CIS 500 — Software Foundations
Midterm I

October 11, 2006

Name: ____________________

Student ID: ____________________

Email: ____________________

Status:
- ____ registered for the course
- ____ not registered: sitting in to improve a previous grade
- ____ not registered: just taking the exam for practice

Section:
- ____ 500-001 (Ph.D.)
- ____ 500-002 (MSE / undergraduate)

	Score
1	
2	
3	
4	
5	
6	
7	
8	
9	
Total	

Instructions

- This is a closed-book exam: you may not use any books or notes.
- You have 80 minutes to answer all of the questions. The entire exam is worth 80 points for students in section 002 and 90 points for students in section 001 (there is one PhD-section-only problem).
- 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!

OCaml

1. (5 points) The `forall` function takes a predicate `p` (a one-argument function returning a boolean) and a list `l`; it returns `true` if `p` returns `true` on every element of `l` and `false` otherwise.

```
# forall (fun x -> x >= 3) [2;11;4];;
- : bool = false

# forall (fun x -> x >= 3) [3;4;5];;
- : bool = true
```

   (a) What is the type of `forall`?

   (b) Complete the following definition of `forall` as a recursive function:

```
let rec forall p l =
```

2. (5 points) Recall the function `fold` discussed in class:

```
# let rec fold f l acc =
  match l with
    [] -> acc
  | a::l -> f a (fold f l acc);;
val fold : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b
```

Complete the following definition of `forall` by supplying appropriate arguments to `fold`:

```
let forall p l =

    fold  _________________________________________________________________________
```

Untyped lambda-calculus

The following questions are about the untyped lambda calculus. For reference, the definition of this language appears on page 13 at the end of the exam.

Recall the definitions of the following lambda-terms from the book and/or lecture notes:

```
/* A dummy "unit value", for forcing thunks */
unit = λx. x;

/* Standard definition of booleans */
tru = λt. λf. t;
fls = λt. λf. f;
not = λb. b fls tru;
test = λb. λt. λf. b t f unit;

/* Standard definition of pairs */
fst = λp. p tru;
snd = λp. p fls;
pair = λx. λy. λsel. sel x y;

/* Standard call-by-value fixed point function. */
fix = λf. (λx. f (λy. x x y)) (λx. f (λy. x x y));

/* Standard definitions of church numerals and arithmetic operations */
c0 = λs. λz. z;
c1 = λs. λz. s z;
c2 = λs. λz. s (s z);
c3 = λs. λz. s (s (s z));
c4 = λs. λz. s (s (s (s z)));
c5 = λs. λz. s (s (s (s (s z))));
c6 = λs. λz. s (s (s (s (s (s z)))));
scc = λn. λs. λz. s (n s z);
iszro = λm. m (λdummy. fls) tru;
zz = pair c0 c0;
ss = λp. pair (snd p) (scc (snd p));
prd = λm. fst (m ss zz);
```

3. (6 points) Circle the term that each of the following lambda calculus terms steps to, using the *single-step* evaluation relation $\mathtt{t} \longrightarrow \mathtt{t}'$. If the term is a normal form, circle DOESN'T STEP.

(a) `(λx.x) (λx. x x) (λx. x x)`

i. `(λx. x) (λx. x x) (λx. x x)`

ii. `(λx. x x) (λx. x x)`

iii. `(λx'. (λx. x x)) (λx. x x)`

iv. `(λx. x) (λx. x x)`

v. DOESN'T STEP

(b) `(λx. (λx.x) (λx. x x))`

i. `(λx. (λx.x) (λx. x x))`

ii. `(λx. (λx. x x))`

iii. `(λx. (λx. x))`

iv. `(λx. x) (λx. x x)`

v. DOESN'T STEP

(c) `(λx. (λz. λx. x z) x) (λx. x x)`

i. `(λx. (λz. λx. x z) x) (λx. x x)`

ii. `(λz. λx'. (λx. x x) z) (λx. x x)`

iii. `(λz. λx. x z) (λx. x x)`

iv. `(λx. x (λx. x x))`

v. DOESN'T STEP

4. (10 points) Recall the definitions of observational and behavioral equivalence from the lecture notes:

- Two terms `s` and `t` are *observationally equivalent* iff either both are normalizable (i.e., they reach a normal form after a finite number of evaluation steps) or both are divergent.
- Terms `s` and `t` are *behaviorally equivalent* iff, for every finite sequence of values $\mathtt{v}_1, \mathtt{v}_2, \ldots, \mathtt{v}_n$ (including the empty sequence), the applications

  $$\mathtt{s}\ \mathtt{v}_1\ \mathtt{v}_2\ \ldots\ \mathtt{v}_n$$

  and

  $$\mathtt{t}\ \mathtt{v}_1\ \mathtt{v}_2\ \ldots\ \mathtt{v}_n$$

  are observationally equivalent.

