CSE 130, Fall 2006: Final Examination

Name: ________________________________

ID: ________________________________

Instructions, etc.

1. Write your answers in the space provided.

2. Wherever it says explain, write no more than three lines as explanation. The rest will be ignored.

3. The points for each problem are a rough indicator (when converted to minutes), of how long you should take for the problem.

4. Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>135</td>
</tr>
</tbody>
</table>
1. **[15 Points]** For each of the following Ocaml programs, if the code is well-typed, write down the value of \texttt{ans}, otherwise, if the code has a type problem, write “type error”.

(a) \begin{verbatim}
let ans =
  let x = 10 in
  let f y =
    let a = x + 1 in
    let b = y + a in
    a + b in
  f 100
\end{verbatim}

(b) \begin{verbatim}
let ans =
  let f n = 10 in
  let f n = if n > 0 then n + (f (n-1)) else 0 in
  f 5
\end{verbatim}

(c) \begin{verbatim}
let ans =
  let f g x = g (g x) in
  let h0 = fun x -> x * x in
  let h1 = f h0 in
  let h2 = f h1 in
  h2 2
\end{verbatim}

2. **[15 Points]** For each of the following Ocaml programs, write down the type of \texttt{ans}.

(a) \begin{verbatim}
let ans =
  let f f = f + 1 in
  f
\end{verbatim}

(b) \begin{verbatim}
let ans f g x =
  if x > 0 then f x else g x
\end{verbatim}

(c) \begin{verbatim}
let ans l =
  match l with
\end{verbatim}
[] → []
| ⟨hx,hy⟩::t → ⟨hx hy⟩::(ans t)
3. Consider the Ocaml module described below:

```ocaml
module Set : SETSIG =
  struct
    exception Duplicates
    type 'a set = 'a list
    let new x = [x]
    let rec mem s x =
      match s with
      | [] -> false
      | h::t -> if x <> h then mem t x
      else if mem t x then raise Duplicates
      else true
    let add s x =
      if mem s x then s else (x::s)
    let union s1 s2 =
      match s1 with
      | [] -> s2
      | h::t -> union t (add s2 h)
    let choose s =
      match s with
      | [] -> None
      | h::t -> Some (h,t)
  end

and the two possible signatures:

(A) module type SETSIG = struct
  sig
    type 'a set = 'a list
    val new : 'a -> 'a set
    val mem : 'a set -> 'a -> bool
    val add : 'a set -> 'a -> 'a set
    val choose : 'a set -> ('a * 'a set) option
    val union : 'a set -> 'a set -> 'a set
  end

(B) module type SETSIG = struct
  sig
    type 'a set
    val new : 'a set
    val mem : 'a set -> 'a -> bool
    val add : 'a set -> 'a -> 'a set
    val choose : 'a set -> ('a * 'a set) option
    val union : 'a set -> 'a set -> 'a set
  end

(a) [5 Points] For which one of the signatures (A) or (B), can a client cause the exception Duplicates to get raised? Write down a client expression that would cause this exception to get raised. For the other signature explain why the exception will never get raised.

Signature:

Client Expression:

Explanation:
```
(b) [5 Points] Recall the `filter` function described in class:

```ocaml
let rec filter f l =  
  match l with  
  | [] -> []  
  | h::t -> if f h then h::(filter f t) else filter f t
```

Consider the `client` function:

```ocaml
let intersection s1 s2 =  
  filter (mem s2) s1
```

For one of the signatures (A) or (B), the client function `intersection` compiles, i.e. is well typed. Which one? What is the inferred type of `intersection` using this signature?

