Machine Problem Set 5

Assigned: April 03, 2006
Due: April 21, 2006 9:00pm

Objectives and Background

Upon completion of this MP, you should understand:

- How to create your own types, and how they interact with data abstraction boundaries.
- How to write interpreters for object-oriented languages

Problem 1

In this problem, you will add the ability to define your own types to a (provided) typed interpreter.

The suggested steps are:

- Make sure you understand the provided code, especially the way letrec works. It is similar to the interpreter you worked with in the previous MP.

- Add a typedef-record to the type-environment datatype, modify apply-tenv to ignore typedef-records, and add a function find-typedef to map type names to types by searching through the environment.

- Add lettype to the grammar definition.

- Modify expand-type-expression (and expand-type-expressions) to convert new type-expressions to the actual types.

- Extend type-of-expression to handle lettype-exp. This is the biggest change you have to make. This must check the expression types actually match the declared types. You should treat the new type as the implementation type in the function definitions of lettype, and as a new atomic type in the body of lettype.
Example:

```lisp
(run "
(lettype ff = (int -> int)
   ff zero-ff () = proc (int k) 0
   ff extend-ff (int k, int val, ff old-ff) =
      proc (int k1) if zero?( -(k1,k) ) then val else (old-ff k1)
   int apply-ff ( ff f, int k ) = (f k)
in  let ff1 = (extend-ff 1 11 (extend-ff 2 22 (zero-ff)))
in (apply-ff ff1 2)"
)
```

**Problem 2**

This problem is only required for all the graduate students. Undergraduates may do this problem for extra credit.

Exercise 5.22 in the book.

**Problem 3**

Exercise 5.23 In the book. We will not test the case where a static field in the derived class overrides a field in the base class. A solution can be written by adding/changing around 25 lines of code to the starter code.

Example:

```lisp
(run "
class c1 extends object
   static next_serial_number = 5
   field my_serial_number
   method get_serial_number () my_serial_number
   method initialize ()
      begin
         set my_serial_number = next_serial_number;
         set next_serial_number = add1(next_serial_number)
      end

class c2 extends c1
   static next_to_last_ser_num = sub1(next_serial_number)
   method get_next_to_last_ser_num() next_to_last_ser_num
let o1 = new c1()
o2 = new c1()
o3 = new c2()
```


in list(send o1 get_serial_number(),
    send o2 get_serial_number(),
    send o3 get_serial_number(),
    send o3 get_next_to_last_ser_num())"
(5 6 7 4)

Problem 4

Modify the interpreter so that the class fields are always available via ‘self->’ keyword. The solution can be written in about 12 lines. (Brownie points: Can you write an interpreter that is a solution to both this and previous problem?) Example:

> (run "
class c1 extends object
    field x
    method initialize () set x = 1
    method m2()
    let x = 5 in
    x
    method m3()
    let x = 5 in
    self->x
    let o1 = new c1()
in send o1 m3()"
1

Handin

You should hand in four files named ‘mp5p1.scm’, ‘mp5p2.scm’, ‘mp5p3.scm’, and ‘mp5p4.scm’ with the implementations of the interpreters described above. There are two starter files, one for problem 1 and one for problems 2-4. If you choose not to do problem x, copy the starter file into the mp5px.scm and hand it in.

Make sure this assignment runs the way you want on the EWS computers under the MzScheme dialect.

Please see the CS 421 FAQ web page for handin instructions.