Foundations of Programming Languages

Seyed H. Roosta

Chapter Four

Syntax Specification

Formal specification of a programming language

- Help language comprehension
- Supports language standardization
- Guides language design
- Aids compiler and language system writing
- Supports program correctness verification
- Models software specification

- The only restriction on a language is that each string must be finite length and must contain characters chosen from some fixed finite alphabet of symbols

- Ex:
  - If a<b then x else y fi.
  - Keyword: if,fi,else,then
  - Symbols {a,b,x,y}

- Any programming language description can be classified
  - Syntax– formation of phrases
  - Semantics– meaning of phrases
  - Pragmatics– practical use of phrases

- A programming language definition enable us to determine whether the program is valid and understand its underlying meaning

- Syntax : similar to the grammer of natural language
- semantics: interaction
- Pragmatics: translator

and implementation of programming languages

- Regular expressions
- Formal grammars
- Attribute grammars

Regular expression

- Invented by Stephen Kleene 1950
- The alphabet of the language is a finite set of symbols athat are assembled to form the strings or sentences of the language

Regular expression

- Alternation (a+b)
- Concatenation (a*b)
- Kleene closure a*
- Positive closure a^+
- Empty $\emptyset$
- Atom {a}

ex

- L1L2={01,1001} and L2={11,00,1}
- L1L2={0111,0100,011,10011,100100,10011}
- L1+L2={01,1001,11,00,1}

ex

- {00}
- (0+1)*
- (0+1)*00(0+1)*
- (1+10)*
- 0*1*2
- $0^+1^+2^+$
- 11*22*

Formal grammars

- Each programming language has a vocabulary of symbols and rules for how these symbols may be put together to form phrases.

Formal Grammars

- while expression do command
- if (expression)
  - Statement1
- else
  - statement2

SYNTAX SPECIFICATION

- A grammar defines the set of all possible phases that constitute programs in the subject language together with their syntactic structures.

- The grammar of a programming language consists of four components
  - A set symbols known as terminal symbols that are the atomic symbols in the language.
  - A set of nonterminal symbols known as variable symbols
  - A set of rules known as production rules that are used to define the formation of the constructs.
  - A variable symbol, or distinguished symbol, called start symbol.

- Each string in the derivation is called a sentential form.
- A language is formally defined as the set of sentential forms wherein each form consists solely of terminal symbols that can be derived from the initial symbol of a grammar.
- The derivation continues until the sentential form contain no variable.

- C programming language contains more than 150 rules.

p.112

- The length of a string is the number of symbols in it. The empty string denoted   ,has length 0
- The notation Sn is used for the set od strings over S with length n(n>0)
- The notation S* is used for the set of all finite strings over S of any length,or

Classification of grammars

- Type 0 grammar
- Type 1 Grammar
- Type 2 Grammar
- Type 3 Grammar

Type 0 grammar

- Unrestricted grammar
- Recursively enumerable
- Phrase structured

Exmaple

- A unrestricted grammar for describing strnig aaaa can be defined as G=(V,T,P,S) where
- V={S,A,B,C,D,E}
- T={a, }
- S={S}
- The production rules are as follows
- 1. S=> ACaB
- 2. Ca=> aaC
- 3.CB=>DB
- 4.CB=>E
- aD=>Da
- AD=>AC
- aE=>Ea
- AE=>

Type 1 Grammar

- Context-sensitive grammar
- A Thing => Thing a b.

ex

- G=(V,T,P,S)
- V={Sentence,Thing,Other}
- T={a,b,c}
- S={Sentence}
- P:
  - Sentence =>a b c
  - Sentence => a Thing b c
  - Thing b => b thing
  - Thing c => Other b c c
  - a Other a a
  - a Other a a Thing
  - b Other => Other b

- a a b b c c ?

Type 2 Grammar

- Context-free grammar
  - Expression => Value + Expression
- Backus-Naur Form(BNF)
- Chomsky's type 2 garmmar

<expression> ::=<value>+<expression>

- Left side of a production rule is a single variable symbol.
- Right side is a combination of ternimal and variable symbols

ex

- G=(V,T,P,S)
- V={Real-Number,Integer-Part,Fraction,Digit}
- T={0,1,2,3,4,5,6,7,8,9}
- S={Real-Number}
- P:
  - Real-Number=>Integer-Part.Fraction
  - Ineger-Part=>Digit
  - Integer-Part=>Integer-Part Digit
  - Fraction=>Digit
  - Fraction=>Digit Fraction
  - Digit=>0|1|2|3|4|5|6|7|8|9