**Signature:**

**Inferred Type:** `intersection : ___________

(c) [10 Points] Write an equivalent version of `intersection` that would compile with both signatures.
4. Consider the following Ocaml datatype used to represent trees.

\[
\text{type } 'a \text{ tree } = \text{Leaf of } 'a \mid \text{Node } 'a \text{ tree } \times 'a \text{ tree}
\]

(a) [5 Points] Write the value of type \text{int tree} that corresponds to the following pictorial representation of a tree.

```
          .
         / \   \\
        /   \\
       .     \\
     / \   / \\
    /   /   \\
   1  2  3  4
```

(b) [5 Points] Consider the following function:

\[
\text{let rec tf f b t} =
\text{match t with}
\text{  Leaf } x \rightarrow f (b,x)
| \text{Node } (t1,t2) \rightarrow tf f (tf f b t1) t2
\]

What is the type of the function \text{tf}? Answer this by filling in the blanks:

------------ -> -------------- -> -------------- -> --------------

(c) [5 Points] Fill in the blanks below to obtain an implementation of:

\text{to_list : 'a tree } \rightarrow 'a \text{ list}

that returns the list of values occurring as leaves of the tree.

\[
\text{let to_list t} =
\text{let f ____ = ________________________ in}
\text{let b = ___________________________ in}
\text{tf f b t}
\]

(d) [5 Points] Fill in the blanks below to obtain an implementation of:

\text{size : 'a tree } \rightarrow \text{ int}

that returns the list of values occurring as leaves of the tree.

\[
\text{let size t} =
\text{let f ____ = ________________________ in}
\text{let b = ___________________________ in}
\]
(e) [5 Points] Write a tail-recursive version of tf. **Hint:** This is difficult. You may need a helper function.
5. For each of the following Scala programs, write down the value of ans, or write error together with an explanation, if an error occurs. Write your answers on the blank space on the right.

(a) [5 Points]
val x = Array(1,2,3)
val y = Array("a","b","c")
def f(y: Any) {
  val x = y
}
val _ = f(x)
val ans = x(0)

(b) [5 Points]
def f(x: Int) : Int => Int = {
def g(y: Int) = {
a(x+y)
}
g
}
val a: Int => Int = f(10)
val ans = a(0)

(c) [8 Points]
val a = Array(0)
def f(x: Int) = {
def g = (y: Int) => { a(0) += (x + y)
a(0) }
g}
val foo = f(10)
val _ = foo(1000)
val ans = (a(0), foo(1))
(d) [12 Points]

```scala
trait A {
    var x : List[String] = List()
    def d(): Unit
    def a() {
        x = x ++ List("a")
        d()
    }
}

trait B extends A {
    def b() {
        x = x ++ List("b")
    }
}

trait C extends A {
    def c() {
        x = x ++ List("c")
    }
}

class D extends B with C {
    def d() {
        x = x ++ List("d")
        b()
        c()
    }
}

val o = new D
val _ = o.a
val ans = o.x
```

(e) [5 Points]

```scala
def foo(n: Int) = {
    var xs : List[Int] = List()
    var i = 1
    while (i <= n) {
        xs = i :: xs
        i += i
    }
    xs
}

var ans = 0
for (i <- foo(10)) { ans += i }
```
6. (a) [5 Points]
Use `yield` to write a function

```scala
def elementAndRest[A](xs: List[A]): Iterator[(A, List[A])]
```
that takes a list as input and returns an iterator over tuples which consist of an element of the list, and the list with that element removed.
The elements of the list should be in the same order as in the original list. The function `elementAndRest` should not return a list. When you are done, the following:

```scala
scala> for (t <- elementAndRest(List(1,2,3,4,5))) println(t)
```

```scala
(1,List(2, 3, 4, 5))
(2,List(1, 3, 4, 5))
(3,List(1, 2, 4, 5))
(4,List(1, 2, 3, 5))
(5,List(1, 2, 3, 4))
```
Write the function by filling in the 3 blanks below with suitable expressions.

```scala
def elementAndRest[A](xs: List[A]): Iterator[(A, List[A])] = {
  val iter = ___________________________________________________________
  for (i <- iter
    ; x = ______________________________________________________________
    ; y = ______________________________________________________________
  ) yield (x, y)
}
```

