Midterm #2

CMSC 330: Organization of Programming Languages

April 16, 2013

Name

Instructions

Do not start until told to do so!

• This exam has 12 pages (including this one); make sure you have them all
• You have 75 minutes to complete the exam
• The exam is worth 100 points. Allocate your time wisely: some hard questions are worth only a few points, and some easy questions are worth a lot of points.
• If you have a question, please raise your hand and wait for the instructor.
• You may use the back of the exam sheets if you need extra space.
• In order to be eligible for partial credit, show all of your work and clearly indicate your answers.
• Write neatly. Credit cannot be given for illegible answers.

<table>
<thead>
<tr>
<th></th>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Programming language concepts</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OCaml types</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Pattern matching &amp; polymorphism</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>User-defined types &amp; higher-order functions</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Parameter passing &amp; scoping</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Operational semantics</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Multithreading</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>
1. (Programming language concepts, 15 points total)

(a) (2 points) What are the two components of a closure?

Answer:

*The function code and the environment*

(b) (2 points) Name one key difference between lists and tuples in OCaml

Answer:

Either of: (1) Lists must be homogeneous—all elements have the same type—but tuples may be heterogeneous—each element may have a different. (2) Lists may be variable-length, whereas tuples are fixed length.

(c) (4 points) What is the output of the following OCaml execution?

```ocaml
let f x =  
  let c = ref x in 
  ((fun () -> !c), 
   (fun () -> c := !c + 1)) 
;; 
let (g,h) = f 3;; 
h ();; 
h ();; 
h ();; 
print_int (g ());;
```

Answer:

6

(d) (3 points) What is the difference between a non-reentrant lock and a reentrant lock?

Answer:

*The latter can be reacquired by the same thread, while the former cannot.*

(e) (3 points) Consider the following two functions:

```ocaml
let rec pow' (x,y) =  
  if y = 0 then 1 
  else x * (pow' (x,y-1)) 
let rec pow x y =  
  if y = 0 then 1 
  else x * (pow x (y-1))
```

What is the technical term that characterizes how the second function is different from the first?

Answer:

*Currying*

(f) (1 points) OCaml has type inference. Does Ruby?

Answer:

*No.*
2. (OCaml types, 15 points total: 2 points for a–f, 3 points for g)

What are the types of the following functions?

(a) let f (x,y) = x+y
   Answer: int*int -> int

(b) let f x = x::[]
   Answer: 'a -> 'a list

(c) let f x y z = z
   Answer: 'a -> 'b -> 'c -> 'c

(d) let f x y z = if x then y else z
   Answer: bool -> 'a -> 'a -> 'a

(e) let unit x = Some x
(Recall: 'a option = None | Some of 'a.)
   Answer: 'a -> 'a option

(f) let bind x f =
    match x with
    None -> None
    | Some a -> f a
   Answer: 'a option -> ('a -> 'b option) -> 'b option

(g) let rec f x y = (* worth 3 points *)
    match y with
    [] -> []
    | h::t -> (x h) :: (f x t)
   Answer: ('a -> 'b) -> 'a list -> 'b list
3. Coding: pattern matching, polymorphism (10 points total)

(a) (4 points) The following function (pairup : 'a list -> 'a list list) takes a list of elements and converts it to a list of pairs of elements. E.g., pairup [1;2;3;4] returns [[1;2]; [3;4]]. Modify this code so that when the input list has an odd number of elements, the last element of the output list is a singleton list; i.e., pairup [1;2;3] should produce [[1;2];[3]].

``` Ocaml
let rec pairup' l =  
    match l with  
    | h1::h2::t -> (h1::[h2])::(pairup' t)  
    | [] -> []
```

Answer:

``` Ocaml
let rec pairup l =  
    match l with  
    | h1::h2::t -> (h1::[h2])::(pairup t)  
    | [x] -> [[x]] (* ADDED CASE *)  
    | [] -> []
```

(b) (3 points) This function has a type error in it. Point out or correct the type error.

``` Ocaml
let rec mystery x =  
    match x with  
    | h::t -> (mystery t)::h  
    | [] -> []
```

Answer:

``` Ocaml
let rec mystery x =  
    match x with  
    | h::t -> (mystery t)@[h]  
    | [] -> []
```