ex

- 125.78

ex

- Grammar for a calculator language in BNF notation

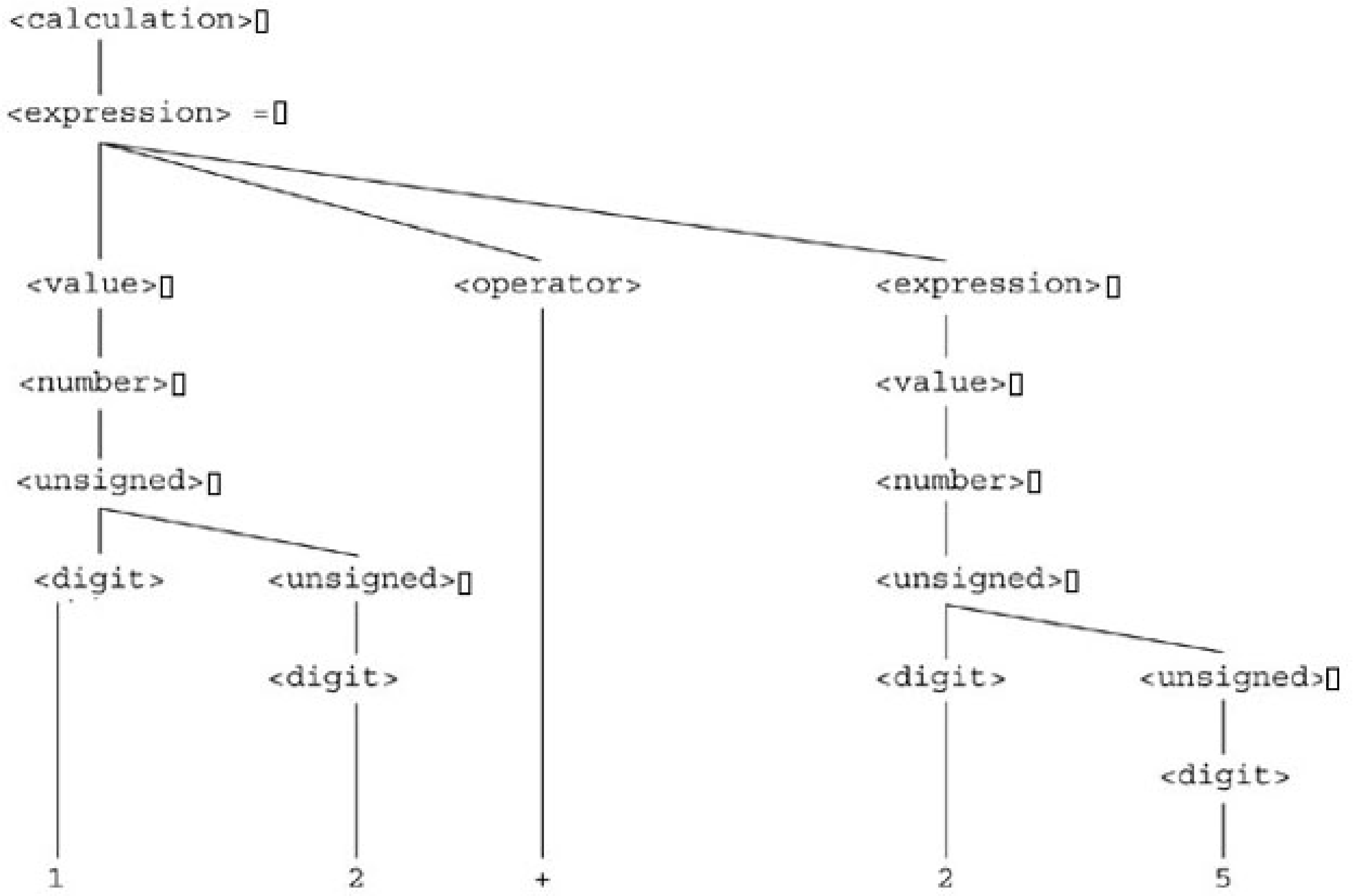


Figure 4.1 Tree representation for 12 + 25 =

Ex:

- S=>0S|1A
- A=>0S|1B
- B=>0S|1C|1
- C=>1C|0C|1|0
- 011101011?

Type3 grammar

- Regular grammar
- Restrictive grammar
- Only one terminal or one terminal and one variable on the right side of the production rules
- Right-linear/left –linear grammar

- Right-linear grammar
  - A=>xB or A=>x
- Left-linear grammar
  - A=>Bx or A=>x

- A complete grammar is a set of production rules that together define a hierarchy of constructs leading to a synatactic category, which for a programming language is called a program.
- Ex:
  - Thing =>a Thing
  - Thing =>Thing a

SYNTAX Tree

- The syntax of a programming language is commonly divided into two parts:
  - The lexical syntax that describes the smallest units with significance called token,
  - the phrase-structure syntax that explains how the token are arranged into programs

- Grammar-oriented compiling technique – syntax-oriented translation
- Lexical analyzer (scanner)
  - Convert the stream of input characters to a stream of tokens that becomes the input to the second phase of the process
- Syntactic analyzer
  - Is a combination of a parser and an intermediate code generator and forms a derivation tree from the token list based on the syntax definition of the programming language

