CS 558 Homework 3 – due 2pm, Monday, January 25, 2010

Homework must be submitted by mail to cs558acc@cs.pdx.edu, with the subject line “HW3”.

Place your solution in a Scheme file names sol3.ss and send this is a plain-text attachment to the mail message (the contents of the message itself will be ignored). It is your responsibility to submit the homework in the proper format.

General rules:

- Your files must start with the language declaration #lang planet plai/plai:1:6.
- You must provide a sensible set of tests (using the PLAI test form) for each top-level function you define. No credit without the tests.
- For questions that call for English text, just put your answers in a block comment.
- Try to stick to the subset of Scheme forms that were introduced in Homework 1.

1. Implement the F1WAE interpreter with “deferred substitutions” (which most authors call “environments”) from PLAI Chapter 5, but extended with subtraction, multiplication, and division, and with an if0 conditional construct. The form (if0 a b c) is intended to evaluate to b if a is zero, and to c if a is non-zero.

Here’s the expression syntax of the extended F1WAE language.

```
<F1WAE> ::= <num>
    | {+ <F1WAE> <F1WAE>}
    | {- <F1WAE> <F1WAE>}
    | {* <F1WAE> <F1WAE>}
    | {/ <F1WAE> <F1WAE>}
    | {with {<id> <F1WAE>} <F1WAE>}
    | {if0 <F1WAE> <F1WAE> <F1WAE>}
    | <id>
    | {<id> <F1WAE>}
```

The function AST (FunDef) remains the same as in Chapter 5. Deviating somewhat from Chapter 5, we’ll use an explicit AST for whole programs too, consisting of a list of named functions together with a “main” expression. Then we’ll use the obvious S-expression form of the program AST as a concrete syntax. For example, here is an F1WAE program that should evaluate to 50:

```
{({g x {with {d (= - x 42)} {if0 d {h 100} {h 200}}} }
 {h y (/ y 2)}}
 {with {a {+ 22 (* 5 4)}} {g a}}}
```

A datatype definition, parser, and interp function for the original F1WAE described in Chapter 5 are provided in hw3.ss.

(a) Write the datatype definition (define-type) for extended F1WAE expressions.
(b) Write the parse function for expressions in this extended language.
(c) Write the interp function for this extended language.
Note: When evaluating division, you should not let your interpreter crash with a Scheme error. Instead, you should issue your own polite error (similar to the “free identifier” error that’s shown in the book’s calc). Note that you can use the test/exn form to test that your code issues errors when it should.

2. Write and test the factorial function (on non-negative integers) in your F1WAE interpreter.

3. We discussed the map and filter functions in lecture 6. The lecture notes go on to define and use an even more useful higher-order function called fold-right (or just foldr in the standard library), which abstracts over the traversal of a list (from right to left) accumulating an answer. For reference, here’s the definition:

   (define (fold-right c n xs)
     (if (empty? xs)
         n
         (c (first xs) (fold-right c n (rest xs)))))

Write Scheme code for the each of the following functions using fold-right and without using any other recursion (or imperative features).

(a) my-forall, which takes two arguments: a function returning booleans and a list, such that (my-forall p xs) returns true if and only if (p x) returns true for each x in xs. For example, (my-forall (lambda (x) (> x 2)) '(3 4 5)) returns true but (my-forall (lambda (x) (> x 2)) '(3 2 4)) returns false. You may use the standard and library function.

(b) my-reverse : listof(X) -> listof(X), which returns the elements of its argument list in reverse order. For example, (my-reverse '(1 2 3)) is (3 2 1). You may use the standard append library function.

(c) my-filter, with the same behavior as specified in lecture.