HW 3 – Lambda Calculus and Floyd-Hoare Logic

CS 421 – Fall 2011
Revision 1.0

Assigned Tuesday,November 29, 2011
Due Tuesday December 6, 2011, 2:00pm - in class
Extension 48 hours (20% penalty)

1 Change Log

1.0 Initial Release.

2 Turn-In Procedure

Your answers to the following questions are to be hand-written neatly or printed on one or more sheets of paper, each with your name in the upper right corner. The homework is to be turned in in class at the start of class. Alternately, you may hand it to Prof. Elsa Gunter in person before the deadline.

3 Objectives and Background

The purpose of this HW is to test your understanding of:

- Alpha and beta conversion in the lambda calculus
- The consequences of different evaluation schemes
- How to represent datatypes in the lambda calculus
- How to do proofs in Floyd-Hoare Logic

Another purpose of HW3 is to provide you with experience answering non-programming written questions of the kind you may experience on the final.

4 Problems

1. (15 pts) Prove that $\lambda x.(\lambda y.xy)x$ is α-equivalent $\lambda y.(\lambda x.yx)y$. You should label every use of α-conversion and congruence.

2. (15 pts) Given the following term:

   $$(\lambda f.f(\lambda x.x))((\lambda f.\lambda x.f(fx))(\lambda f.\lambda x.fx))$$

   reduce this term as much as possible using each of

   a. eager evaluation
   b. lazy evaluation
   c. unrestricted $\alpha\beta$-reduction (*i.e.* by $\alpha\beta$ conversion that can be applied anywhere)

Label each step of reduction with the rule justifying it. You do not need to label uses of congruence, or break them out as separate steps, in this problem.

3. (12 pts) Give a proof in Floyd-Hoare Logic of the following partial correctness assertion:

$$\{x \geq a\} \texttt{ if x > y then x:= x + x - y else x := y + 1} \{x > a\}$$

You may either give a proof tree, or write out your proof in English, but you must cite each Floyd-Hoare Logic Rule used.

4. (Extra Credit) (10 pts) Using the methodology discussed in class, give the lambda terms that encode the constructors, and the functions `fold` and `reflect` for the OCaml datatype `'a tree` given as follows:

```
type 'a tree = Leaf of 'a | Node of ('a tree) * ('a tree)
let rec fold l n tree =
    match tree
    with Leaf x -> l x
       | Node(g, d) -> n (fold l n g) (fold l n d)

let reflect tree = fold (fun x -> x) (fun g -> fun d -> Node(d, g))
```