Programming Languages
Fall 2013
Midterm Exam
10/22/13
Time Limit: 100 Minutes

Name (Print): ______________________

Graduate Center I.D. ______________________

- This is a closed-book, closed-notes exam.
- Various definitions are provided in the exam.
- Do not get stuck on a single problem.

Do not write in the table to the right.

Problem	Points	Score
1	5	
2	5	
3	9	
4	6	
5	5	
6	3	
7	7	
8	7	
9	6	
10	6	
11	13	
12	5	
13	6	
14	9	
15	8	
Total:	100	

Operational Semantics

The first few questions concern the following simple programming language:

`t ::=`		*terms*
	`true`	*constant true*
	`false`	*constant false*
	`if t then t else t`	*conditional*
	`pair t t`	*pairing*
	`fst t`	*first component*
	`snd t`	*second component*
`v ::=`		*values*
	`true`	*true value*
	`false`	*false value*
	`pair v v`	*pair value*

and its *big-step* operational semantics.

$$\texttt{true} \Downarrow \texttt{true} \qquad \text{(B-True)}$$

$$\texttt{false} \Downarrow \texttt{false} \qquad \text{(B-False)}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{true} \qquad \texttt{t}_2 \Downarrow \texttt{v}}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \Downarrow \texttt{v}} \qquad \text{(B-IfTrue)}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{false} \qquad \texttt{t}_3 \Downarrow \texttt{v}}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \Downarrow \texttt{v}} \qquad \text{(B-IfFalse)}$$

$$\frac{\texttt{t}_1 \Downarrow \texttt{v}_1 \qquad \texttt{t}_2 \Downarrow \texttt{v}_2}{\texttt{pair } \texttt{t}_1 \ \texttt{t}_2 \Downarrow \texttt{pair } \texttt{v}_1 \ \texttt{v}_2} \qquad \text{(B-Pair)}$$

$$\frac{\texttt{t} \Downarrow \texttt{pair } \texttt{v}_1 \ \texttt{v}_2}{\texttt{fst } \texttt{t} \Downarrow \texttt{v}_1} \qquad \text{(B-Fst)}$$

$$\frac{\texttt{t} \Downarrow \texttt{pair } \texttt{v}_1 \ \texttt{v}_2}{\texttt{snd } \texttt{t} \Downarrow \texttt{v}_2} \qquad \text{(B-Snd)}$$

1. (5 points) Draw the derivation tree of the *big-step* evaluation of the following term. Remember to include the rule name for each rule applied.

   `fst (if true then pair true false else pair false true)`

2. (5 points) We might also want to define a *small-step* semantics for this language, such that

$$\texttt{t} \Downarrow \texttt{v} \text{ if and only if } \texttt{t} \longrightarrow^* \texttt{v}$$

Recall that a small-step semantics is composed of both computation and congruence rules. Here is a list of rules. Please fill in the rule type (congruence or computation) for each rule (except for the first one which I did for you).

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

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

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

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{pair t}_1 \texttt{ t}_2 \longrightarrow \texttt{pair t}_1' \texttt{ t}_2} \quad \text{(E-Pair)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{fst t}_1 \longrightarrow \texttt{fst t}_1'} \quad \text{(E-Fst)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{snd t}_1 \longrightarrow \texttt{snd t}_1'} \quad \text{(E-Snd)}$$

3. (9 points) However, this list is not complete. List the remaining rules and their types (Hint: there are three of them). You can name them as you like, but the names will be used in your answers to later questions.

	rule	name	type
1			
2			
3			

4. (6 points) Show the small-step evaluation steps of the following term until reaching a value:

```
fst (if fst (pair true false) then pair true false else false)
```

$\rightarrow$

$\rightarrow$

$\rightarrow$

5. (5 points) Draw the derivation tree for the first evaluation step above. Again, remember the rule names.

6. (3 points) Are there any "stuck" terms in this language? (i.e. a term that fails to produce a value). If so, give an example. If not, explain why not.

Functional programming

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

Recall the Church encoding of lists and booleans in the untyped lambda calculus.

$$
\begin{aligned}
\texttt{tru} &= \lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{x}\\
\texttt{fls} &= \lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{y}\\
\texttt{not} &= \lambda\texttt{b}.\ \texttt{b fls tru}\\
\texttt{and} &= \lambda\texttt{b1}.\ \lambda\texttt{b2}.\ \texttt{b1 b2 fls}\\
\texttt{or} &= \lambda\texttt{b1}.\ \lambda\texttt{b2}.\ \texttt{b1 tru b2}
\end{aligned}
$$

$$
\begin{aligned}
\texttt{nil} &= \lambda\texttt{c}.\ \lambda\texttt{n}.\ \texttt{n}\\
\texttt{cons} &= \lambda\texttt{h}.\ \lambda\texttt{t}.\ \lambda\texttt{c}.\ \lambda\texttt{n}.\ \texttt{c h (t c n)}\\
\texttt{head} &= \lambda\texttt{l}.\ \texttt{l}\ (\lambda\texttt{h}.\ \lambda\texttt{t}.\ \texttt{h})\ \texttt{fls}\\
\texttt{tail} &= \lambda\texttt{l}.\ \texttt{fst}\ (\texttt{l}\ (\lambda\texttt{x}.\ \lambda\texttt{p}.\ \texttt{pair (snd p) (cons x (snd p))})\ (\texttt{pair nil nil}))\\
\texttt{isnil} &= \lambda\texttt{l}.\ \texttt{l}\ (\lambda\texttt{h}.\ \lambda\texttt{t}.\ \texttt{fls})\ \texttt{tru}
\end{aligned}
$$

7. (7 points) Which of the following terms defines the function `all` that takes a list of boolean terms and determines of **all** of the terms are true? For example,
`all (cons tru (cons fls nil))` should be equivalent to `fls`
and `all nil` should be equivalent to `tru`. Circle the correct answer.

