CS 558 Homework 7 – due 6:00pm, Wednesday, Nov. 26, 2003

Homework must be submitted by mail to cs558acc@cs.pdx.edu. All submitted files (two for this assignment) must be sent as plain-text attachments 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.

All programs mentioned can be downloaded from the course web page.

1. Consider the following functional language, which we’ll call “E7.”

\[
\begin{align*}
\text{prog} & := \text{exp} \\
\text{exp} & := \text{var} \\
& \mid \text{int} \\
& \mid '(' \text{if} \ exp \ exp \ exp ')' \\
& \mid '(' \text{let} \ var \ exp \ exp ')' \\
& \mid '(' \text{letfun} \ var \ var \ exp \ exp ')' \\
& \mid '(' \text{@} \ exp \ exp ')' \\
& \mid '(' + \ exp \ exp ')' \\
& \mid '(' - \ exp \ exp ')' \\
& \mid '(' * \ exp \ exp ')' \\
& \mid '(' / \ exp \ exp ')' \\
& \mid '(' <= \ exp \ exp ')' \\
& \mid '(' \text{pair} \ exp \ exp ')' \\
& \mid '(' \text{fst} \ exp ')' \\
& \mid '(' \text{snd} \ exp ')' \\
& \mid '(' \text{ispair} \ exp ')' \\
\text{var} & := \text{letter} \{ \text{letter} | \text{digit} \}
\end{align*}
\]

As usual, comments may be included by enclosing them between ‘{’ and ‘}’ characters, and they may be nested.

E7 is similar to our earlier (untyped) languages, but lacks imperative features including assignment, while, block, and write. The local expression has been renamed let (to emphasize that it acts like an ML-style immutable binding), and only one variable can be defined. (Nested lets can be used to define multiple variables.)

The most important changes are in the treatment of functions, which are now defined using a locally-scoped letfun expression form. Functions are treated as just another kind of value, and they share the same name space as other values. To evaluate \((\text{letfun} \ f \ x \ b \ e)\), first create a function value whose formal argument is \(x\), whose body is \(b\), and whose environment is the current environment; then bind \(f\) to that function value and evaluate \(e\) in the resulting environment. Applications now take an arbitrary expression in the function position; this must evaluate to a function value. All functions take exactly one argument; pairs can be used to encode multiple arguments (or multiple results). Functions are completely “first-class,” i.e., they can be passed as arguments to, or returned as results of, other functions, and can be stored in pairs. Since there is now no need for a separate (fun ...) declaration form, a program is once again just an expression.
The web site gives several example E7 programs. Program static.e7 illustrates that nested functions use static scoping rules; program compose.e7 shows how to write a higher-order function that composes two existing functions.

The web site provides an E7 interpreter in Java (hw7_1.java). Your task is to modify this interpreter to allow E7 functions to be recursive. For example, program map.e7 (which currently fails with an unbound variable error) should work properly when you're done. This requires only a very small change. But be careful: programs like static.e7 must also continue to work when you're done!

Call your revised interpreter sol7_1.java and submit it.

2. Implement the following functions in ML using foldr but without using any imperative features or explicit recursion. Submit your solutions to both the exercises in a single file sol7_2.sml.

a. reverse : 'a list -> 'a list reverses an arbitrary list. You make take the “built-in” library function @ (append) as primitive. (Ignore the warning message you may get when testing this function on empty lists.)

b. forall : ('a -> bool) * 'a list -> bool, where forall(f,l) returns true iff for every element x in l, f(x) evaluates to true, or l is empty.