λ-Calculus: Exercises 1

1 Exercise 1

All of these functions are computed iteratively.

1.1 Exercise 1(a)

My first effort:

$$\text{predStep} = \lambda p. < \text{if } (\text{snd } p) \text{ then } \overline{0} \text{ else } (\text{succ } (\text{fst } p)), \text{false} > \quad (1)$$

$$\text{pred} = \lambda n. \text{fst } (n \text{ predStep } < \overline{0}, \text{true} >) \quad (2)$$

A better definition, due to Kleene:

$$\text{predStep} = \lambda p. < \text{snd } p, \text{succ } (\text{snd } p) > \quad (3)$$

$$\text{pred} = \lambda n. \text{fst } (n \text{ predStep } < \overline{0}, \overline{0} >) \quad (4)$$

1.2 Exercise 1(b)

$$\text{minus} = \lambda n. \lambda m. m \text{ pred } n \quad (5)$$

1.3 Exercise 1(c)

$$\text{factStep} = \lambda p. < \text{succ } (\text{fst } p), \text{mult } (\text{succ } (\text{fst } p)) \ (\text{snd } p) > \quad (6)$$

$$\text{fact} = \lambda n. \text{snd } (n \text{ factStep } < \overline{0}, \overline{1} >) \quad (7)$$

2 Exercise 2

The functions in exercises 2(a) and 2(b) are computed recursively. This poses a problem in that it requires a function to invoke itself, and there is no way to directly embed a function definition within itself if the definition is to be of finite length. Instead, a function f can be rewritten in terms of a function f' such that $(f'\ f')$ evaluates to the desired f.

2.1 Exercise 2(a)

$$\text{length}' = \lambda f. \lambda l. \text{if } (\text{isNil } l) \text{ then } \overline{0} \text{ else } (\text{succ } (f\ f\ (\text{tail } l))) \quad (8)$$

$$\text{length} = (\lambda f. f\ f)\ \text{length}' \quad (9)$$

2.2 Exercise 2(b)

$$\text{sumList}' = \lambda f.\lambda l.\,\text{if}\,(\text{isNil}\ l)\ \text{then}\ \overline{0}\ \text{else}\,(\text{plus}\,(\text{head}\ l)\,(f\ f\,(\text{tail}\ l))) \tag{10}$$

$$\text{sumList} = (\lambda f.f\ f)\ \text{sumList}' \tag{11}$$

2.3 Exercise 2(c)

As an example, start by unpacking a particular Church numeral:

$$\overline{4} = \lambda f.\lambda x.f\,(f\,(f\,(f\,x))) \tag{12}$$

(12) serves as a model in creating an iterated list of length 4. Replace each bound f in (12) with a free f_n such that the order of their application is $f_0, f_1, f_2 \ldots$

$$\lambda f.\lambda x.f_3\,(f_2\,(f_1\,(f_0\,x))) \tag{13}$$

Each f_n takes a single Church numeral as an argument and returns a Church numeral as a result. If the bound f in (13) is a two-place function that takes some a as its first argument, a Church numeral as a second argument, and returns a Church numeral as its result; then each f_n may be rewritten as $(f\,a_n)$ so that (13) becomes

$$\lambda f.\lambda x.(f\ a_3)\,((f\ a_2)\,((f\ a_1)\,((f\ a_0)\,x))) \tag{14}$$

The left associativity of application allows the removal of the parentheses enclosing the application of f to each a_n:

$$\lambda f.\lambda x.f\ a_3\,(f\ a_2\,(f\ a_1\,(f\ a_0\,x))) \tag{15}$$

As with Church numerals, this pattern generalizes to lists of any length.