Name: ________________________________

CSE 505, Fall 2009, Final Examination
14 December 2009

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 12:20.
• You can rip apart the pages.
• There are 100 points total, distributed unevenly among 7 questions.
• The questions 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, make sure you get to all the problems.
• If you have questions, ask.
• Relax. You are here to learn.
For your reference (page 1 of 2):

Name: __________________________

For your reference (page 2 of 2):

$e ::= \lambda x. e \mid x \mid \text{int} \mid e \mid e \mid \Delta; \Gamma \mid e.l \mid \text{fix } e$

$\tau ::= \text{int} \mid \tau \to \tau \mid \{l_1 : \tau_1, \ldots, l_n : \tau_n\}$

$\vdash e \to e'$ and $\Gamma \vdash e : \tau$ and $\tau_1 \leq \tau_2$

$\Gamma \vdash e : \tau_1 \ldots \Gamma \vdash e : \tau_n \quad \text{labels distinct}$

$\Gamma \vdash \{l_1 = \tau_1, \ldots, l_n = \tau_n\} \leq \{l_1 = \tau_1, \ldots, l_n = \tau_n\}$
\[
\begin{align*}
    e & ::= \ldots | A(e) | B(e) | \text{match } e \text{ with } A x \ | B x \ | \text{roll}_\tau e | \text{unroll } e | (e, e) | e.1 | e.2 \\
    \tau & ::= \ldots | \tau_1 + \tau_2 \ | \mu \alpha.\tau | \tau_1 * \tau_2 \\
    v & ::= \ldots | A(v) | B(v) | \text{roll}_\tau v | (v, v)
\end{align*}
\]

\[
\begin{align*}
    e & \rightarrow e' \text{ and } \Delta; \Gamma \vdash e : \tau \\
    e & \rightarrow e' \quad e & \rightarrow e' \\
    A(e) & \rightarrow A(e') \quad B(e) & \rightarrow B(e') \\
    \text{match } e \text{ with } A x & \quad \text{match } e \text{ with } B y \\
    \text{match } A(v) & \text{ with } A x \ e_1 \quad \text{match } B(v) & \text{ with } A x \ e_2 \\
    \text{match } e & \text{ with } A x \ e_1 \quad \text{match } e & \text{ with } B y \ e_2 \\
    \text{match } e & \text{ with } A x \ e_1 \quad \text{match } e & \text{ with } B y \ e_2 \\
    \text{match } e & \text{ with } A x \ e_1 \quad \text{match } e & \text{ with } B y \ e_2 \\
    \text{match } e & \text{ with } A x \ e_1 \quad \text{match } e & \text{ with } B y \ e_2 \\
    \text{match } e & \text{ with } A x \ e_1 \quad \text{match } e & \text{ with } B y \ e_2 \\
    \text{match } e & \text{ with } A x \ e_1 \quad \text{match } e & \text{ with } B y \ e_2 \\
\end{align*}
\]

Module Thread:

\[
\begin{align*}
    \text{type } t \\
    \text{val create : ('a -> 'b) -> 'a -> t} \\
    \text{val join : t -> unit}
\end{align*}
\]

Module Mutex:

\[
\begin{align*}
    \text{type } t \\
    \text{val create : unit -> t} \\
    \text{val lock : t -> unit} \\
    \text{val unlock : t -> unit}
\end{align*}
\]

Module Event:

\[
\begin{align*}
    \text{type 'a channel} \\
    \text{type 'a event} \\
    \text{val new_channel : unit -> 'a channel} \\
    \text{val send : 'a channel -> 'a -> unit event} \\
    \text{val receive : 'a channel -> 'a event} \\
    \text{val choose : 'a event list -> 'a event} \\
    \text{val wrap : 'a event -> ('a -> 'b) -> 'b event} \\
    \text{val sync : 'a event -> 'a}
\end{align*}
\]

1. (15 points) Consider a typed-lambda calculus with functions, integers, records, and subtyping as considered in class. Note this problem considers only the subtyping judgment with the six inference rules on page 2 of this exam, not the typing judgment. For each of the following claims, if it is true, prove it. If it is false, provide a counterexample.

(a) If $\tau_1 \leq \tau_2$ and $\tau_1$ is a record type, then $\tau_2$ is a record type.

(b) If $\tau_1 \leq \tau_2$ and $\tau_1$ contains a function type somewhere in it, then $\tau_2$ contains a function type somewhere in it.
2. (10 points) Consider System F.

(a) Write a well-typed term that implements function composition and is as polymorphic as possible. Function composition takes two (curried) arguments (say \( f \) and \( g \)) and returns a function that given \( x \) returns \( f(g(x)) \).

