Assignment 6 — Interpretation — assigned Saturday 19 October — due Monday 4 November

This assignment is worth 100 points.

6.1 Interpreters (50pts)

Given a representation of an *expression* in the subject (source) language, an interpreter computes a representation of the *value* of that expression.

An interpreter can be viewed as a mechanism for evaluating a source expression, or as a mechanism for defining the meaning of source expressions. The latter is commonly used, but fraught with danger (Reynolds, 1972). We prefer the first option. The meaning of expressions is given by a formal dynamic semantics, for instance in the style of structured operational semantics,[1] and we must prove that the interpreter is faithful to the semantics.

In the manner of semantics of PCF as developed in class, write a formal semantics of the core lambda language (the target language of assignment 3).

6.2 Substitution-based interpreter for the core lambda language (50pts)

Write an interpreter for the core lambda language that uses substitution. This should be very similar to the substitution-based interpreter for the pure lambda calculus which you developed in assignment 1. The interpreter operates exclusively within the realm of core lambda language expressions.

An incomplete sketch of the interpreter is given by this code:

```
fun subst = ...

exception BadPrim and BadSwitch and BadSelect

fun eval e =
   case e of
      VAR v => VAR v
    | ABS (v, e) => ABS (v, e)
    | APP (e1, e2) => apply (eval e1, eval e2)
    | INT i => INT i
    ...
    | PRIM p => PRIM p
```

[1] Denotational semantics, especially action semantics lead to almost direct derivation of interpreters (Watt, 1986), but the interpreters may not be very efficient.

```
    | RECORD el => RECORD (List.map eval el)
    ...

and apply (e1, e2) =
    case e1 of
      ABS (v, e) => eval (subst e2 v e)
                         (* substitute e2 for v in e *)
    | PRIM IADD =>
        (case e2 of
           RECORD [INT i1, INT i2] => INT [Int64.+ (i1, i2)]
         | _ => raise BadPrim
        )
    ...
```

Note that the order of evaluation in the core lambda language is implicitly given by the order of evaluation in SML. Carefully consider the order of evaluation in your implementation language.

How to turn in

Turn in your code by running *~darko/handin your-file* on a regular UNM CS machine or on *delta*.

You should use whatever filename is appropriate in place of your-file. You can put multiple files on the command line, or even directories. Directories will have their entire contents handed in, so please be sure to clean out any cruft.

Remember to submit extensive tests of your programs!

Homework must be accompanied by the following statement: *"I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual, including Section 4.8, Academic Dishonesty."* The manual is available at `http://www.unm.edu/~brpm/index.html`.