For each of the following pairs of terms, write *Yes* if the terms are behaviorally equivalent and *No* if they are not.

(a) $\mathtt{plus}\ \mathtt{c}_2\ \mathtt{c}_1$
$\mathtt{c}_3$

(b) `tru`
$\lambda\mathtt{x}.\ \lambda\mathtt{y}.\ (\lambda\mathtt{z}.\ \mathtt{z})\ \mathtt{x}$

(c) $\lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}\ \mathtt{y}$
$\lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}\ (\lambda\mathtt{z}.\ \mathtt{z})\ \mathtt{y}$

(d) $(\lambda\mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda\mathtt{x}.\ \mathtt{x}\ \mathtt{x})$
$\lambda\mathtt{x}.\ (\lambda\mathtt{x}.\ \mathtt{x}\ \mathtt{x})\ (\lambda\mathtt{x}.\ \mathtt{x}\ \mathtt{x})$

(e) $\lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}\ \mathtt{y}$
$\lambda\mathtt{x}.\ \mathtt{x}$

5. (12 points) Complete the following definition of a lambda-term `equal` that implements a *recursive* equality function on Church numerals. For example, `equal c0 c0` and `equal c2 c2` should be behaviorally equivalent to `tru`, while `equal c0 c1` and `equal c5 c0` should be behaviorally equivalent to `fls`. You may freely use the lambda-terms defined on page 3.

```
equal =

  fix (λe.

         λm. λn.

         test (iszro m)





      )
```

Simple types for numbers and booleans

6. (18 points) Recall the following properties of the language of numbers and booleans:

   - **Progress**: If $\vdash \texttt{t} : \texttt{T}$, then either $\texttt{t}$ is a value or else $\texttt{t} \longrightarrow \texttt{t}'$ for some $\texttt{t}'$.
   - **Preservation**: If $\Gamma \vdash \texttt{t} : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\Gamma \vdash \texttt{t}' : \texttt{T}$.
   - **Uniqueness of types**: Each term $\texttt{t}$ has at most one type, and if $\texttt{t}$ has a type, then there is exactly one derivation of that typing.

   Each part of this exercise suggests a different way of changing the language of typed arithmetic and boolean expressions (see page 11 for reference). Note that these changes are not cumulative: each part starts from the original language. In each part, for each property, indicate (by circling TRUE or FALSE) whether the property remains true or becomes false after the suggested change. If a property becomes false, give a counterexample.

   (a) Suppose we add the following typing axiom:

   $$\texttt{pred (succ 0) : Bool}$$

   Progress: TRUE FALSE, for example...

   Preservation: TRUE FALSE, for example...

   Uniqueness of types: TRUE FALSE, for example...

(b) Suppose we add the following evaluation axiom:

$$\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_1$$

Progress: TRUE FALSE, for example...

Preservation: TRUE FALSE, for example...

Uniqueness of types: TRUE FALSE, for example...

(c) Suppose we add a new type `Foo` and two new typing rules:

$$\frac{\texttt{t}_1 \texttt{ : Nat}}{\texttt{pred } \texttt{t}_1 \texttt{ : Foo}}$$

$$\frac{\texttt{t}_1 \texttt{ : Foo}}{\texttt{succ } \texttt{t}_1 \texttt{ : Nat}}$$

Progress: TRUE FALSE, for example...

Preservation: TRUE FALSE, for example...

Uniqueness of types: TRUE FALSE, for example...

7. (10 points) **[For students in the PhD section only.]** Suppose we add to the language of numbers and booleans two new types, called `True` and `False`, plus the following rules. (Note how the two rules for `if` allow types to be given to conditionals where the branches are not of the same type.)

$$\mathtt{true : True}$$

$$\mathtt{false : False}$$

$$\frac{\mathtt{t_1 : True} \qquad \mathtt{t_2 : T_2} \qquad \mathtt{t_3 : T_3}}{\mathtt{if\ t_1\ then\ t_2\ else\ t_3 : T_2}}$$

$$\frac{\mathtt{t_1 : False} \qquad \mathtt{t_2 : T_2} \qquad \mathtt{t_3 : T_3}}{\mathtt{if\ t_1\ then\ t_2\ else\ t_3 : T_3}}$$

(a) What type(s) can be derived for the following term?

```
if (if true then true else 0) then false else 0
```

(b) The *inversion lemma* tells us, for each syntactic form of terms, how terms of this form can be given types by the typing rules—intuitively, it allows us to "read the typing relation backwards."

