CIS 500 — Software Foundations

Midterm II

November 13, 2002

Name: ________________________________

Student ID: ________________________________
(from your PennCard)

Email: ________________________________

<table>
<thead>
<tr>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

Total: ________________________________
Instructions

- This is a closed-book exam: you may not make use of any books or notes.
- You have 80 minutes to answer all of the questions. The entire exam is worth 80 points.
- Questions vary significantly in difficulty, and the point value of a given question is not always exactly proportional to its difficulty. Do not spend too much time on any one question.
- Partial credit will be given. All correct answers are short. The back side of each page may be used as a scratch pad.
- Good luck!
Simply typed lambda-calculus

The definition of the simply typed lambda-calculus with Unit is reproduced on page 15.

1. (8 points) For each of the following untyped \( \lambda \)-terms, either give a well-typed term of the simply typed lambda-calculus with Unit whose erasure is the given term, or else write "not typable" if no such term exists.

   The type annotations in your answers should only involve Unit and \( \rightarrow \).

   (a) \( \lambda x. x \ (x \ \text{unit}) \)

   (b) \( \lambda x. x \ \text{unit} \ x \)

   (c) \( \lambda x. x \ \text{unit} \ \text{unit} \)

   (d) \( \lambda x. \lambda y. \lambda z. (x \ y) \ (y \ z) \)
The definition of the simply typed lambda-calculus with references is reproduced on page 15.

2. (9 points) Suppose, for this question, that our language also has \texttt{let} expressions and numbers. Then evaluating the expression
   \begin{verbatim}
   let x = ref 5 in
   let y = x in
   let z = ref (\lambda a: Nat. y := a; succ (!y)) in
   (!z) (!y)
   \end{verbatim}
   beginning in an empty store yields:

   Result: 6
   Store: \begin{align*}
   l_1 & \rightarrow 5 \\
   l_2 & \rightarrow \lambda a: Nat. l_1 := a; \text{succ} (!l_1)
   \end{align*}

   Fill in the results and final stores (when started with an empty store) of the following terms:

   (a) \begin{verbatim}
   let x = ref 2 in
   let y = x in
   let f = \lambda a:Ref Nat. \lambda b:Ref Nat. a := 5; b := 6; !a in
   f x y
   \end{verbatim}

   Result: \begin{align*}
   \end{align*}
   Store: \begin{align*}
   \end{align*}

   (b) \begin{verbatim}
   let x = ref 2 in
   let y = ref x in
   let z = ref y in
   !z
   \end{verbatim}

   Result: \begin{align*}
   \end{align*}
   Store: \begin{align*}
   \end{align*}
let x = ref 0 in
let f = ref (\u:Unit. !x) in
x := 2;
let g = \u:Unit. (!f) unit in
x := 3;
f := \u:Unit. succ (!x) in
let r = g unit in
x := 9;
r
Result: 9
Store: store
3. (3 points) Is there any well-typed term that, when started with an empty store, will yield the following store?

\[ l_1 \rightarrow l_1 \]

If so, give one. If not, explain (briefly!) why not.
4. (8 points) We saw in homework 8 that, using references, we can achieve the effect of a recursive function definition by building a “cyclic store” in which the function’s body refers to its own definition indirectly, via a reference cell. The same idea extends straightforwardly to mutually recursive definitions.

Fill in the blanks in the following expressions so that, after evaluating them, even will be a function that checks whether its argument n is even (by returning true if it is 0 and otherwise checking whether (pred n) is odd).

\[
\begin{align*}
even_{ref} &= \text{ref (\(\lambda n:\text{Nat}.\text{true}\))}; \\
\text{odd}_{ref} &= \text{ref (\(\lambda n:\text{Nat}.\text{true}\))}; \\
\text{even}_{body} &= \lambda n:\text{Nat}. \text{if iszero n then true else ((\_\_\_\_) (pred n))}; \\
\text{odd}_{body} &= \lambda n:\text{Nat}. \text{if iszero n then false else ((\_\_\_\_) (pred n))}; \\
\text{even}_{ref} &= \_\_\_\_; \\
\text{odd}_{ref} &= \_\_\_\_; \\
\text{even} &= !\text{even}_{ref}; \\
\text{odd} &= !\text{odd}_{ref};
\end{align*}
\]
5. (20 points) In Chapter 13 of TAPL, the following lemmas were used in proving the preservation property for the simply typed lambda-calculus with references. (We’ve given all the lemmas names here, for easy reference.)

