Mid-term Exam

Foundations of Software

November 16, 2007

Last Name : _____

First Name : _____

Section : _____

Exercise	**Points**	**Achieved Points**
1	10	
2	12	
3	8	
Total	30	

Exercise 1 : Equivalence of lambda terms (10 points)

For each of the following pairs of lambda terms, say if they are behaviorally equivalent or not. If they are not equivalent, write down the arguments for which *only one* of the terms does not terminate. You do not need to prove it when they are equivalent. We assume the *call-by-value* pure lambda calculus.

$\lambda t.\lambda f.\ t$
$\lambda t.\lambda f.\ f$

$\lambda t.\lambda f.\ t$
$\lambda x.\lambda y.\ (\lambda z.z)\ x$

$\lambda x.\lambda y.\ x\ y$
$\lambda x.x$

$\lambda x.\lambda y.\ x\ y$
$\lambda x.\lambda y.\ x\ (\lambda z.z)\ y$

Exercise 2 : Linear terms (12 points)

A lambda-term t is said to be *linear* if, for every sub-term t of the form $\lambda x.s$ the bound variable x appears exactly once in the body s. For example: $\lambda x.x$ and $\lambda x.\lambda y.x\ y$ are linear, while $\lambda x.x\ x$ and $\lambda x.\lambda y.y$ are not.

Show that all linear terms have a normal form.

Hint:
Think about what happens to the number of lambda abstractions after one reduction step.

Exercise 3 : References (8 points)

Consider the simply-typed lambda calculus with references. Design an extension that acts like *free* in C or Pascal, that is, given a term that evaluates to a label in the store, it removes it from the store. There is no need to ensure in the type system that labels that have been removed are not dereferenced.
The new grammar should look like:

$$\begin{array}{rcll} t & ::= & \ldots & \\ & | & free\ t & \text{deallocation} \end{array}$$

Write down the typing and evaluation rules for the new case. Does *preservation* still hold? What about *progress*? If either of them doesn't hold, give an example program for which it fails (you can use *let*, sequencing and natural numbers in your examples).