CS421 Fall 2009 Midterm 1

Thursday, October 1, 2009

Name:	
NetID:	

- You have **75 minutes** to complete this exam.
- This is a **closed-book** exam. You are allowed one 3inch by 5 inch card of notes prepared by yourself. This card is **not to be shared**. All other materials, besides pens, pencils and erasers, are to be away.
- Do not share anything with other students. Do not talk to other students. Do not look at another student's exam. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.
- If you believe there is an error, or an ambiguous question, you may seek clarification from myself or one of the TAs. You must use a whisper, or write your question out. Speaking out aloud is not allowed.
- Including this cover sheet and rules at the end, there are 8 sheets, 15 pages to the exam, including one blank page for workspace. The exam is printed double sided. Please verify that you have all 15 pages.
- Please write your name and NetID in the spaces above, and also at the top of every sheet.

Problems	**Possible Points**	**Points Earned**
1	5	
2	9	
3	15	
4	5	
5	14	
6	12	
7	8	
8	15	
9	17	
PreTotal	100	
Extra Credit	10	
PostTotal	110	

CS 421 Midterm 1 **Name:**____________________________________

1. (5 pts total) Suppose that the following code is input into OCaml:

```
let a = "Hi";;
let salute n = a ^ ", " ^ n;;
let a = "Hello";;
let b = salute "Joe";;
let a = true;;
let s = salute "Sally";;
```

For each of the following, write true or false after each statement.

a. (2 pts) **b** will have a value of

1) "Hello, Joe" **false**

2) "Hi, Joe" **true**

b. (3 pts) After the declaration of **b**,

1) The declaration **let a = true;;** will cause a type error.

false

2) The declaration **let s = salute "Sally";;** will cause a type error.

false

3) The identifier **s** will have the value **"Hello, Sally"**

false

2. (9 pts total)
   a. (3 pts) Write a function
      do_each: **(('a -> 'b) * ('c -> 'd)) -> ('a * 'c) -> ('b * 'd)**
      that takes a pair of functions as a first argument and a pair of values as a second argument, then returns the pair formed by applying the first function to the first argument and the second function to the second argument.

```
let do_each (f, g) (x, y) = ((f x), (g y));;
```

What is the result of each of the following applications:

b. (2 pts) **do_each (fun x -> x + 1) (fun y -> y > 3) (2, 7);;**

A type eror because the first argument to do_each must be a pair of function, not a function

c. (2 pts) **do_each ((fun a -> a ^ "!") , (fun z -> z > 0));;**

A function of type : (string * int) -> (string * bool) taking a pair of a string and an integer, and returning a pair of the original string with ! on the end, and a boolean saying whether the integer was strictly greater than 0.

d. (2 pts) **do_each ((fun x -> x – 2) , (fun x -> x + 3)) (3, 19);;**

(1, 22)

CS 421 Midterm 1 **Name:___________________________________**

3. (15 pts total) Consider the following OCaml code

```
let a = 10;;
let f =  fun x -> x + a;;
let a = 20;;
let g = fun z -> f (2 * z);;
```

Describe the final environment that results from the execution of the above code if execution is begun in an empty environment. Your answer should be written as a set of bindings of variables to values, with only those bindings visible at the end of the execution present. Your answer should be a precise mathematical answer, with a precise description of values involved in the environment.

{ f → < (x) → x + a, { a → 10 }>, a → 20,
g → <(z) → f(2 * z), { f → < (x) → x + a, {a → 10}>, a → 20 }

4. (5 pts) Write **count : 'a list -> 'a -> int** that such that **count l a** returns the number of times that **a** occurs in **l**. You may use any form of recursion, and any standard Library functions. Executing your code should give the following behavior:

```
# count [1;2;0;3;4;0;1] 0;;
- : int = 2
```

```
let rec count list y =
    match list with [] -> 0
     | (x::xs) ->  if x = y then 1+count xs y
                   else count xs y;;
```

