Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- Please stop promptly at 1:20.
- You can rip apart the pages, but please write your name on each page.
- There are 140 points total, distributed unevenly among 6 questions (which have multiple parts).

Advice:

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit.
- The questions are not necessarily in order of difficulty. Skip around. In particular, do not spend so much time on a proof that you do not get to all the problems.
- If you have questions, ask.
- Relax. You are here to learn.
Name: ____________________________

For your reference:

\[
\begin{align*}
\text{s} &::= \text{skip} | x := e | s ; s | \text{if } e \text{ } s \text{ } s | \text{while } e \text{ } s \\
\text{e} &::= c | x | e + e | e \cdot e \\
(c) &\in \{\ldots, -2, -1, 0, 1, 2, \ldots \} \\
(x) &\in \{x_1, x_2, \ldots, y_1, y_2, \ldots, z_1, z_2, \ldots, \ldots \} \\
\end{align*}
\]

\[
\begin{array}{c}
H ; e \Downarrow c \\
\end{array}
\]

\[
\begin{array}{lcccc}
\text{CONST} & \text{VAR} & \text{ADD} & \text{MULT} \\
H ; c \Downarrow c & H ; x \Downarrow H(x) & H ; e_1 \Downarrow c_1 & H ; e_2 \Downarrow c_2 & H ; e_1 \Downarrow c_1 & H ; e_2 \Downarrow c_2 \\
H_1 ; s_1 \rightarrow H_2 \rightarrow s_2 & H ; x := e \rightarrow H, x \rightarrow c ; \text{skip} & H ; \text{skip}; s \rightarrow H ; s & H ; s_1 \rightarrow H' ; s_1' & H ; s_1 \rightarrow H ; s_2 & H ; s_1 \rightarrow H ; s_2 \\
\end{array}
\]

\[
\begin{array}{c}
\text{IF1} & \text{IF2} \\
H ; e \Downarrow c & c > 0 & H ; e \Downarrow c & c \leq 0 \\
H ; \text{if } e \text{ } s_1 \text{ } s_2 \rightarrow H ; s_1 & H ; \text{if } e \text{ } s_1 \text{ } s_2 \rightarrow H ; s_2 \\
\end{array}
\]

\[
\begin{align*}
\text{e} &::= \lambda x. \ e | x | e \cdot e | c \\
\text{v} &::= \lambda x. \ e | c \\
\tau &::= \text{int} | \tau \rightarrow \tau \\
\end{align*}
\]

\[
\begin{array}{c}
\text{e} \rightarrow \text{e}' \\
(\lambda x. \ e) \text{ } v \rightarrow e[v/x] & e_1 \rightarrow e'_1 & e_2 \rightarrow e'_2 \\
\end{array}
\]

\[
\begin{array}{c}
e[e'/x] = e'' \\
x[e/x] = e & e_1[e/x] = e'_1 & y \neq x \text{ } y \notin \text{FV}(e) \\
(\lambda y. \ e_1)[e/x] = \lambda y. \ e'_1 \\
y \neq x & \text{y} \neq x \text{ } y \notin \text{FV}(e) \\
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash e : \tau \\
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash c : \text{int} & \Gamma \vdash x : \Gamma(x) & \Gamma, x : \tau_1 \vdash e : \tau_2 & \Gamma \vdash \text{e}_1 : \tau_2 \rightarrow \tau_1 & \Gamma, e_1 : \tau_2 \vdash e_2 : \tau_1 \\
\end{array}
\]

2
1. (IMP with choice)

(a) (10 points) Let “?” be a choice operator for IMP expressions: \( e_1 ? e_2 \) chooses either \( e_1 \) or \( e_2 \) and evaluates its choice to produce an answer. Give semantic rules for this extension.

(b) (20 points) Theorem: If \( e_1 \) is equivalent to \( e_2 \), then \( e_1 \) is equivalent to \( e_1 ? e_2 \).

- Restate this theorem formally.
- Prove this theorem formally.

