The Theory of Computation
Bernard M.E. Moret
Addison-Wesley-Longman, 1998

Detailed Solution
Chapter 5, Problem 3

We begin by defining a two-case scheme; multiple cases are an easy generalization from there. We have seen two mechanisms for definition by cases already, both somewhat specialized: one is the primitive recursion scheme, which distinguishes between a base case and a recursive case; and the other is the guard function, which is very similar to our problem, but returns 0 in one of the cases.

A fairly easy solution is to use the guard function twice, once with predicate P and once with predicate $notP$, and concatenate the two results—one of the two will always be 0. This idea yields the definition

$$f(x_1, \ldots, x_n) = \text{con}_2(P(x_1, \ldots, x_n)\#g(x_1, \ldots, x_n), (\text{not}P(x_1, \ldots, x_n))\#f(x_1, \ldots, x_n))$$

which is a valid primitive recursive definition by substitution and clearly does what is intended. It remains to verify that the semantics of the definition allow the function to return one of g or h without computing the other. We have already noted that the guard function, when its predicate is false (when it returns 0), does not evaluate its other argument; thus, when our function evaluates to the value of g, the other argument (the function h) of the second guard is not evaluated, and vice versa.

To generalize to multiple cases, we use concatenation of all of the cases, each guarded with a predicate that excludes all other cases; since the logical connectives are all primitive recursive, we can construct the appropriate predicates; the rest is clearly primitive recursive.