(c) (3 points) Write a function app that has the following type:

``` Ocaml
('a -> 'b) -> 'a -> 'b
```

Answer:

``` Ocaml
let app f x = f x
```
4. Coding: user-defined types and higher order programming (16 points total)

Consider the following implementation of a binary search tree and a higher-order function called foldi.

```ocaml
type 'a bst =  
  Leaf  
| Node of 'a * 'a bst * 'a bst

let rec insert (x:'a) (t:'a bst) : 'a bst =  
  match t with  
    Leaf -> Node (x, Leaf, Leaf)  
  | Node (y,left,right) ->  
    if x < y then Node(y,insert x left,right)  
    else if x > y then Node(y,left,insert x right)  
    else t

let rec foldi (f:'a -> 'b -> 'b) (acc:'b) (t:'a bst) : 'b =  
  match t with  
    Leaf -> acc  
  | Node (x,left,right) ->  
    let accleft = foldi f acc left in  
    let acc0 = f x accleft in  
    foldi f acc0 right
```

As an example of the use of these functions, note that the following code will print "125" to the console:

```ocaml
let t = insert 1 (insert 5 (insert 2 Leaf));;  (* Variable t above will be Node (2, Node (1, Leaf, Leaf), Node (5, Leaf, Leaf)) *)
foldi (fun x y -> print_int x) () t;;
```

Questions on the next page.
(a) (5 points) Write a function (tolist: 'a bst -> 'a list) that returns the elements as a sorted list. Implement your function using foldi.

Answer:

```ocaml
let tolist (t:'a bst) : 'a list =
foldi (fun x y -> y @ [x]) [] t;;
```

(b) (5 points) Write a function (mapi: 'a bst -> ('a -> 'b) -> 'b bst). Implement your function using foldi.

Answer:

```ocaml
let mapi (f:'a -> 'b) (t:'a bst) : 'b bst =
foldi (fun x u -> insert (f x) u) Leaf t
```

(c) (6 points) Write function (max: 'a bst -> 'a) that returns the largest element in your tree. Your function should throw an exception if the tree is empty. Note that the comparison operators in OCaml are polymorphic, e.g., "greater than" has type 'a -> 'a -> 'a. You may or may not use foldi in your implementation. For one point extra credit, make it so your function does not have to look at every element of the tree, to find the maximum.

Answer:

```ocaml
let rec max (t:'a bst) : 'a =
match t with
  | Leaf -> failwith "empty tree"
  | Node (y, _, Leaf) -> y
  | Node (y, _, right) -> max right

(* Slower answer *)
let max' (t:'a bst) : 'a =
match t with
  | Leaf -> failwith "empty tree"
  | Node (y, _,) ->
    foldi (fun x y -> if x > y then x else y) y t
```
5. Parameter passing (10 points total)

Consider the following function:

```ocaml
let mytry e h =  
  match e with  
    Some x -> x  
  | None -> h  
;;
```

This function approximates exception handling. Consider its use in the following code:

```ocaml
let div x y =  
  mytry  
    (if y = 0 then None else Some (x / y))  
    (print_string "oops!\n"; 0)  
;;
```

Here, the function div will divide the first argument by the second, unless the second argument is zero, in which case it returns prints "oops!" to the console and returns 0.

(a) (3 points) Recall that OCaml is call-by-value. As such, when I run the following, what output will I get?

```ocaml
print_int (div 2 1);;
```

**Answer:**

```
  oops!  
  2
```

(b) (2 points) If OCaml were instead call-by-name, what output I get from running the same thing?

**Answer:**

```
  2
```
Consider the following code:

```ocaml
let x = 6;;
let f y = x + y;;
let x = 5;;
print_int (f 6);;
```

(c) (3 points) What does this program print when static scoping is used?

**Answer:**

12

(d) (2 points) What does the program print when dynamic scoping is used?

**Answer:**

11
6. Operational semantics (16 points total)

Consider the following programming language, and operational semantics having the form \( A; E \Rightarrow V \).

\[
E ::= X \mid N \mid E+E \mid \text{if } E \text{ then } E \text{ else } E \mid \text{let } X = E \text{ in } E
\]

\[
A ::= . \mid A,X=N
\]

In this grammar, letters in caps are non-terminals, and those in lowercase are terminals. \( X \) represents variables, and \( N \) represents integers. \( A \) is a list pairing variables with their bound values, and "." if there are no bindings.