Solution:

(a)

\[
\begin{array}{ll}
\text{LEFT} & \text{RIGHT} \\
\hline
H ; e_1 \Downarrow c & H ; e_2 \Downarrow c \\
\hline
H ; e_1 ? e_2 \Downarrow c & H ; e_1 ? e_2 \Downarrow c \\
\end{array}
\]

(b) For all \( H, e_1, e_2, \) and \( c \), suppose \( H ; e_1 \Downarrow c \) if and only if \( H ; e_2 \Downarrow c \). Then \( H ; e_1 \Downarrow c \) if and only if \( H ; e_1 ? e_2 \Downarrow c \).

We prove the two directions of the if-and-only-if separately. First assume \( H ; e_1 \Downarrow c \). Then we can use LEFT to derive \( H ; e_1 ? e_2 \Downarrow c \). Now assume \( H ; e_1 ? e_2 \Downarrow c \). Then inverting the derivation ensures the derivation ends with either LEFT or RIGHT. If LEFT, then \( H ; e_1 \Downarrow c \) directly. If RIGHT, then \( H ; e_2 \Downarrow c \), but by assumption that means \( H ; e_1 \Downarrow c \).
2. (Bad statement rules)

(a) (10 points) Why do we not have this rule in our IMP statement semantics?

\[
\begin{align*}
H ; s_1 & \rightarrow H' ; s'_1 \\
\hline
H ; s_1 ; (s_2 ; s_3) & \rightarrow H' ; s'_1 ; (s_2 ; s_3)
\end{align*}
\]

(b) (10 points) Why do we not have this rule in our IMP statement semantics?

\[
\begin{align*}
H ; s_1 & \rightarrow H' ; s'_1 \\
\hline
H ; s_2 ; s_1 & \rightarrow H' ; s_2 ; s'_1
\end{align*}
\]

Solution:

(a) It is unnecessary because we can use seq2 to conclude \(H ; s_1 ; (s_2 ; s_3) \rightarrow H' ; s'_1 ; (s_2 ; s_3)\) given \(H ; s_1 \rightarrow H' ; s'_1\) – we just instantiate the \(s_2\) in the rule with \(s_2 ; s_3\).

(b) It is not what we “want” – the purpose of a sequence of statements is to execute the statements in order. This rule would make our language non-deterministic in a way we don’t want because it lets us execute the two parts of a sequence in either order (or in fact we can interleave their execution in any way).
3. (Functional programming)

(a) (10 points) Consider this Caml code:

\[
\text{type t} = \text{A of int} \mid \text{B of (int->int)} \\
\text{let } x = 2 \\
\text{let } f \ y = x + y \\
\text{let } \text{ans1} = (\text{let } x = 3 \text{ in} \\
\text{\quad let } a = \text{A (f 4) in} \\
\text{\quad let } x = 5 \text{ in} \\
\text{\quad match } a \text{ with } \text{A } x \rightarrow x \mid \text{B } x \rightarrow x 6) \\
\text{let } \text{ans2} = (\text{let } x = 3 \text{ in} \\
\text{\quad let } b = \text{B f in} \\
\text{\quad let } x = 5 \text{ in} \\
\text{\quad match } b \text{ with } \text{A } x \rightarrow x \mid \text{B } x \rightarrow x 6)
\]

After evaluating this code, what values are \text{ans1} and \text{ans2} bound to?

(b) (10 points) Consider this Caml code:

\[
\text{let rec } g \ x = \\
\text{\quad match } x \text{ with} \\
\text{\quad \quad [] } \rightarrow \text{ []} \\
\text{\quad \quad | hd::tl } \rightarrow \text{ (fun } y \rightarrow \text{ hd } + y) :: (g \ tl)
\]

i. What does this function do?

ii. What is this function’s type?

iii. Write a function \( h \) that is the inverse of \( g \). That is, \( \text{fun } x \rightarrow h (g x) \) would return a value equivalent to its input.

Solution:

(a) \text{ans1} is bound to 6 and \text{ans2} is bound to 8.

(b) This function takes a list of integers and returns a list of functions where the \( i^{th} \) element in the output list returns the sum of its input and the \( i^{th} \) element of the input list.

