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.
For your reference:

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

\[
\begin{array}{cccc}
\text{CONST} & \text{VAR} & \text{ADD} & \text{MULT} \\
\hline
H ; c \downarrow c & H ; x \downarrow H(x) & H ; e_1 \downarrow c_1 & H ; e_2 \downarrow c_2 \\
& & H ; e_1 + e_2 \downarrow c_1 + c_2 & H ; e_1 \cdot e_2 \downarrow c_1 \cdot c_2 \\
H_1 ; s_1 \rightarrow H_2 ; s_2 & & & \\
\end{array}
\]

\[
\begin{align*}
\text{ASSIGN} & & \text{SEQ1} & & \text{SEQ2} \\
H ; e \downarrow c & & H ; \text{skip}; s \rightarrow H ; s & & H ; s_1 \rightarrow H' ; s'_1; s_2 \\
& & H ; x := e \rightarrow H, x := c ; \text{skip} & & H ; e_1 \rightarrow H ; s_1 \\
& & & & H ; e_2 \rightarrow H ; s_2 \\
& & & & \text{IF1} \\
& & & & H ; e \downarrow c \quad c > 0 \\
& & & & H ; e_1 \rightarrow H ; s_1 \\
& & & & \text{IF2} \\
& & & & H ; e \downarrow c \quad c \leq 0 \\
& & & & H ; e_2 \rightarrow H ; s_2 \\
\end{align*}
\]

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

\[
\begin{align*}
  e \rightarrow e' \\
  (\lambda x. e) \; v \rightarrow e[v/x] & & e_1 \rightarrow e'_1 & & e_2 \rightarrow e'_2 \\
  e_1 \; e_2 \rightarrow e'_1 \; e_2 & & v \; e_2 \rightarrow v \; e'_2 \\
  e[e'/x] = e'' \\
\end{align*}
\]

\[
\begin{align*}
  x[e/x] = e & & e_1[e/x] = e'_1 & & y \neq x & & y \notin \text{FV}(e) \\
  (\lambda y. e_1)[e/x] = \lambda y. e'_1 & & (\lambda y. e_1)[e/x] = \lambda y. e'_1 \\
  e_1[e/x] = e'_1 & & e_2[e/x] = e'_2 \\
  (e_1 \; e_2)[e/x] = e'_1 \; e'_2 & & (e_1 \; e_2)[e/x] = e'_1 \; e'_2
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash e : \tau \\
\Gamma \vdash \lambda x. e : \tau_1 \rightarrow \tau_2 & & \Gamma \vdash \tau_2 \rightarrow \tau_1 & & \Gamma \vdash e_1 \; e_2 : \tau_1 \\
\Gamma \vdash c : \text{int} & & \Gamma \vdash x : \Gamma(x) & & \Gamma \vdash \lambda x. e : \tau_1 \rightarrow \tau_2 & & \Gamma \vdash e_1 : \tau_2 & & \Gamma \vdash e_2 : \tau_2
\end{align*}
\]
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.
2. (Bad statement rules)

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

\[
\frac{H ; s_1 \rightarrow H' ; s_1'}{H ; s_1 ; (s_2 ; s_3) \rightarrow H' ; s_1' ; (s_2 ; s_3)}
\]

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

\[
\frac{H ; s_1 \rightarrow H' ; s_1'}{H ; s_2 ; s_1 \rightarrow H' ; s_2 ; s_1'}
\]
3. (Functional programming)

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

```caml
type t = A of int | B of (int->int)
let x = 2
let f y = x + y
let ans1 = (let x = 3 in
    let a = A (f 4) in
    let x = 5 in
    match a with A x -> x | B x -> x 6)
let ans2 = (let x = 3 in
    let b = B f in
    let x = 5 in
    match b with A x -> x | B x -> x 6)
```

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

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

```caml
let rec g x =
    match x with
    | [] -> []
    | hd::tl -> (fun y -> 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.
4. (λ encodings) Recall this encoding of booleans in the λ-calculus:

“true” λx. λy. x
“false” λx. λy. y
“if” λb. λt. λf. 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.
5. (Simply-Typed $\lambda$ calculus)
   For all subproblems, assume the simply-typed $\lambda$ calculus.
   
   (a) (6 points) Give a $\Gamma, e_1, e_2, \tau$ such that $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$ and $e_1 \neq e_2$.
   
   (b) (6 points) Give a $\Gamma_1, \Gamma_2, e, \tau$ such that $\Gamma_1 \vdash e : \tau$ and $\Gamma_2 \vdash e : \tau$ and $\Gamma_1 \neq \Gamma_2$.
   
   (c) (8 points) Give a $\Gamma, e, \tau_1, \tau_2$ such that $\Gamma \vdash e : \tau_1$ and $\Gamma \vdash e : \tau_2$ and $\tau_1 \neq \tau_2$. 

6. (Type-Safety)

We add an explicit infinite-loop to the simply-typed λ-calculus: The term ∞ simply “reduces to itself”.

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

(b) (10 points) Extend the type system of the simply-typed λ-calculus to include ∞. 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 ⊢ e : τ and e → e', then ⊢ e' : τ.
- Progress: If ⊢ e : τ, then e is a value or there exists an e' such that e → e'.
- Substitution: If Γ, x:τ' ⊢ e_1 : τ and Γ ⊢ e_2 : τ', then Γ ⊢ e_1[e_2/x] : τ.