**Lemma [Inversion]:**

(a) If \( \Gamma \vdash x : T \), then \( x : T \in \Gamma \).
(b) If \( \Gamma \vdash \lambda x : T_1 . t_2 : T \), then \( T = T_1 \rightarrow T_2 \) for some \( T_2 \) with \( \Gamma, x : T_1 \vdash t_2 : T_2 \).
(c) If \( \Gamma \vdash t_1 \ t_2 : T \), then there is some type \( T_{11} \) such that \( \Gamma \vdash t_1 : T_{11} \rightarrow T \) and \( \Gamma \vdash t_2 : T_{11} \).
(d) If \( \Gamma \vdash \text{unit} : T \), then \( T = \text{Unit} \).
(e) If \( \Gamma \vdash \text{ref} \ t_1 : T \), then \( T = \text{Ref} T_1 \) and \( \Gamma \vdash t_1 : T_1 \).
(f) If \( \Gamma \vdash !t_1 : T \), then \( T = T_1 \) with \( \Gamma \vdash t_1 : \text{Ref} T_1 \).
(g) If \( \Gamma \vdash t_1 := t_2 : T \), then \( T = \text{Unit} \) and \( \Gamma \vdash t_1 : \text{Ref} T_{11} \) and \( \Gamma \vdash t_2 : T_{11} \).
(h) If \( \Gamma \vdash l : T \), then \( T = \text{Ref} \Sigma(l) \).

**Lemma [Substitution]:** If \( \Gamma, x : S \vdash t : T \) and \( \Gamma \vdash s : S \), then \( \Gamma \vdash [x \rightarrow s]t : T \).

**Lemma [Replacement]:** If \( \Gamma \vdash \mu : T \) and \( \Sigma' \supseteq \Sigma \), then \( \Gamma \vdash \mu : T \).

**Lemma [Weakening]:** If \( \Gamma \vdash t : T \) and \( \Sigma' \supseteq \Sigma \), then \( \Gamma \vdash t : T \).

For each case in the proof on the next page, write down the *skeleton* of the argument. A skeleton contains the same sequence of steps as the full argument, but omits all details. The rules for writing skeletons are as follows:

- Steps of the form “By part (x) of the inversion lemma, we obtain...” in the full argument become “inversion(x)” in the skeleton.
- Steps of the form “By the substitution lemma, we obtain...” become “substitution.” (Similarly for replacement and weakening.)
- Steps of the form “By the induction hypothesis, we obtain...” become “IH.”
- Steps of the form “By typing rule T-XXX, we obtain...” become “T-XXX.”
- If the full argument doesn’t use any of the lemmas or the induction hypothesis, then its skeleton is “Direct.”

For example, if the full argument is

**Case E­DerefLoc:** \( t = !l \quad t' = \mu(l) \quad \mu' = \mu \)

By part (f) of the inversion lemma, \( T = T_{11} \), and \( \Gamma \vdash l : \text{Ref} T_{11} \). By part (h) of the inversion lemma, \( T_{11} = \text{Ref} \Sigma(l) \), i.e., \( T = T_{11} = \Sigma(l) \). But now, from the assumption that \( \Gamma \vdash \mu \), we can conclude (by the definition of \( \Gamma \vdash \mu \)) that \( \Gamma \vdash \mu(l) : \Sigma(l) \).

the skeleton is written:

**Case E­DerefLoc:** \( t = !l \quad t' = \mu(l) \quad \mu' = \mu \)

Inversion(f), inversion(h)

As a second example, the case for E­Ref is also given below.
Theorem [Preservation]: If

\[ \Gamma \vdash \Sigma \vdash t : T \]
\[ \Gamma \vdash \mu \quad \text{(i.e., } \text{dom}(\mu) = \text{dom}(\Sigma) \text{ and } \Gamma \vdash \mu(l) : \Sigma(l) \text{ for every } l \in \text{dom}(\mu)) \]
\[ t \vdash \mu' \]
then, for some \( \Sigma' \supseteq \Sigma \),

\[ \Gamma \vdash \Sigma' \vdash t' : T \]
\[ \Gamma \vdash \Sigma' \vdash \mu' \]

Proof: By induction on evaluation derivations, with a case analysis on the final rule used.

Case E-App:
\[ t = t_1 \ t_2 \quad t_1 \vdash \mu' \quad t' = t_1' \ t_2 \]

Case E-App2:
Similar.

