Goal

Getting used to different reduction strategies, to the notions normal form and weak head normal form (WHNF), strong and weak normalization.

Material

Chapter 3 of the course notes λ-calculus, and the slides of lecture 3. Also the text part of the description of the first set of Haskell exercies.

Exercises

1. From exercise class 2: Consider the term $M = (\lambda x.\,((\lambda u.\,u)\,x)\,(\lambda z.\,x\,\Omega))\,\lambda y.\,z$.

   (a) Depict the term tree of M. (Unfold the definition of Ω.)

   (b) Give two different α-equivalent renderings of M.

   (c) Reduce M using the leftmost-innermost (call-by-value) strategy.

   (d) Reduce M using the leftmost-outermost (call-by-need) strategy.

2. We consider the following terms, with $\mathsf{Y} = \lambda f.\,(\lambda x.\,f\,(x\,x))\,(\lambda x.\,f\,(x\,x))$ the fixed point combinator that we will also encounter later.

   $$M_1 = (\lambda f.\,f\,\mathsf{3})\,(\lambda x.\,(\lambda y.\,y)\,x)$$
   $$M_2 = \omega_3\omega_3 \text{ with } \quad \omega_3 = \lambda x.\,x\,x\,x$$
   $$M_3 = \mathsf{Y}\,u \text{ with } u \text{ a variable}$$
   $$M_4 = (\lambda x.\,(\lambda y z.\,z\,y)\,x)\,u\,(\lambda x.\,x)$$

   Answer for these terms the following questions:

   (a) Give a reduction sequence using the leftmost-innermost reduction stategy.

   (b) Give a reduction sequence using the leftmost-outermost reduction stategy.

   (c) Does the term have a normal form? If not, why not? If yes, what is its normal form?

   (d) Does the term have a WHNF? If no, why not? If yes, what is the WHNF?

3. Reduce in the term $(\lambda x.\,\lambda y.\,x\,(\mathsf{I}\,\mathsf{I}))\,(\lambda z.\,v\,z)\,\Omega$ repeatedly the lazy redex until the WHNF is reached.

4. Reduce in the term $(\lambda x.\, f\,(x\,\Omega\,\mathsf{I}))\,(\lambda u.\,\lambda v.\, v\,w)$ repeatedly the lazy redex until a WHNF is reached.

5. Reduce the following term in a minimum number of steps to normal form: $(\lambda x.\,(\lambda y.\, z\,y\,y)\,(\mathsf{I}\,\mathsf{I}))\,(\mathsf{I}\,\mathsf{I})$.

6. Consider the following Haskell function:

```
foldr f b []    = b
foldr f b (h:t) = f h (foldr f b t)
```

   Give the first four steps of the evaluation of `foldr (:) [] [1,2,3,4]`. Use informal equational reasoning ('replace equals by equals').

7. Consider the function `foldr`:

```
foldr f b []    = b
foldr f b (h:t) = f h (foldr f b t)
```

   Use it to give a definition of the function `myconcat` that takes as input a list of lists, and that gives back as output the concatenation of those lists. Example:

```
*Main> myconcat [[1,2,3] , [4,5,6]]
[1,2,3,4,5,6]
```

   Give the first four steps of the evaluation of `myconcat [[1,2,3] , [4,5,6]]` using informal equational reasoning.