B NOTES ON LANGUAGES AND ALGORITHMS

B.1 DEFINING LANGUAGES WITH BACKUS–NAUR FORM (BNF)

In this book, we define several languages, including the languages of propositional logic (page 243), first-order logic (page 293), and a subset of English (page 899). A formal language is defined as a set of strings where each string is a sequence of symbols. The languages we are interested in consist of an infinite set of strings, so we need a concise way to characterize the set. We do that with a **grammar**. The particular type of grammar we use is called a **context-free grammar**, because each expression has the same form in any context. We write our grammars in a formalism called **Backus–Naur form (BNF)**. There are four components to a BNF grammar:

CONTEXT-FREE GRAMMAR
BACKUS–NAUR FORM (BNF)

TERMINAL SYMBOL

- A set of **terminal symbols**. These are the symbols or words that make up the strings of the language. They could be letters (**A, B, C,** . . .) or words (**a, aardvark, abacus,** . . .), or whatever symbols are appropriate for the domain.

NONTERMINAL SYMBOL

- A set of **nonterminal symbols** that categorize subphrases of the language. For example, the nonterminal symbol $NounPhrase$ in English denotes an infinite set of strings including "you" and "the big slobbery dog."

START SYMBOL

- A **start symbol**, which is the nonterminal symbol that denotes the complete set of strings of the language. In English, this is $Sentence$; for arithmetic, it might be $Expr$, and for programming languages it is $Program$.
- A set of **rewrite rules**, of the form $LHS \rightarrow RHS$, where LHS is a nonterminal symbol and RHS is a sequence of zero or more symbols. These can be either terminal or nonterminal symbols, or the symbol ϵ, which is used to denote the empty string.

A rewrite rule of the form

$$Sentence \rightarrow NounPhrase\ VerbPhrase$$

means that whenever we have two strings categorized as a $NounPhrase$ and a $VerbPhrase$, we can append them together and categorize the result as a $Sentence$. As an abbreviation, the two rules ($S \rightarrow A$) and ($S \rightarrow B$) can be written ($S \rightarrow A \mid B$).

Here is a BNF grammar for simple arithmetic expressions:

$$\begin{aligned} \textit{Expr} &\rightarrow \textit{Expr Operator Expr} \mid \textbf{(} \textit{Expr} \textbf{)} \mid \textit{Number} \\ \\ \textit{Number} &\rightarrow \textit{Digit} \mid \textit{Number Digit} \\ \\ \textit{Digit} &\rightarrow \textbf{0} \mid \textbf{1} \mid \textbf{2} \mid \textbf{3} \mid \textbf{4} \mid \textbf{5} \mid \textbf{6} \mid \textbf{7} \mid \textbf{8} \mid \textbf{9} \\ \textit{Operator} &\rightarrow + \mid - \mid \div \mid \times \end{aligned}$$

We cover languages and grammars in more detail in Chapter 22. Be aware that other books use slightly different notations for BNF; for example, you might see $\langle \textit{Digit} \rangle$ instead of *Digit* for a nonterminal, 'word' instead of **word** for a terminal, or **::=** instead of $\rightarrow$ in a rule.

B.2 Describing Algorithms with Pseudocode

The algorithms in this book are described in pseudocode. Most of the pseudocode should be familiar to users of languages like Java, C++, or Lisp. In some places we use mathematical formulas or ordinary English to describe parts that would otherwise be more cumbersome. A few idiosyncrasies should be noted.

- **Persistent variables**: We use the keyword **persistent** to say that a variable is given an initial value the first time a function is called and retains that value (or the value given to it by a subsequent assignment statement) on all subsequent calls to the function. Thus, persistent variables are like global variables in that they outlive a single call to their function, but they are accessible only within the function. The agent programs in the book use persistent variables for *memory*. Programs with persistent variables can be implemented as *objects* in object-oriented languages such as C++, Java, Python, and Smalltalk. In functional languages, they can be implemented by *functional closures* over an environment containing the required variables.
- **Functions as values**: Functions and procedures have capitalized names, and variables have lowercase italic names. So most of the time, a function call looks like FN(x). However, we allow the value of a variable to be a function; for example, if the value of the variable f is the square root function, then $f(9)$ returns 3.
- **for each**: The notation "**for each** x **in** c **do**" means that the loop is executed with the variable x bound to successive elements of the collection c.
- **Indentation is significant**: Indentation is used to mark the scope of a loop or conditional, as in the language Python, and unlike Java and C++ (which use braces) or Pascal and Visual Basic (which use **end**).
- **Destructuring assignment**: The notation "$x, y \leftarrow \textit{pair}$" means that the right-hand side must evaluate to a two-element tuple, and the first element is assigned to x and the second to y. The same idea is used in "**for each** x, y **in** *pairs* **do**" and can be used to swap two variables: "$x, y \leftarrow y, x$"
- **Generators** and **yield**: the notation "**generator** G(x) **yields** numbers" defines G as a generator function. This is best understood by an example. The code fragment shown in

```
generator POWERS-OF-2() yields ints
  i ← 1
  while true do
    yield i
    i ← 2 × i

for p in POWERS-OF-2() do
  PRINT(p)
```

Figure B.1 Example of a generator function and its invocation within a loop.

Figure B.1 prints the numbers 1, 2, 4, ..., and never stops. The call to POWERS-OF-2 returns a generator, which in turn yields one value each time the loop code asks for the next element of the collection. Even though the collection is infinite, it is enumerated one element at a time.

- **Lists**: $[x, y, z]$ denotes a list of three elements. $[first|rest]$ denotes a list formed by adding *first* to the list *rest*. In Lisp, this is the `cons` function.
- **Sets**: $\{x, y, z\}$ denotes a set of three elements. $\{x : p(x)\}$ denotes the set of all elements x for which $p(x)$ is true.
- **Arrays start at 1**: Unless stated otherwise, the first index of an array is 1 as in usual mathematical notation, not 0, as in Java and C.

B.3 ONLINE HELP

Most of the algorithms in the book have been implemented in Java, Lisp, and Python at our online code repository:

`aima.cs.berkeley.edu`

The same Web site includes instructions for sending comments, corrections, or suggestions for improving the book, and for joining discussion lists.