   (a) `all` = λ`l`. (λ`a`. λ`b`. `a and b`) `l fls`

   (b) `all` = λ`l`. `l` (λ`a`. λ`b`. `a tru b`) `fls`

   (c) `all` = λ`l`. `all (head l) (tail l)`

   (d) `all` = λ`l`. `l and tru`

   Explain your intuition.

   Show that `all (cons tru (cons fls nil))` is equivalent to `fls`. For convenience, you may use full beta-reduction.

8. (7 points) Which of the following terms defines the function `map` that takes a term `l`, representing a list, and a function `f`, applies `f` to each element of `l`, and yields a list of the results (just like the `map` in Haskell). For example:
`map not (cons tru (cons fls nil))`
should be equivalent to
`(cons fls (cons tru nil))`. Circle the correct answer.

   (a) `map` = λ`f.` λ`l. l` (λ`h.` λ`t. cons t (f h)) nil`

   (b) `map` = λ`f.` λ`l.` λ`c.` λ`n. l` (λ`h.` λ`t. c (f h) t)` n

   (c) `map` = λ`f.` λ`l. l (f cons) nil`

   Show that `map not (cons tru (cons fls nil))` is equivalent to `cons fls (cons tru nil)`. Again, you may use full-beta reduction.

9. (6 points) Implement `all` and `map` in Haskell, using recursion.

```
all f [] =

all f (x:xs) =

map f [] =

map f (x:xs) =
```

Proof by Induction

10. (6 points) Recall that $FV(\texttt{t})$ is the set of free variables in $\texttt{t}$. Compute:

- $FV(\texttt{x})$
- $FV(\lambda\texttt{x}.\ \texttt{y})$
- $FV((\lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{z})\ \ \texttt{y})$

11. (13 points) Complete the following proof of a property of the untyped lambda calculus, by induction on the structure of lambda terms.

Theorem: If $\texttt{t}$ is closed (i.e., there is no free variable in $\texttt{t}$), and $\texttt{t} \longrightarrow \texttt{t}'$, then $\texttt{t}'$ is closed.

You may use, without proving, the following lemma about substitution.
Lemma: $(FV(\texttt{t}_1)\backslash\{\texttt{x}\}) \cup FV(\texttt{t}_2) \supseteq FV([\texttt{x} \mapsto \texttt{t}_2]\texttt{t}_1)$.

We prove the theorem by induction on the structure of the lambda term $\texttt{t}$.

- Suppose $\texttt{t}$ is a variable $\texttt{x}$. This is trivial because:

- Suppose $\texttt{t}$ is a lambda term $\lambda\texttt{x}.\ \texttt{t}_1$. This case is also trivial because:

- Suppose $\texttt{t}$ is an application $\texttt{t}_1\ \ \texttt{t}_2$.
  -
  -
  -

12. (5 points) Re: the lemma, show an example that $(FV(\texttt{t}_1)\backslash\{\texttt{x}\}) \cup FV(\texttt{t}_2) \neq FV([\texttt{x} \mapsto \texttt{t}_2]\texttt{t}_1)$.

Untyped lambda calculus

Call-by-Value Evaluation

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1}\ \mathtt{t_2} \longrightarrow \mathtt{t_1'}\ \mathtt{t_2}} \qquad \text{(E-App1)}$$

$$\frac{\mathtt{t_2} \longrightarrow \mathtt{t_2'}}{\mathtt{v_1}\ \mathtt{t_2} \longrightarrow \mathtt{v_1}\ \mathtt{t_2'}} \qquad \text{(E-App2)}$$

$$(\lambda \mathtt{x}.\ \mathtt{t_{12}})\ \mathtt{v_2} \longrightarrow [\mathtt{x} \mapsto \mathtt{v_2}]\mathtt{t_{12}} \qquad \text{(E-AppAbs)}$$

13. (6 points) What do the following lambda calculus terms step to (in one step), using the call-by-value *single-step* evaluation relation $\mathtt{t} \longrightarrow \mathtt{t'}$. Write *NONE* if the term does not step. For reference, the semantics of call-by-value evaluation is given above.

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

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

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

14. (9 points) Now redo the above question with full-beta reduction (i.e., can reduce anywhere): write *all* possible $\mathtt{t'}$ that $\mathtt{t}$ can step to in one-step. Again, write *NONE* if $\mathtt{t}$ does not step.

15. (8 points) Write out the single-step evaluation rules for "full-beta reduction".