The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

Solution Manual
Chapter 5, Problem 5.1

Almost all of these are easy to define informally; the point of the exercise is to write completely formal definitions—i.e., to write programs in the primitive recursive definition language that a compiler would accept as syntactically correct.

1. We use primitive recursion to distinguish the case $n = 0$ from the others:

$$\begin{cases} \text{lev}(0) = 0 \\ \text{lev}(n+1) = \text{Succ}(\text{Zero}(\Pi_1^2(n, \text{lev}(n)))) \end{cases}$$

2. Much the same definition:

$$\begin{cases} \text{lev}(0) = 1 \\ \text{lev}(n+1) = \text{Zero}(\Pi_1^2(n, \text{lev}(n))) \end{cases}$$

3. We use recursion on the second argument (we have seen that we can shift the recursion onto any argument).

$$\begin{cases} \text{minus}(x, 0) = \Pi_1^1(x) \\ \text{minus}(x, n+1) = \text{dec}(\Pi_2^3(x, \text{minus}(x, n), n)) \end{cases}$$

4. This one follows the model of addition, this time using repeated additions to carry out the multiplication..

$$\begin{cases} \text{mult}(0, x) = \text{Zero}(x) \\ \text{mult}(n+1, x) = \text{add}(\Pi_2^3(n, \text{mult}(n, x), x), \Pi_3^3(n, \text{mult}(n, x), x)) \end{cases}$$

5. This is the most interesting of the bunch, mostly because of the lazy evaluation. We write $\#(x, y)$ to preserve function notation in the definition, even though we normally use the function in infix notation.

$$\begin{cases} \#(0, y) = \text{Zero}(y) \\ \#(n+1, y) = \Pi_3^3(n, \#(n, y), y) \end{cases}$$

Note that, since the projection function is lazy, it will not evaluate its first or second argument and thus recursion is not actually invoked; more importantly, the Zero function does not evaluate its argument either, so that the base case functions without examining or evaluating the argument y at all.