Chapter 8

Languages and Meanings

In this chapter we look at how programming languages are defined. There are two aspects to consider. We must develop a method to specify exactly what words are valid as program expressions. And we must develop a method to say precisely what computation a valid program expresses. We now have the basic mathematical tools to make these specifications: programming languages are specified by inductive set definitions and their meanings by recursive function definitions. However, we still must develop techniques to use these tools effectively.

8.1 Language Definitions

We shall start with a seemingly simple language of expressions. The idea is to put the language together by showing how to build more complicated expressions from simpler ones. This is the way almost all computer languages are defined.

Let the set $A = \mathbb{N} \cup \{\$, \#\}$ be our alphabet of symbols. Define the language $L \subseteq A^+$ inductively, according to

1.	$\mathbb{N} \subseteq L$
2a.	$u, v \in L \Rightarrow u \; \$ \; v \in L$
2b.	$u, v \in L \Rightarrow u \; \# \; v \in L$
3.	nothing else

The first kind of question that might be asked is whether a particular word in A^+ is in (i.e. an element of) the language L. To answer such a question, we must analyze the given word to see whether it could be built using the rules of L's definition.

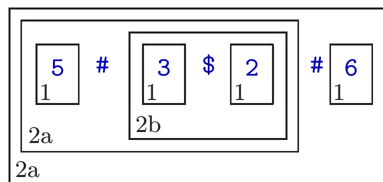
Example

Ex 8.1 *Is the word* $5 \# 3 \$ 2 \# 6$ *in* L*?*

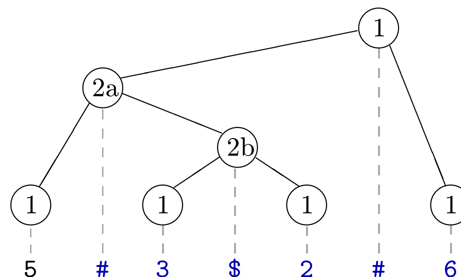
The answer is “yes,” because there is a construction sequence:

1.	$3 \in L$	by rule 1
2.	$2 \in L$	by rule 1
3.	3 $ 2 $\in L$	by rule 2b, 1 and 2
4.	$5 \in L$	by rule 1
5.	5 # 3 $ 2 $\in L$	by rule 2a, 4 and 3
6.	$6 \in L$	by rule 1
7.	5 # 3 $ 2 # 6 $\in L$	by rule 2a, 5 and 6

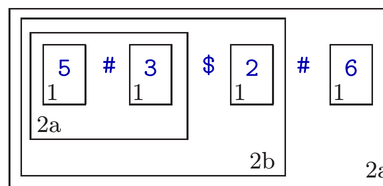
The analysis showing that a word is in a language is called *parsing*. There are two ways to diagram derivations like this. A *parsing diagram* uses nested boxes to show how the word decomposes:



The second way is to draw a A *parse tree*, whose interior nodes are labeled by rules and whose leaves are symbols of the alphabet:



Each of the diagrams above reflects the same parse. Here is a different parse showing that 5 # 3 $ 2 # 6 is in L:



That there are several ways to prove that a given word is in L is a problem, as we shall see next.

Copyright © 2008 Steven D. Johnson

8.2 Defining How Languages are Interpreted

The interpretation of a language associates a value with each word. The word is said to express the value. If the language is inductively defined, then the interpretation can be defined recursively.

Using the language L of the previous section, let us define a function $\mathcal{V}\colon L \to \mathbb{N}$, which gives a natural-number interpretation where symbols # and $ express addition and multiplication, respectively.

1.	for $k \in \mathbb{N}$,	$\mathcal{V}[k]$	$= k$
2a.	for words of the form $w = u \,\#\, v$,	$\mathcal{V}[w]$	$= \mathcal{V}[u] + \mathcal{V}[v]$
2b.	for words of the form $w = u \,\$\, v$,	$\mathcal{V}[w]$	$= \mathcal{V}[u] \times \mathcal{V}[v]$

Based on this definition and the first parsing analysis, we can determine an interpretation of $5 \,\#\, 3 \,\$\, 2 \,\#\, 6$:

1.	$3 \in \mathbb{N}$	$\Rightarrow \mathcal{V}[3] = 3$
2.	$2 \in \mathbb{N}$	$\Rightarrow \mathcal{V}[2] = 2$
3.	$\left.\begin{array}{lcl} u &=& 3 \in L \\ v &=& 2 \in L \end{array}\right\}$	$\Rightarrow \mathcal{V}[u \,\$\, v] = \mathcal{V}[u] \times \mathcal{V}[v] = 3 \times 2 = 6$
4.	$5 \in \mathbb{N}$	$\Rightarrow \mathcal{V}[5] = 5$
5.	$\left.\begin{array}{lcl} u &=& 5 \in L \\ v &=& 3 \,\$\, 2 \in L \end{array}\right\}$	$\Rightarrow \mathcal{V}[u \,\#\, v] = \mathcal{V}[u] + \mathcal{V}[v] = 5 + 6 = 11$
6.	$6 \in \mathbb{N}$	$\Rightarrow \mathcal{V}[6] = 6$
7.	$\left.\begin{array}{lcl} u &=& 5 \,\#\, 3 \,\$\, 2 \in L \\ v &=& 6 \in L \end{array}\right\}$	$\Rightarrow \mathcal{V}[u \,\#\, v] = \mathcal{V}[u] + \mathcal{V}[v] = 11 + 6 = 17$

That is, $\mathcal{V}[5 \,\#\, 3 \,\$\, 2 \,\#\, 6] = 17$. However, a different parse leads to a different interpretation. The derivation that follows is "top-down" in the sense that the interpreted expression is decomposed as the interpretation is applied. At each step, you should verify that the word is broken down in a manner that is consistent with L's definition.

$$
\begin{array}{lll}
\mathcal{V}[5 \,\#\, 3 \,\$\, 2 \,\#\, 6] & & \\
\quad = & \mathcal{V}[5] + \mathcal{V}[3 \,\$\, 2 \,\#\, 6] & \text{defn. } \mathcal{V}\text{, case 2a} \\
\quad = & 5 + \mathcal{V}[3 \,\$\, 2 \,\#\, 6] & \mathcal{V}(1) \\
\quad = & 5 + \big(\mathcal{V}[3] \times \mathcal{V}[2 \,\#\, 6]\big) & \mathcal{V}(2b) \\
\quad = & 5 + \big(3 \times \mathcal{V}[2 \,\#\, 6]\big) & \mathcal{V}(1) \\
\quad = & 5 + \big(3 \times \big(\mathcal{V}[2] + \mathcal{V}[6]\big)\big) & \mathcal{V}(2a) \\
\quad = & 5 + \big(3 \times \big(2 + 6\big)\big) & \mathcal{V}(1)\text{, twice} \\
\quad = & 29 & \text{arithmetic}
\end{array}
$$

Copyright © 2008 Steven D. Johnson

That is, $\mathcal{V}[5 \;\texttt{\#}\; 3 \;\texttt{\$}\; 2 \;\texttt{\#}\; 6] = 17 = 29$ (?!). The apparent contradiction is due to the assumption that $\mathcal{V}$ is a function; use of the '=' symbol, in defining $\mathcal{V}$ and in deriving a value, is simply wrong. Both interpretations are correct: $\mathcal{V}$ is a *relation* associating the values 17, 29, and several others, with the word $5 \;\texttt{\#}\; 3 \;\texttt{\$}\; 2 \;\texttt{\#}\; 6$.

When computer languages are defined, it is usually intended that each word have a unique interpretation—we *want* $\mathcal{V}$ to be a function. When multiple interpretations exist, it is said that the language is *ambiguous*. As we shall see later, ambiguity is not necessarily the fault of the language, but even so, languages can be designed to be unambiguous. In the following two examples, variations of L are defined, by which the problems just encountered are avoided.

