Programming Languages

2nd edition

Tucker and Noonan

Chapter 2
Syntax

A language that is simple to parse for the compiler is also simple to parse for the human programmer.

N. Wirth

Contents

Thinking about Syntax

The *syntax* of a programming language is a precise description of all its grammatically correct programs.

Precise syntax was first used with Algol 60, and has been used ever since.

Three levels:

- *Lexical syntax*
- *Concrete syntax*
- *Abstract syntax*

Levels of Syntax

Lexical syntax = all the basic symbols of the language (names, values, operators, etc.)

Concrete syntax = rules for writing expressions, statements and programs.

Abstract syntax = internal representation of the program, favoring content over form. E.g.,

– *C:* *if (expr) ...* *discard ( )*

– *Ada:* *if (expr) then discard then*

2.1 Grammars

A *metalanguage* is a language used to define other languages.

A *grammar* is a metalanguage used to define the syntax of a language.

Our interest: using grammars to define the syntax of a programming language.

2.1.1 Backus-Naur Form (BNF)

- Stylized version of a context-free grammar (cf. Chomsky hierarchy)
- Sometimes called Backus Normal Form
- First used to define syntax of Algol 60
- Now used to define syntax of most major languages

BNF Grammar

Set of *productions*: P

terminal symbols: T

nonterminal symbols: N

start symbol: $S \in N$

A *production* has the form

$$A \rightarrow \omega$$

where $A \in N$ and $\omega \in (N \cup T)^*$

Example: Binary Digits

Consider the grammar:

$$binaryDigit \to 0$$

$$binaryDigit \to 1$$

or equivalently:

$$binaryDigit \to 0 \mid 1$$

Here, | is a metacharacter that separates alternatives.

2.1.2 Derivations

Consider the grammar:

$$Integer \rightarrow Digit \mid Integer\ Digit$$
$$Digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

We can *derive* any unsigned integer, like 352, from this grammar.

Derivation of 352 as an *Integer*

A 6-step process, starting with:

Integer

Derivation of 352 (step 1)

Use a grammar rule to enable each step:

$$Integer \Rightarrow Integer\ Digit$$

Derivation of 352 (steps 1-2)

Replace a nonterminal by a right-hand side of one of its rules:

$$\begin{aligned} \mathit{Integer} &\Rightarrow \mathit{Integer}\ \mathit{Digit} \\ &\Rightarrow \mathit{Integer}\ 2 \end{aligned}$$

Derivation of 352 (steps 1-3)

Each step follows from the one before it.

$$
\begin{aligned}
&\mathit{Integer} \Rightarrow \mathit{Integer}\ \mathit{Digit} \\
&\quad \Rightarrow \mathit{Integer}\ 2 \\
&\quad \Rightarrow \mathit{Integer}\ \mathit{Digit}\ 2
\end{aligned}
$$

Derivation of 352 (steps 1-4)

$$\begin{aligned} Integer &\Rightarrow Integer\ Digit \\ &\Rightarrow Integer\ 2 \\ &\Rightarrow Integer\ Digit\ 2 \\ &\Rightarrow Integer\ 5\ 2 \end{aligned}$$

Derivation of 352 (steps 1-5)

$$
\begin{aligned}
Integer &\Rightarrow Integer\ Digit \\
&\Rightarrow Integer\ 2 \\
&\Rightarrow Integer\ Digit\ 2 \\
&\Rightarrow Integer\ 5\ 2 \\
&\Rightarrow Digit\ 5\ 2
\end{aligned}
$$

Derivation of 352 (steps 1-6)

You know you're finished when there are only terminal symbols remaining.

$$
\begin{aligned}
\mathit{Integer} &\Rightarrow \mathit{Integer}\ \mathit{Digit} \\
&\Rightarrow \mathit{Integer}\ 2 \\
&\Rightarrow \mathit{Integer}\ \mathit{Digit}\ 2 \\
&\Rightarrow \mathit{Integer}\ 5\ 2 \\
&\Rightarrow \mathit{Digit}\ 5\ 2 \\
&\Rightarrow 3\ 5\ 2
\end{aligned}
$$

A Different Derivation of 352

$$
\begin{aligned}
Integer &\Rightarrow Integer\ Digit \\
&\Rightarrow Integer\ Digit\ Digit \\
&\Rightarrow Digit\ Digit\ Digit \\
&\Rightarrow 3\ Digit\ Digit \\
&\Rightarrow 3\ 5\ Digit \\
&\Rightarrow 3\ 5\ 2
\end{aligned}
$$

This is called a *leftmost derivation*, since at each step the leftmost nonterminal is replaced.

(The first one was a *rightmost derivation*.)

Notation for Derivations

$Integer \Rightarrow^* 352$

Means that 352 can be derived in a finite number of steps using the grammar for *Integer*.

$352 \in L(G)$

Means that 352 is a member of the language defined by grammar *G*.

