CIS 500 — Software Foundations

Midterm I

Answer key

October 13, 2004

Name: ______________________________

Student ID: ______________________________

Email ______________________________

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

Program
- ____ undergrad
- ____ undergrad (MSE submatriculant)
- ____ CIS MSE
- ____ CIS MCIT
- ____ CIS PhD
- ____ other

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!

Please mark your preferences for the time and date of the final exam.

Date	Time	can't do it	rather not	ok with me
12/16	8:30-10:30			
12/20	1:30-3:30			
12/21	11-1			
12/21	1:30-3:30			

Semantics of simple programming languages

The following four questions concern the following simple programming language:

`t` ::=		*terms*
	`true`	*constant true*
	`false`	*constant false*
	`maybe`	*constant maybe*
	`perhaps` t_1 `then` t_2 `else` t_3	*conditional*
	`definitely` t_1 `then` t_2 `else` t_3	*another conditional*
`v` ::=		*values*
	`true`	*true value*
	`false`	*false value*
	`maybe`	*maybe value*

and its *small-step* operational semantics.

$$\texttt{perhaps true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \text{(E-PT)}$$

$$\texttt{perhaps maybe then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \text{(E-PM)}$$

$$\texttt{perhaps false then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \qquad \text{(E-PF)}$$

$$\texttt{definitely true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \text{(E-DT)}$$

$$\texttt{definitely maybe then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \qquad \text{(E-DM)}$$

$$\texttt{definitely false then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \qquad \text{(E-DF)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{perhaps } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{perhaps } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad \text{(E-P)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{definitely } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{definitely } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad \text{(E-M)}$$

1. (15 points)

   (a) State the structural induction principle for the syntax of this language.
   Answer: For all terms `t`*, P(*`t`*) is true if and only if*

   - *P(*`true`*), P(*`false`*), and P(*`maybe`*) are true*
   - *P(*`perhaps` t_1 `then` t_2 `else` t_3*) is true given P(*t_1*), P(*t_2*) and P(*t_3*).*
   - *P(*`definitely` t_1 `then` t_2 `else` t_3*) is true given P(*t_1*), P(*t_2*) and P(*t_3*).*

(b) Prove by structural induction, the following statement: For all $\mathtt{t}$, either $\mathtt{t}$ is a value or $\mathtt{t} \longrightarrow \mathtt{t}'$. Note: If two cases of this proof are extremely similar, you may say that the second case is analogous to the first, instead of writing the case out in full.

Answer: The property that we would like to show is $P(\mathtt{t})$ *= either* $\mathtt{t}$ *is a value or* $\mathtt{t} \longrightarrow \mathtt{t}'$.

- *Showing that the property is true when* $\mathtt{t}$ *is* `true`, `false` *or* `maybe` *is trivial, because* $\mathtt{t}$ *is a value in each of these cases.*
- *Suppose* $\mathtt{t}$ *is* `perhaps` $\mathtt{t}_1$ `then` $\mathtt{t}_2$ `else` $\mathtt{t}_3$. *By induction, we know that either* $\mathtt{t}_1$ *is a value, or* $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$.
  - *If* $\mathtt{t}_1$ *is* `true` *then* $\mathtt{t} \longrightarrow \mathtt{t}_2$ *by E-PT.*
  - *If* $\mathtt{t}_1$ *is* `false` *then* $\mathtt{t} \longrightarrow \mathtt{t}_3$ *by E-PF.*
  - *If* $\mathtt{t}_1$ *is* `maybe` *then* $\mathtt{t} \longrightarrow \mathtt{t}_2$ *by E-PM.*
  - *If* $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$ *then* $\mathtt{t} \longrightarrow$ `perhaps` $\mathtt{t}_1'$ `then` $\mathtt{t}_2$ `else` $\mathtt{t}_3$ *by E-P.*
- *Suppose* $\mathtt{t}$ *is* `definitely` $\mathtt{t}_1$ `then` $\mathtt{t}_2$ `else` $\mathtt{t}_3$. *This case is analogous to the previous.*

2. (4 points) We can define a new term form $\texttt{and}\ \texttt{t}_1\ \texttt{t}_2$ with the following operational semantics rules:

$$\texttt{and true v} \longrightarrow \texttt{v}$$

$$\texttt{and maybe true} \longrightarrow \texttt{maybe}$$

$$\texttt{and maybe maybe} \longrightarrow \texttt{maybe}$$

$$\texttt{and maybe false} \longrightarrow \texttt{false}$$

$$\texttt{and false v} \longrightarrow \texttt{false}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{and}\ \texttt{t}_1\ \texttt{t}_2 \longrightarrow \texttt{and}\ \texttt{t}_1'\ \texttt{t}_2}$$

$$\frac{\texttt{t}_2 \longrightarrow \texttt{t}_2'}{\texttt{and v}\ \texttt{t}_2 \longrightarrow \texttt{and v}\ \texttt{t}_2'}$$

However, this term is *definable* using the existing constructs of the language. What is its definition?