- Basic approaches:
  - Top-down parsers
  - Bottom-up parsers

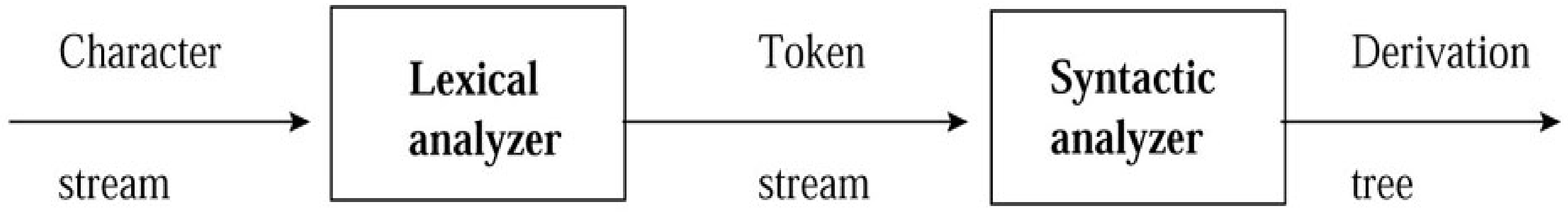


Figure 4.3 Program translation by scanner and parser

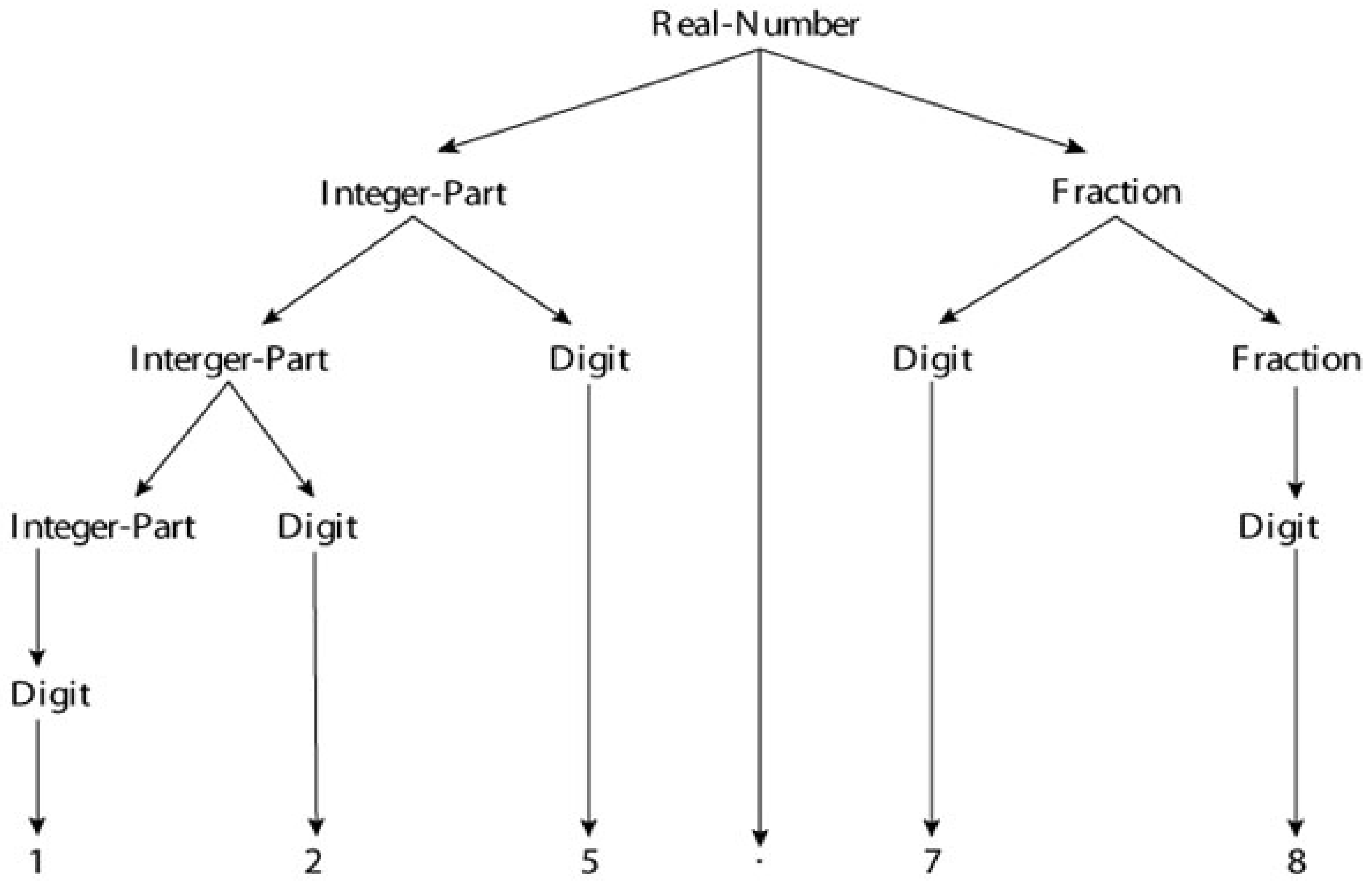


Figure 4.4 A top-down parse tree for the real number 125.78

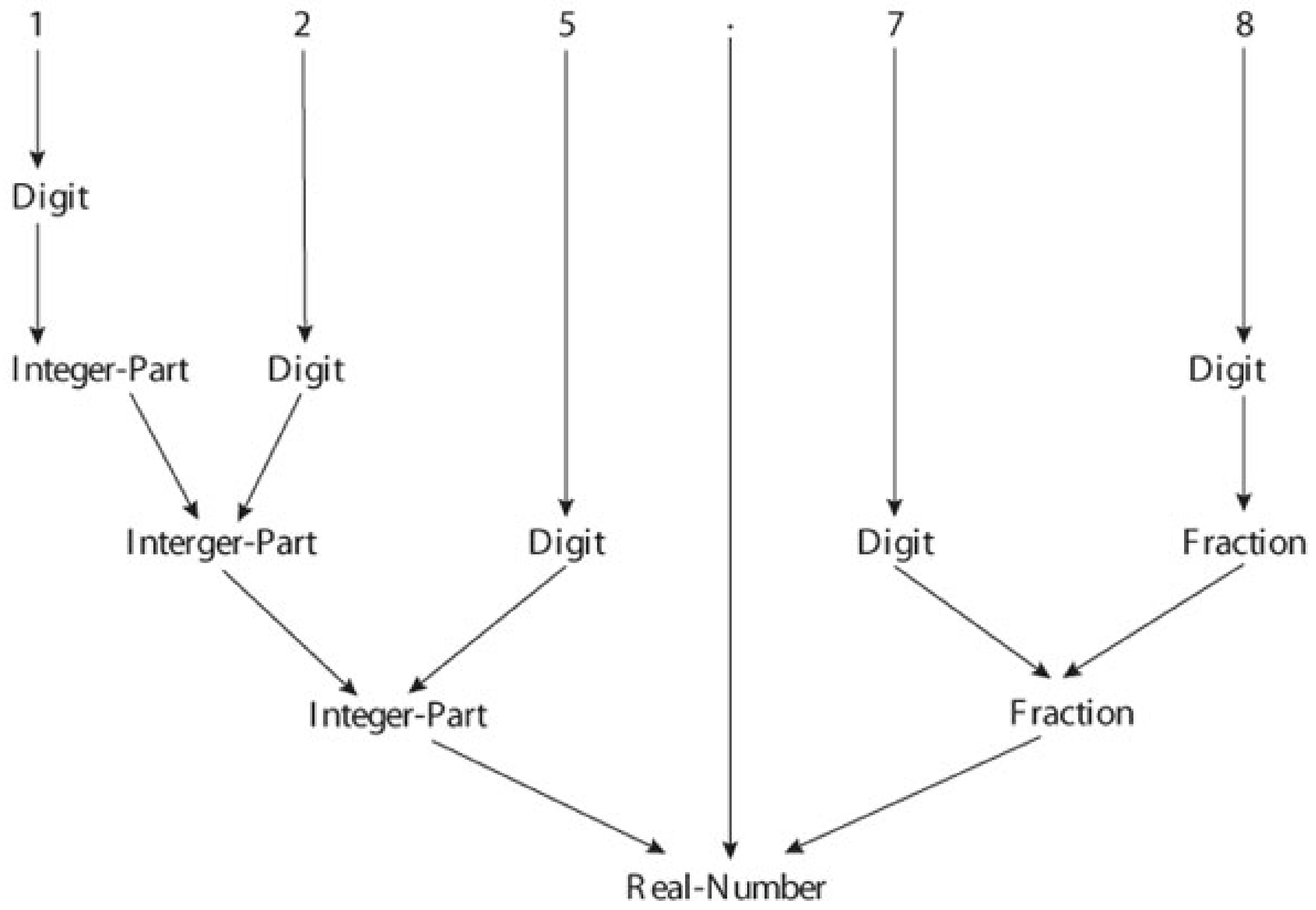


Figure 4.5 A bottom-up parse tree for the real number 125.78

- Well-known UNIX tools
  - LEX
  - YACC

- Derivation tree has the following properties
  - Each terminal node is labeled with a terminal symbol
  - Each internal node is labeled with a variable symbol
  - The label of an internal node is the left side of the production rule, and the labels of the children of the node, from left to right, and the right side of that production rule
  - The root of the tree is labeled with the start symbol

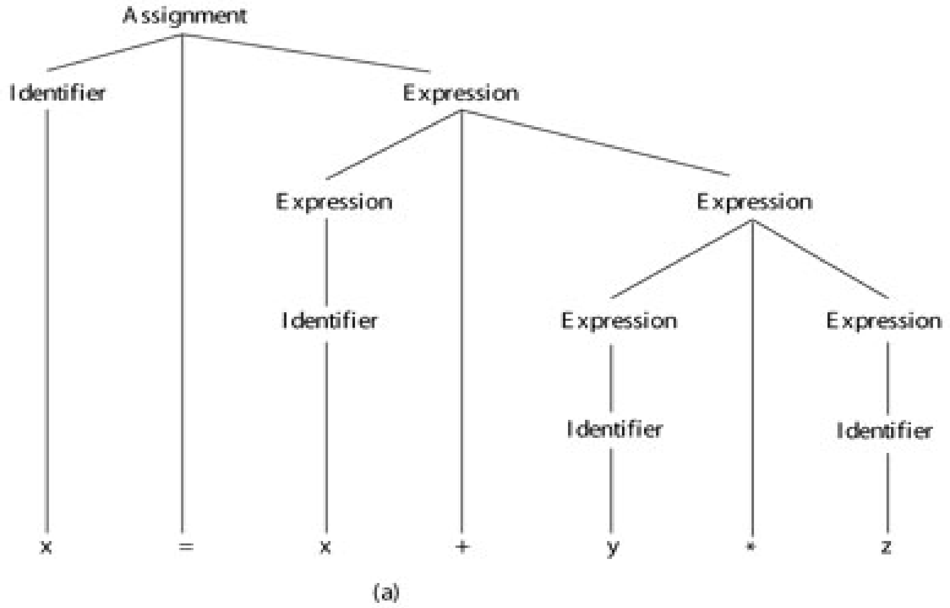
- If the phrase can be successfully represented, it belongs to the language
- Determining whether the phrase is valid is called recognition or representation

Ambiguity

- A grammar that represents a phrase associated with its language in two ore more distinct derivation trees is known as a syntactically ambiguous grammar.