Here is the inversion lemma from class for the original language of numbers and booleans:
Lemma:

- If `true` : R, then R = `Bool`.
- If `false` : R, then R = `Bool`.
- If `if` $\mathtt{t_1}$ `then` $\mathtt{t_2}$ `else` $\mathtt{t_3}$: R, then $\mathtt{t_1}$: `Bool`, $\mathtt{t_2}$: R, and $\mathtt{t_3}$: R.
- If `0` : R, then R = `Nat`.
- If `succ` $\mathtt{t_1}$: R, then R = `Nat` and $\mathtt{t_1}$: `Nat`.
- If `pred` $\mathtt{t_1}$: R, then R = `Nat` and $\mathtt{t_1}$: `Nat`.
- If `iszero` $\mathtt{t_1}$: R, then R = `Bool` and $\mathtt{t_1}$: `Nat`.

Complete the statements of the following clauses for the enriched language.

Lemma [Inversion]:

- If `true` : T, then

- If `if` $\mathtt{t_1}$ `then` $\mathtt{t_2}$ `else` $\mathtt{t_3}$: T, then

Simply typed lambda-calculus

The following questions are about the simply typed lambda-calculus over the base type `Nat` (not `Bool`, as in the book!). For reference, the definition of this language appears on page 14 at the end of the exam.

8. (6 points) Draw a typing derivation for the statement

$$\emptyset \vdash \texttt{(}\lambda\texttt{f:Nat}\rightarrow\texttt{Nat. f 0) (}\lambda\texttt{g:Nat. pred g) : Nat}$$

9. (18 points) Here are the weakening and permutation lemmas for $\lambda_{\rightarrow}$:

> *Lemma [Weakening]:* If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{x} \notin dom(\Gamma)$, then $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$. Moreover, the latter derivation has the same depth as the former.
>
> *Lemma [Permutation]:* If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and Δ is a permutation of Γ, then $\Delta \vdash \mathtt{t} : \mathtt{T}$. Moreover, the latter derivation has the same depth as the former.

Fill in the missing parts of the proof of the substitution lemma on the following page.

- In the T-Succ case, you need to fill in *both* the assumptions coming from the case analysis (the three blank lines at the beginning of the case) *and* the body of the argument.
- Your wording does not need to exactly match what is in the book or lecture notes, but every step required in the proof (use of an assumption, application of a lemma, use of the induction hypothesis, or use of a typing rule) must be mentioned explicitly.
- The cases for application, zero, and predecessor are omitted; you don't need to worry about these.

Lemma [Substitution]: If $\Gamma, \mathtt{x}:\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

Proof: By induction on the depth of a derivation of $\Gamma, \mathtt{x}:\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$. Proceed by cases on the final typing rule used in the derivation.

Case T-VAR: $\mathtt{t} = \mathtt{z}$
with $\mathtt{z}:\mathtt{T} \in (\Gamma, \mathtt{x}:\mathtt{S})$

Case T-ABS: $\mathtt{t} = \lambda\mathtt{y}:\mathtt{T}_2.\mathtt{t}_1$ $\quad \mathtt{T} = \mathtt{T}_2 \to \mathtt{T}_1$
$\Gamma, \mathtt{x}:\mathtt{S}, \mathtt{y}:\mathtt{T}_2 \vdash \mathtt{t}_1 : \mathtt{T}_1$

By our conventions on choice of bound variable names, we may assume $\mathtt{x} \neq \mathtt{y}$ and $\mathtt{y} \notin FV(\mathtt{s})$.

Case T-SUCC: ______________________________

____

____

For reference: Boolean and arithmetic expressions

Syntax

`t ::=`		*terms*
	`true`	*constant true*
	`false`	*constant false*
	`if t then t else t`	*conditional*
	`0`	*constant zero*
	`succ t`	*successor*
	`pred t`	*predecessor*
	`iszero t`	*zero test*
`v ::=`		*values*
	`true`	*true value*
	`false`	*false value*
	`nv`	*numeric value*
`nv ::=`		*numeric values*
	`0`	*zero value*
	`succ nv`	*successor value*
`T ::=`		*types*
	`Bool`	*type of booleans*
	`Nat`	*type of numbers*

