jpress-02written

Exercise 2.18

f(n) = {0 if n = 0; 2 + f(n-1) otherwise}

base case:

f(0)

= <definition of f(0)>

0

=> <0 = 2 * n [n := 0]>

2 * n //

inductive case, for `f(n + 1) = 2 * (n + 1)` assuming `f(n) = 2 * n` as the inductive hypothesis

f(n + 1)

= <definition of f(n)[n := n + 1]>

2 + f(n)

= <inductive hypothesis>

2 + 2 * n

= <2 = 2 * 1>

2 * 1 + 2 * n

= <combine like terms>

2 * (n + 1) //

Exercise 3.14(a)

The `closest-power` process generates a recursive process because the function calls itself on line 5 and uses itself to do work.

Exercise 3.15

(a)

`(f 1) => (g 0) => 1`

`(f 2) => (g 1) => (f 0) => 0`

`(f 3) => (g 2) => (f 1) => (g 0) => 1`

(b)

Even arguments cause `f` to return 0, and odd arguments consider `f` to return 1.

(c)

The process is iterative because the final function call does the work, rather than having the first call doing the return.