Ex:

- Assignment =>identifier =Expression
- Expression => expression +expression
- Expression => expression+expression
- Expression=>identifier
- Identifier=> x|y|z



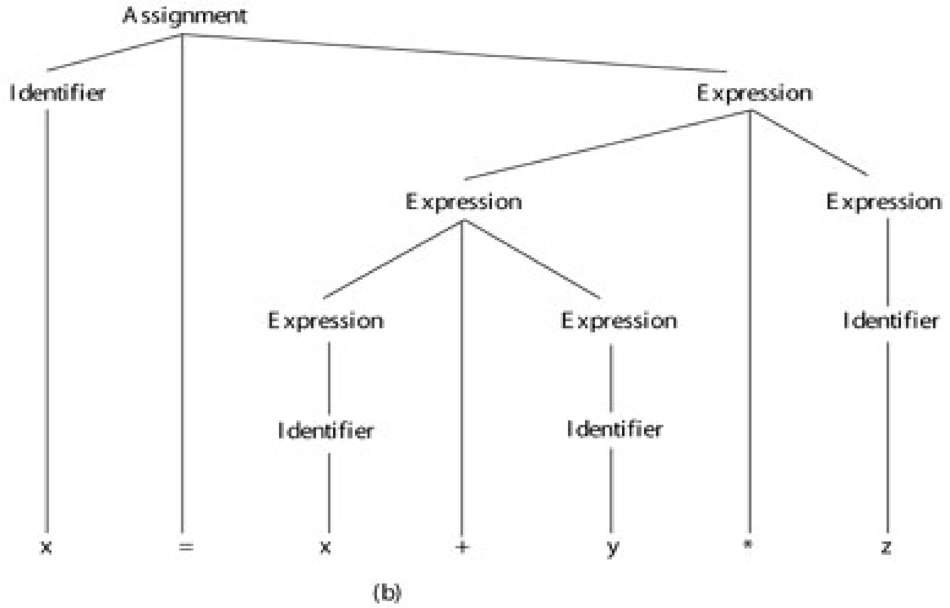


Figure 4.6 Two derivation trees for the assignment statement x = x + y * z

- Deciding whether grammar is ambiguous is a theoretically difficult task
- In practice , ambiguities can be avoided

- Assignment => identifier=expression
- Expression => element +expression
- Expression => element* expression
- Expression => element
- Element =>identifier
- Identifier=>x|y|z

BNF Variations

- a grammar written in BNF may be expressed in many other notation
- Two popular notational variations of BNF are the Extended BNF grammar and Syntax Diagram

extended BNF grammar

- Doesn't enhance the descriptive powerof BNF
- Merely increases the readability and writability of the production rules

Recommand notation

- Braces {}
  - represent a sequence of zero or more instance of elements
- Brackets[]
  - Used to represent an optional element
- Parenthese
  - Used to represent a group of elements

