
HW 1 – Evaluation and Environments

CS 421 – Fall 2009

Revision 1.1

Assigned September 22, 2009

Due October 6, 2009, 2:00 pm, in class

Extension 48 hours (20% penalty)

1 Change Log

1.1 Corrected that Problem 3 ran off the page.

1.0 Initial Release.

2 Turn-In Procedure

Your answers to the following questions are to be hand-written, or printed, neatly on one or more sheets of paper, each with your name in the upper right corner. The homework is to be turned in in class at the start of class. Alternately, you may hand it to Prof. Elsa Gunter in person before the deadline.

3 Objectives and Background

The purpose of this HW is to test your understanding of

- the order of evaluation of expressions in OCaml;
- the scope of variables, and the state of environments used during evaluation
- how to use typing rules to perform type derivations in simplified OCaml

Another purpose of HWs is to provide you with experience answering non-programming written questions of the kind you may experience on the midterms and final.

4 Problems

1. (15 pts) Below is a fragment of OCaml code, with various program points indicated by numbers with comments.

```
let x = "a";;  
let y = 8;;  
(* 1 *)  
let f x z = x + y + z;;  
(* 2 *)  
let z =  
  let y = "b" in  
(* 3 *)  
    x ^ y;;  
(* 4 *)
```

```

let g w = f w w;;
let y = g y;;
(* 5 *)

```

For each of program points 1, 2, 3, 4 and 5, please describe the environment in effect after evaluation has reached that point. You may assume that the evaluation begins in an empty environment, and that the environment is cumulative thereafter. The program points are supposed to indicate points at which all complete preceding declarations (including local ones) have been fully evaluated. In describing the environments 1 through 4, you may use set notation, as done in class, or you may use the update operator $+$. If you use set notation, no duplicate bindings should occur. The answer for program point 5 should be written out fully in set notation.

Solution:

1. $\rho_1 = \{y \rightarrow 8, x \rightarrow "a"\}$
2. $\rho_2 = \{f \rightarrow \langle x \mapsto \text{fun } z \rightarrow x + y + z, \rho_1 \rangle\} + \rho_1$
 $= \{f \rightarrow \langle x \mapsto \text{fun } z \rightarrow x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\}, y \rightarrow 8, x \rightarrow "a"\}$
3. $\rho_3 = \{y \rightarrow "b"\} + \rho_2$
 $= \{y \rightarrow "b", f \rightarrow \langle x \mapsto \text{fun } z \rightarrow x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\}, x \rightarrow "a"\}$
4. $\rho_4 = \{z \rightarrow "ab"\} + \rho_3$
 $= \{z \rightarrow "ab", f \rightarrow \langle x \mapsto \text{fun } z \rightarrow x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\}, y \rightarrow 8, x \rightarrow "a"\}$
5. $\{y \rightarrow 24,$
 $g \rightarrow \langle w \mapsto f \ w \ w, \{z \rightarrow "ab", f \rightarrow \langle x \mapsto \text{fun } z \rightarrow x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\}, y \rightarrow 8, x \rightarrow "a"\}\},$
 $z \rightarrow "ab",$
 $f \rightarrow \langle x \mapsto \text{fun } z \rightarrow x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\},$
 $x \rightarrow "a"\}$

Because I may have been ambiguous in class, or even misspoke, I will also accept

1. $\rho_1 = \{y \rightarrow 8, x \rightarrow "a"\}$
2. $\rho_2 = \{f \rightarrow \langle (x, z) \mapsto x + y + z, \rho_1 \rangle\} + \rho_1$
 $= \{f \rightarrow \langle (x, z) \mapsto x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\}, y \rightarrow 8, x \rightarrow "a"\}$
3. $\rho_3 = \{y \rightarrow "b"\} + \rho_2$
 $= \{y \rightarrow "b", f \rightarrow \langle (x, z) \mapsto x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\}, x \rightarrow "a"\}$
4. $\rho_4 = \{z \rightarrow "ab"\} + \rho_3$
 $= \{z \rightarrow "ab", f \rightarrow \langle (x, z) \mapsto x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\}, y \rightarrow 8, x \rightarrow "a"\}$
5. $\{y \rightarrow 24,$
 $g \rightarrow \langle w \mapsto f \ w \ w, \{z \rightarrow "ab", f \rightarrow \langle (x, z) \mapsto x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\}, y \rightarrow 8, x \rightarrow "a"\}\},$
 $z \rightarrow "ab",$
 $f \rightarrow \langle (x, z) \mapsto x + y + z, \{y \rightarrow 8, x \rightarrow "a"\}\},$
 $x \rightarrow "a"\}$

This answer is really for the version of the code where f is in uncurried form.