The operational semantics for this language are as follows:

\[
\begin{align*}
A; N & \Rightarrow N \\
A(X) = N & \quad A; X \Rightarrow N \\
A; E_1 \Rightarrow N_1 & \quad A; E_2 \Rightarrow N_2 & \quad N = N_1+N_2 & \quad A; E_1+E_2 \Rightarrow N \\
A; E \Rightarrow N & \quad N \neq 0 & \quad A; E_1 \Rightarrow N & \quad A; \text{if } E \text{ then } E_1 \text{ else } E_2 \Rightarrow N \\
A; E \Rightarrow 0 & \quad A; E_2 \Rightarrow N & \quad A; \text{if } E \text{ then } E_1 \text{ else } E_2 \Rightarrow N \\
A; E_1 \Rightarrow N_1 & \quad A,X=N_1; E_2 \Rightarrow N & \quad A; \text{let } X = E_1 \text{ in } E_2 \Rightarrow N
\end{align*}
\]

Questions on the following page.
Indicate whether the following are legal derivations of executions, according to these rules. If they are not, circle the part of the derivation that is incorrect.

(a) (4 points) Let $A = \cdot$ in the following:

\[
\frac{\_}{A; \color{red} (3 \Rightarrow 2)}
\]

(b) (4 points) Let $A = \cdot$, $X = 2$ in the following:

\[
\frac{A; X \Rightarrow 2}{A; \text{let } Y = X \text{ in } Y \Rightarrow 2}
\]

(c) (4 points) Let $A = \cdot$ in the following:

\[
\frac{\_}{A; X \Rightarrow 2} \quad \frac{A(Y) = 2}{A; \text{let } Y = X \text{ in } Y \Rightarrow 2}
\]

(d) (4 points) Let $A = \cdot$ in the following:

\[
\frac{\_}{A; 0 \Rightarrow 0} \quad \frac{A; 2 \Rightarrow 2}{(A,X=2)(X) = 2} \quad \frac{A; X=2; X \Rightarrow 2}{A; \text{let } X = 2 \text{ in } X \Rightarrow 2}
\]

\[
\frac{\_}{A; \text{if } 0 \text{ then let } X = 2 \text{ in } X \text{ else } 2 \Rightarrow 2}
\]
7. (Multithreading, 18 points total)

(a) (4 points total, 1 point each) Indicate whether the following are true ("T") or false ("F"):

____ No synchronization is necessary to protect immutable objects from data races.
____ Calling wait() on some object requires that thread to hold the lock on that object.
____ Threads within the same process are typically run on separate processors.
____ Threads within the same process share memory.

Answer:
(a) T (b) T (c) F (d) T

(b) (6 points total) Consider the following (possibly broken) implementation of producer/consumer:

```java
void produce(Object o) {
    lock.lock();
    while(valueReady) {
        lock.unlock();
        lock.lock();
    }
    value = o;
    valueReady = true;
    lock.unlock();
}

Object consume() {
    lock.lock();
    while(!valueReady) {
        lock.unlock();
        lock.lock();
    }
    Object o = value;
    valueReady = false;
    lock.unlock();
}
```

(3 points each) Identify each as true ("T") or false ("F").

____ The code will potentially waste time doing useless work.
____ The code may exhibit a race condition.

Answer:
(T,F)

Continued on the next page.
Consider the following code running in two separate threads:

Thread A: A1: lock(lock1)  Thread B: B1: lock(lock2)
   A2: lock(lock2)          B2: lock(lock1)
   A3: (A’s critical segment)  B3: (B’s critical segment)
   A4: unlock(lock1)         B4: unlock(lock1)
   A5: unlock(lock2)         B5: unlock(lock2)

(c) (4 points) Describe what happens when this code is run with a round-robin scheduler (where each thread executes one statement before the next thread is run), and where initially thread A is at A1 and thread B is at B1.

Answer:
   Deadlocks

(d) (4 points) Describe how you would fix the code.

Answer:
   Lock in the same order (swap A1/A2 or B1/B2).