CS 421 Midterm 1 **Name:**___________________________________

5. (14 pts total)
   a. (6 pts) Write a function **separate : (’a -> bool) -> ’a list -> int * int** such that **separate p l** returns a pair of integers, where the first indicates the number of elements of **l** for which **p** returns **true**, and the second indicates the number of elements for which **p** returns **false**. The function is required to use (only) forward recursion (no other form of recursion). You may **not** use any library functions. Executing your code should give the following behavior:

```
# let rec separate p l = ... ;;
val separate : (’a -> bool) -> ’a list -> int * int = <fun>
# separate (fun x -> x mod 2 = 0) [-3; 5; 2; -6];;
- : int * int = (2, 2)
```

```
let rec separate p l =
    match l with [] -> (0, 0)
       | (x::xs) ->
        (match separate p xs with (a,b) -> if p x then (a + 1, b) else (a, b +1));;
```

   b. (8 pts) Rewrite **separate** as described above, using

      List.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b

   but no other Library functions and no explicit recursion.

```
let separate p l = List.fold_right
                    (fun x -> fun (a,b) -> if p x then (a + 1, b) else (a, b +1))
                    l
                   (0,0);;
```

6. (12 pts) Write a function **remove_allk** that is a complete Continuation Passing Style transformation of the following code:

```
let rec remove_all list m =
  match list with [] -> []
  | (x::xs) -> if x = m then remove_all xs m else x :: remove_all xs m;;
```

You may treat **+, *, -, mod, /, ::, =** all as built-in operators that do not need to be converted to CPS. Your function should have the following type:

remove_allk : 'a list -> 'a -> ('a list -> 'b) -> 'b

```
let rec remove_allk list m k =
  match list with [] -> k []
  | (x::xs) -> if x=m then remove_allk xs m k
               else remove_allk xs m (fun r -> k(x::r));;
```

CS 421 Midterm 1
Name:__________________________________

7. (8 pts). I have a short wave radio at home. It can be off, or it can be on an FM station, or an AM station, or an LW station or an MW station or an SW station. If stations are all represented by floats, give an Ocaml datatype **radio** describing the possible states of my radio. Assuming my radio can always be set to any station (and ignoring the fact that floats may be negative, but stations can not), every state of my radio should be describable by a term of type **radio**, and every term of type **radio** (using positive floats) should describe some possible state of my radio.

type radio = Off | FM of float | AM of float | LW of float | MW of float | SW of float

8. (15 pts total) Consider the following Ocaml datatype:

```
type webpage = Content of (string * string * webpage list)
                      | Data of (string * int);;
```

This OCaml data type **webpage** represents the following different kinds of information about simple webpages: a content page with its URL as a **string**, its content as a **string**, and its links as a list of **webpage**; or a data page with its URL as a **string**, and its data as an **int**. Note that in this representation, a URL does not determine webpage; multiple webpages may have the same URL. Write a function **get_all_data : webpage -> int list** that, given a webpage, returns a list containing the data (integer values) from all data pages reachable from that page. A page is reachable if either it is the current page, or there is some series of links from the current page that leads to it. You may use recursion and library functions from OCaml freely.

```
let rec get_all_data wp =
    match wp with
     Content(url, content, links) ->
       (List.fold_right (fun w -> fun r -> (get_all_data w) @ r)
         links
         [])
   | Data (url, data) -> [data];;
```

CS 421 Midterm 1 **Name:__________________________________**

9. (17 pts total) Give a type derivation for the following type judgment:

{ } |- (let rec f = fun x -> if x > 0 then x + f (x – 1) else 17 in f 3) : int

You may use the attached sheet of typing rules. Label every use of a rule with the rule used. You may abbreviate, but you must define your abbreviations. You may find it useful to break you derivation into pieces. If you do, give names to your pieces, which you may then use in describing the whole.

Let Γ_1 = {f:int -> int} and Γ_2 = {f:int -> int, x: int}
Let C=Constant Rule, V=Variables Rule, PO= Primitive Operations Rule, R=Relations Rule, Conn=Connectives Rule, I= If_then_else Rule, A = Application Rule, F= Function Rule, L=Let Rule, and LR=LetRec Rule

$$\text{LR}\;\dfrac{\textbf{Proof1} \qquad \text{A}\;\dfrac{\text{V}\;\dfrac{}{\Gamma_1 \text{ |- f : int -> int}} \qquad \text{C}\;\dfrac{}{\Gamma_1 \text{ |- 3 : int}}}{\Gamma_1 \text{ |- f 3 : int}}}{\text{\{ \} |- (let rec f = fun x -> if x > 0 then x + f (x – 1) else 17 in f 3) : int}}$$