Example

Ex 8.2 We can remove the ambiguity in L by introducing parenthesis symbols. Take $V = \mathbb{N} \cup \{\texttt{\#}, \texttt{\$}, \texttt{(}, \texttt{)}\}$. The language L_2 and its interpretation relation $\mathcal{V}_2$ are defined simultaneously below:

	$L_2 \subseteq V^+$	$\mathcal{V}_2 \colon L_2 \to \mathbb{N}$
1.	$\mathbb{N} \subseteq L_2$	$\mathcal{V}_2[k] = k,$ for $k \in \mathbb{N}$
2a.	$u, v \in L_2 \Rightarrow \texttt{(}\, u \;\texttt{\#}\; v \,\texttt{)} \in L_2$	$\mathcal{V}_2[\texttt{(}\, u \;\texttt{\#}\; v \,\texttt{)}] = \mathcal{V}_2[u] + \mathcal{V}_2[v]$
2b.	$u, v \in L_2 \Rightarrow \texttt{(}\, u \;\texttt{\$}\; v \,\texttt{)} \in L_2$	$\mathcal{V}_2[\texttt{(}\, u \;\texttt{\$}\; v \,\texttt{)}] = \mathcal{V}_2[u] \times \mathcal{V}_2[v]$
3.	nothing else	

In this language,

$$\mathcal{V}_2[\texttt{(}\,\texttt{(}\,5 \;\texttt{\#}\; \texttt{(}\,3 \;\texttt{\$}\; 2\,\texttt{)}\,\texttt{)} \;\texttt{\#}\; 6\,\texttt{)}\,/] = 17$$

and

$$\mathcal{V}_2[\texttt{(}\,5 \;\texttt{\#}\; \texttt{(}\,3 \;\texttt{\$}\; \texttt{(}\,2 \;\texttt{\#}\; 6\,\texttt{)}\,\texttt{)}\,\texttt{)}] = 29.$$

This version of L_2 forces every expression to be fully parenthesized, but the interpretation is unambiguous. $\mathcal{V}_2$ is a function because there is only one way to parse any word in L_2.

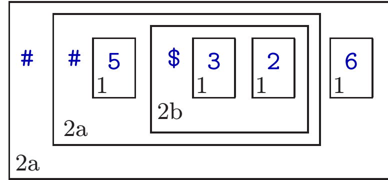
Example

Ex 8.3 In the version of L shown below, there are no parentheses but the operator symbols have been moved from an infix position to a *prefix* position:

	$L_3 \subseteq V^+$	$\mathcal{V}_3 \colon L_3 \to \mathbb{N}$
1.	$\mathbb{N} \subseteq L_3$	$\mathcal{V}_3[k] = k,$ for $k \in \mathbb{N}$
2a.	$u, v \in L_3 \Rightarrow \texttt{\#}\; u\; v \in L_3$	$\mathcal{V}_3[\texttt{\#}\; u\; v] = \mathcal{V}_3[u] + \mathcal{V}_3[v]$
2b.	$u, v \in L_3 \Rightarrow \texttt{\$}\; u\; v \in L_3$	$\mathcal{V}_3[\texttt{\$}\; u\; v] = \mathcal{V}_3[u] \times \mathcal{V}_3[v]$
3.	nothing else	

Copyright © 2008 Steven D. Johnson

To express 17, one would write:



This is the only way to parse # # 5 $ 3 2 6 because its decomposition under the definition of L_3 is determined by the initial symbol of the word. Only one symbol can be the initial symbol, so there can only be one parse. Consequently, the only possible interpretation is:

$$
\begin{array}{lll}
\mathcal{V}_3[\# \ \# \ 5 \ \$ \ 3 \ 2 \ 6] & & \\
\quad = & \mathcal{V}_3[\# \ 5 \ \$ \ 3 \ 2] + \mathcal{V}_3[6] & \mathcal{V}_3(2a) \\
\quad = & \big([\mathcal{V}_3[5] + \mathcal{V}_3[\$ \ 3 \ 2]\big) + \mathcal{V}_3[6] & \mathcal{V}_3(2a) \\
\quad = & \big([\mathcal{V}_3[5] + \big([\mathcal{V}_3[3] \times \mathcal{V}_3[2]\big)\big) + \mathcal{V}_3[6] & \mathcal{V}_3(2b) \\
\quad = & (5 + (3 \times 2)) + 6 & \mathcal{V}_3[1], \text{ four times} \\
\quad = & 17 & \text{arithmetic}
\end{array}
$$

Exercises 8.2

1. For the language L and interpretation relation $\mathcal{V}$ defined at the beginning of this section, list all the values that are associated with the expression $\mathcal{V}$[5 # 3 $ 2 # 6].

2. Draw the tree corresponding to the second parse of 5 # 3 $ 2 # 6. with respect to the language L.

3. Using the definition of $\mathcal{V}_3$, check that the expression shown in Example 3 evaluates to 17. Then evaluate the following words of L_3:

(a) # $ 3 # 8 9 $ 2 5

(b) # # # $ 3 4 5 6 7

(c) # 3 # 4 # 5 $ 6 7

4. Give an expression in the language L_3 of Example 8.3 which evaluates to 29.

5. Consider the following language, $L_4 \subseteq V^+$, for $V = \mathbb{N} \cup \{ \texttt{\#}, \texttt{\$} \}$.

	$L_4 \subseteq V^+$
1.	$\mathbb{N} \subseteq L_4$
2*a*.	$u, v \in L_4 \Rightarrow \texttt{\#}\ u\ v \in L_4$
2*b*.	$u, v \in L_4 \Rightarrow u\ v\ \texttt{\$} \in L_4$
3.	nothing else

Define an interpretation function for L_4 under which '$\texttt{\#}$' stands for addition and '$\texttt{\$}$' for multiplication.

6. For each of the expressions (a)–(c) in Exercise 3, give the corresponding expression in L_4.

7. Determine whether the following words are in L_4, and if so evaluate them:

(a) $\texttt{\#}\ \texttt{\#}\ 2\ 3\ 4\ \texttt{\$}\ 4$

(b) $\texttt{\#}\ 2\ 3\ \texttt{\$}\ 4\ 5\ \texttt{\$}$

(c) $2\ \texttt{\#}\ 3\ \texttt{\#}\ 5\ \texttt{\$}\ 6$

(d) $2\ \texttt{\#}\ 3\ \texttt{\#}\ 5\ \texttt{\$}\ 6$

(e) $2\ 3\ \texttt{\#}\ 4\ \texttt{\#}\ 5\ 6\ \texttt{\$}\ 7\ \texttt{\$}\ \texttt{\$}$

(f) $1\ \texttt{\#}\ \texttt{\#}\ 2\ 3\ 4\ 5\ 6\ \texttt{\$}\ 7\ 8\ \texttt{\$}\ \texttt{\$}\ 9\ \texttt{\$}$

8.3 Specifying Precedence

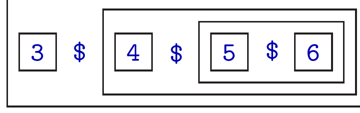
We have seen in Examples 8.2 and 8.3 that one can control the way operations are applied by using disambiguating syntax, like parentheses, or otherwise changing the grammar of the language. One can also deal with the problem mathematically by introducing more structure to the language definition. Let $V = \mathbb{N} \cup \{ \texttt{\$}, \texttt{\#} \}$, as before. First, define a language F and interpretation function $\mathcal{V}_F : F \to \mathbb{N}$, involving only the '$\texttt{\$}$' operation symbol:

	$F \subseteq V^+$	$\mathcal{V}_F : F \to \mathbb{N}$
1.	$\mathbb{N} \subseteq F$	$\mathcal{V}_F[k] = k$, for $k \in \mathbb{N}$
2.	$u \in \mathbb{N},\ v \in F \Rightarrow u\ \texttt{\$}\ v \in F$	$\mathcal{V}_F[u\ \texttt{\$}\ v] = u \times \mathcal{V}_F[v]$
3.	nothing else	

If you study these definitions carefully, you will see a subtle difference from the earlier definition of L (aside from the fact that part 2a is missing!). Part 2 of the definition, builds and interprets words in such a way that multiplications are carried out from right to left. It is said that '$\texttt{\$}$' *associates to the right.* To

Copyright © 2008 Steven D. Johnson

illustrate why this must be, let us consider the word $w = 3 \,\$\, 4 \,\$\, 5 \,\$\, 6$. There is exactly one way to parse w:



Hence, there is exactly one interpretation:

$$
\begin{array}{lll}
\mathcal{V}_F[3 \,\$\, 4 \,\$\, 5 \,\$\, 6] & & \\
\quad = & 3 \times \mathcal{V}_F[4 \,\$\, 5 \,\$\, 6] & \mathcal{V}_F(2) \\
\quad = & 3 \times \big(4 \times \mathcal{V}_F[5 \,\$\, 6]\big) & \mathcal{V}_F(2) \\
\quad = & 3 \times \big(4 \times \big(5 \times \mathcal{V}_F[6]\big)\big) & \mathcal{V}_F(2) \\
\quad = & 3 \times \big(4 \times \big(5 \times 6\big)\big) & \mathcal{V}_F(1) \\
\quad = & 360 & \text{arithmetic}
\end{array}
$$

In this case, the order of evaluating multiplications doesn't matter; but it *would* matter if '$' were to denote, say, subtraction (or *real* computer multiplication with overflow).

