Polymorphic types; Hoare axioms

CS 421 – Spring 2011
Revision 1.0

Assigned Thursday, April 28, 2011
Due Tuesday, May 3, 2011, in class
Extension no lateness permitted
Total points 30

1 Change Log

1.0 Initial Release.

2 Objectives and Background

After completing this MP, you should have a better understanding of

- OCaml's polymorphic type system
- Hoare axioms and loop invariants

3 Collaboration

Collaboration is NOT allowed in this assignment.

4 Turn-In Procedure

Your solutions are to be typeset (presumably with TEX) on one or more sheets of paper, each with your name in the upper right corner. The homework is to be turned in at the beginning of class.

5 OCaml type system

The OCaml type system is presented in lectures 24-25. We used the standard mathematical notation for presenting those systems. However, that notation is hard to use because proofs are difficult to typeset and they run off the page. For this homework, we present the proof systems in a different way, using indentation. In other words, we will write proof trees with the root at the top and the children below them, indented by a certain amount. The systems are otherwise identical.

Here is the type system in the new format:

Variable: $\Gamma \vdash x : \tau$, if $\Gamma(x) = \sigma$ and $\tau \leq \sigma$

Application:

$$\Gamma \vdash e_1 e_2 : \tau$$
$$\Gamma \vdash e_1 : \tau' \to \tau$$
$$\Gamma \vdash e_2 : \tau'$$

Abstraction:

$$\Gamma \vdash \text{fun } x \text{ -> } e : \tau \to \tau'$$
$$\Gamma[x : \tau] \vdash e : \tau'$$

Tuple:

$$\Gamma \vdash (e_1, e_2) : \tau_1 * \tau_2$$
$$\Gamma \vdash e_1 : \tau_1$$
$$\Gamma \vdash e_2 : \tau_2$$

Let:

$$\Gamma \vdash \text{let } x\text{=}e \text{ in } e' : \tau'$$
$$\Gamma \vdash e : \tau$$
$$\Gamma[x : GEN_\Gamma(\tau)] \vdash e' : \tau'$$

For example, following is the proof of $\Gamma_0 \vdash$ fst(3,true): int, which is in the annotated notes from lecture 25. There is one change: We decided the use of the empty set for the initial environment was too abusive, so we use Γ_0 instead. Recall that this initial environment gives type schemes for all names; in particular, we are assuming Γ_0(fst) = $\forall\alpha, \beta.\alpha * \beta \to \alpha$, $\Gamma_0(3)$ = int, and $\Gamma_0(true)$ = bool.

$\Gamma_0 \vdash$ fst(3,true): int (*App*)
- $\Gamma_0 \vdash$ fst: int*bool→int, since Γ_0(fst) $= \forall\alpha, \beta.\alpha * \beta \to \alpha$ and int*bool→int $\leq \forall\alpha, \beta.\alpha * \beta \to \alpha$ (*Var*)
- $\Gamma_0 \vdash$ (3,true): int*bool (*Tuple*)
  - $\Gamma_0 \vdash$ 3: int, since $\Gamma_0(3) =$int (*Var*)
  - $\Gamma_0 \vdash$ true: bool, since Γ_0(true) =bool (*Var*)

The following problems are to be typeset (presumably with TEX) and handed in on a separate sheet of paper. The first one was already done in class, except that in class we replaced `fun y -> y+1` with `incr` and assumed that Γ_0(incr) = int→int. The second one is a more elaborate version of the first. Keep in mind that the structure of the type proof exactly mimics the abstract syntax of the expression being typed, so your solutions should have exactly that number of lines (just as the proof above has five lines).

For these problems, we are using prefix forms of the built-in operations + and `::`, so as not to have to introduce any new rules, and we're using "`[]`" as a "variable." So, assume we have Γ_0(+) = int→int, Γ_0(::) = $\forall\alpha.\alpha \to \alpha$ list $\to$ α list , and Γ_0([]) = $\forall\alpha.\alpha$ list . Also, keep in mind that these operations are curried; so, for example, "`(+) y 1`" is really "`((+) y) 1`," and contains five AST nodes (two app nodes and three var nodes).

1. (9 pts) $\Gamma_0 \vdash$ let f=fun x -> x 0 in f (fun y -> (+) y 1) : int
2. (9 pts) $\Gamma_0 \vdash$ let f=fun x -> x 0 in (f (fun y -> (+) y 1), f(fun n -> (::) n [])) : int * int list

6 Hoare axioms

For these questions, we will not ask you to write proofs, but just to give the invariants of some loops. Note that the invariant must actually be an invariant, and, together with the termination of the loop, should prove the "post-condition" — the conclusion given after the loop.

You should use mathematical notation, as we have in giving the pre- and post-conditions. We have used one abbreviation, which you can also use: we write "A has the same elements as B" to mean that arrays A and B are

of the same length and have the same elements, but possibly in a different order; similarly, we write "A[i..j] has the same elements as B[k..l]" with the obvious meaning. It is somewhat complicated to write this out completely in mathematical notation. (If you're not familiar with the notation $\forall a < i < b. \ldots$, it is just shorthand for $\forall i. a < i \& i < b \Rightarrow \ldots$.)

Another important notational convention is that we always assume, for any variable x, that we can refer to its *original* value — the value before the loop started — as x_0. In particular, for arrays, this means A_0 has the same elements as A, in their original order. Finally, we write $|A|$ for the length of array A.

This homework should be typed and handed in with the solutions for problems 1 and 2.

3. (3 pts) Give the invariant for this loop which finds the location of the smallest element in A. The pre-condition is $min = 0$ & $i = 1$, and the post-condition is $\forall 0 \leq i < |A|. A[min] \leq A[i]$.

```
while (i<n) { if (A[i] < A[min]) min = i;
              i = i+1; }
```

4. (3 pts) This loop performs "selection sort;" you are to give its invariant. `min(A, j)` is the index of the smallest element in `A[j..|A|-1]`. `swap` exchanges the indicated elements of `A`. The pre-condition is $i = 0$ & $n = |A|$, and the post-condition is $\forall 0 \leq j < |A| - 1. A[j] \leq A[j+1]$ & A has the same elements as A_0.

```
while (i < n-1) {
   m = min(A,i); swap(A, i, m); i = i+1;
}
```

5. (3 pts) This loop performs a partition operation like the one at the heart of quicksort. That is, it moves the elements of A so that all the elements less than or equal to some value x come before all the elements greater than x. The pre-condition is $lo = 0$ & $hi = |A| - 1$, and the post-condition is $\forall 0 \leq i < lo. A[i] \leq x$ & $\forall lo \leq i < |A|. A[i] > x$.

```
while (lo<hi)
   if (A[lo] > x) {
      swap(A, lo, hi); hi = hi-1;
   }
   else if (A[hi] <= x) {
      swap(A, lo, hi); lo = lo+1;
   }
   else { lo = lo+1; hi = hi-1; }
```

6. (3 pts) Give the invariant for this loop which performs in-place reversal of an array. The pre-condition is $i = 0$ & $j = |A| - 1$, and the post-condition is $\forall 0 \leq i < |A|. A[i] = A_0[|A| - i - 1]$.

```
while (i < j) { swap(A, i, j);  i = i+1; j = j-1;}
```