Homework 6: λ-calculus *Due: December 5, 2018, 9:00 AM*

5.8.5, exercise 2.

book problem 1. Consider the following term

(λx.λy.x) (λz.z) ((λx.x)(λy.y))

(a) For this term, show both the left-to-right and right-to-left call-by-value reduction sequences which end in values, using underlining notation.

(b) Write down the contexts used for each step in those reduction sequences, and confirm that all contexts are accepted by the appropriate grammer in the chapter.

solution 1. left-to-right
$\underline{(\lambda \text{x}.\lambda \text{y}.\text{x})\ (\lambda \text{z}.\text{z})}\ ((\lambda \text{x}.\text{x})(\lambda \text{y}.\text{y}))$
$\rightsquigarrow (\lambda \text{y}.\lambda \text{z}.\text{z})\ \underline{((\lambda \text{x}.\text{x})(\lambda \text{y}.\text{y}))}$
$\rightsquigarrow \underline{(\lambda \text{y}.\lambda \text{z}.\text{z})(\lambda \text{y}.\text{y})}$
$\rightsquigarrow \lambda \text{z}.\text{z}$

right-to-left
$(\lambda \text{x}.\lambda \text{y}.\text{x})\ (\lambda \text{z}.\text{z})\ \underline{((\lambda \text{x}.\text{x})(\lambda \text{y}.\text{y}))}$
$\rightsquigarrow (\lambda \text{x}.\lambda \text{y}.\text{x})\ \underline{(\lambda \text{z}.\text{z})(\lambda \text{y}.\text{y})}$
$\rightsquigarrow \underline{(\lambda \text{x}.\lambda \text{y}.\text{x})(\lambda \text{y}.\text{y})}$
$\rightsquigarrow [\text{x}/\text{z}]\lambda \text{z}.\text{z}$

5.9, exercise 2.

book problem 2. for the purposes of this problem define t↓t' to means $\exists t''.t \rightsquigarrow *t'' \wedge t' \rightsquigarrow *t''$ where $\rightsquigarrow$ is normal-order reduction (where the outersmost β-redex is reduced, and reduction proceeds under λ-binders). As usual, variables can be safely renamed, so that we consider $\lambda x.x$ equivalent to $\lambda y.y$, for example.

For which of the following terms do we have t↓$\lambda x.\lambda y.y$? Please indicated all the terms which satisfy this property.

(a) $\lambda x.\lambda y.y$
(b) $(\lambda x.x)\lambda y.y$
(c) $(\lambda x.xx)(\lambda y.y)\lambda x.\lambda y.y$
(d) $\lambda x.(\lambda x.\lambda y.yy)(\lambda x.x)$
(e) $\lambda x.\lambda x.(\lambda y.y)x$

solution 2. .
(a) $\lambda x.\lambda y.y$
sol. If we safely rename x to z we have $\lambda z.\lambda y.y$ and therefore we have t↓t'
(b) $(\lambda x.x)\lambda y.y$
sol. This does not reduce to the form $\lambda x.\lambda y.y$ and therefore does not have the property t↓t'
(c) $(\lambda x.xx)(\lambda y.y)\lambda x.\lambda y.y$
sol. reducing to normal form we have.
$\underline{(\lambda x.xx)(\lambda y.y)\lambda x.\lambda y.y}$
$\rightsquigarrow \underline{((\lambda y.y)\lambda x.\lambda y.y)}((\lambda y.y)\lambda x.\lambda y.y)$
$\rightsquigarrow \underline{(\lambda x.\lambda y.y)((\lambda y.y)\lambda x.\lambda y.y)}$
$\rightsquigarrow \lambda x.\lambda y.y$
again if we safely rename x to z we have $\lambda z.\lambda y.y$ and therefore we have t↓t' (d) $\lambda x.(\lambda x.\lambda y.yy)(\lambda x.x)$
sol. reducing to normal form we have.
$\lambda x.\underline{(\lambda x.\lambda y.yy)(\lambda x.x)}$
$\rightsquigarrow \lambda x.(\lambda y.yy)$

We cannot reduce this any further and hence this does not reduce to the form $\lambda x.\lambda y.y$ and therefore does not have the property t↓t'
(e) $\lambda x.\lambda x.(\lambda y.y)x$
sol. This is already in normal for and hence This does not reduce to the form $\lambda x.\lambda y.y$ and therefore does not have the property t↓t'

6.8.1, exercises 2–3.

book problem 3. (2) Write a function $add - components$ that takes in a pair (x, y) and returns the pair $x + y$

(3) Write a function $swap - pair$ that takes in a pair (x, y) and returns the pair (y, x)

solution 3. (3) $\lambda fxy.fyx$

6.8.2, exercises 2–3.

book problem 4. .
(2)Suppose we wish to encode a data type consisting of basic colors *red* and *blue*, with possible repeated modifier *light*. So example data elements are: *red*, *blue*, *lightblue*, *light*(*lightred*), etc.
give definitionsfor the three constructors, *red*, *blue*, and *light*, using the Scott encoding.