Building from the sublanguage F, we can now create a language T by introducing the addition symbol.

	$T \subseteq \big(F \cup \{\,\$\,\}\big)^+$	$\mathcal{V}_F : F \to \mathbb{N}$
1.	$F \subseteq T$	$\mathcal{V}_T[u] = \mathcal{V}_F[u],\ \text{for } u \in F$
2.	$u \in F, v \in T \Rightarrow u \,\#\, v \in F$	$\mathcal{V}_T[u \,\#\, v] = \mathcal{V}_F[u] + \mathcal{V}_T[v]$
3.	nothing else	

The language T is *exactly the same* as the language L that we originally defined. However, the interpretation, V_T, is constrained to perform both additions and multiplications from right to left. In addition, multiplications are performed before additions. It is said that '$' takes *precedence* over '#'. Let us check this with the original problem expression.

$$
\begin{array}{lll}
\mathcal{V}_T[5 \,\#\, 3 \,\$\, 2 \,\#\, 6] & & \\
\quad = & \mathcal{V}_T[5 \,\#\, 3 \,\$\, 2] + \mathcal{V}_T[6] & {}_T(2) \\
\quad = & \big(\mathcal{V}_T[5] + \mathcal{V}_T[3 \,\$\, 2]\big) + \mathcal{V}_T[6] & {}_T(2) \\
\quad = & \big(\mathcal{V}_T[5] + \mathcal{V}_T[3 \,\$\, 2]\big) + \mathcal{V}_T[6] & {}_T(1) \\
\quad = & \big(\mathcal{V}_T[5] + \big(3 \times \mathcal{V}_T[2]\big)\big) + \mathcal{V}_T[6] & {}_T(2) \\
\quad = & \big(5 + \big(3 \times 2\big)\big) + 6 & {}_T(1), \text{ three times} \\
\quad = & 17 & \text{arithmetic}
\end{array}
$$

Since this is the only way to evaluate the word, we are correct in regarding $\mathcal{V}_T$ and V_F as functions.

8.4 Environments

The languages we have seen so far have contained only constants and operations. Computer languages also contain program variables; this is what gives them much of their power. In most languages, a program variable designates some computer-memory location which contains the value. The language compiler translates the symbolic variable name into an address, which is used by the computer to retrieve the designated value.

For our purposes, it is all right to think of the computer's memory as a device that maps the symbolic variable directly to a value; we shall not concern ourselves with the hidden translation from identifier to address. Thus, a memory can be modeled as a function from the domain of program variables to the range of interpreted values. We call such a mapping an *environment*.

As a program executes, its environment changes. The value associated with a program variable is altered by assignment statements, the binding of procedure parameters, and so forth. Our language specifications must reflect this fact and this is done by including the environment in the definition of the interpretation.

Example

Ex 8.4 Let IDE be a set of program variables (such as `x` , `alpha` , `I5` , *etc.*). Let ENV be the set of all environments,

$$\text{ENV} = \{\sigma \mid \sigma\colon \text{IDE} \to \mathbb{N}\}$$

Finally, take V to be the alphabet

$$V = \mathbb{N} \cup \text{IDE} \cup \{\texttt{\#}\}$$

and define a prefix (hence unambiguous) language $L_6 \subseteq V^+$ and interpretation function $\mathcal{V}\colon L_6 \times \text{ENV} \to \mathbb{N}$ according to:

	$L_6 \subseteq V^+$	$\mathcal{V}_6\colon L_6 \times \text{ENV} \to \mathbb{N}$
1*a*.	$\mathbb{N} \subseteq L_6$	$\mathcal{V}_6[k](\sigma) = k,$ for $k \in \mathbb{N}$
1*b*.	$\text{IDE} \subseteq L_6$	$\mathcal{V}_6[v](\sigma) = \sigma(v),$ for $v \in \text{IDE}$
2.	$u, v \in L_6 \Rightarrow \texttt{\#}\ u\ v \in L_6$	$\mathcal{V}_6[\texttt{\#}\ u\ v](\sigma) = \mathcal{V}_6[u](\sigma) + \mathcal{V}_6[v](\sigma)$
3.	nothing else	

We write $\mathcal{V}_6[u](\sigma)$ rather than $\mathcal{V}_6[u, \sigma]$ or $\mathcal{V}(u, \sigma)$ because it is a little clearer. Since the environment variable is always a single letter, we will later drop the surrounding parentheses.

Copyright © 2008 Steven D. Johnson

Clause 1(b) is new; it specifies the interpretation of a program variable, $v \in$ IDE, whose value is provided by the environment σ.

Figure 8.4 defines a language of infix arithmetic expressions involving operations of addition, subtraction, multiplication, and negation. Among these operations, negation has the highest precedence, then multiplication, and addition and subtraction have lower, but equal, precedence. Precedence may be superseded by parentheses. All operations are performed from left to right. This example shows that by building the mathematical structures in the right way, we can be precise about the meaning of "ambiguous" languages.

Example

Ex 8.5 If environment $\sigma = \{(\texttt{row}, 5), (\texttt{col}, 8)\}$, then

$$
\begin{array}{ll}
\mathcal{V}_E[\texttt{5 * (row + 2) * - col)}]\sigma & \\
\quad = \mathcal{V}_T[\texttt{5 * (row + 2) * - col)}]\sigma & \mathcal{V}_E(1) \\
\quad = \mathcal{V}_F[\texttt{5}]\sigma \times \mathcal{V}_T[\texttt{(row + 2) * - col)}]\sigma & \mathcal{V}_T(2) \\
\quad = 5 \times \mathcal{V}_T[\texttt{(row + 2) * - col)}]\sigma & \mathcal{V}_F(1) \\
\quad = 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times \mathcal{V}_T[\texttt{- col)}]\sigma\big) & \mathcal{V}_F(1) \\
\quad = 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times \mathcal{V}_F[\texttt{- col)}]\sigma\big) & \mathcal{V}_T(1) \\
\quad = 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-\mathcal{V}_E[\texttt{col)}]\sigma\big) & \mathcal{V}_F(2b) \\
\quad = 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-\mathcal{V}_T[\texttt{col)}]\sigma\big) & \mathcal{V}_E(1) \\
\quad = 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-\mathcal{V}_F[\texttt{col)}]\sigma\big) & \mathcal{V}_T(1) \\
\quad = 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-\sigma(\texttt{col}))\big) & \mathcal{V}_F(1b) \\
\quad = 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-8)\big) & (\texttt{col}, 8) \in \sigma \\
\quad = 5 \times \big(\mathcal{V}_E[\texttt{row + 2}]\sigma \times (-8)\big) & \mathcal{V}_F(2a) \\
\quad = 5 \times \big(\mathcal{V}_T[\texttt{row}]\sigma + \mathcal{V}_E[\texttt{2}]\sigma \times (-8)\big) & \mathcal{V}_E(2a) \\
\quad = 5 \times \big(\mathcal{V}_F[\texttt{row}]\sigma + \mathcal{V}_T[\texttt{2}]\sigma \times (-8)\big) & \mathcal{V}_T(1),\ \mathcal{V}_E(1) \\
\quad = 5 \times \big(\sigma(\texttt{row}) + \mathcal{V}_F[\texttt{2}]\sigma) \times (-8)\big) & \mathcal{V}_F(1b),\ \mathcal{V}_T(1) \\
\quad = 5 \times ((5 + 2) \times (-8)) & (\texttt{row}, 5) \in \sigma,\ \mathcal{V}_F(1a) \\
\quad = -280 & \text{arithmetic}
\end{array}
$$

Exercises 8.4

1. Add a division operation symbol '`/` ' to the language E in Figure 8.4 in such a way that 'T/' and '`*` ' have equal precedence but there is no ambiguity.