Where Proof1 =

$$\text{F}\;\dfrac{\text{I}\;\dfrac{\text{R}\;\dfrac{\text{V}\;\dfrac{}{\Gamma_2 \text{ |- x :int}} \quad \text{C}\;\dfrac{}{\Gamma_2 \text{ |- 0:int}}}{\Gamma_2 \text{ |- (x > 0):bool}} \qquad \text{PO}\;\dfrac{\text{V}\;\dfrac{}{\Gamma_2 \text{ |- x :int}} \quad \text{A}\;\dfrac{\text{V}\;\dfrac{}{\Gamma_2 \text{ |- f:int -> int}} \quad \text{PO}\;\dfrac{\text{V}\;\dfrac{}{\Gamma_2 \text{ |- x :int}} \quad \text{C}\;\dfrac{}{\Gamma_2 \text{ |- 0:int}}}{\Gamma_2 \text{ |- (x – 1) :int}}}{\Gamma_2 \text{ |- f(x – 1) :int}}}{\Gamma_2 \text{ |- x + f (x – 1):int}} \qquad \text{C}\;\dfrac{}{\Gamma_2 \text{ |- 17 : int}}}{\Gamma_2 \text{ |- (if x > 0 then x + f (x – 1) else 17) : int}}}{\Gamma_1 \text{ |- (fun x -> if x > 0 then x + f (x – 1) else 17) : int -> int}}$$

Extra Credit: (10 pts) Write a function **filterk** that is a complete Continuation Passing Style transformation of the following code:

```
let rec filter p list =
 match list with [] -> []
 | (x::xs) -> if p x then filter p xs else x :: filter xs
```

You may treat **+, *, -, mod, /, ::, =** all as built-in operators that do not need to be converted to CPS. All procedures must be put in CPS form. Your function should have the following type:

```
filterk : ('a -> (bool -> 'b) -> 'b) -> 'a list -> ('a list -> 'b) -> 'b
```

```
let rec filterk pk list k =
   match list with [] -> k [] |
  (x::xs) ->
  pk x
  (fun b -> if b then filterk pk xs (fun r -> k(x::r))
              else filterk pk xs k) ;;
```

CS 421 Midterm 1
Name:____________________________________

Worksheet (If extra space is needed).

CS 421 Midterm 1 **Name:______________________________**

Rules for type derivations:

Constants:

$$\frac{}{\Gamma \vdash n : \text{int}} \quad \text{(assuming } n \text{ is an integer constant)}$$

$$\frac{}{\Gamma \vdash \text{true} : \text{bool}} \qquad \frac{}{\Gamma \vdash \text{false} : \text{bool}}$$

Variables:

$$\frac{}{\Gamma \vdash x : \sigma} \quad \text{if } \Gamma(x) = \sigma$$

Primitive operators ($\oplus \in \{ +, -, *, \text{mod}, \ldots\}$):

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}}$$

Relations ($\sim \in \{ <, >, =, <=, >= \}$):

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \sim e_2 : \text{bool}}$$

Connectives :

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \mathbin{\&\&} e_2 : \text{bool}} \qquad \frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \mathbin{||} e_2 : \text{bool}}$$

If_then_else rule:

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash \mathrm{e}_2 : \tau \quad \Gamma \vdash \mathrm{e}_3 : \tau}{\Gamma \vdash (\text{if } e_1 \text{ then } \mathrm{e}_2 \text{ else } \mathrm{e}_3) : \tau}$$

Application rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash (e_1\ e_2) : \tau_2}$$

Function rule:

$$\frac{[x : \tau_1] + \Gamma \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \text{ -> } e : \tau_1 \rightarrow \tau_2}$$

Let rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad [x : \tau_1] + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let x} = e_1 \text{ in } e_2) : \tau_2}$$

Let Rec rule:

$$\frac{[x : \tau_1] + \Gamma \vdash e_1 : \tau_1 \quad [x : \tau_1] + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let rec x} = e_1 \text{ in } e_2) : \tau_2}$$