HW 1 – Evaluation and Environments

CS 421 – Fall 2011

Revision 1.0

Assigned September 20, 2011
Due September 27, 2011, 2:00 pm, in class
Extension 48 hours (20% penalty)

1 Change Log

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 OCaml variant types (a.k.a algebraic data types) to model specific examples
- 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 = 3;;
let a = x + 4;;
(* 1 *)
let g x = a + (2 * x);;
(* 2 *)
let a = g 3;;
(* 3 *)
let b =
    let f x y = g x + g (y + a) in
(* 4 *)
f x a;;
(* 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 without the update operator or duplications.

2. (15 pts) Give an OCaml variant data type to capture the set of all polynomials over an arbitrary set of variables, named by strings, with integer coefficients and closed under binary addition and multiplication. Your data type should be able to accurately represent the following polynomials:

$$0 \qquad x \qquad 5 * z$$

$$x * (x * (4 * y)) \qquad (2 * x) + (y * y) \qquad (2 + x) * (y + z)$$

It should also be the case that every data structure that can be built using your data type should accurately represent a polynomial.

3. (20 points) Give a complete type derivation for the following typing judgment:

```
{}⊢ let fact = fun n ->
      let rec fact_aux = fun n -> (fun prod ->
              if n = 0 then prod else fact_aux (n -1) (n * prod))
      in fact_aux n 1
    in fact 4 : int
```

You should use the monomorphic typing rules for `let` and `let rec`.

4. (Extra Credit) (5 pts) Write OCaml data type(s) to present red-black trees without the constraint that every path from the root to a leaf has the same number of black nodes. More precisely, we wish to describe trees made from leaves containing only values and red and black nodes as follows: Every black node contains a value and a pair, with each component of the pair being either a red node, a black node or a value. Every red node contains a value and a pair, each component of which is either a black node or a value, but not a red node. The root of a red-black tree is a black node.

You are free to define as many types as you feel appropriate, but you should clearly label which type is representing red-black trees. Your red-black tree type should be able to describe all such trees, with distinct representations. Moreover, every data structure of that type should correspond to a red-black tree in a clear manner.