Copyright © 2008 Steven D. Johnson

Let $V = \mathbb{N} \cup \text{IDE} \cup \{$ `(`, `)`, `+`, `-`, `*` $\}$. Simultaneously define (see Exercise 5, Section 4) the languages F, T, E and interpretations $\mathcal{V}_F, \mathcal{V}_T, \mathcal{V}_E$ as follows:

	$F \subseteq V^+$	$\mathcal{V}_F : F \times \text{ENV} \to \mathbb{N}$
$1a.$	$\mathbb{N} \subseteq F$	$\mathcal{V}_F[k]\sigma = k$, for $k \in \mathbb{N}$
$1b.$	$\text{IDE} \subseteq F$	$\mathcal{V}_F[v]\sigma = \sigma(v)$, for $v \in \text{IDE}$
$2a.$	$e \in E \Rightarrow$ `(` e `)` $\in F$	$\mathcal{V}_F[$ `(` e `)` $]\sigma = \mathcal{V}_E[e]\sigma$
$2b.$	$e \in E \Rightarrow$ `-` $e \in F$	$\mathcal{V}_F[$ `-` $e]\sigma = -\mathcal{V}_E[e]\sigma$
3.	nothing else	

	$T \subseteq V^+$	$\mathcal{V}_T : T \times \text{ENV} \to \mathbb{N}$
1.	$F \subseteq T$	$\mathcal{V}_T[f]\sigma = \mathcal{V}_F[f]\sigma$ for $f \in F$
2.	$f \in F, t \in T \Rightarrow f$ `*` $t \in T$	$\mathcal{V}_T[f$ `*` $t]\sigma = \mathcal{V}_F[f]\sigma \times \mathcal{V}_T[t]\sigma$
3.	nothing else	

	$E \subseteq V^+$	$\mathcal{V}_E : E \times \text{ENV} \to \mathbb{N}$
1.	$T \subseteq E$	$\mathcal{V}_E[t]\sigma = \mathcal{V}_T[t]\sigma$ for $t \in T$
$2a.$	$t \in T, e \in E \Rightarrow t$ `+` $e \in E$	$\mathcal{V}_E[t$ `+` $e]\sigma = \mathcal{V}_T[t]\sigma + \mathcal{V}_E[e]\sigma$
$2b.$	$t \in T, e \in E \Rightarrow t$ `-` $e \in E$	$\mathcal{V}_E[t$ `-` $e]\sigma = \mathcal{V}_T[t]\sigma - \mathcal{V}_E[e]\sigma$
3.	nothing else	

Figure 8.1: A language of infix arithmetic expressions and its interpretation

2. Add a division operation symbol '`/`' to the language E in Figure 8.4 in such a way that division operations are performed from right to left. Can right-to-left division and left-to-right multiplication have equal precedence?

3. Let environment $\sigma = \{(\texttt{a}, 3), (\texttt{b}, -2), (\texttt{c}, 5)\}$. Evaluate one of the following words from the language E of Figure ?.

(a) `a + 6 - 3 * 5` (c) `b * 4 + - - y`
(b) `((b * 2) * c) * 3` (d) `a * - (b + 5)`

8.5 Backus-Naur Form

Languages defined in the manner of the previous sections are called *context free languages*. There is a standard notation in computer science for context free grammars. It is called *Backus-Naur form* or *BNF*, after John Backus and Peter Naur, who used it to specify the syntax of the algol 60 programming language. A BNF description of the language E in Figure 8.1 looks like this:

⟨F⟩ ::= ⟨NATURAL NUMBER⟩
⟨F⟩ ::= ⟨PROGRAM VARIABLE⟩
⟨F⟩ ::= `-` ⟨E⟩
⟨F⟩ ::= `(` ⟨E⟩ `)`

⟨T⟩ ::= ⟨F⟩
⟨T⟩ ::= ⟨F⟩ `*` ⟨T⟩

⟨E⟩ ::= ⟨T⟩
⟨E⟩ ::= ⟨T⟩ `+` ⟨E⟩
⟨E⟩ ::= ⟨T⟩ `-` ⟨E⟩

The names of inductively defined sets are surrounded by angle brackets $\langle \cdots \rangle$ and "$\langle L \rangle$ ::= *rule*" replaces "*rule* $\in L$. BNF allows us to describe languages concisely, without introducing variables for sub-phrases (e.g. f, t, and e in Figure 8.1).

Exercises 8.5

1. Give the BNF forms for the languages defined in Exercises 1 and 2.

2. Give the BNF form for the language of statements from Section 1.3. Assume that sets ⟨*assignmentexpression*⟩ and ⟨*testexpression*⟩ are already defined.

Copyright © 2008 Steven D. Johnson

3. If you were going to define an interpretation function for the language of statements, what would its domain and range be?

8.6 Propositional Formulas

In this section we shall explore applications of the language definition style just introduced. For concreteness, the language of propositional logic is used. However, bear in mind that, except for the syntax, the results at the end of this section are valid for other languages, such as arithmetic expressions.

Figure 8.2 defines a language, PROP, of *propositional formulas*. It follows the style of Figure 8.1 in an abbreviated form:

(a) The language PROP is defined using BNF.

(b) Rather than building the language in stages, as was done with arithmetic terms, 8.2 simply specifies the operator precedence.

(c) The interpretation function $\mathcal{P}$ implicitly assumes that the language has been disambiguated.

It is supposed that the Reader can fill in the necessary details when needed.

In Figure 8.2, PROPs' meanings are given in terms of environments.

$$\text{ENV} = \{\sigma \mid \sigma \colon IVS \to \{T, F\}\}$$

A given environment $\sigma \in \text{ENV}$ corresponds to one row of a truth table. Accordingly, let us redefine *tautology*, *contradiction*, and *logical equivalence* in terms of environments.

Definition 8.1 *A* PROP Q *is a* tautology *iff for every* $\sigma \in \text{ENV}$*,* $\mathcal{P}\sigma\,[Q] = T$*. A* PROP Q *is a* contradition *iff for every* $\sigma \in \text{ENV}$*,* $\mathcal{P}\sigma\,[Q] = F$*.*

Definition 8.2 *Two* PROP*s* P *and* Q *are* logically equivalent*, written* $P \,\text{eq}\, Q$*, iff for every* $\sigma \in \text{ENV}$*,* $\mathcal{P}\sigma\,[P] = \mathcal{P}\sigma\,[Q]$*.*

As a first exercise, let us validate the intuitive correspondence between logical equivalence (eq) and bi-implication ($\Leftrightarrow$). Proposition 8.1 confirms a fact that we have already used. Conversely, its proof helps validate that our new definitions are sensible. For the purpose of this proposition, suppose an implication operator, `=>`, is included in PROP.

Proposition 8.1 *If* P *and* Q *are* PROP*s, then for all* $\sigma \in \text{ENV}$

$$P \text{ eq } Q \text{ iff } \texttt{(}P \texttt{ => } Q\texttt{) \& (}Q \texttt{ => } P\texttt{)} \textit{ is a tautology.}$$

PROOF: The proof is a straightforward application of the definition of $\mathcal{P}$; induction is not required, although we do need to know that $\mathcal{P}$ is a function (not a relation or partial function).

Copyright © 2008 Steven D. Johnson

Let PVAR be a set of propositional variables, and the alphabet $A = \text{PVAR} \cup \{\,\texttt{(},\ \texttt{)},\ \texttt{0}\ \texttt{1},\ \texttt{-},\ \texttt{|},\ \texttt{\&}\ >\}$. The language PROP $\subseteq A^+$ of *propositional formulas* and its interpretation

$$\begin{array}{rl} \text{ENV}: & \text{PVAR} \to \{\mathit{true}, \mathit{false}\} \\ \mathcal{P}: & \text{ENV} \times \text{PROP} \to \{\mathit{true}, \mathit{false}\} \end{array}$$

are defined as follows:

$$\begin{array}{lcll} \langle\text{PROP}\rangle & ::= & \texttt{0} & \mathcal{P}_\sigma[\,\texttt{0}\,] = \mathit{false} \\ & \mid & \texttt{1} & \mathcal{P}_\sigma[\,\texttt{1}\,] = \mathit{true} \\ & \mid & \langle\text{PVAR}\rangle & \mathcal{P}_\sigma[\,v\,] = \sigma(v),\ \text{for } v \in \text{PVAR} \\ & \mid & \texttt{-}\,\langle\text{PROP}\rangle & \mathcal{P}_\sigma[\,\texttt{-}\,f\,] = \neg\,\mathcal{P}_\sigma[\,f\,] \\ & \mid & \texttt{(}\,\langle\text{PROP}\rangle\,\texttt{)} & \mathcal{P}_\sigma[\,\texttt{(}\,f\,\texttt{)}\,] = \mathcal{P}_\sigma[\,f\,] \\ & \mid & \langle\text{PROP}\rangle\ \texttt{\&}\ \langle\text{PROP}\rangle & \mathcal{P}_\sigma[\,f_1\,\texttt{\&}\,f_2\,] = \mathcal{P}_\sigma[\,f_1\,] \wedge \mathcal{P}_\sigma[\,f_2\,] \\ & \mid & \langle\text{PROP}\rangle\ \texttt{|}\ \langle\text{PROP}\rangle & \mathcal{P}_\sigma[\,f_1\ \texttt{|}\ f_2\,] = \mathcal{P}_\sigma[\,f_1\,] \vee \mathcal{P}_\sigma[\,f_2\,] \end{array}$$

with precedence $\boxed{-} \succ \boxed{\&} \succ \boxed{|}$.