Case E-AppAbs:
\[ t = (\lambda x \colon T_{11} \cdot t_{12}) \ v_2 \quad t' = [x \mapsto v_2] t_{12} \quad \mu' = \mu \]

Case E-Ref:
\[ t = \text{ref } t_1 \quad t' = \text{ref } t_1' \quad t_1 \vdash \mu \vdash t_1' \vdash \mu' \]
Inversion(e), IH, T-Ref

Case E-DerefLoc:
\[ t = !l \quad t' = \mu(l) \quad \mu' = \mu \]
Inversion(f), inversion(h)

Case E-Deref:
\[ !t_1 \vdash \mu \vdash !t_1' \vdash \mu' \]

Case E-Assign:
\[ t = l := v_2 \quad t' = \text{unit} \quad \mu' = [l \mapsto v_2] \mu \]

Case E-Assign1:
\[ t = t_1 := t_2 \quad t' = t_1' := t_2 \quad t_1 \vdash t_1' \vdash \mu' \]

Case E-Assign2:
Similar.
Subtyping

The definition of the simply typed lambda-calculus with records and subtyping is reproduced for your reference on page 17.

6. (11 points) For each type $S$ from the left-hand column below, draw a line connecting it to each type $T$ in the right-hand column such that $S <: T$.

**Choices for $S$:**

- $\{a:\{\}, b:\{x:\text{Top}\}\}$
- $\text{Top} \rightarrow \text{Top}$
- $\{\} \rightarrow \{\}$
- $\text{Top}$
- $\{\{a:\text{Top}\} \rightarrow \{\}\} \rightarrow \{b:\text{Top}\}$
- $\{b:\text{Top} \rightarrow \text{Top}\}$

**Choices for $T$:**

- $\{\} \rightarrow \{a:\text{Top}\} \rightarrow \{\}$
- $\text{Top} \rightarrow \text{Top}$
- $\{\} \rightarrow \text{Top}$
- $\text{Top} \rightarrow \{\}$
- $\{b:\text{Top}\}$
- $\{b:\{\}\}$
7. (12 points) It is easy to show, by induction on subtyping derivations, that

**LEMMA A:** If \( \text{Top} \ll T \), then \( T = \text{Top} \).

A similar, but slightly more interesting, lemma holds for supertypes of arrow types.

**LEMMA B:** If \( S_1 \rightarrow S_2 \ll T \), then either \( T = \text{Top} \) or else \( T \) has the form \( T_1 \rightarrow T_2 \), with \( T_1 \ll S_1 \) and \( S_2 \ll T_2 \).

Fill in the arguments for the **S-ARROW** and **S-TRANS** cases of its proof.

**Proof:** By induction on subtyping derivations. Proceed by a case analysis on the last rule used in the derivation.

*Case S-REFL:* \( T = S_1 \rightarrow S_2 \)

\( T \) clearly has the required form, with \( T_1 = S_1 \) and \( T_2 = S_2 \). The inclusions \( T_1 \ll S_1 \) and \( S_2 \ll T_2 \) both follow by **S-REFL**.

*Case S-TRANS:* \( S_1 \rightarrow S_2 \ll U \quad U \ll T \)

*Fill in:*

**Case S-ARROW:** \( T = T_1 \rightarrow T_2 \quad T_1 \ll S_1 \) and \( S_2 \ll T_2 \)

*Fill in:*

**Case S-TOP:** \( T = \text{Top} \)

Immediate.

*Case S-RCDWIDTH, S-RCDDEPTH, S-RCDPERM, S-TOP:*

Can’t happen: \( T \) has the wrong form.
8. (9 points) Suppose we remove rule S-ARROW from the subtype relation. Which of the following properties will remain true? For each one, write either “true” (if it remains true) or else “false” (if it becomes false), plus (in either case) a one-sentence justification of your answer.