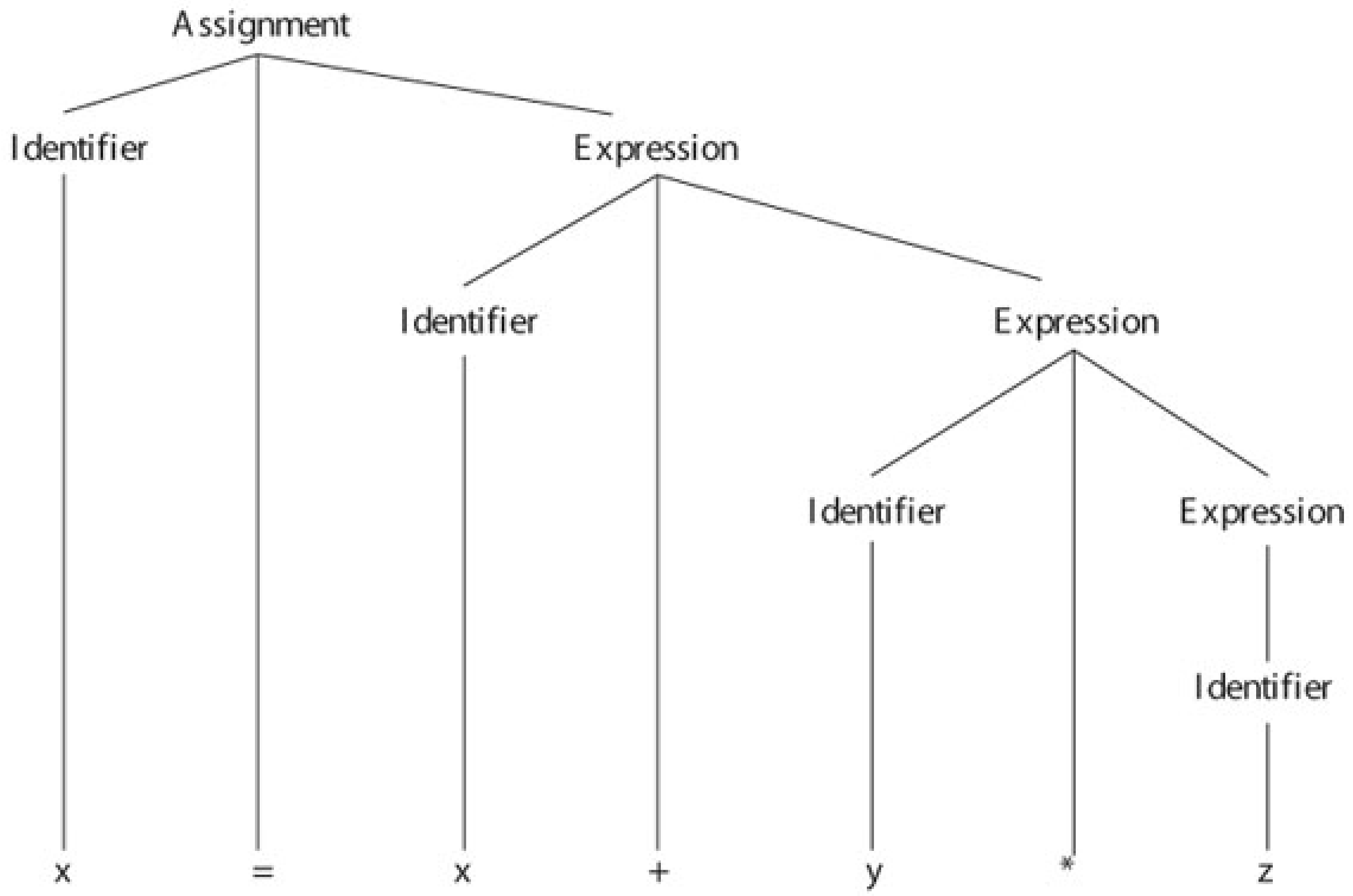


Figure 4.7 Revision of the grammar for the assignment x = x + y * z

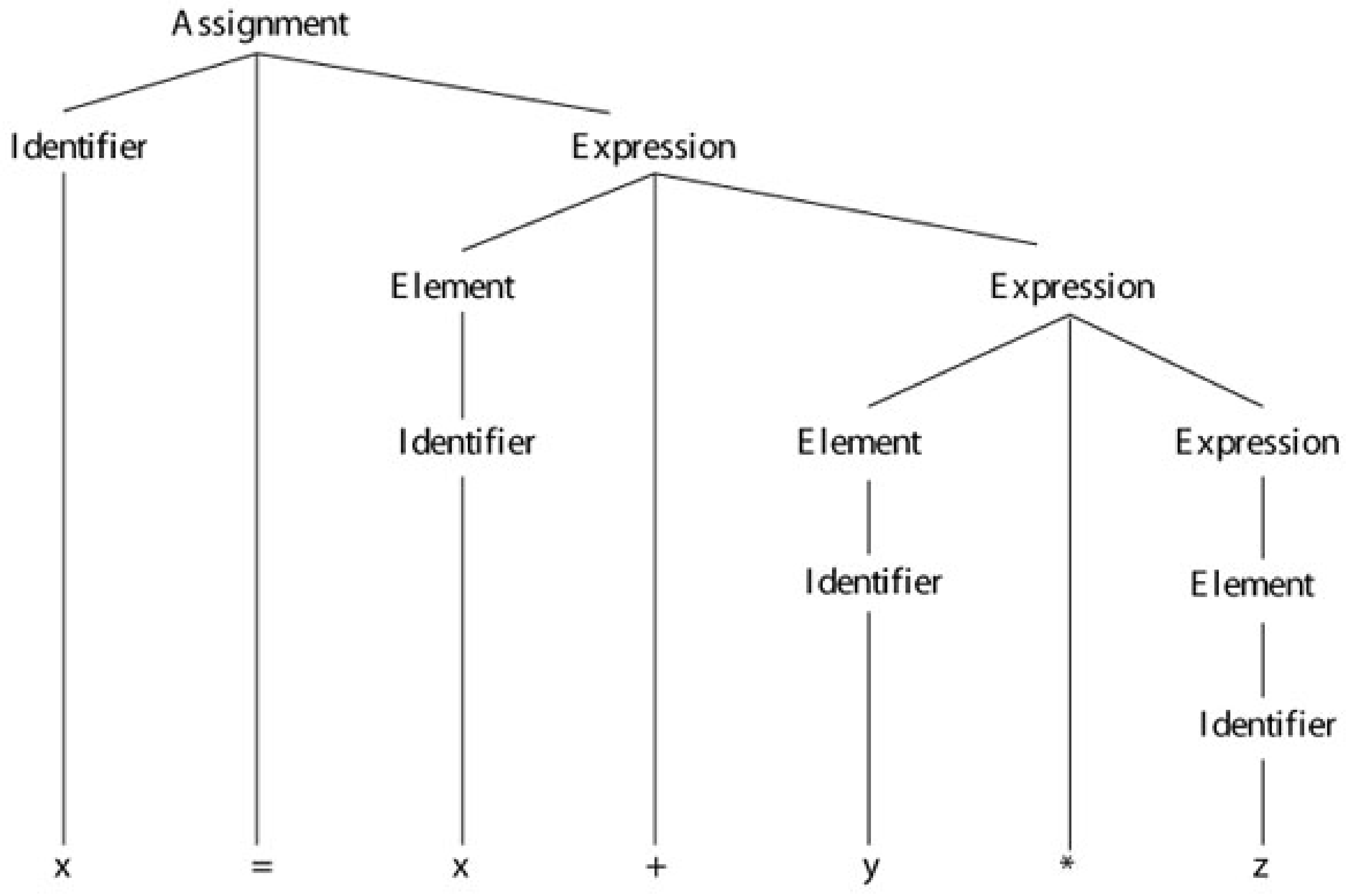


Figure 4.8 Introducing a new variable for the assignment statement x = x + y * z

Syntax diagram

- 1970, definition of pascal programming language

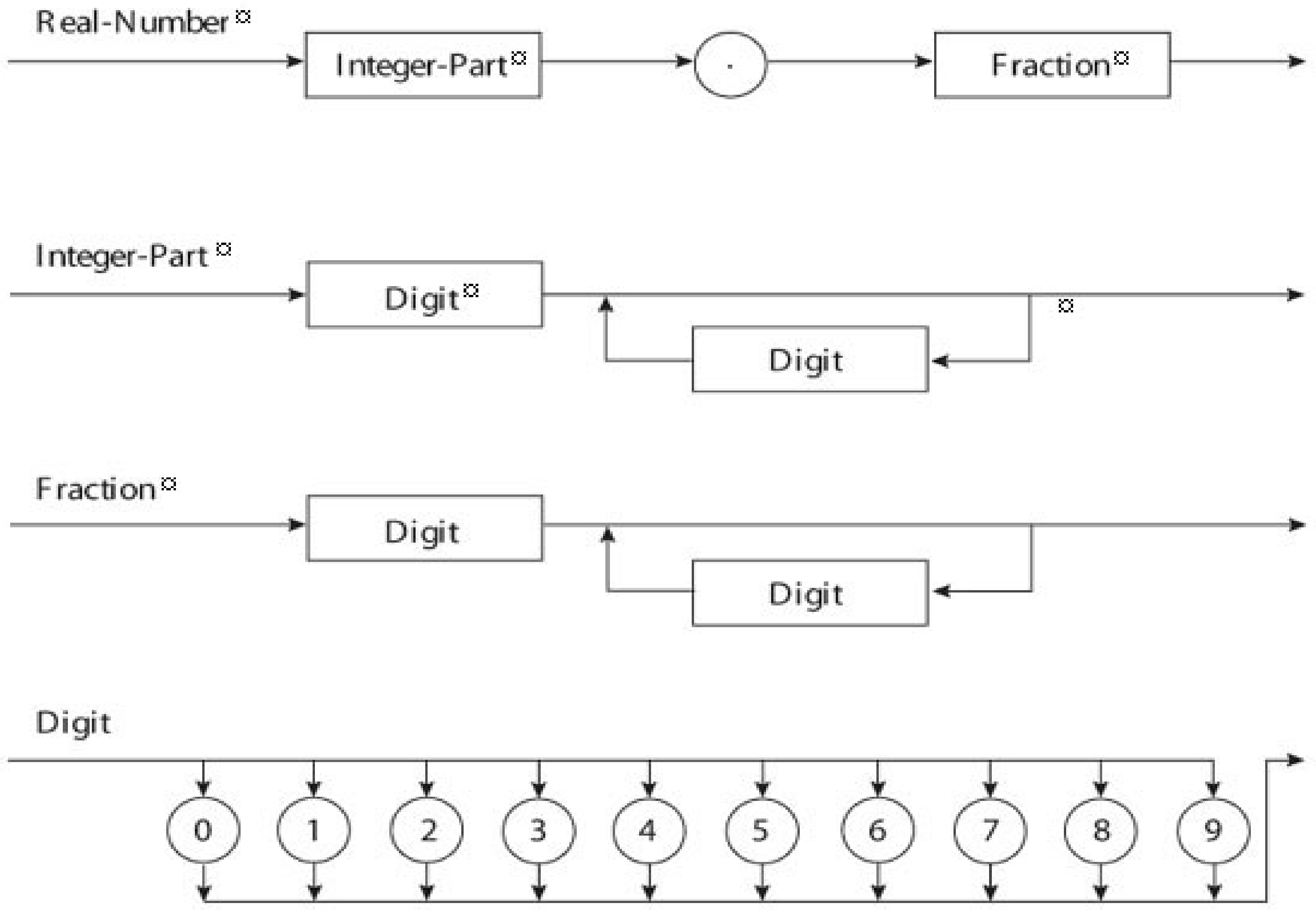


Figure 4.10 Syntax Diagram representation for the Real-Number grammar