(3) Give definitions using the scott encoding for constructors *node* and *leaf* for a datatype of binary trees, with data stored at the nodes but not the leaves. So a tree like this, (insert picture of three with values only in internal nodes) $(node 1 leaf (node 2 leaf leaf))$

solution 4. .
(2)
red = $\lambda r.\lambda b.\lambda l.r$
blue = $\lambda r.\lambda b.\lambda l.b$
light = $\lambda c.\lambda r.\lambda b.\lambda l.lc$
(3)
node = $\lambda v.\lambda l.fvll$
leaf = $\lambda l.l$

6.9.2, exercise 1.

book problem 5. One way to write a function *eqnaut* to test whether two Peano numbers are equal is to remove a successor from each of them, until either both numbers are zero, in which case we return *true*; or else one is zero, and the other is not, in which we return *false*. (a) Based on this idea, define *eqnuat* using recursive equations. (b) Translate your encoding into a lamda term using the Scott encoding.

Problem 1. Given the λ-encoding of lists by:

$$nil = \lambda f.\lambda z.z \qquad cons = \lambda x.\lambda xs.\lambda f.\lambda z.f\ x\ (xs\ f\ z)$$

Show encodings of the *head* and *tail* functions satisfying the following equations

$$head\ (cons\ x\ xs) = x \qquad tail\ (cons\ x\ xs) = xs$$

It may help you to recall the approach used to encode the predecessor function.

Problem 2. Given the λ-encoding of the natural numbers by:

$$zero = \lambda f.\lambda x.x \qquad succ = \lambda n.\lambda f.\lambda x.f\ (n\ f\ x)$$

We have the following two definitions of addition:

$$add = \lambda m.\lambda n.\lambda f.\lambda x.m\ succ\ n\ f\ x \qquad add' = \lambda m.\lambda n.\lambda f.\lambda x.m\ f\ (n\ f\ x)$$

Show (by natural number induction) that, for all encodings of natural numbers m and n, we have $add\ m\ n = add'\ m\ n$.

solution 5. .
I will begin by replacing succ function with it's definition and reducing to normal form.

$$add = \lambda m.\lambda n.\lambda f.\lambda x.m\underline{(\lambda n.\lambda f.\lambda x.f\,(n\,f\,x))n\,f\,x} \rightsquigarrow \lambda m.\lambda n.\lambda f.\lambda x.m\,f\,(n\,f\,x)$$

now I consider the base case for induction. add 0 n and add' 0 n we have

add 0 n =
$\underline{\lambda m.\lambda n.\lambda f.\lambda x.m\,f\,(n\,f\,x)(\lambda f.\lambda x.x)n}$
$\rightsquigarrow \underline{\lambda n.\lambda f.\lambda x.nf(nfx)(\lambda f.\lambda x.x)}$
$\rightsquigarrow \underline{\lambda f.\lambda x.(\lambda f.\lambda x.x)fnfx}$
$\rightsquigarrow \underline{\lambda x.(\lambda f.\lambda x.x)xnx}$
$\rightsquigarrow \underline{(\lambda f.\lambda x.x)xn}$
$\rightsquigarrow n$

add' 0 n =
$\underline{\lambda m.\lambda n.\lambda f.\lambda x.m\,f\,(n\,f\,x)(\lambda f.\lambda x.x)n}$
$\rightsquigarrow \underline{\lambda n.\lambda f.\lambda x.nf(nfx)(\lambda f.\lambda x.x)}$
$\rightsquigarrow \underline{\lambda f.\lambda x.(\lambda f.\lambda x.x)fnfx}$
$\rightsquigarrow \underline{\lambda x.(\lambda f.\lambda x.x)xnx}$
$\rightsquigarrow \underline{(\lambda f.\lambda x.x)xn}$
$\rightsquigarrow n$
Now I will assume add m n = add' m n for m
And now I will show add Sm n = add' Sm n
add Sm n =
$\underline{\lambda m.\lambda n.\lambda f.\lambda x.m\,f\,(n\,f\,x)(\lambda s.\lambda zs(msz))n}$
$\rightsquigarrow \underline{\lambda n.\lambda f.\lambda x.nf(nfx)(\lambda s.\lambda z.s(msz))}$
$\rightsquigarrow \underline{\lambda f.\lambda x.(\lambda s.\lambda z.s(msz))fnfx}$
$\rightsquigarrow \underline{\lambda x.(\lambda s.\lambda z.s(msz))xnx}$
$\rightsquigarrow \underline{(\lambda s.\lambda z.s(msz)xn)}$
$\rightsquigarrow x(mxn)$

add' Sm n =
$\underline{\lambda m.\lambda n.\lambda f.\lambda x.m\,f\,(n\,f\,x)(\lambda s.\lambda zs(msz))n}$
$\rightsquigarrow \underline{\lambda n.\lambda f.\lambda x.nf(nfx)(\lambda s.\lambda z.s(msz))}$
$\rightsquigarrow \underline{\lambda f.\lambda x.(\lambda s.\lambda z.s(msz))fnfx}$
$\rightsquigarrow \underline{\lambda x.(\lambda s.\lambda z.s(msz))xnx}$
$\rightsquigarrow \underline{(\lambda s.\lambda z.s(msz)xn)}$
$\rightsquigarrow x(mxn)$

and hence we can say add m n = add' m n

Problem 3. ($\star$) A *snoc-list* is a cons-list in reverse: the sequence $1, 2, 3$ is represented

$$cons\,1\,(cons\,2\,(cons\,3\,nil))$$

as a cons-list, and

$$snoc\,(snoc\,(snoc\,lin\,1)\,2)\,3$$

as a snoc-list. Give a λ-encoding of snoc-lists; that is, give λ definitions for *lin*, *snoc*, and *foldl* such that

$$foldl\,f\,z\,(snoc\,(\ldots(snoc\,lin\,x_1)\ldots)\,x_n) = f\,(\ldots(f\,z\,x_1)\ldots)\,x_n$$