(a) Existence of minimal types (if term \( t \) is typable in context \( \Gamma \), then there is some type \( S \) such that \( \Gamma \vdash t : S \) and, for every type \( T \) such that \( \Gamma \vdash t : T \), we have \( S \ll T \))

(b) Progress (if \( t \) is a closed, well-typed term, then either \( t \) is a value or else \( t \rightarrow t' \) for some \( t' \))

(c) Preservation (if \( t \) has type \( T \) and \( t \rightarrow t' \), then \( t' \) also has type \( T \))
For reference: Simply typed lambda calculus with \texttt{Unit}

\textbf{Syntax}
\[
\begin{align*}
t & ::= \text{unit} \\
x & \text{variable} \\
\lambda x:T.t & \text{abstraction} \\
t \ t & \text{application}
\end{align*}
\]
\[
\begin{align*}
v & ::= \text{unit} \\
\lambda x:T.t & \text{abstraction value}
\end{align*}
\]
\[
\begin{align*}
T & ::= \text{Unit} \\
T \rightarrow T & \text{type of functions}
\end{align*}
\]
\[
\begin{align*}
\Gamma & ::= \emptyset \\
\Gamma, x:T & \text{term variable binding}
\end{align*}
\]

\textbf{Evaluation}
\[
\begin{align*}
t_1 & \rightarrow t'_1 \\
t_1 \ t_2 & \rightarrow t'_1 \ t_2 \\
t_2 & \rightarrow t'_2 \\
v_1 \ t_2 & \rightarrow v_1 \ t'_2 \\
(\lambda x:T_{11}.t_{12}) \ v_2 & \rightarrow [x \rightarrow v_2]t_{12}
\end{align*}
\]

\textbf{Typing}
\[
\begin{align*}
\Gamma \vdash \text{unit} : \text{Unit} & \text{(T-UNIT)} \\
\ x : T \in \Gamma & \text{(T-VAR)} \\
\Gamma \vdash \lambda x:T_1.t_2 : T_{1} \rightarrow T_2 & \text{(T-ABS)} \\
\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} & \text{and} \\
\Gamma \vdash t_2 : T_{11} & \text{(T-APP)}
\end{align*}
\]
New syntactic forms
\[ t ::= \ldots \]
- `ref t`
- `!t`
- `t:=t`

\[ v ::= \ldots \]
- `l`

\[ T ::= \ldots \]
- `Ref T`

\[ \mu ::= \ldots \]
- `∅`
  - `\mu, l = v`

\[ \Sigma ::= \ldots \]
- `∅`
- `\Sigma, l:T`

New evaluation rules

\[ \frac{t_1|\mu \rightarrow t_1'|\mu'}{t_1 \ t_2|\mu \rightarrow t_1' \ t_2|\mu'} \quad (E-App1) \]

\[ \frac{t_2|\mu \rightarrow t_2'|\mu'}{v_1 \ t_2|\mu \rightarrow v_1 \ t_2'|\mu'} \quad (E-App2) \]

\[ (\lambda x:T_{11} \ . \ t_{12}) \ v_2|\mu \rightarrow [x \rightarrow v_2] t_{12}|\mu \quad (E-AppAbs) \]

\[ \frac{l \notin \text{dom}(\mu)}{\text{ref} \ v_1 |\mu \rightarrow l | (\mu, l \rightarrow v_1)} \quad (E-RefV) \]

\[ \frac{t_1|\mu \rightarrow t_1'|\mu'}{\text{ref} \ t_1 |\mu \rightarrow \text{ref} \ t_1'|\mu'} \quad (E-Ref) \]

\[ \frac{\mu(l) = v}{!l |\mu \rightarrow v |\mu} \quad (E-DerefLoc) \]

\[ \frac{t_1|\mu \rightarrow t_1'|\mu'}{!t_1 |\mu \rightarrow !t_1'|\mu'} \quad (E-Deref) \]

\[ l := v_2 |\mu \rightarrow \text{unit} | [l \rightarrow v_2]\mu \quad (E-Assign) \]

\[ t_1 |\mu \rightarrow t_1'|\mu' \quad (E-Assign1) \]