Attribute grammars

- 1968 Knuth
- Are powerful and elegant mechanisms that formalize both the context-free and context-sensitive aspects of a language syntax
- Ex:
  - Used to determine whether a variable has been declared and whether the use of the variable is consistent with its declaration

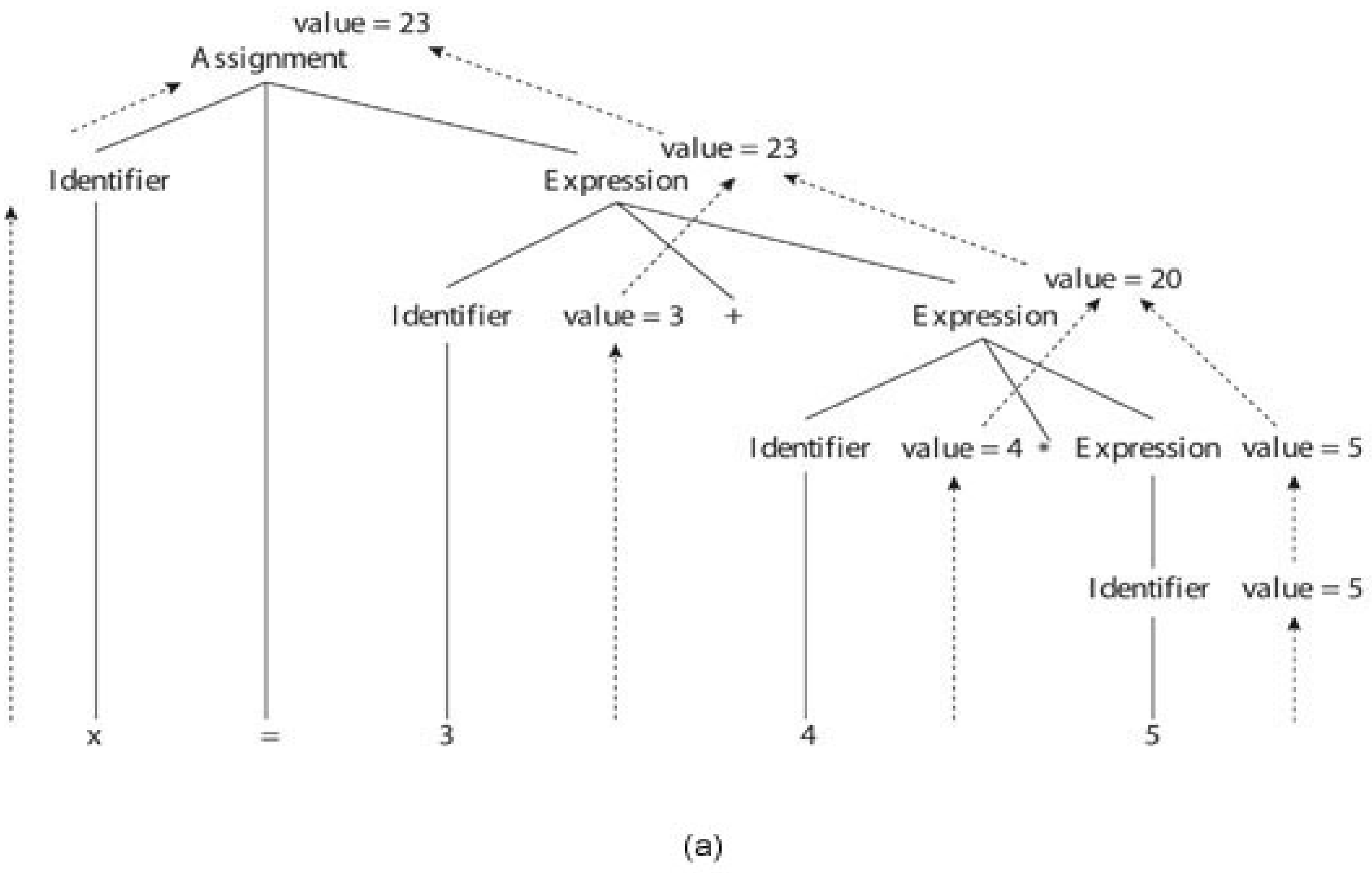


Figure 4.11 (a) An attributed syntax tree expressing the *value* attribute.