$\texttt{and}\ \texttt{t}_1\ \texttt{t}_2$ =

Answer: *definitely* t_1 *then* t_2 *else (perhaps* t_2 *then t1 else false)*.

3. (8 points) We can also encode this language into the untyped lambda calculus. Here is part of the encoding, fill in the missing pieces in the simplest way possible.

$comp(\texttt{true}) = \lambda\texttt{x}.\lambda\texttt{y}.\lambda\texttt{z}.\ \texttt{x}\ (\lambda\texttt{w}.\texttt{w})$

$comp(\texttt{false})$ = ______________________________

Answer: $\lambda x.\lambda y.\ \lambda z.\ z\ (\lambda w.w)$

$comp(\texttt{maybe})$ = ______________________________

Answer: $\lambda x.\lambda y.\ \lambda z.\ y\ (\lambda w.w)$

$comp(\texttt{perhaps}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3) = comp(\texttt{t}_1)\ (\lambda\texttt{w}.comp(t_2))\ (\lambda\texttt{w}.comp(t_2))(\lambda\texttt{w}.comp(t_3))$

$comp(\texttt{definitely}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3)$ =

____

Answer: $comp(t_1)\ (\lambda w.comp(t_2))\ (\lambda w.comp(t_3))(\lambda w.comp(t_3))$

4. (8 points) The following O'Caml definitions implement the small-step evaluation relation *almost correctly*, but there are several mistakes or omissions. Change the code below to repair these mistakes.

```
type term = TmTrue | TmFalse | TmMaybe
      | TmPerhaps of term * term * term
      | TmDefinitely of term * term * term

let rec ss t = match t with

    TmPerhaps(t1,t2,t3) →

      (match t1 with

          TmTrue   → ss t2

        | TmMaybe  → ss t2

        | TmFalse  → ss t3

        | _ →  TmPerhaps(ss t11, ss t2, t3))

  | TmDefinitely(TmTrue,t2,t3)  → ss t2

  | TmDefinitely(TmFalse,t2,t3) → ss t3

  | TmDefinitely(t1,t2,t3) →  TmPerhaps(ss t1, t2, t3)

  | TmTrue → ss TmTrue

  | TmFalse → TmFalse
```

Answer:

- *Cases for* `perhaps true`, `perhaps false`, `perhaps maybe` *and* `definitely true` *shouldn't call* `ss` *recursively. Also, the last case for* `perhaps` *shouldn't call* `ss` *recursively for* `t2`.
- *Last case for* `perhaps` *should say* `t1` *instead of* `t11`.
- *Missing case for* `definitely maybe`.
- *Congruence rule for* `definitely` *goes to* `perhaps`.
- *Because this is a single step semantics, there shouldn't be cases for* `true` *and* `false`, *these are values. For these terms, an exception should be raised.*

Untyped lambda-calculus

For each of the following pairs of untyped lambda-terms, answer the following three questions:

(a) What are their normal forms? If a term does not have a normal form, write *none*.

(b) If they have normal forms, are these normal forms alpha-equivalent? If they are alpha-equivalent write *yes*, if they are not write *no*, if at least one term does not have a normal form write *not applicable*.

(c) Are these terms *behaviorally equivalent*? Write *yes* or *no*.

Recall the following definitions of observational and behavioral equivalence from 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 $\mathtt{v}_1$, ... $\mathtt{v}_n$, the applications

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

  and

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

  are observationally equivalent.

5. (6 points) `(λx. x x)(λx. x x)` and `(λx. x x x)(λx. x x x)`.
   (a) *Answer: none and none*
   (b) *Answer: not applicable*
   (c) *Answer: yes*

6. (6 points) `(λx.λy.x)` and `(λx.λy. (λw.w) x)`.
   (a) *Answer:* `(λx.λy.x)` *and* `(λx.λy. (λw.w) x)`
   (b) *Answer: no*
   (c) *Answer: yes*

7. (6 points) `(λx.λy.x)(λz.y)` and `(λx.λy.x)(λx.w)`.
   (a) *Answer:* `λw.λz.y` *and* `λy. λx. w`
   (b) *Answer: no*
   (c) *Answer: yes*

8. (9 points) The following is a slightly different encoding of natural numbers in the untyped lambda calculus.

```
s₀ = λs.λz.z
s₁ = λs.λz.s s₀ z
s₂ = λs.λz.s s₁ (s s₀ z)
s₃ = λs.λz.s s₂ (s s₁ (s s₀ z))

scc = λn.λs.λz.s n (n s z)
```

(a) Define the predecessor function `prd` for this encoding, using the simplest term you can.