Figure 8.2: A language of propositional formulas and its interpretation

IF PART: Let $\sigma \in$ ENV and assume that P eq Q, that is, by Definition 8.2, $\mathcal{P}\sigma\,[P] = \mathcal{P}\sigma\,[Q]$. Then

$$
\begin{array}{lll}
\multicolumn{3}{l}{\texttt{(}P\ T{=}{>}\ Q\texttt{)}\ \texttt{\&}\ \texttt{(}Q\ \texttt{=>}\ P\texttt{)}} \\
= & (\mathcal{P}\sigma\,[P] \Rightarrow \mathcal{P}\sigma\,[Q]) \wedge (\mathcal{P}\sigma\,[Q] \Rightarrow \mathcal{P}\sigma\,[P]) & \text{(Defn. of } \mathcal{P}) \\
= & (\mathcal{P}\sigma\,[P] \Rightarrow \mathcal{P}\sigma\,[P]) \wedge (\mathcal{P}\sigma\,[P] \Rightarrow \mathcal{P}\sigma\,[P]) & \text{(Assumption that } \mathcal{P}\sigma\,[P] = \mathcal{P}\sigma\,[Q]) \\
= & T & \text{(meanings of } \Rightarrow, \wedge)
\end{array}
$$

Thus, by Defintion 8.1, (P => Q) & (Q => P) is a tautology.

ONLY-IF PART: Assume that (P => Q) & (Q => P) is a tautology, and let $\sigma \in$ ENV be any environment.

$$
\begin{array}{lll}
T & = & \mathcal{P}\sigma\,[\texttt{(}P\ \texttt{=>}\ Q\texttt{)}\ \texttt{\&}\ \texttt{(}Q\ \texttt{=>}\ P\texttt{)}] \\
 & = & (\mathcal{P}\sigma\,[P] \Rightarrow \mathcal{P}\sigma\,[Q]) \wedge (\mathcal{P}\sigma\,[Q] \Rightarrow \mathcal{P}\sigma\,[P]) \qquad \text{(defn. } \mathcal{P})
\end{array}
$$

This equality can hold only if $\mathcal{P}\sigma\,[P] = \mathcal{P}\sigma\,[Q]$. Therefore, by Definition 8.2, P *eq* Q. ∎

Example

Ex 8.6 A function that translates sentences from one language to another (possibly the same) language called a *transliteration*. We are often interested in whether and how transliterations change the meaning of the original sentence. This example applies this idea to define a generalization of DeMorgan's identity for boolean algebras.

The *DeMorgan Dual* of $f \in$ PROP is obtained by switching all 0's and 1's, +'s and *'s, and inserting a - just before every variable symbol.

For instance, the DeMorgan dual of `( a &  b) | (( c | 0) & ( -b |  a))`
is `(-a | -b) & ((-c & 1) | (--b & -a))`

Inspecting the DNFs of these formulas: $\left\{ \begin{array}{l} \overline{a}\,\overline{b}\,c + a\,\overline{b}\,c + a\,b\,\overline{c} + a\,b\,c \\ \overline{a}\,\overline{b}\,\overline{c} + \overline{a}\,b\,\overline{c} + \overline{a}\,b\,c + a\,\overline{b}\,\overline{c}. \end{array} \right\}$ reveals that they are negations of each other—they have no clauses in common and together contain all eight possible clauses.

(a) *Define a recursive function* $\mathbb{D}$: PROP $\to$ PROP *that gives the DeMorgan dual of any* $F \in$ PROP.

$$
\begin{array}{rcl}
\mathbb{D}[\,\texttt{0}\,] & = & \texttt{1} \\
\mathbb{D}[\,\texttt{1}\,] & = & \texttt{0} \\
\mathbb{D}[\,v\,] & = & \texttt{-}\hat{}v \text{ for } v \in IVS \\
\mathbb{D}[\,\texttt{-}\ P\,] & = & \texttt{-}\hat{}\mathbb{D}[\,P\,] \\
\mathbb{D}[\,P\ \texttt{|}\ Q\,] & = & \mathbb{D}[\,P\,]\hat{}\texttt{\&}\hat{}\mathbb{D}[\,Q\,] \\
\mathbb{D}[\,P\ \texttt{\&}\ Q\,] & = & \mathbb{D}[\,P\,]\hat{}\,\texttt{|}\,\hat{}\mathbb{D}[\,Q\,]
\end{array}
$$

(b) *Prove that the DeMorgan dual of a propositional formula is its logical negation:* For all $\sigma \in \text{ENV}$ and $F \in \text{PROP}$, $\mathcal{P}_\sigma[\,\mathbb{D}[\,F\,]\,] = \neg\,\mathcal{P}_\sigma[\,F\,]$.

The proof is a straightforward induction on words in PROP. The crux of the argument is in the base case for variables, where for any $v \in \mathit{IVS}$ we have

$$\mathcal{P}_\sigma[\,\mathbb{D}[\,v\,]\,] = \mathcal{P}_\sigma[\,\texttt{-}\hat{}v\,] = \neg\mathcal{P}_\sigma[\,v\,]$$

All steps above are justfied by the definitions of $\mathcal{P}$ or $\mathbb{D}$. An example of the inductive cases, for formulas of the form $P \texttt{ + } Q$, is

$$\begin{array}{rcll} \mathcal{P}_\sigma\big[\mathbb{D}[\,P \;\texttt{|}\; Q\,]\big] & = & \mathcal{P}_\sigma\big[\mathbb{D}[\,P\,]\hat{}\texttt{\&}\hat{}\mathbb{D}[\,Q\,]\big] & \text{(defn. } \mathbb{D}) \\ & = & \mathcal{P}_\sigma\big[\mathbb{D}[\,P\,]\big] \wedge \mathcal{P}_\sigma\big[\mathbb{D}[\,Q\,]\big] & \text{(defn. } \mathcal{P}) \\ & \overset{H}{=} & \neg\mathcal{P}_\sigma[\,P\,] \;\wedge\; \neg\mathcal{P}_\sigma[\,Q\,] & \text{(I.H. used twice)} \\ & = & \neg\,(\mathcal{P}_\sigma[\,P\,] \;\vee \mathcal{P}_\sigma[\,Q\,]) & \text{(DeMorgan's Identity)} \\ & = & \neg\mathcal{P}_\sigma[\,P\hat{}\,\texttt{|}\,\hat{}Q\,] & \text{(defn. } \mathcal{P}) \end{array}$$

∎

Exercises 8.6

1. Add an implication operator, ‘`=>`’ to PROP.

2. Use transliteration to add an implication “macro” to PROP. That is, define a language $\text{PROP}_\Rightarrow$ that includes ‘`=>`’ and a translation function $F\colon \text{PROP}_\Rightarrow \to \text{PROP}$ that correctly re-interprets f_1 `=>` f_2 as `(-` f_1 `+` f_2 `)` .

8.7 Substitution

A *substitution* is the simultanesous replacement of formulas for variables in an expression.

Definition 8.3 *Let* $F, P_1, \ldots, P_k \in \text{PROP}$ *and* $v_1, \ldots, v_k \in \text{PVAR}$. *The* substitution

$$F\left[{}^{P_1,\;\ldots,\;P_k}_{v_1,\;\ldots,\;v_k}\right]$$

denotes the result, $\mathcal{S}[\,F\,]$*, of a substitution function* $S\colon \text{PROP} \to \text{PROP}$ *defined*

as follows:

$$\begin{array}{rcl} \mathcal{S}[\,\texttt{0}\,] &=& 0 \\ \mathcal{S}[\,\texttt{1}\,] &=& 1 \\ \mathcal{S}[\,x\,] &=& \begin{cases} P_j & \textit{if } x = v_j \\ x & \textit{otherwise} \end{cases} \\ \mathcal{S}[\,\texttt{-}\;Q\,] &=& \texttt{-}\hat{\ }\mathcal{S}[\,P\,] \\ \mathcal{S}[\,Q\;\texttt{|}\;Q'\,] &=& \mathcal{S}[\,Q\,]\hat{\ }\texttt{|}\hat{\ }\mathcal{S}[\,Q'\,] \\ \mathcal{S}[\,Q\;\texttt{\&}\;Q'\,] &=& \mathcal{S}[\,Q\,]\hat{\ }\texttt{\&}\hat{\ }\mathcal{S}[\,Q'\,] \end{array}$$

According to the definition, we often consider the substitution *function*

$$\mathcal{S} \text{ as specified by } \left[{}^{P_1,\ \ldots,\ P_k}_{v_1,\ \ldots,\ v_k}\right]$$

which may be applied to any formula $F \in$ PROP, writing write $\mathcal{S}[\,P\,]$ to denote the result.