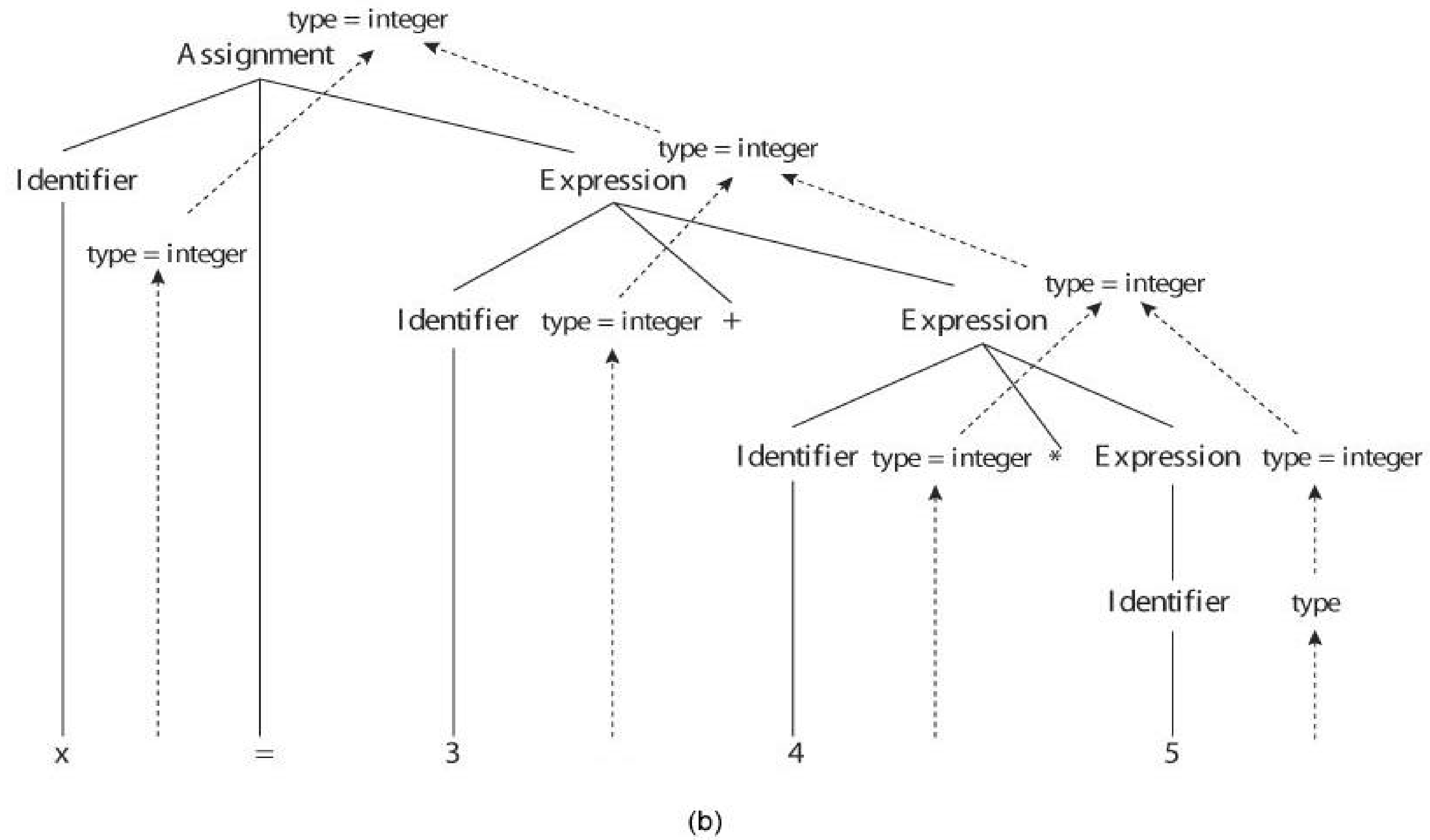


Figure 4.11 (b) An attributed syntax tree expressing the *actual-type* attribute

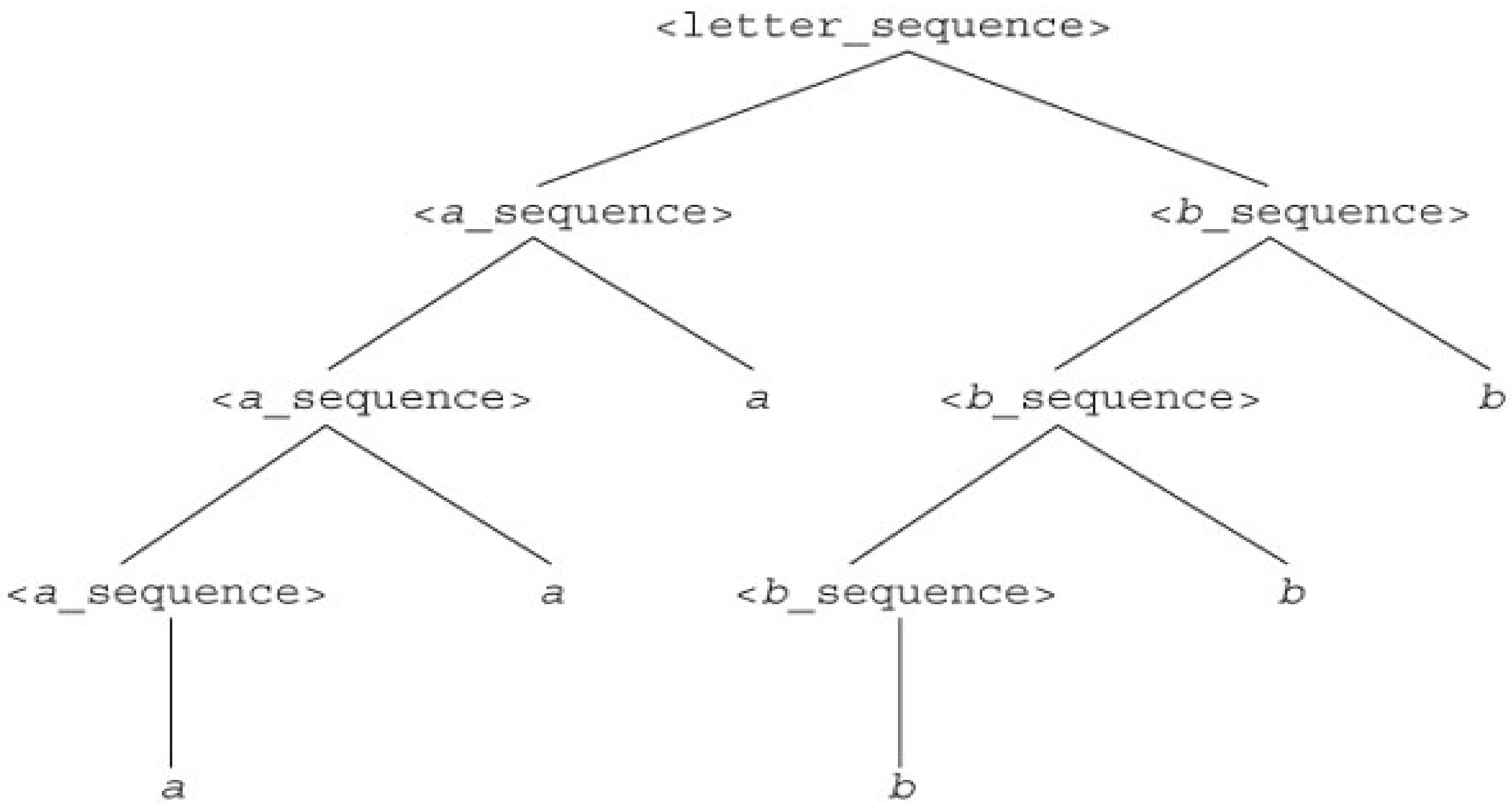


Figure 4.12 Syntax tree for string *aaabbb*

