CIS 705 — Programming Languages — Spring 2009

Final Examination

Tuesday, May 12, 9:40 a.m.–11:30 a.m.

Question 1 (45 points)

This question is about programming in OCaml.

For the purposes of this question:

- A list *xs* is a *prefix* of a list *ys* iff there is a list *us* such that $xs \mathbin{@} us = ys$.
- A list *xs* is a *suffix* of a list *ys* iff there is a list *us* such that $us \mathbin{@} xs = ys$.
- A list *xs* is a *sublist* of a list *ys* iff there are lists *us* and *vs* such that $us \mathbin{@} xs \mathbin{@} vs = ys$.
- A list *xs* is *strictly ascending* iff *xs* has no sublist of length two whose first element is greater-than-or-equal-to its second element. E.g., `[]`, `[1]` and `[1;2;5;10]` are strictly ascending, but `[1;2;2;4]` and `[1;-1;2]` are not strictly ascending.

Write an OCaml program defining a function

```
val longest : 'a list -> int
```

such that, for all lists *xs*, `longest` *xs* returns the largest n such that there is a strictly ascending sublist of *xs* with length n.

Your program should be well-structured and well-documented. You should write abstract input/output specifications for each of its functions. Your program should be reasonably efficient, but you *don't* need to make your functions tail-recursive.

Question 2 (55 Points)

This question is about a reformulation of the simply typed lambda calculus that is similar—but different in important ways—to the one used in Assignment 5.

Definitions

If f is a function and x, y are elements of our universe, we define the function $f[x \mapsto y]$ from $\mathsf{dom}(f) \cup \{x\}$ to $\mathsf{ran}(f) \cup \{y\}$ by, for all $z \in \mathsf{dom}(f) \cup \{x\}$,

$$f[x \mapsto y](z) = \begin{cases} y, & \text{if } z = x, \\ f(z), & \text{if } z \neq x. \end{cases}$$

Our *types* are defined by:

$T ::=$	types:
Unit	unit type
$T \to T$	type of functions

Our *terms* are defined by:

$t ::=$	terms:
unit	unit constant
x	variables
$\lambda x.\, t$	abstraction
$t\, t$	application

And our *values* are defined by:

$v ::=$	values:
unit	unit constant
$\lambda x.\, t$	abstraction value

The *free variables* of a term t ($\mathsf{FV}(t)$) is defined recursively as follows:

$$\begin{aligned} \mathsf{FV}(\mathsf{unit}) &= \emptyset, \\ \mathsf{FV}(x) &= \{x\}, \\ \mathsf{FV}(\lambda x.\, t) &= \mathsf{FV}(t) \setminus \{x\}, \\ \mathsf{FV}(t_1\, t_2) &= \mathsf{FV}(t_1) \cup \mathsf{FV}(t_2). \end{aligned}$$

A term is *closed* iff it has no free variables; otherwise it is *open*.

The *substitution of* a *closed* term s *for the free occurrences of* a variable x *in* a term t ($[x \mapsto s]t$) is defined recursively by:

$$[x \mapsto s]\mathsf{unit} = \mathsf{unit},$$

$$[x \mapsto s]y = \begin{cases} s, & \text{if } y = x, \\ y, & \text{if } y \neq x, \end{cases}$$

$$[x \mapsto s](\lambda y.\, t) = \begin{cases} \lambda y.\, t, & \text{if } y = x, \\ \lambda y.\, [x \mapsto s]t, & \text{if } y \neq x, \end{cases}$$

$$[x \mapsto s](t_1\, t_2) = [x \mapsto s]t_1\, [x \mapsto s]t_2.$$

The *evaluation relation* $\boxed{t \to t'}$ between *closed* terms is defined inductively by:

$$\frac{t_1 \to t_1'}{t_1\, t_2 \to t_1'\, t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \to t_2'}{t_1\, t_2 \to t_1\, t_2'} \qquad \text{(E-App2)}$$

$$(\lambda x.\, t)t' \to [x \mapsto t']t \qquad \text{(E-AppAbs)}$$

So, in the above, t_1, t_1', t_2, t_2' and t' range over *closed* terms, whereas $\mathsf{FV}(t) \subseteq \{x\}$. A closed term t is a *normal form* iff there is no closed term t' such that $t \to t'$. A closed term t is *stuck* iff t is a normal form but t is not a value.

A *typing context* (or just *context*) Γ is a function such that $\mathsf{dom}(\Gamma)$ is a finite subset of the variables, and $\mathsf{ran}(\Gamma)$ is a subset of the types. The *typing relation* $\boxed{\Gamma \vdash t : T}$ between typing contexts, terms and types is defined inductively by:

$$\Gamma \vdash \mathsf{unit} : \mathsf{Unit} \qquad \text{(T-Unit)}$$

$$\frac{(x, T) \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-Var)}$$

$$\frac{\Gamma[x \mapsto T_1] \vdash t : T_2}{\Gamma \vdash \lambda x.\, t : T_1 \to T_2} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\, t_2 : T_2} \qquad \text{(T-App)}$$

We say that a closed term t is *well-typed* iff $\emptyset \vdash t : T$ for some type T.

An easy induction on the typing relation suffices to show that, for all contexts Γ, terms t and types T, if $\Gamma \vdash t : T$, then $\mathsf{FV}(t) \subseteq \mathsf{dom}(\Gamma)$. And, we have the **typing of substitutions lemma**: for all contexts Γ, variables y, terms t, t' and types T, T', if $\Gamma[y \mapsto T'] \vdash t : T$ and $\emptyset \vdash t' : T'$, then $\Gamma \vdash [y \mapsto t']t : T$.

Questions

Either *prove* or *disprove* each of the following statements:

Determinacy of Evaluation For all closed terms t, t' and t'', if $t \to t'$ and $t \to t''$, then $t' = t''$.

Uniqueness of Typing For all contexts Γ, terms t and types T and T', if $\Gamma \vdash t : T$ and $\Gamma \vdash t : T'$, then $T = T'$.

Progress For all closed terms t, if t is well-typed, then t is not stuck.

Preservation For all closed terms t and t' and types T, if $\emptyset \vdash t : T$ and $t \to t'$, then $\emptyset \vdash t' : T$.

Reverse Preservation For all closed terms t and t' and types T, if $\emptyset \vdash t' : T$ and $t \to t'$, then $\emptyset \vdash t : T$.