Example

Ex 8.7

$$\texttt{p * (-q + r)}\left[{}^{\texttt{q (s+t) p}}_{\texttt{p}\quad\texttt{q}\quad\texttt{r}}\right] \equiv \texttt{q * ((s+t) + p)}$$

In performing substitutions, one must take care to preserve operator precedence. Above, this is done by parenthesizing `(s + t)`*.*

The three theorems that follow verify the fundamental rules for logical manipulations involving substitution. Since we have been using these results all our lives, another way to look at these theorems is that they validate the definitions given so far.

Lemma 8.2 (Substitution Lemma) *Let $\mathcal{S}$ be a substitution and σ an environment. Define an new environment σ' as follows:*

$$\sigma'(v) = \mathcal{P}\sigma\,[\mathcal{S}(v)] \text{ for } v \in \mathit{IVS}$$

Then for all PROP*s P,*

$$\mathcal{P}\sigma'\,[P] = \mathcal{P}\sigma\,[\mathcal{S}\,[P]]$$

PROOF: The proof is a straightforward structural induction on PROP. The inductive cases hold because the operations are functions. The interesting base case is the one for a variable $v \in \mathit{IVS}$. In that case, we have

$$\mathcal{P}\sigma'\,[v] = \mathcal{P}\sigma\,[\mathcal{S}\,[p]]$$

which is exactly what we need to make the Theorem true. ∎

The Substitution Lemma states that for the language of propositions, a call-by-value style evaluation—in which variables are bound to expressions' values—is equivalent to a call-by-name style evaluation. This is an exact equivalence because there is no form of looping in the language.

More significantly, the lemma says that our notion of substitution interacts well with our notion of evaluation. For example,

Theorem 8.3 (Tautology Theorem) *Let P be a* PROP *and $\mathcal{S}$ a substitution, If P is a tautology, then so is $\mathcal{S}[P]$.*

PROOF: Apply the definition of *tautology* and the Substitution Lemma. The details are left as Exercise **??**. ∎

For example, the formula

$$Q \equiv \texttt{(p | q \& (p => r)) | -(p | q \& (p => r))}$$

is a tautology because `p | -p` is a tautology and there is a subsitition,

$$\mathcal{S}[\texttt{p}] = \texttt{(p | q \& (p => r))}$$

under which

$$Q \equiv \mathcal{S}[\texttt{p | -p}]$$

Theorem 8.3 gives us one way to abbreviate the analysis of PROPs. The next theorem states that we can analyze PROP *schemes* as well as individual PROPs.

Theorem 8.4 (Substitution Theorem) *If $P \,\mathrm{eq}\, Q$ then for any substitution $\mathcal{S}$, $\mathcal{S}[P] \,\mathrm{eq}\, \mathcal{S}[Q]$*

PROOF: Use the Substitution Lemma. ∎

Thus, not only is formula

$$\texttt{p | (q | r)} \;\mathrm{eq}\; \texttt{(p | q) | r}$$

but the two formula *schemes*,

$$P \texttt{ | (} Q \texttt{ | } R \texttt{)} \quad \text{and} \quad \texttt{(} P \texttt{ | } Q \texttt{) |} R$$

are equivalent for arbitray sub-PROPs P, Q, and R. In fact, we reason about PROP schemes far more often than we reason about PROP individuals.

The following result is very important to the way we do proofs. It says you can "replace equals with equals" and still preserve equivalence.

Theorem 8.5 (Replacement Theorem) *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be substitutions such that, for all $v \in IVS$, $\mathcal{S}_1(v) \;\mathrm{eq}\; \mathcal{S}_2(v)$. Then for any* PROP *$P$,*

$$\mathcal{S}_1[P] \;\mathrm{eq}\; \mathcal{S}_2[P]$$

Copyright © 2008 Steven D. Johnson

PROOF: Let σ be any environment and define σ' to be

$$\sigma'(v) = \mathcal{P}\sigma\,[\mathcal{S}_1[v]]$$

Since $\mathcal{S}_1(v)$ eq $\mathcal{S}_2(v)$, it is also the case that $\sigma'(v) = \mathcal{P}\sigma\,[\mathcal{S}_2[v]]$ for all $v \in IVS$. Using the Substitution Lemma twice, we have

$$\mathcal{P}\sigma\,[\mathcal{S}_1[P]] = \mathcal{P}\sigma'\,[P] = \mathcal{P}\sigma\,[\mathcal{S}_2[P]]$$

as desired. ∎

For example, `p⇒q` eq `¬q ⇒ ¬p` so

`(q ⇒ p) ∧ (p ⇒ q)` eq `(q ⇒ p) ∧ (¬q ⇒ ¬p)`

under the substitutions

v	$\mathcal{S}_1[v]$	$\mathcal{S}_2[v]$
`p`	`p`	`p`
`q`	`q`	`q`
`r`	`p ⇒ q`	`¬q ⇒ ¬p`

These results about substitution and replacement do not depend in any fundamental way on the syntax of formulas in PROP. Exercise **??** asks you to define substitution for arithmetic expressions. Exercise 3 asks you to consider a more general definition of substitution, applicable to any language defined in the style developed in this chapter.

The question of interpretation is more important. Is it the case that all recursively defined interpretations allow substitution? One answer is that substitution is so important that only those interpretations that make the Substitution Lemma (Lemma 8.2) true are allowed.

Exercises 8.7

1. Let $F \equiv p \wedge (q \vee r)$. Perform the following substitutions

(a) $F\left[\begin{smallmatrix} r, & q, & p \\ p, & q, & r \end{smallmatrix}\right]$

(b) $F\left[\begin{smallmatrix} p \vee r \\ p \end{smallmatrix}\right]$

(c) $F\left[\begin{smallmatrix} p \vee r, & q \Rightarrow r \\ p, & r \end{smallmatrix}\right]$

(d) $\left(F\left[\begin{smallmatrix} p \vee r \\ p \end{smallmatrix}\right]\right)\left[\begin{smallmatrix} q \\ p \end{smallmatrix}\right]$

(e) $\left(F\left[\begin{smallmatrix} q \\ p \end{smallmatrix}\right]\right)\left[\begin{smallmatrix} p \vee r \\ p \end{smallmatrix}\right]$

Copyright © 2008 Steven D. Johnson

2. Define substitution for arithmetic terms as defined in Figure 8.1. Show that the Substitution Lemma (Lemma 8.2) holds for their interpretation.

3. Describe in general terms the process of defining substitution for any language that contains variables. Try to write down the appropriate generalized definitions and theorems.

8.8 The Programming Language of Statements

The STMT programming language was first introduced in Chapter 1 and has been referred to often throughout this textbook. In this section we shall apply the definitional style developed in this chapter to write a more rigorous specification of program syntax and meaning. These specifications appear in Definitions 8.4 and 8.5.

Sentences in STMT—programs—are built from a set of keywords,

{`begin`, `end`, `if`, `then`, `else`, `while`, `do`, `:=`, `;`}

and phrases coming from:

(a) The language of *arithmetic terms* used in assignment statements. This language and its interpretation are essentially the same as that of Figure 8.1.

(b) *Test expressions* used in `if` and `while` statements. Test expressions are logical combinations of arithmetic comparisons. Their interpretation is straightforward to define and is left as an exercise.

Discussion Points The clauses in Definition 8.5 are subtle. They must be studied carefully.

- Programs express computation in terms of *assignment*: recording information in a memory. The interpretation of statements reflects this model. A program starts with an initial memory, runs for a while, and then stops, leaving its results in an updated memory. We model memories abstractly with environments mapping program variables to values. Thus, the interpretation function maps from environments to environments.

$$\mathcal{M}\colon \text{ENV} \times \text{STMT} \xrightarrow{p} \text{ENV}$$

  $\mathcal{M}$ is a *partial* function. For non-terminating programs it does not give a value, as discussed below.

