothing to do with meaning
- Context-free grammars (CFGs) cannot describe all of the syntax of programming languages
- Categories of constructs that are trouble:
  - Context-free, but cumbersome (e.g., types of operands in expressions)
  - Non-context-free (e.g., variables must be declared before they are used)

- Switch(var){
- case 3:
- print()
- break;
- case 4:
- print()
- default:
- print()
- }

has (switch)

has

optional

has

<switch>-> switch (<expr>) { case <item>: <stmt_lists>

{case <item>: <stmt_lists> }

[default : <stmt_lists>]

}

