

Functional Languages

10.9 Exercises

- 10.18 In Figure 10.4 we evaluated our expression in normal order. Did we really have any choice? What would happen if we tried to use applicative order?
- 10.19 Prove that for any lambda expression f , if the normal-order evaluation of Yf terminates, where Y is the fixed-point combinator $\lambda h.(\lambda x.h(x\ x))$ ($\lambda x.h(x\ x)$), then $f(Yf)$ and Yf will reduce to the same simplest form.
- 10.20 Given the definition of structures (lists) on page 244, what happens if we apply `car` or `cdr` to `nil`? How might you introduce the notion of “type error” into lambda calculus?
- 10.21 Let

$$\text{zero} \equiv \lambda x.x$$

$$\text{succ} \equiv \lambda n.(\lambda s.(s\ \text{select_second})\ n)$$

where $\text{select_second} \equiv \lambda x.\lambda y.y$. Now let

$$\text{one} \equiv \text{succ zero}$$

$$\text{two} \equiv \text{succ one}$$

Show that

$$\text{one select_second} = \text{zero}$$

$$\text{two select_second select_second} = \text{zero}$$

In general, show that

$$\text{succ}^n\ \text{zero select_second}^n = \text{zero}$$

Use this result to define a predecessor function `pred`. You may ignore the issue of the predecessor of zero.

Note that our definitions of T and F allow us to check whether a number is equal to zero:

$$\text{iszero} \equiv \lambda n. (n \text{ select_first})$$

Using `succ`, `pred`, `iszero`, and `if`, show how to define `plus` and `times` recursively. These definitions could of course be made nonrecursive by means of beta abstraction and **Y**.