CIS 705 — Programming Languages — Spring 2010

Final Examination

Tuesday, May 11, 9:40 a.m.–11:30 a.m.

Question 1 (45 points)

This question is about programming in OCaml.

If *xs* is a list with no duplicates (i.e., a given value appears no more than once in *xs*), then a list *ys* is a *permutation* of *xs* iff *ys* can be formed from *xs* by reordering its elements. Thus, if *xs* is a list of length n with no duplicates, there are exactly $n!$ permutations of *xs*.

E.g., the $6 = 3!$ permutations of the list `[1; 3; 2]` are `[1; 2; 3]`, `[1; 3; 2]`, `[2; 1; 3]`, `[2; 3; 1]`, `[3; 1; 2]` and `[3; 2; 1]`.

Write an OCaml program defining a function

```
val perms : 'a list -> 'a list list
```

such that, for all lists *xs*, if *xs* contains no duplicates, then `perms` *xs* returns the list of all permutations of *xs*, listed in some order without duplicates.

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.

Question 2 (55 Points)

This question is about a simply typed lambda calculus that is similar to the ones we have studied, but is novel in one respect.

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 $z \in \mathsf{dom}(f) \cup \{x\}$,

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

In the following, variables x are as usual, and n ranges over the natural numbers. Our *types* are defined by:

$T ::=$	types:
Nat	type of natural numbers
$T \to T$	type of functions

As usual, $\to$ associates to the right. Our *terms* are defined by:

$t ::=$	terms:
n	natural number constant
$\mathsf{succ}\, t$	successor
$\mathsf{pred}\ t\ \mathsf{normal}\ t\ \mathsf{error}\ t$	predecessor with error recovery
x	variables
$\lambda x : T.\, t$	abstraction
$t\, t$	application

And our *values* are defined by:

$v ::=$	values:
n	natural number constant
$\lambda x : T.\, t$	abstraction

As usual, application associates to the left and abstractions extend as far as possible.

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

$$\begin{aligned}
\mathsf{FV}(n) &= \emptyset, \\
\mathsf{FV}(\mathsf{succ}\, t) &= \mathsf{FV}(t), \\
\mathsf{FV}(\mathsf{pred}\ t_1\ \mathsf{normal}\ t_2\ \mathsf{error}\ t_3) &= \mathsf{FV}(t_1) \cup \mathsf{FV}(t_2) \cup \mathsf{FV}(t_3), \\
\mathsf{FV}(x) &= \{x\}, \\
\mathsf{FV}(\lambda x : T.\, t) &= \mathsf{FV}(t) \setminus \{x\}, \\
\mathsf{FV}(t_1\, t_2) &= \mathsf{FV}(t_1) \cup \mathsf{FV}(t_2).
\end{aligned}$$

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:

$$\begin{aligned}
[x \mapsto s]n &= n, \\
[x \mapsto s](\mathsf{succ}\, t) &= \mathsf{succ}([x \mapsto s]t), \\
[x \mapsto s](\mathsf{pred}\ t_1\ \mathsf{normal}\ t_2\ \mathsf{error}\ t_3) &= \mathsf{pred}\ [x \mapsto s]t_1\ \mathsf{normal}\ [x \mapsto s]t_2\ \mathsf{error}\ [x \mapsto s]t_3, \\
[x \mapsto s]y &= \begin{cases} s, & \text{if } y = x, \\ y, & \text{if } y \neq x, \end{cases} \\
[x \mapsto s](\lambda y : T.\, t) &= \begin{cases} \lambda y : T.\, t, & \text{if } y = x, \\ \lambda y : T.\, [x \mapsto s]t, & \text{if } y \neq x, \end{cases} \\
[x \mapsto s](t_1\, t_2) &= [x \mapsto s]t_1\, [x \mapsto s]t_2.
\end{aligned}$$