2. (20 pts) Below is a fragment of Ocaml code. Describe everything that is displayed on the screen (its observable behavior) after this code has been cut-and-pasted into an interactive Ocaml session, and explain why this behavior is observed. This should include both the type information that the compiler gives back for each declaration, and any other things printed to the screen. For the type information, no explanation is required (but it should be correct). Give explanation for all other things printed, and the order in which they occur.

```

let f g x =
  (let r =
    if ((print_string "a"; x > 2) && (print_string "b"; x > 10)) &&
      (let y = (print_string "c"; 14) in
        (print_string "d"; x > y) || (print_string "e"; x < 5)))
    then
      let z = (print_string "f"; 7) in (print_string "g"; x - z)
    else
      let z = (print_string "h"; 15) in (print_string "i"; z)
  in (g(); r));;
let u = (f (fun () -> print_string "j\n") (f (fun () -> print_string "k\n") 3));;

```

Solution: Printed on the screen is:

```

val f : (unit -> 'a) -> int -> int = <fun>
# abhik
abcdfgj
val u : int = 8

```

First, the evaluation of the declaration of the `f` generates no observable behavior, beside the final printing of its type information, because it immediately evaluates to a closure $\langle\langle g \mapsto \text{fun } x \rightarrow (\text{let } r = \dots), \rho_{init} \rangle\rangle$ where ρ_{init} is the environment with which we start.

The evaluation of `u` proceeds as follows:

- To evaluate `u`, we must evaluate $(f (fun () \rightarrow \text{print_string } "j\n") (f (fun () \rightarrow \text{print_string } "k\n") 3))$.
- `f` is already a closure, and $(fun () \rightarrow \text{print_string } "j\n")$ evaluates immediately to a closure. The application $(f (fun () \rightarrow \text{print_string } "j\n") (f (fun () \rightarrow \text{print_string } "k\n") 3))$ evaluates to a closure with $g \mapsto (fun () \rightarrow \text{print_string } "j\n")$ added to the environment of the closure for the application. No observable behavior occurs.
- To evaluate the application of $(f (fun () \rightarrow \text{print_string } "j\n"))$ to $(f (fun () \rightarrow \text{print_string } "k\n") 3)$, we must first evaluate $(f (fun () \rightarrow \text{print_string } "k\n"))$. This is an application of an application: `f` is applied to $(fun () \rightarrow \text{print_string } "k\n")$ and then to 3. The first application yields a closure derived from `f`'s, with $g \mapsto \langle(), \text{print_string } "k\n", \rho_f \rangle$ added to the closure environment (which is empty). (ρ_f is the environment containing the the mapping of `f` to its closure.) The second application the body of `f` to be evaluated in the second closure environment with $x \mapsto 3$ added.
- Next we must evaluate the expression for `r`. It is a conditional, so we must evaluate the boolean guard first.
- The boolean guard is conjunction; the left-hand side is evaluated first. That in turn is again a conjunction; the left-hand side of it is evaluated first.
- `a` is printed out; $x > 2 \rightarrow 3 > 2 \rightarrow \text{true}$
- Since the first conjunct evaluated to `true`, next we evaluate the second half of the inner conjunction. `b` is printed. $x > 10 \rightarrow 3 > 10 \rightarrow \text{false}$. Thus second conjunct evaluates to `false`, and hence the first conjunct of the outer conjunction also evaluates to `false`. Short-circuit evaluation, prevents the second half of the conjunction from evaluating.
- Next we evaluate the `else` branch of the conditional. This begins with evaluating the `let`. `h` is printed out, then $z \mapsto 15$ is added to the current environment. Next `i` is printed, then the binding for `z` is removed from the current environment, and `r` is bound the value of `z`, 15, in the environment,

Then the main type derivation is as follows:

$$\begin{array}{c}
 \frac{}{\{f:\text{int} \rightarrow \text{int}\} \vdash f:\text{int} \rightarrow \text{int}} \text{V} \quad \frac{}{\{f:\text{int} \rightarrow \text{int}\} \vdash 0:\text{int}} \text{C} \\
 \hline
 \frac{}{\{f:\text{int} \rightarrow \text{int}\} \vdash (f\ 3):\text{int}} \text{APP} \quad \frac{}{\{f:\text{int} \rightarrow \text{int}\} \vdash 0:\text{int}} \text{C} \\
 \hline
 \frac{}{\{f:\text{int} \rightarrow \text{int}\} \vdash (f\ 3) > 0 : \text{bool}} \text{AR} \quad \frac{}{\{f:\text{int} \rightarrow \text{int}, y:\text{bool}\} \vdash y:\text{bool}} \text{V} \\
 \hline
 \frac{}{\{f:\text{int} \rightarrow \text{int}\} \vdash \text{let } y = ((f\ 3) > 0) \text{ in } y : \text{bool}} \text{LR} \\
 \hline
 \text{decproof} \quad \frac{}{\{\} \vdash (\text{let rec } f = \text{fun } x \rightarrow \text{if } x \leq 0 \text{ then } 0 \text{ else } f\ (x - 1) \\ \text{in let } y = ((f\ 3) > 0) \text{ in } y : \text{bool})} \text{LR}
 \end{array}$$

4. (Extra Credit) (5 pts) What is the running time of the following Ocaml function, in proportion to the size of the integers input? How can you rewrite this code so that its running time is linear in the size of the input?

```

let rec weird n =
  if n <= 0 then 0
  else if weird(n - 1) mod 2 = 0 then weird (n - 1) / 2
  else weird (n - 1) - 1;;

```

Solution:

Because at each step of the recursion potentially two recursive calls are made, each to pieces of size one smaller than the current call, this code is exponential in the size of n . To eliminate these duplicate recursive calls, we can use a local binding to record the value of the recursive call and reuse it:

```

let rec weird n =
  if n <= 0 then 0
  else let r = weird (n - 1) in if r mod 2 = 0 then r / 2 else r - 1;;

```

I will also accept the solution that observes that this is the constant function that always returns 0 (we never enter the else branch).