- The interpretation of assignment says to take the environment function σ, remove the ordered pair for program variable V, and replace it with one that binds V to the value of T.

$$[\sigma \setminus \{(V, \sigma(V)\}] \cup \{(v, \mathcal{T}_\sigma[\,T\,]\}$$

Copyright © 2008 Steven D. Johnson

- The compound-statement interpretations says, execute statement S_1, and then execute S_2 using this resulting memory, σ'. It might have been written even more mysteriously as

$$\mathcal{M}_{(\mathcal{M}_\sigma[S_1])}[\, S_2 \,]$$

- Unlike all other rules, the rule for `while`-statements does not reduce interpretation to a *proper* sub-sentence. The interpretation of certain statements is undefined. For instance,

$$\begin{array}{ll} & \mathcal{M}_\sigma[\,\texttt{while}\ x = x\ \texttt{do}\, x := x\,] \\ & \boxed{\mathcal{P}_\sigma[\, x = x \,] = \cdots = \mathit{true}} \\ \stackrel{(d)}{==} & \mathcal{M}_{\sigma'}[\,\texttt{while}\ x = x\ \texttt{do}\, x := x\,] \\ & \boxed{\text{where } \sigma'(x) = \mathcal{M}_\sigma[\, x := x \,] = \cdots = \sigma} \\ = & \mathcal{M}_\sigma[\,\texttt{while}\ x = x\ \texttt{do}\, x := x\,] \\ \vdots & \end{array}$$

  Of course, this is just what we want our model to describe: a non-terminating program does not produce a "final" memory.

Let us use Definition 8.5 prove an interesting fact about compound statements.

Proposition 8.6 *The compound operator, '`;`' is associative in the sense that for all $\sigma \in$ ENV and statements S_1, S_2, and S_3,*

$$\begin{array}{ll} & \mathcal{M}_\sigma[\,\texttt{begin begin}\ S_1\ \texttt{;}\ S_2\ \texttt{end ;}\ S_3\ \texttt{end}\,] \\ = & \mathcal{M}_\sigma[\,\texttt{begin}\ S_1\ \texttt{; begin}\ S_2\ \texttt{;}\ S_3\ \texttt{end end}\,] \end{array}$$

PROOF: Apply Definition 8.5. ∎

Thus, it is not ambiguous to write `begin` S_1 `;` S_2 `;` S_3 `end` because it doesn't matter how `begin`-`end`s are associated.

The next relatively simple fact is the first step in reducing program correctness statements to purely logical conditions. Recall that the notation `{`P`} S {`Q`}` says, "If statement S starts with a memory in which property P holds (and if it terminates), property Q holds in the final memory. In more precise terms, for all $\sigma \in$ ENV,

$$\mathcal{P}_\sigma[\, P \,] \text{ implies } \mathcal{P}_{\sigma'}[\, Q \,] \text{ where } \sigma' = \mathcal{M}_\sigma[\, S \,]$$

Proposition 8.7 $\{P\}\, V$ `:=` $T\, \{Q\}$ *iff* $P \Rightarrow Q^T_V$.

PROOF: ($\Rightarrow$) Let $\sigma \in$ ENV, and assume $\{P\}\, V$ `:=` $T\, \{Q\}$ is true. Then $\mathcal{P}_\sigma[\, P \,] \Rightarrow \mathcal{P}_{\sigma'}[\, Q \,]$ where $\sigma' = \mathcal{M}_\sigma[\, V \texttt{:=} T \,]$. By Definition 8.5, $\sigma'(V) = \mathcal{T}_\sigma[\, T \,]$. Hence, by the Substitution Lemma (Lemma 8.2, suitably generalized), $\mathcal{P}_{\sigma'}[\, Q \,]$ is logically equivalent to $\mathcal{P}_\sigma[\, Q^T_V \,]$. Therefore, $\mathcal{P}_\sigma[\, P \Rightarrow Q^T_V \,]$ is true.

($\Rightarrow$) By the same argument as above. ∎

Definition 8.4 *The languages of arithmetic terms (*TERM*), comparisons (*COMP*), tests (*TEST*) and program statements (*STMT*) are defined according to the grammar below. All operators associate to the right with precedence* $\boxed{-}_1 \succ \boxed{*} \succ \boxed{+} = \boxed{-}_2$ *and*

$$
\begin{array}{lll}
\langle\text{TERM}\rangle & ::= & \langle\text{NUMERAL}\rangle \\
 & ::= & \langle\text{IDENTIFIER}\rangle \\
 & ::= & \texttt{-}_1\ \langle\text{TERM}\rangle \\
 & ::= & \texttt{(}\ \langle\text{TERM}\rangle\ \texttt{)} \\
 & ::= & \langle\text{TERM}\rangle\ \texttt{*}\ \langle\text{TERM}\rangle \\
 & ::= & \langle\text{TERM}\rangle\ \texttt{+}\ \langle\text{TERM}\rangle \\
 & ::= & \langle\text{TERM}\rangle\ \texttt{-}_2\ \langle\text{TERM}\rangle
\end{array}
$$

$$
\begin{array}{lll}
\langle\text{COMP}\rangle & ::= & \langle\text{TERM}\rangle\ \texttt{=}\ \langle\text{TERM}\rangle \\
 & ::= & \langle\text{TERM}\rangle\ <\ \langle\text{TERM}\rangle \\
\langle\text{TEST}\rangle & ::= & \langle\text{COMP}\rangle \\
 & ::= & \texttt{-}_3\ \langle\text{COMP}\rangle \\
 & ::= & \texttt{(}\ \langle\text{COMP}\rangle\ \texttt{)} \\
 & ::= & \langle\text{COMP}\rangle\ \&\ \langle\text{COMP}\rangle \\
 & ::= & \langle\text{COMP}\rangle\ \texttt{|}\ \langle\text{COMP}\rangle
\end{array}
$$

$$
\begin{array}{llll}
\langle\text{STMT}\rangle & ::= & \langle\text{IDENTIFIER}\rangle\ \texttt{:=}\ \langle\text{TERM}\rangle & \textit{(}\text{assignment}\textit{)} \\
 & ::= & \texttt{begin}\ \langle\text{STMT}\rangle\ \texttt{;}\ \langle\text{STMT}\rangle\ \texttt{end} & \textit{(}\text{compound}\textit{)} \\
 & ::= & \texttt{if}\ \langle\text{TEST}\rangle\ \texttt{then}\ \langle\text{STMT}\rangle\ \texttt{else}\ \langle\text{STMT}\rangle & \textit{(}\text{conditional}\textit{)} \\
 & ::= & \texttt{while}\ \langle\text{TEST}\rangle\ \texttt{do}\ \langle\text{STMT}\rangle & \textit{(}\text{repetition}\textit{)}
\end{array}
$$

Definition 8.5 *Given interpretations* $\mathcal{T}$: ENV × TERM → $\mathbb{N}$ *and* $\mathcal{P}$: ENV × TEST → $\{true,\ false\}$, *the operational meaning of programs in* STMT *(Definition 8.4) is given by the partial function* $\mathcal{M}$: ENV × STMT → ENV *defined as follows:*

(*a*) $\mathcal{M}_\sigma[\,V\ \texttt{:=}\ T\,] = [\sigma \setminus \{(V, \sigma(V)\}] \cup \{(V, \mathcal{T}_\sigma[\,T\,]\}$

(*b*) $\mathcal{M}_\sigma[\,\texttt{begin}\ S_1\ \texttt{;}\ S_2\ \texttt{end}\,] = \mathcal{M}_{\sigma'}[\,S_2\,]$ where $\sigma' = \mathcal{M}_\sigma[\,S_1\,]$

(*c*) $\mathcal{M}_\sigma[\,\texttt{if}\ Q\ \texttt{then}\ S_1\ \texttt{else}\ S_2\,] = \begin{cases} \mathcal{M}_\sigma[\,S_1\,] & \text{if } \mathcal{P}_\sigma[\,Q\,] = \textit{true} \\ \mathcal{M}_\sigma[\,S_2\,] & \text{if } \mathcal{P}_\sigma[\,Q\,] = \textit{false} \end{cases}$

(*d*) $\mathcal{M}_\sigma[\,\texttt{while}\ Q\ \texttt{do}\,S\,] = \begin{cases} \sigma, & \text{if } \mathcal{P}_\sigma[\,Q\,] = \textit{false} \\ \mathcal{M}_{\sigma'}[\,\texttt{while}\ Q\ \texttt{do}\ S\,] & \text{if } \mathcal{P}_\sigma[\,Q\,] = \textit{true} \\ \quad \text{where } \sigma' = \mathcal{M}_\sigma[\,S\,], & \end{cases}$