(b) Give the type for the term you wrote in part (a).

Be sure to use parentheses appropriately.
3. (10 points) For each of the following Caml definitions, does it type-check in Caml? If so, what type does it have? If not, why not?

(a) let part_a = (fun g -> (fun x y -> x) (g 0) (g 17))
(b) let part_b = (fun g -> (fun x y -> x) (g 0) (g true))
(c) let part_c = (fun g -> (fun x y -> x) (g 0) (g (g 17)))
4. (15 points) Consider a typed λ-calculus with recursive types, sums, pairs, int, string, and unit. Assume the language uses explicit roll and unroll coercions (not subtyping) for recursive types.

(a) Give a recursive type for binary trees where interior nodes have no data and each leaf has either a string or an int.

(b) In this part, you can use tr as an abbreviation for the type you gave in part (a). Using fix for recursion, write a program of type \( tr \rightarrow (\text{int} + \text{unit}) \). When called with a tree, the program should return \( A(i) \) if \( i \) is the left-most integer in the tree and \( B(()) \) if the tree has no integers. Give explicit types to all function arguments. If you get confused by fix, use let rec instead for significant partial credit.
5. (20 points) In this problem you will use CML to implement a server for “rock-paper-scissors”. Rock-paper-scissors is a game where normally there are two players who each pick “rock”, “paper”, or “scissors” and either one player wins or there is a tie. This Caml code defines the rules for this two-player game:

```caml
let pick_winner p1 p2 = (* useful helper function *)
  match (p1,p2) with
  (Rock,Rock) -> Tie
  | (Scissors,Scissors) -> Tie
  | (Paper,Paper) -> Tie
  | (Rock,Scissors) -> Left
  | (Rock,Paper) -> Right
  | (Scissors,Paper) -> Left
  | (Scissors,Rock) -> Right
  | (Paper,Rock) -> Left
  | (Paper,Scissors) -> Right
```

You will implement the `new_game` function in this interface:

```caml
val new_game : unit -> game
val play_game : game -> play -> result
```

`new_game` creates a new server. Players can play by calling `play_game`. There are two differences from the two-player version:

- Players do not know their opponent. The server can choose any opponent. A player simply learns whether he/she won or lost.
- There are no ties. The server must match up players so that each player wins or loses. For example, the server can pair a “rock” with a “scissors” or a “paper” but not with another “rock.”

To avoid ties, the server may need to make players wait for other players to arrive. However, to avoid any unnecessary waiting, all the zero-or-more waiting players at any one time will have picked the same play — otherwise the server should have paired up two players that picked differently. So, when a new player arrives, if there are no waiters or the waiters “tie” with the new player, then the new player waits. Otherwise, the new player and one waiter complete.

Do not change the partial implementation below; just complete `new_game`. You do not need `choose` and `wrap`. The sample solution is 15–20 lines. Use `pick_winner`, defined above. Remember that when players are paired up the winner and the loser need to be informed.

```caml
let new_game () = (* for you *)
let play_game g p =
  let c = new_channel () in
  sync (send g (p,c));
  sync (receive c)
```
Name:__________________________________________

This page intentionally blank.
6. (15 points) Consider a class-based OOP language like we did in class where method-name reuse means overriding. Consider this code skeleton:

```java
class A {
    A m() { return self; } ... }
class B extends A {
    B m() { return super(); } ... }
class C extends B {
    A m() { return new A(); } ... }

class Main { void main() { ... } }
```

(a) Assuming all code not shown (i.e., the code in the ... ) type-checks, there are two reasons the code above does not type-check. What are they?

(b) One of your two answers to part (a) cannot actually lead to a program getting stuck. Explain why not.

(c) The other of your two answers to part (a) can lead to a program getting stuck. Fill in the ... as necessary (you may not need to use all of them) such that all the code you add type-checks but running the `main` method would produce a “method not found” error.
7. (15 points) Consider a single-inheritance class-based OOP language. Assume booleans are provided as primitives.

Assume method-name reuse means either static overloading or multimethods. For each part give a single answer that is correct under either assumption. This is not intended to make the problem harder.

(a) Write a program (class definitions and client code) such that:
   • The program type-checks.
   • The program evaluates to true.
   • There is one method definition you can remove from the program (comment-out) such that the program still type-checks but the program now evaluates to false.

   Clearly indicate which method should be commented out.

(b) Write a program (class definitions and client code) such that:
   • The program type-checks.
   • The program evaluates to true.
   • There is one method definition you can remove from the program (comment-out) such that the program would now have a “no best match” error.

   Clearly indicate which method should be commented out.