Evaluation

$$\texttt{if true then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_3 \qquad \text{(E-IfFalse)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{if } \texttt{t}_1' \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3} \qquad \text{(E-If)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{succ } \texttt{t}_1 \longrightarrow \texttt{succ } \texttt{t}_1'} \qquad \text{(E-Succ)}$$

$$\texttt{pred 0} \longrightarrow \texttt{0} \qquad \text{(E-PredZero)}$$

$$\texttt{pred (succ } \texttt{nv}_1\texttt{)} \longrightarrow \texttt{nv}_1 \qquad \text{(E-PredSucc)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{pred } \texttt{t}_1 \longrightarrow \texttt{pred } \texttt{t}_1'} \qquad \text{(E-Pred)}$$

$$\texttt{iszero 0} \longrightarrow \texttt{true} \qquad \text{(E-IszeroZero)}$$

$$\texttt{iszero (succ } \texttt{nv}_1\texttt{)} \longrightarrow \texttt{false} \qquad \text{(E-IszeroSucc)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{iszero } \texttt{t}_1 \longrightarrow \texttt{iszero } \texttt{t}_1'} \qquad \text{(E-IsZero)}$$

continued on next page...

Typing

$$\texttt{true} : \texttt{Bool} \qquad \text{(T-True)}$$

$$\texttt{false} : \texttt{Bool} \qquad \text{(T-False)}$$

$$\frac{\texttt{t}_1 : \texttt{Bool} \quad \texttt{t}_2 : \texttt{T} \quad \texttt{t}_3 : \texttt{T}}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{T}} \qquad \text{(T-If)}$$

$$\texttt{0} : \texttt{Nat} \qquad \text{(T-Zero)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{succ } \texttt{t}_1 : \texttt{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{pred } \texttt{t}_1 : \texttt{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{iszero } \texttt{t}_1 : \texttt{Bool}} \qquad \text{(T-IsZero)}$$

For reference: Untyped lambda calculus

Syntax

$\mathtt{t}$ ::=		*terms*
	$\mathtt{x}$	*variable*
	$\lambda\mathtt{x.t}$	*abstraction*
	$\mathtt{t\ t}$	*application*
$\mathtt{v}$ ::=		*values*
	$\lambda\mathtt{x.t}$	*abstraction value*

Evaluation

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \qquad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \qquad \text{(E-App2)}$$

$$(\lambda\mathtt{x.t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \qquad \text{(E-AppAbs)}$$

For reference: Simply typed lambda-calculus with numbers

Syntax

t ::=		*terms*
	$\mathtt{x}$	*variable*
	$\lambda\mathtt{x}\mathtt{:T.t}$	*abstraction*
	$\mathtt{t\ t}$	*application*
	$\mathtt{0}$	*constant zero*
	$\mathtt{succ\ t}$	*successor*
	$\mathtt{pred\ t}$	*predecessor*
v ::=		*values*
	$\lambda\mathtt{x}\mathtt{:T.t}$	*abstraction value*
	$\mathtt{nv}$	*numeric value*
nv ::=		*numeric values*
	$\mathtt{0}$	*zero value*
	$\mathtt{succ\ nv}$	*successor value*
T ::=		*types*
	$\mathtt{Nat}$	*type of numbers*
	$\mathtt{T}\to\mathtt{T}$	*type of functions*

Evaluation

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-App2)}$$

$$(\lambda\mathtt{x}\mathtt{:T}_1\mathtt{.t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-AppAbs)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{succ}\ \mathtt{t}_1 \longrightarrow \mathtt{succ}\ \mathtt{t}_1'} \quad \text{(E-Succ)}$$

$$\mathtt{pred\ 0} \longrightarrow \mathtt{0} \quad \text{(E-PredZero)}$$

$$\mathtt{pred\ (succ\ nv}_1\mathtt{)} \longrightarrow \mathtt{nv}_1 \quad \text{(E-PredSucc)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{pred}\ \mathtt{t}_1 \longrightarrow \mathtt{pred}\ \mathtt{t}_1'} \quad \text{(E-Pred)}$$

continued on next page...

Typing

$$\frac{\mathtt{x}{:}\mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \tag{T-Var}$$

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1 \to \mathtt{T}_2} \tag{T-Abs}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \tag{T-App}$$

$$\Gamma \vdash \mathtt{0} : \mathtt{Nat} \tag{T-Zero}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{succ}\ \mathtt{t}_1 : \mathtt{Nat}} \tag{T-Succ}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{pred}\ \mathtt{t}_1 : \mathtt{Nat}} \tag{T-Pred}$$