Copyright © 2008 Steven D. Johnson

Example

Ex 8.8 In Proposition 4.1 (p. **??**) it was shown that

```
{z + xy = AB}
while x ≠ 0 do
   begin
   x := x − 1;
   z := z + y
   end;
end {z = AB}
```

The proof of invariance involved reasoning about the values of identifiers before (x, y and z) and after (x', y' and z') executing the loop body. Proposition 8.7 says that this kind of temporal reasoning may be reduced to "pure logic":

$$\{z + xy = AB \ \wedge\ x \neq 0\}\,\texttt{begin}\ x \,\texttt{:=}\, x - 1\,\texttt{;}\ z \,\texttt{:=}\, z + 1\ \texttt{end}\{z + xy = AB\}$$

eq (by Prop. 8.7)

$$\{z + xy = AB \ \wedge\ x \neq 0\}\ x \,\texttt{:=}\, x - 1\ \left\{\big(z + xy = AB\big)\left[{}^{z+1}_{\ z}\right]\right\}$$

eq (by Prop. 8.7)

$$\big(z + xy = AB \ \wedge\ x \neq 0\big)\ \Rightarrow\ \Big(\big(z + xy = AB\big)\left[{}^{z+y}_{\ z}\right]\Big)\left[{}^{x-1}_{\ x}\right]$$

Performing these substitutions, we get

$$\big(z + xy = AB \ \wedge\ x \neq 0\big)\ \Rightarrow\ \Big(\big(z + xy = AB\big)\left[{}^{z+y}_{\ z}\right]\Big)\left[{}^{x-1}_{\ x}\right]$$

eq

$$\big(z + xy = AB \ \wedge\ x \neq 0\big)\ \Rightarrow\ \big((z + y) + xy = AB\big)\left[{}^{x-1}_{\ x}\right]$$

eq

$$\big(z + xy = AB \ \wedge\ x \neq 0\big)\ \Rightarrow\ \big[(z + y) + (x - 1)y = AB\big]$$

eq (simplifying $(x + y) + (x - 1)y$)

$$\big(z + xy = AB \ \wedge\ x \neq 0\big)\ \Rightarrow\ \big(z + xy = AB\big)$$

Which is tautologically true.

Exercises 8.8

1. Define interpretations for arithmetic and test expressions. $\mathcal{T}$: ENV × TERM → $\mathbb{N}$ $\mathcal{P}$: ENV × TEST → $\{true,\ false\}$.

2. In Example 8.8 the subformula

$$\Big(\big(z + xy = AB\big)\left[{}^{z\,+\,y}_{z}\right]\Big)\left[{}^{x\,-\,1}_{x}\right]$$

is derived. Is this the same as

$$\big(z + xy = AB\big)\left[{}^{z\,+\,y,\; x\,-\,1}_{z,\qquad x}\right]$$

3. Suppose that, for all $\sigma \in \textsc{env}$, $T \in \textsc{term}$ and $C \in \textsc{comp}$, $\mathcal{T}_\sigma[\,T\,] = 42$ and $\mathcal{P}_\sigma[\,C\,] = \mathit{true}$. Describe how programs in STMT behave.

8.9 *Discussions

8.9.1 Parenthesized Expressions

In Example 8.2 it is claimed that fully parenthesized expressions are unambiguous. This fact seems intuitively obvious but proving it rigourously requires delving into that intuition. The essential property of this language is that its parentheses are "balanced."

As in the example, Let alphabet $A = \mathbb{N} \cup \{ \texttt{\#}, \texttt{\$}, \texttt{(}, \texttt{)} \}$. For $w \in A^+$, define $\Delta[w]$ to be the number of left parentheses in w minus the number of right parentheses. For example, $\Delta[\texttt{((}5\,\texttt{\$}\,2\,\texttt{()}] = 2$ and $\Delta[\texttt{)}3\texttt{()}7\texttt{(}] = 0$.

The language L_2 of 8.2 was defined

	$L_2 \subseteq A^+$			$\mathcal{V}\colon L_2 \to \mathbb{N}$
1.	$\langle L_2 \rangle$	$::=$	$\mathbb{N}$	$\mathcal{V}[\,n\,] = n$ for $n \in \mathbb{N}$
2a.		|	$\texttt{(}\langle L_2\rangle\,\texttt{\#}\,\langle L_2\rangle\texttt{)}$	$\mathcal{V}[\,\texttt{(}\,u\,\texttt{\#}\,v\,\texttt{)}\,] = \mathcal{V}[\,u\,] + \mathcal{V}[\,v\,]$
2b.		|	$\texttt{(}\langle L_2\rangle\,\texttt{\$}\,\langle L_2\rangle\texttt{)}$	$\mathcal{V}[\,\texttt{(}\,u\,\texttt{\$}\,v\,\texttt{)}\,] = \mathcal{V}[\,u\,] \times \mathcal{V}[\,v\,]$

Proposition 8.8 *Let L_2 be the language defined in Example 8.2. Then for all $w \in L_2$, $\Delta[w] = 0$.*

PROOF: The proof is by structural induction on L_2. In the base case, if $k \in \mathbb{N}$ then k contains no parentheses, so $\Delta[k] = 0$. For the induction case, assume $\Delta[u] = \Delta[v] = 0$. Then

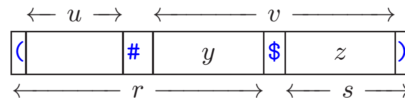
$$\Delta[\,\texttt{(}\,u\,\texttt{*}\,v\,\texttt{)}\,] = 1 + \Delta[u] + \Delta[v] - 1 = 0$$

And similarly for $\Delta[\,\texttt{(}\,u\,\texttt{+}\,v\,\texttt{)}\,]$. ∎

Proposition 8.8 captures only part of the quality of being balanced. How can we capture the notion of being *properly* balanced? The answer is not obvious, but a little experimentation will convince you that the next proposition is what we need:

Proposition 8.9 *Let L_2 be the language defined in Example 8.2, and let $w \in L_2$. If $w = r\,\texttt{\#}\,s$ (or $r\,\texttt{\$}\,s$) for any two words $r, s \in V^+$, then $\Delta[r] > 0$ and $\Delta[s] < 0$.*

PROOF: The proof is by structural induction on L_2 and the base case holds vacuously. For the induction case, assume $w = \texttt{(}\,u\,\texttt{\#}\,v\,\texttt{)}$ and that the induction hypothesis holds for u and v. Now suppose that w also equals $r\,\texttt{\$}\,s$, so that we have the following:



Copyright © 2008 Steven D. Johnson

By Proposition 8.8 $\Delta[u] = 0$, and by the induction hypothesis, $\Delta[y] > 0$. Hence,

$$\Delta[r] = \Delta[\texttt{(}u\ \texttt{\#}\ y] = 1 + \Delta[y] > 0$$

Again by Proposition 8.8, $\Delta[w] = 0$, so it must be the case that

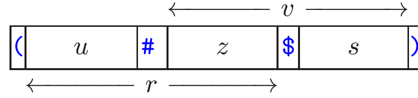
$$\Delta[s] = \Delta[z\texttt{)}] = -\Delta[r] < 0$$

A similar argument holds if the '$' occurs to the right of the '#'. ∎

We are now in a position to prove that each word in L_2 has a unique parse.

Proposition 8.10 *Let L_2 be the language defined in Example 8.2, and let $w \in L_2$, $w \notin \mathbb{N}$. Then there is exactly one pair of words, $u, v \in L_2$ such that $w = \texttt{(}u\ \texttt{\#}\ v\texttt{)}$ (or $\texttt{(}u\ \texttt{\$}\ v\texttt{)}$).*

PROOF: We will assume that there are two such pairs and reach a contradiction. Without loss in generality, assume that $w = \texttt{(}u\ \texttt{\#}\ v\texttt{)}$ and $w = \texttt{(}r\ \texttt{\$}\ s\texttt{)}$ and u, v, r, $s \in L_2$, so that we have the following:



By Proposition 8.8, $r \in L_2$ implies $\Delta[z] < 0$ while $u \in L_2$ implies $\Delta[z] > 0$. This is a contradiction, so either u and v or r and s do not exist. ∎

Copyright © 2008 Steven D. Johnson DRAFT MATERIAL (v08r0) December 3, 2008