**Hint:**

`List(0, 10, 20, 30, 40, 50, 60, 70).slice(2, 4) == List(20, 30)`

(b) [10 Points] Write a function `permutations` which takes a list as input and returns an iterator over permutations of the given list. The function should not compute all permutations before returning. When you are done, you should get the following behavior:

```scala
scala> for (p <- permutations(List(1,2,3))) println(p)
```

```scala
List(1,2,3)
List(1,3,2)
List(2,1,3)
List(2,3,1)
List(3,1,2)
List(3,2,1)
```
The body of the function should be at most 5 lines long. Write it by filling in the blanks below:
def permutations[A](xs: List[A]): Iterator[List[A]] =
xs match {
  case Nil => ___________________________________________________
  case x::rest => ___________________________________________________
  _____________________________________________________
  _____________________________________________________
  _____________________________________________________
}

7. Recall that $P <: Q$ if $P$ is a structural subtype of $Q$. Assume that everything is a subtype of Any.

```scala
trait A {
  val a: Any
}

trait B {
  val a: Int
  val b: Int
}

trait C {
  def f(x: B): A
}

trait D {
  def f (x: /* IN */ ____): /* OUT */ ____
}
```

(a) [2 Points] True or False: $A <: B$ ?
(b) [2 Points] True or False: $B <: A$ ?
(c) [6 Points] Write four possible ways of filling in the blanks in the definition of D (i.e. of completing the type of f) such that $D <: C$.

i. /* IN */ _____________ , /* OUT */ _____________
ii. /* IN */ _____________ , /* OUT */ _____________
iii. /* IN */ _____________ , /* OUT */ _____________
iv. /* IN */ _____________ , /* OUT */ _____________

8. [5 Points] Consider the following C-like code.

```c
int y = 1;
void f(int x){
  int y;
  y = x + 1;
  x = x + 10;
  g(x);
  printf("x = %d \n",x);
}
void g(int x){
  y = x + 1;
}
void main(){
  f(y);
  printf("y = %d \n",y)
}
```

What is the output of executing this code under
(a) *static scoping*?

(b) *dynamic scoping*?
9. Consider the following Prolog code:

\[
\begin{align*}
\text{actor}(xmen,jackman). \\
\text{actor}(xmen,berry). \\
\text{actor}(scoop,jackman). \\
\text{actor}(scoop,johanssen). \\
\text{actor}(\text{lost\_in\_translation},murray). \\
\text{actor}(\text{lost\_in\_translation},johanssen). \\
\text{actor}(\text{ghostbusters},murray). \\
\text{actor}(\text{ghostbusters},akroyd). \\
\text{actor}(\text{batmanreturns},bale). \\
\text{actor}(\text{batmanreturns},caine). \\
\text{actor}(\text{dirtyrottenscoundrels},martin). \\
\text{actor}(\text{dirtyrottenscoundrels},caine). \\
\text{actor}(\text{shopgirl},danes). \\
\text{actor}(\text{shopgirl},martin).
\end{align*}
\]

(a) [2 Points] Write a predicate \texttt{costar}(X,Y) that is true when \textit{X,Y} have acted in the same movie.

(b) [3 Points] Write a predicate \texttt{busy}(X) that is true when \textit{X} has acted in more than one movie.

(c) [5 Points] Write a predicate \texttt{bacon}(X,Y) that is true when there is a sequence of actors \textit{Z}_1, \textit{Z}_2, \ldots, \textit{Z}_n such that for each \textit{i}, the pair \textit{Z}_i, \textit{Z}_{i+1} have acted in the same movie, and \textit{X} is \textit{Z}_1 and \textit{Y} is \textit{Z}_n.
10. For this problem, you will write Prolog code to implement the magic algorithm whereby ML is able to infer the types of all expressions. First, we shall encode (nano) ML expressions as Prolog terms via the following grammar:

\[
expr ::= \text{const}(i) \\
| \text{var}(x) \\
| \text{plus}(expr, expr) \\
| \text{leq}(expr, expr) \\
| \text{ite}(expr, expr) \\
| \text{letin}(x, expr, expr) \\
| \text{fun}(\text{var}(x), expr) \\
| \text{app}(expr, expr)
\]

Similarly, we shall encode ML types as Prolog terms using the following grammar:

\[
type ::= \text{int} | \text{bool} | \text{arrow}\text{(type, type)}
\]

The table below shows several examples of Ocaml expressions, the Prolog term encoding that expression, and the Prolog term encoding the type of the expression.