\[ t \ |\mu \rightarrow t' \ |\mu' \]
New typing rules

\[
\begin{align*}
\frac{\tau_2 \mid \mu \rightarrow \tau'_2 \mid \mu'}{v_1 := \tau_2 \mid \mu \rightarrow v_1 := \tau'_2 \mid \mu'} & \quad (E\text{-}\text{ASSIGN2}) \\
\frac{\Gamma \mid \Sigma \vdash \text{unit : Unit}}{(T\text{-}\text{UNIT})} \\
\frac{x : T \in \Gamma}{\Gamma \mid \Sigma \vdash x : T} & \quad (T\text{-}\text{VAR}) \\
\frac{\Gamma, x : T_1 \mid \Sigma \vdash t_2 : T_2}{\Gamma \mid \Sigma \vdash \lambda x : T_1. t_2 : T_1 \rightarrow T_2} & \quad (T\text{-}\text{ABS}) \\
\frac{\Gamma \mid \Sigma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1 \ t_2 : T_{12}} & \quad (T\text{-}\text{APP}) \\
\frac{\Sigma(l) = T_1}{\Gamma \mid \Sigma \vdash l : \text{Ref } T_1} & \quad (T\text{-}\text{LOC}) \\
\frac{\Gamma \mid \Sigma \vdash t_1 : T_1}{\Gamma \mid \Sigma \vdash \text{ref } t_1 : \text{Ref } T_1} & \quad (T\text{-}\text{REF}) \\
\frac{\Gamma \mid \Sigma \vdash t_1 : \text{Ref } T_{11}}{\Gamma \mid \Sigma \vdash !t_1 : T_{11}} & \quad (T\text{-}\text{DEREF}) \\
\frac{\Gamma \mid \Sigma \vdash t_1 : \text{Ref } T_{11} \quad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1 := t_2 : \text{Unit}} & \quad (T\text{-}\text{ASSIGN})
\end{align*}
\]
New syntactic forms

\[ t ::= \ldots \]
\[ \{ l_i = t_i \}_{i=1}^n \]
\[ t.t \]

\[ v ::= \ldots \]
\[ \{ l_i = v_i \}_{i=1}^n \]

\[ T ::= \ldots \]
\[ \{ l_i : T_i \}_{i=1}^n \]

New evaluation rules

\[ \{ l_i = v_i \}_{i=1}^n \], \[ l_j \rightarrow v_j \] \hspace{1cm} (E-PROJ\text{RCD})

\[ t_i \rightarrow t'_i \]
\[ \frac{}{t_i.t \rightarrow t'_i.t} \] \hspace{1cm} (E-PROJ)

\[ t_j \rightarrow t'_j \]
\[ \{ l_i = v_i \}_{i=1}^{j-1}, l_j = t_j, l_k = t_k \}_{k=j+1}^n \]
\[ \rightarrow \{ l_i = v_i \}_{i=1}^{j-1}, l_j = t'_j, l_k = t_k \}_{k=j+1}^n \] \hspace{1cm} (E-RCD)

New subtyping rules

\[ S <: S \]
\[ S <: U U <: T \]
\[ S <: T \]
\[ S <: \text{Top} \]
\[ T_1 <: S_1 \quad S_2 <: T_2 \]
\[ S_1 - S_2 <: T_1 - T_2 \]
\[ \{ l_i : T_i \}_{i=1}^n \} <: \{ l_i : T_i \}_{i=1}^n \] \hspace{1cm} (S-RcdWidth)

\[ \text{for each } i \quad S_i <: T_i \]
\[ \{ l_i : S_i \}_{i=1}^n \} <: \{ l_i : T_i \}_{i=1}^n \] \hspace{1cm} (S-RcdDepth)

\[ \{ k_j : S_j \}_{j=1}^n \} \text{ is a permutation of } \{ l_i : T_i \}_{i=1}^n \]
\[ \{ k_j : S_j \}_{j=1}^n \} <: \{ l_i : T_i \}_{i=1}^n \] \hspace{1cm} (S-RcdPerm)

New typing rules

\[ \Gamma \vdash t_i : T_i \]
\[ \Gamma \vdash \{ l_i = t_i \}_{i=1}^n \} : \{ l_i : T_i \}_{i=1}^n \} \]
\[ \Gamma \vdash t_i : \{ l_i : T_i \}_{i=1}^n \} \]
\[ \Gamma \vdash t_i : \{ l_i : T_i \}_{i=1}^n \} \]
\[ \Gamma \vdash t \vdash S \quad S <: T \]
\[ \Gamma \vdash t : T \] \hspace{1cm} (T-Rcd)

\[ \Gamma \vdash v_i : \{ l_i = v_i \}_{i=1}^n \} \]
\[ \Gamma \vdash t_i : \{ l_i : T_i \}_{i=1}^n \} \]
\[ \Gamma \vdash t_i : \{ l_i : T_i \}_{i=1}^n \} \]
\[ \Gamma \vdash t_i : \{ l_i : T_i \}_{i=1}^n \} \]
\[ \Gamma \vdash t : S \quad S <: T \]
\[ \Gamma \vdash t : T \] \hspace{1cm} (T-Proj)

\[ \Gamma \vdash t : T \] \hspace{1cm} (T-Sub)