(c) \text{int list } \rightarrow ((\text{int } \rightarrow \text{int}) \text{ list})

(d) \text{let rec } h \ x = \text{match } x \text{ with } \text{[]} \rightarrow \text{ []} \mid \text{hd::tl } \rightarrow (\text{hd } 0) :: (h \ tl)
4. (λ encodings) Recall this encoding of booleans in the λ-calculus:
   “true” \( \lambda x. \lambda y. x \)
   “false” \( \lambda x. \lambda y. y \)
   “if” \( \lambda b. \lambda t. \lambda f. b \ t \ f \)
   (a) (10 points) Extend this encoding with a λ term that encodes (inclusive) or.
   (b) (10 points) Extend this encoding with a λ term that encodes not.

Solution:

(a) “or” \( \lambda b_1. \lambda b_2. \ b_1 \ (\lambda x. \lambda y. x)\ b_2 \)
(b) “not” \( \lambda b. \ b \ (\lambda x. \lambda y. y) \ (\lambda x. \lambda y. x) \)
5. (Simply-Typed λ calculus)
For all subproblems, assume the simply-typed λ calculus.

(a) (6 points) Give a $\Gamma$, $e_1$, $e_2$, and $\tau$ such that $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$ and $e_1 \neq e_2$.

(b) (6 points) Give a $\Gamma_1$, $\Gamma_2$, $e$, and $\tau$ such that $\Gamma_1 \vdash e : \tau$ and $\Gamma_2 \vdash e : \tau$ and $\Gamma_1 \neq \Gamma_2$.

(c) (8 points) Give a $\Gamma$, $e$, $\tau_1$, and $\tau_2$ such that $\Gamma \vdash e : \tau_1$ and $\Gamma \vdash e : \tau_2$ and $\tau_1 \neq \tau_2$.

Solution:

(a) $\Gamma = x : \text{int}, y : \text{int}, e_1 = x, e_2 = y, \tau = \text{int}$.

(b) $\Gamma_1 = x : \text{int}, \Gamma_2 = x : \text{int}, y : \text{int}, e = x, \tau = \text{int}$.

(c) $\Gamma = \cdot, e = \lambda x. x, \tau_1 = \text{int} \rightarrow \text{int}, \tau_2 = (\text{int} \rightarrow \text{int}) \rightarrow (\text{int} \rightarrow \text{int})$.
6. (Type-Safety)
We add an explicit infinite-loop to the simply-typed λ-calculus: The term \( \infty \) simply "reduces to itself".

(a) (5 points) Extend the semantics of the call-by-value λ-calculus to include \( \infty \).

(b) (10 points) Extend the type system of the simply-typed λ-calculus to include \( \infty \). Be as permissive as possible considering the next problem.

(c) (15 points) Prove that your extensions maintain type safety. Do not repeat the entire type-safety proof. Rather, for each of these lemmas, remind us the structure of the proof (i.e., the induction hypothesis) and then prove any new cases introduced by your extensions.

- Preservation: If \( \cdot \vdash e : \tau \) and \( e \rightarrow e' \), then \( \cdot \vdash e' : \tau \).
- Progress: If \( \cdot \vdash e : \tau \), then \( e \) is a value or there exists an \( e' \) such that \( e \rightarrow e' \).
- Substitution: If \( \Gamma, x: \tau' \vdash e_1 : \tau \) and \( \Gamma \vdash e_2 : \tau' \), then \( \Gamma \vdash e_1[e_2/x] : \tau \).

Solution:

(a)

\[
\begin{align*}
\text{INF} \\
\infty \rightarrow \infty \\
\infty[e/x] = \infty
\end{align*}
\]

(b)

\[\Gamma \vdash \infty : \tau\]

(c) • Preservation: By induction on the (height of the) derivation that \( e \rightarrow e' \). The new case is that the derivation ends with \( \text{INF} \). But then \( e' \) is \( \infty \) so we can use our new typing rule to conclude \( \cdot \vdash e' : \tau \).

• Progress: By induction on the structure (height) of expressions. The new case is that \( e \) is \( \infty \), in which case we can use \( \text{INF} \) to take a step.

• Substitution: By induction on the typing derivation of \( e_1 \). The new case is \( e_1 = \infty \), in which case \( e_1[e_2/x] = \infty \), so we can use our new typing rule to derive \( \Gamma \vdash e_1[e_2/x] : \tau \).