Mid-term Exam – solution

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.

$\lambda t.\lambda f.\ t$
$\lambda t.\lambda f.\ f$
Not equivalent:

```
(λt.λf.t) λx.omega λx.x λx.x
  → (λf.λx.omega) λx.x λx.x
  → λx.omega λx.x
  → omega → diverges
```

and

```
(λt.λf.f) λx.omega λx.x λx.x
  → (λf.f) λx.x λx.x
  → λx.x λx.x
  → λx.x
```

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

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

$\lambda x.\lambda y.\ x\ y$
$\lambda x.\lambda y.\ x\ (\lambda z.z)\ y$
Not equivalent:

```
(λx.λy.x y) (λt.λf.t) λx.omega λx.x λx.x
  → (λy.(λt.λf.t) y) λx.omega λx.x λx.x
  → (λt.λf.t) λx.omega λx.x λx.x
  → (λf.λx.omega) λx.x λx.x
  → λx.omega λx.x
  → omega → diverges
```

and

```
(λx.λy.x (λz.z) y) (λt.λf.t) λx.omega λx.x λx.x
  → (λy.(λt.λf.t) (λz.z) y) λx.omega λx.x λx.x
  → (λt.λf.t) (λz.z) λx.omega λx.x λx.x
  → (λf.λz.z) λx.omega λx.x λx.x
  → λz.z λx.x λx.x
  → λx.x λx.x
  → λx.x
```

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.

Proof. We define the size of terms to be the number of lambda abstractions a term has. More precisely, it is:

$$size(t) = \begin{cases} 0 & \text{if } t \text{ is a variable} \\ 1 + size(t_1) & \text{if } t = \lambda x.t_1 \\ size(t_1) + size(t_2) & \text{if } t = t_1\ t_2 \end{cases}$$

We show by induction on evaluation derivations that if $t \to t'$ then $size(t') < size(t)$.

- E-App1 We have: $t = t_1\ t_2$, $t' = t_1'\ t_2$ and $t_1 \to t_1'$ Then:

$$\begin{array}{rcl} size(t) & = & size(t_1) + size(t_2) \\ size(t') & = & size(t_1') + size(t_2) \end{array}$$

  by using the Induction Hypothesis, we have $size(t_1') < size(t_1)$. By using this in the above equation, we get

$$\begin{array}{rcl} size(t') & < & size(t_1) + size(t_2) \\ size(t') & < & size(t) \end{array}$$

- E-App2 idem

- E-AppAbs We have: $t = (\lambda x.t_1)\ t2$, $t' = [x \mapsto t_2]t_1$

$$\begin{array}{rcl} size(t) & = & 1 + size(t_1) + size(t_2) \\ size(t') & = & size([x \mapsto t_2]t_1) \end{array}$$

  But we know that x appears exactly once in t_1, therefore $size([x \mapsto t_2]t_1 = size(t_1) + size(t_2)$. It follows that $size(t') < size(t)$.

We can use *size* to map any infinite sequence of reduction steps to an infinite decreasing sequence of natural numbers. Since this is impossible, we conclude there are no infinite sequences of reductions, for any linear lambda term. □

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:

$$\begin{array}{lll} t & ::= & \dots \\ & | & \mathit{free}\ t \quad \text{deallocation} \end{array}$$

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).

Answer:

$$\text{T-Free}\ \frac{\Gamma, \Sigma \vdash t :\ \mathit{Ref}\ T}{\Gamma, \Sigma \vdash \mathit{free}\ t :\ \mathit{unit}}$$

$$\text{E-Cong}\ \frac{t|\mu \to t'|\mu'}{\mathit{free}\ t|\mu \to \mathit{free}\ t'|\mu'} \qquad \text{E-Red}\ \frac{}{\mathit{free}\ l|\mu \to \mathit{unit}|\mu \setminus \{l\}}$$

Preservation still holds: the new rules can be easily added to the existing proof.
Progress does not hold anymore. Consider the following program:

```
let x = ref 0 in
  (\a: Ref Nat.\b: Ref Nat. free a; !b) x x
```

The code is well typed, but the evaluation gets stuck trying to dereference b, after it has been freed.