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

$$\mathsf{succ}\, n \to n + 1 \qquad \text{(E-SuccNat)}$$

$$\frac{t \to t'}{\mathsf{succ}\, t \to \mathsf{succ}\, t'} \qquad \text{(E-Succ)}$$

$$\mathsf{pred}\ 0\ \mathsf{normal}\ t_2\ \mathsf{error}\ t_3 \to t_3 \qquad \text{(E-PredZero)}$$

$$\mathsf{pred}\ n + 1\ \mathsf{normal}\ t_2\ \mathsf{error}\ t_3 \to t_2\, n \qquad \text{(E-PredNonZero)}$$

$$\frac{t_1 \to t_1'}{\mathsf{pred}\ t_1\ \mathsf{normal}\ t_2\ \mathsf{error}\ t_3 \to \mathsf{pred}\ t_1'\ \mathsf{normal}\ t_2\ \mathsf{error}\ t_3} \qquad \text{(E-Pred)}$$

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

$$\frac{t_2 \to t_2'}{v_1\, t_2 \to v_1\, t_2'} \qquad \text{(E-App2)}$$

$$(\lambda x : T.\, u)v \to [x \mapsto v]u \qquad \text{(E-AppAbs)}$$

So, in the above, t, t', t_1, t'_1, t_2, t'_2 and t_3 range over *closed* terms, v and v_1 range over *closed* values, and u is a term such that $\mathsf{FV}(u) \subseteq \{x\}$.

A closed term t is a *normal form* iff there is no closed term t' such that $t \to t'$. Thus a closed term t is not a normal form iff there is a 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. Thus a closed term t is not stuck iff t is a value or t is not a normal form.

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:

$$\Gamma \vdash n : \mathsf{Nat} \qquad \text{(T-Nat)}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\, t : \mathsf{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{Nat} \qquad \Gamma \vdash t_2 : \mathsf{Nat} \to T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathsf{pred}\ t_1\ \mathsf{normal}\ t_2\ \mathsf{error}\ t_3 : T} \qquad \text{(T-Pred)}$$

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

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

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

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

The **Free Variables Lemma** says that, for all contexts Γ, terms t and types T, if $\Gamma \vdash t : T$, then $\mathsf{FV}(t) \subseteq \mathsf{dom}(\Gamma)$.

The **Typing of Substitutions Lemma** says that, for all contexts Γ, variables y, terms t and t', and types T and T', if $\Gamma[y \mapsto T'] \vdash t : T$ and $\emptyset \vdash t' : T'$, then $\Gamma \vdash [y \mapsto t']t : T$.

The **Canonical Forms Lemma** says that:

- for all values v, if $\emptyset \vdash v : \mathsf{Nat}$, then $v = n$, for some $n \in \mathbb{N}$;
- for all values v and types T_1 and T_2, if $\emptyset \vdash v : T_1 \to T_2$, then $v = \lambda x : T_1 . t$ for a variable x and term t such that $\mathsf{FV}(t) \subseteq \{x\}$.

Questions

When answering the following questions, you may use—without proof—the facts and lemmas stated above, as well as the inversion lemmas and principles of induction for the inductively defined sets and relations. When doing a proof by induction on a set or relation, you *must* write down the predicate P you are using, and then do your proof using P.

Question A (5 Points)

Let t be the closed term

$$\mathsf{pred}\ 1\ \mathsf{normal}\ (\lambda x : \mathsf{Nat}.\ \mathsf{pred}\ x\ \mathsf{normal}\ (\lambda x : \mathsf{Nat}.\ 2)\ \mathsf{error}\ 3)\ \mathsf{error}\ 4.$$

What is the type T such that $\emptyset \vdash t : T$? What is the closed value v such that $t \to^* v$?

Question B (25 Points)

Use induction on the typing relation to prove the **Progress Theorem**:

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

You may omit the cases (T-Var), (T-Abs) and (T-App), assuming your predicate $P(\Gamma, t, T)$ is the standard one.

Question C (25 Points)

Use induction on the evaluation relation to prove the **Preservation Theorem**:

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

You may omit the cases (E-App1), (E-App2) and (E-AppAbs), assuming your predicate $P(t, t')$ is the standard one.