Exercises for Computability and Complexity, Spring 2017, Sheet 9 – Solutions

Please return your solutions in the Thursday lecture on April 20

For this exercise sheet please prepare your solutions with a text processor and submit a printout!

Exercise 1 (easy) Design a λ-expression LISTSUM, which applied to a list whose entries are Church numerals returns the sum of the list elements, and returns $\underline{0}$ if the list is empty.

Solution. The intuitive recursion equation is

LISTSUM l = **if** (**null** l) $\underline{0}$ (**add** (**head** l) (LISTSUM (**tail** l))).

Using the fixed point operator, this becomes

LISTSUM ≡ Y (λ $gl.$ (**if** (**null** l) $\underline{0}$ (**add** (**head** l) (g (**tail** l)))))

Problem 2 (medium) Design a λ-expression **sortincreasing**, which applied to a list whose entries are Church numerals returns a list of the same length with the same entries, but sorted in ascending order. You may assume that you already have combinators $<$, $>$, $\leq$, $\geq$, which when applied to two Church numerals reduce to **true** or **false** in the obvious fasion. Also you may use all the combinators for Boolean logic, list processing and arithmetics introduced in the lecture notes. Example of what your λ-expression should do: **sortincreasing** (1::3::2::1::**nil**) →* (1::1::2::3::**nil**). I suggest to lean on bubblesort in your construction. You will find it necessary (or at least, helpful) to lean on a modular programming style, where you first define lambda expressions for useful subroutines, which you then can use in your main function.

Solution. There are many ways to do this. One solution that leans on bubblesort goes as follows. We first write down "naïve self-referential" pseudo-definitions of a predicate **ordered** that checks whether a list is already ordered:

```
ordered l = if (or (null l)(null (tail l)))
                true
                (and (≤ (head l)(head (tail l))) (ordered l))
```

We then use this **ordered** combinator in the design of **sortincreasing**:

```
sortincreasing l = if (ordered l)
                      l
                      if (and (> (head l) (head (tail l)))
                              (ordered (cons (head l) (tail (tail l)) )))
                         cons (head (tail l)) (cons (head l) (tail (tail l)))
                         if (> (head l) (head (tail l))
                            sortincreasing (cons  (head (tail l)) (cons (head l) (tail (tail l))))
                            sortincreasing (cons (head l) (sortincreasing (tail l)))
```

The naïve pseudo-definition of **ordered** is turned into a valid λ-expression:

ordered ≡ Y (λ *gl.* **if** (**or** (**null** *l*)(**null** (**tail** *l*)))
true
(**and** (≤ (**head** *l*)(**head** (**tail** l))) (*g l*)))

Likewise for **sortincreasing**:

sortincreasing ≡
Y (λ *gl.* **if** (**ordered** *l*)
l
if (**and** (> (**head** *l*) (**head** (**tail** *l*)))
(**ordered** (**cons** (**head** *l*) (**tail** (**tail** *l*)))))
cons (**head** (**tail** *l*)) (**cons** (**head** *l*) (**tail** (**tail** *l*)))
if (> (**head** *l*) (**head** (**tail** *l*))
g (**cons** (**head** (**tail** *l*)) (**cons** (**head** *l*) (**tail** (**tail** *l*))))
g (**cons** (**head** *l*) (*g* (**tail** *l*))))

What remains to be done is to insert the valid λ-expression obtained for **ordered** verbatim into the valid λ-expression obtained for **sortincreasing** (where variables can but need not be renamed for better readability):

sortincreasing ≡
Y (λ *gl.* **if** (*Y* (λ *hk.* **if** (**or** (**null** *k*)(**null** (**tail** *k*)))
true
(**and** (≤ (**head** *k*)(**head** (**tail** *k*))) (*h k*))) *l*)
l
if (**and** (> (**head** *l*) (**head** (**tail** *l*)))
(**ordered** (**cons** (**head** *l*) (**tail** (**tail** *l*)))))
cons (**head** (**tail** *l*)) (**cons** (**head** *l*) (**tail** (**tail** *l*)))
if (> (**head** *l*) (**head** (**tail** *l*))
g (**cons** (**head** (**tail** *l*)) (**cons** (**head** *l*) (**tail** (**tail** *l*))))
g (**cons** (**head** *l*) (*g* (**tail** *l*))))