Answer: $\texttt{prd} = \lambda n.\ n\ (\lambda m.\lambda r.\ m)\ s_0$

(b) Define the addition function `plus` for this encoding, using the simplest term you can.

Answer: $\texttt{plus} = \lambda n.\ \lambda m.\ n\ (\lambda x.\ scc)\ m$

or the same definition for `plus` *as for Church numerals:*

$\texttt{plus} = \lambda n.\ \lambda m.\ \lambda s.\lambda z.\ n\ s\ (m\ s\ z)$

(c) Define the function `sumupto` that, given the encoding of a number `m`, calculates the sum of all the numbers less than or equal to `m`. Use the simplest term you can, and do not use `fix`.

Answer: $\texttt{sumupto} = \lambda m.\ m\ \texttt{plus}\ m$ *is the simplest answer.*

Several people gave a function that sums all of the numbers less than `m`*, such as* $\lambda m.\ m\ \texttt{plus}\ s_0$*. Partial credit was awarded for this function.*

Typed arithmetic expressions

The full definition of the language of typed arithmetic and boolean expressions is reproduced, for your reference, on page 10.

9. (6 points) Suppose we add the following two new rules to the evaluation relation:

$$\texttt{pred true} \longrightarrow \texttt{pred false}$$

$$\texttt{pred false} \longrightarrow \texttt{pred true}$$

Which of the following properties will remain true in the presence of this rule? For each one, circle either "remains true" or else "becomes false." If a property becomes false, also write down a counter-example to the property.

(a) Termination of evaluation (for every term $\texttt{t}$ there is some normal form $\texttt{t}'$ such that $\texttt{t} \longrightarrow^* \texttt{t}'$)

remains true becomes false, because ....

Answer: Becomes false. $\texttt{pred true} \longrightarrow \texttt{pred false} \longrightarrow \texttt{pred true}$ ...

(b) Progress (if $\texttt{t}$ is well typed, then either $\texttt{t}$ is a value or else $\texttt{t} \longrightarrow \texttt{t}'$ for some $\texttt{t}'$)

remains true becomes false, because ....

Answer: Remains true

(c) Preservation (if $\texttt{t}$ has type $\texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\texttt{t}'$ also has type $\texttt{T}$)

remains true becomes false, because ....

Answer: Remains true

10. (6 points) Suppose, instead, that we add this new rule to the typing relation:

$$\frac{\texttt{t}_2 : \texttt{Nat}}{\texttt{if true then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{Nat}}$$

Which of the following properties remains true? (Answer in the same style as the previous question.)

(a) Termination of evaluation (for every term $\texttt{t}$ there is some normal form $\texttt{t}'$ such that $\texttt{t} \longrightarrow^* \texttt{t}'$)

remains true becomes false, because ....

Answer: Remains true

(b) Progress (if $\texttt{t}$ is well typed, then either $\texttt{t}$ is a value or else $\texttt{t} \longrightarrow \texttt{t}'$ for some $\texttt{t}'$)

remains true becomes false, because ....

Answer: Remains true

(c) Preservation (if $\texttt{t}$ has type $\texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\texttt{t}'$ also has type $\texttt{T}$)

remains true becomes false, because ....

Answer: Remains true

11. (6 points) Suppose, instead, that we add this new rule to the typing relation:

$$\frac{\texttt{t : Bool}}{\texttt{succ t : Bool}}$$

Which of the following properties remains true? (Answer in the same style as the previous question.)

(a) Termination of evaluation (for every term $\texttt{t}$ there is some normal form $\texttt{t}'$ such that $\texttt{t} \longrightarrow^{*} \texttt{t}'$)

remains true   becomes false, because ....

Answer: Remains true

(b) Progress (if $\texttt{t}$ is well typed, then either $\texttt{t}$ is a value or else $\texttt{t} \longrightarrow \texttt{t}'$ for some $\texttt{t}'$)

remains true   becomes false, because ....

Answer: Becomes false. `succ true` *is well-typed, but stuck.*

(c) Preservation (if $\texttt{t}$ has type $\texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\texttt{t}'$ also has type $\texttt{T}$)

remains true   becomes false, because ....

Answer: Remains true

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

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \qquad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad \text{(E-If)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{succ } t_1 \longrightarrow \texttt{succ } t_1'} \qquad \text{(E-Succ)}$$

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

$$\texttt{pred (succ } nv_1\texttt{)} \longrightarrow nv_1 \qquad \text{(E-PredSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{pred } t_1 \longrightarrow \texttt{pred } t_1'} \qquad \text{(E-Pred)}$$

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

$$\texttt{iszero (succ } nv_1\texttt{)} \longrightarrow \texttt{false} \qquad \text{(E-IszeroSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{iszero } t_1 \longrightarrow \texttt{iszero } t_1'} \qquad \text{(E-IsZero)}$$

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*
	$\lambda\mathtt{x:T.t}$	*abstraction value*

Evaluation

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

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

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