<table>
<thead>
<tr>
<th>ML Expression</th>
<th>Prolog Expression Term</th>
<th>Prolog Type Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>const(2)</td>
<td>int</td>
</tr>
<tr>
<td>x</td>
<td>var(x)</td>
<td></td>
</tr>
<tr>
<td>2 + 3</td>
<td>plus(const(2),const(3))</td>
<td>int</td>
</tr>
<tr>
<td>2 &lt;= 3</td>
<td>leq(const(2),const(3))</td>
<td>bool</td>
</tr>
<tr>
<td>fun x -&gt; x &lt;= 4</td>
<td>fun(var(x),leq(var(x),const(4)))</td>
<td>arrow(int,bool)</td>
</tr>
</tbody>
</table>
| fun x y ->      | fun(var(x),fun(var(y), 
| if x then y else 0) | ite(var(x),var(y),const(0))) | arrow(bool,arrow(int,int)) |
| let x = 10 in x | letin(var(x),const(10),var(x)) | int             |
| fun x ->        | fun(var(x),              
| let y = x in y + y | letin(var(y),var(x) 
|                      | plus(var(y),var(y))) | arrow(int,int) |

(a) [5 Points] Write a Prolog predicate \text{envtype}(\text{Env},X,T), such that \text{envtype}([[\text{x1},\text{t1}], [\text{x2},\text{t2}], \ldots, [\text{xn},\text{tn}]], X, T) is true if X equals the first term \text{xi} corresponding to variable \text{xi} and T equals the corresponding \text{ti} corresponding to the type of the variable \text{xi} in the type environment \text{ti}. When you are done, you should get the following behavior:

?- envtype([[x,int],[y,bool]],x,T).
T = int
Yes

?- envtype([[x,int],[x,bool]],x,T).
T = int
Yes

?- envtype([[x,int],[x,bool]],x,bool).
No
(b) [20 Points] Write a Prolog predicate `typeof(Env,E,T)` that is true when the term T is the correct ML type of the ML expression corresponding the term E in the type environment corresponding to the list Env. Write your solution by filling in the grid below:

| typeof(Env,const(I),T) :- |
| typeof(Env,var(X),T) :- |
| typeof(Env,plus(E1,E2),T) :- |
| typeof(Env,leq(E1,E2),T) :- |
| typeof(Env,ite(E1,E2,E3),T) :- |
| typeof(Env,letin(var(X),E1,E2),T) :- |
| typeof(Env,fun(var(X),E),T) :- |
| typeof(Env,app(E1,E2),T) :- |

When you are done, you should get the following output:

?- typeof([[x,int],[y,bool]],Var(x),T).
   T = int
   Yes

?- typeof([],plus(const(2),const(3)),T).
   T = int
   Yes

?- typeof([],leq(const(2),const(3)),T).
   T = bool
   Yes

?- typeof([],fun(var(x),leq(var(x),const(4))),T).
   T = arrow(int,bool)
   Yes

?- typeof([],fun(var(x),fun(var(y),ite(var(x),var(y),const(0)))),T).
   T = arrow(bool,arrow(int,int))
   Yes

?- typeof([],letin(var(x),const(10),var(x)),T).
   T = int
   Yes

?- typeof([],fun(var(x),letin(var(y),var(x),plus(var(y),var(y)))),T).
   T = arrow(int, int)
   Yes

?- typeof([],app(fun(var(x),plus(var(x),const(1))),const(19)),T).
   T = int
   Yes

(c) [5 Points] Does your predicate infer polymorphic types? In other words, using your implementation of `typeof` will the result of the following query be Yes or No? Explain.

?- typeof([],letin(var(id),fun(var(x),var(x)),
   letin(var(y),app(var(id),leq(const(2),const(3)))),
   int
   Yes
app(var(id),const(1))),T).

(d) [[Extra Credit] Points] Extend your solution so that the the above query succeeds. type inference is polymorphic. That is, it should successfully find an appropriate solution for T for the query above.