$L(G) = \{ \omega \in T^* \mid Integer \Rightarrow^* \omega \}$

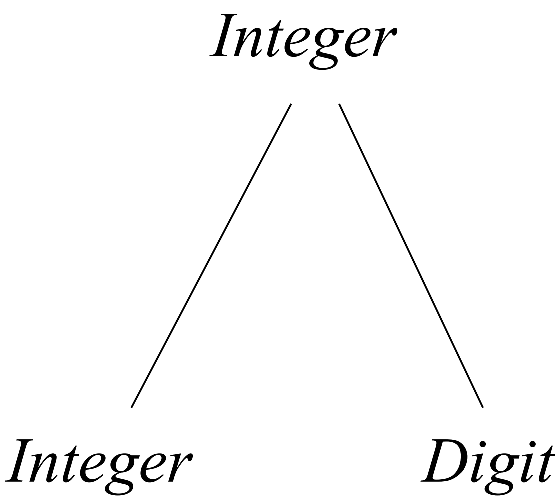
Means that the language defined by grammar *G* is the set of all symbol strings ω that can be derived as an *Integer*.

2.1.3 Parse Trees

A *parse tree* is a graphical representation of a derivation.

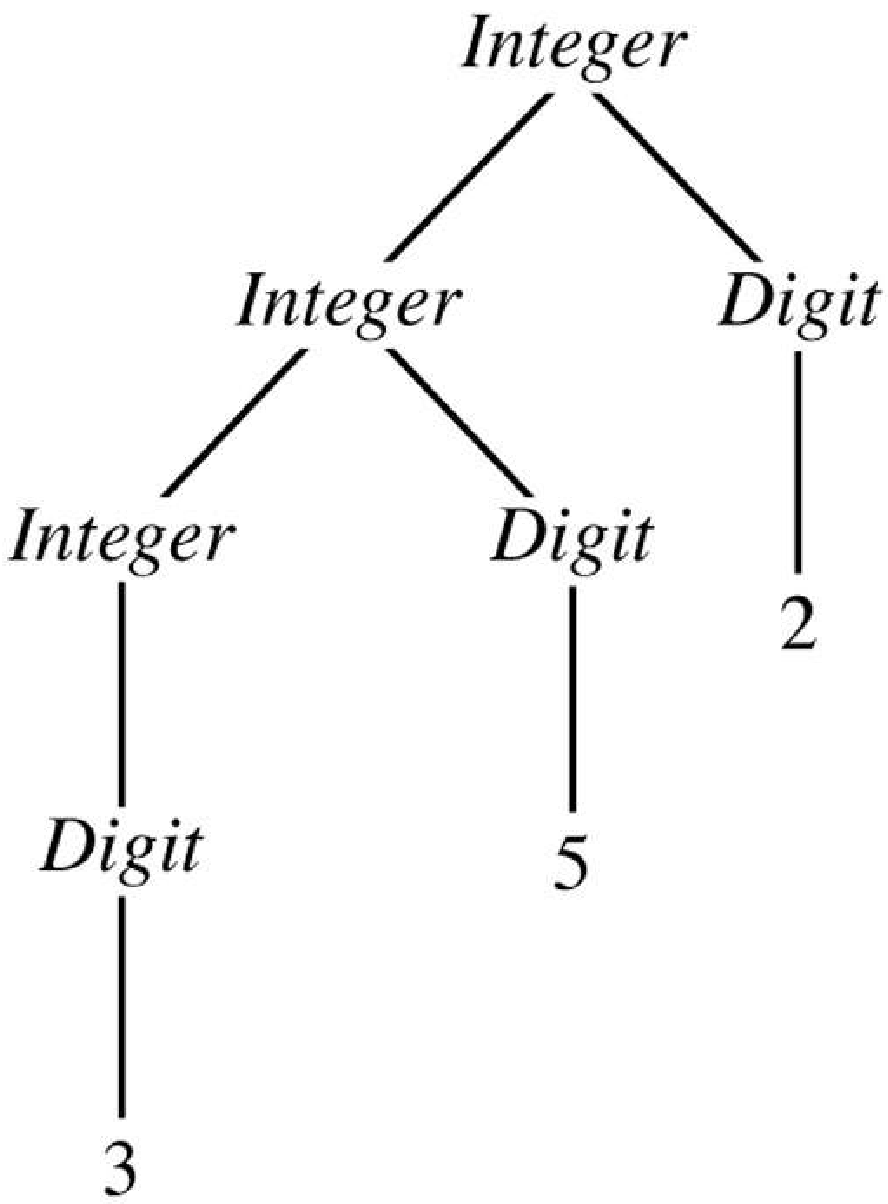
- *Each internal node of the tree corresponds to a step in the derivation.*
- *Each child of a node represents a right-hand side of a production.*
- *Each leaf node represents a symbol of the derived string, reading from left to right.*

E.g., The step $\mathit{Integer} \Rightarrow \mathit{Integer}\ \mathit{Digit}$ appears in the parse tree as:



Parse Tree for 352 as an *Integer*

Figure 2.1



Arithmetic Expression Grammar

The following grammar defines the language of arithmetic expressions with 1-digit integers, addition, and subtraction.

$$Expr \rightarrow Expr + Term \mid Expr - Term \mid Term$$

$$Term \rightarrow 0 \mid \ldots \mid 9 \mid (\ Expr \ )$$

Parse of the String 5-4+3

Figure 2.2