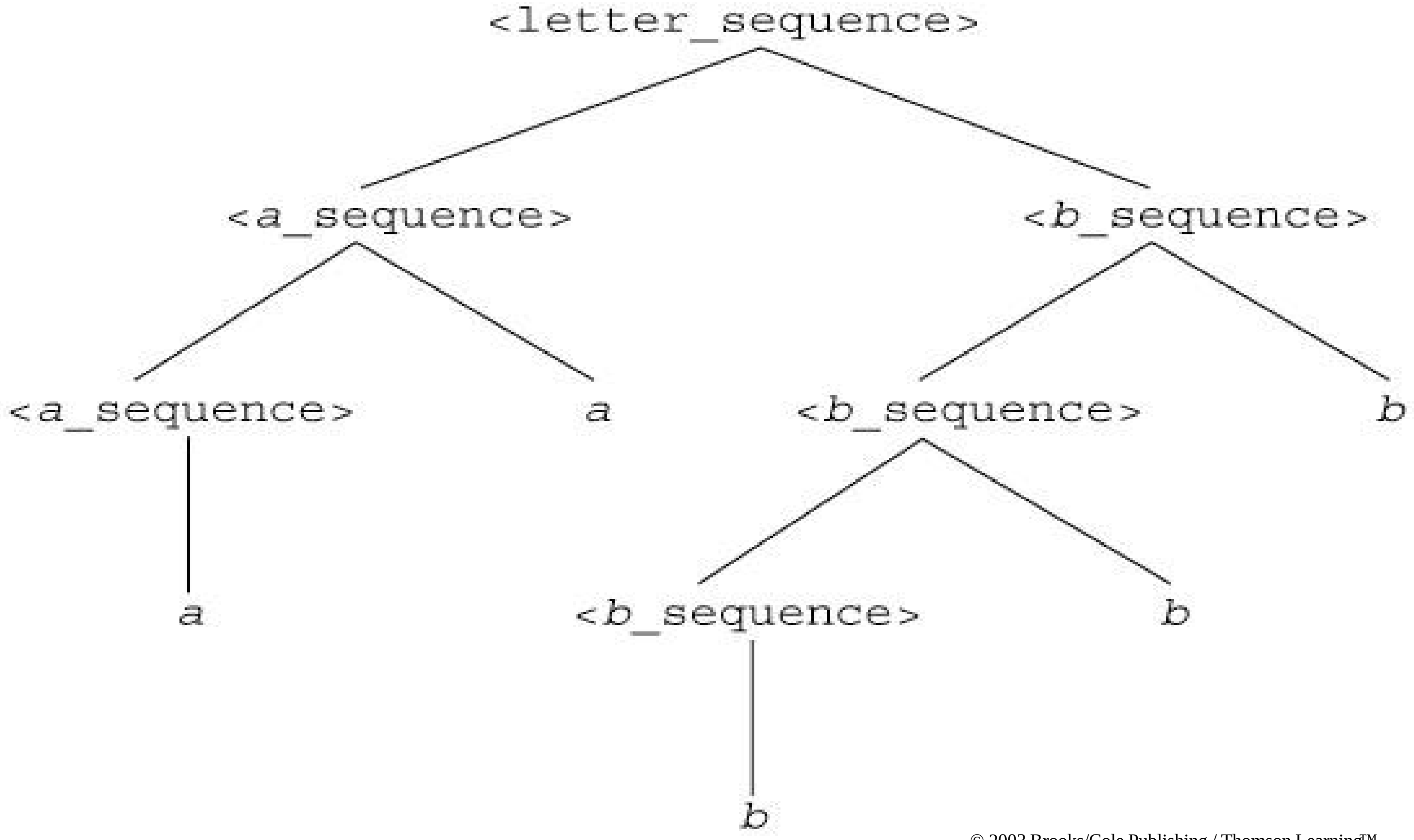


Figure 4.13 Syntax tree for string *aabbb*

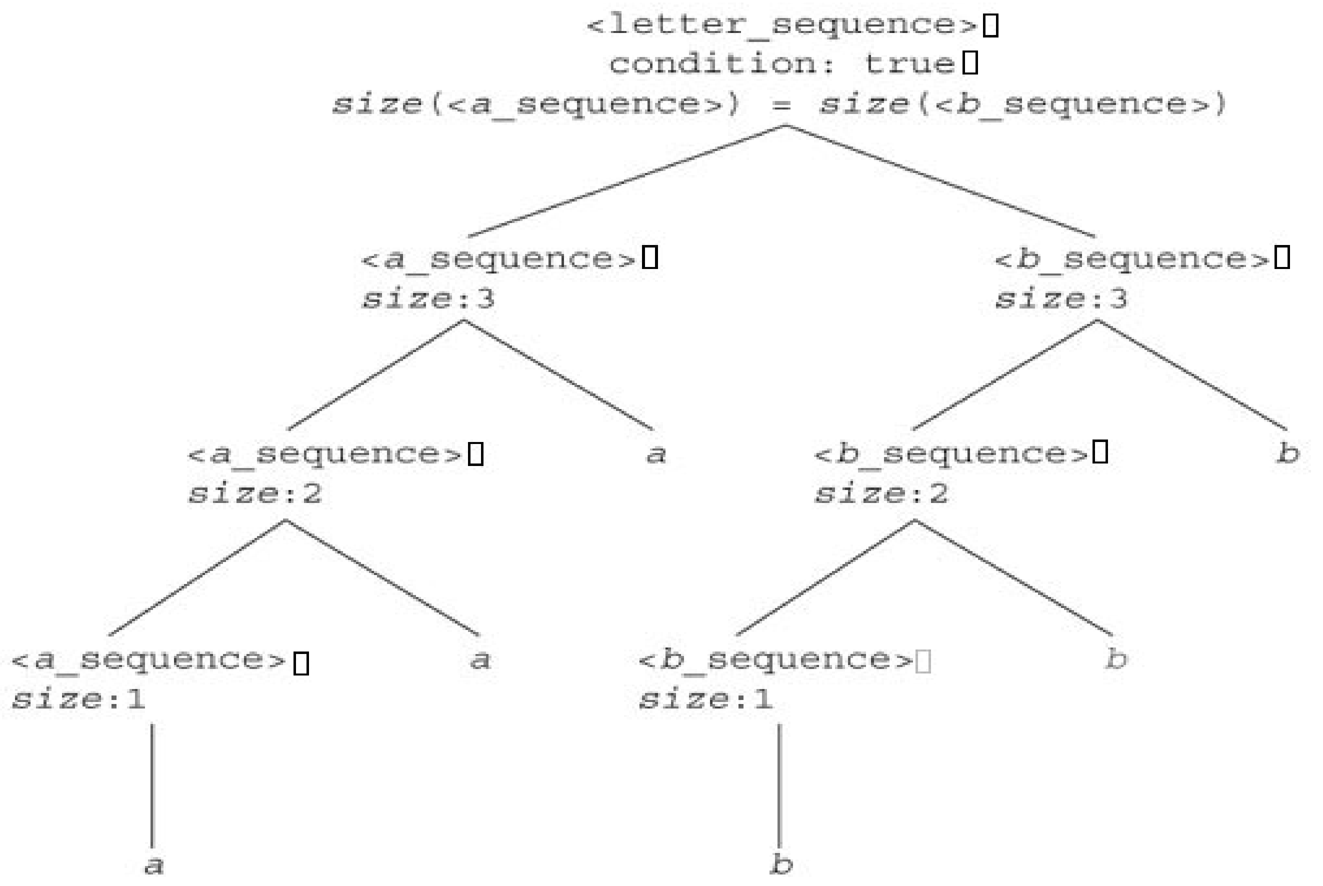


Figure 4.14 Attributed syntax tree for the string *aaabbb*