Homework 5 — ML — assigned Wednesday 6 April — due Thursday 21 April

All ML code in this assignment must be encapsulated in appropriate modules (signatures and structures). Be careful to distinguish internal (auxiliary) components from those that should be visible to the outside, and write the signatures accordingly. Document the purpose of signatures and structures.

Reading assignment

Read Chapter 8 of *ML for the Working Programmer*, focusing on input-output; explore available input-output primitives in the online SML Basis Library documentation. Read about the mutable store in ML (type constructors `ref` and `array`), but do not use it in your work.

Read Chapter 7 of *ML for the Working Programmer*, but ignore all references to `abstype`, which is obsolete.

5.1 Real functions [A.2; A.4; C; K.1.1; K.1.3; K.2.3; K.2.5; K.2.7; K.3.1] (50pts)

Warning: In this exercise, the word *function* refers both to ML functions and to mathematical functions. The intended meaning should be clear from context.

In this exercise we use the following representation for a class of mathematical functions that is a subset of $\mathbf{R} \to \mathbf{R}$:

```
datatype expr = Num of real
              | IntNum of int
              | ConstE
              | ConstPi
              | Var of string
              | Let of {var: string, value: expr, body: expr}
              | Neg of expr
              | Add of expr * expr
              | Sub of expr * expr
              | Mul of expr * expr
              | Div of expr * expr
              | Sin of expr
              | Cos of expr
              | Tan of expr
              | Arctan of expr
              | Exp of expr
              | Ln of expr
              | Power of expr * expr

type env = string -> real
```

```
exception Unbound of string
val emptyEnv: env = fn s => raise (Unbound s)
fun extendEnv oldEnv s n s' = if s' = s then n else oldEnv s'

exception FunctionUndefinedAtArgument of string
```

Note that we have an explicit `Let` construct, similar to a *where* clause in ordinary mathematical usage. The environment type `env` can be used to supply values for the free variables of an expression ("parameters", in mathematical parlance).

Usage example: the mathematical expression $\sqrt{2\pi}\frac{\sin x}{x}$ can be represented as

```
Mul (Power (Mul (IntNum 2, ConstPi), Div (IntNum 1, IntNum 2)),
     Div (Sin (Var "x"), Var "x")
    ) : expr
```

All the code should be placed in a single structure `RealFunctions: REALFUNCTIONS`. Carefully design the signature `REALFUNCTIONS` so that it specifies no more components than absolutely necessary.

5.1.1 Evaluator (10pts)

Write the ML function:

```
evalExpr: {
           func: expr,
           env: env
          } -> real
```

which evaluates the expression `func` in the environment `env`. If there exists some variable that is free in the expression `func` but not defined in the environment `env`, the ML function `evalExpr` should raise the exception `Unbound`.

Usage example: evaluating the mathematical function $x \mapsto e^{-x^2}$ at the point $x = 2.3$ can be represented as

```
evalExpr {func = Exp (Neg (Power (Var "x", IntNum 2))),
          env = extendEnv emptyEnv "x" 2.3
         }
```

which should evaluate to $5.04 \cdot 10^{-3}$.

5.1.2 Drawing (40pts)

Write the ML function:

```
plotExprs: {
            fies: {func: expr, indep: string, env: env} list,
            interval: {lower: real, upper: real},
            numPoints: int
           } -> string
```

which is a combination of an expression evaluator and a function plotter that generates PostScript output. The ML function `plotExprs` should plot each supplied function expression over the given interval with the given number of internal points. The `indeps` provided are the variables that are to be considered as the independent variables of their respective `funcs`. The value of each such independent variable ranges over the same given `interval`; note that the particular variable name may be different in each of the function expressions. The `envs` should provide the values for any other free variables of each of the `funcs`.

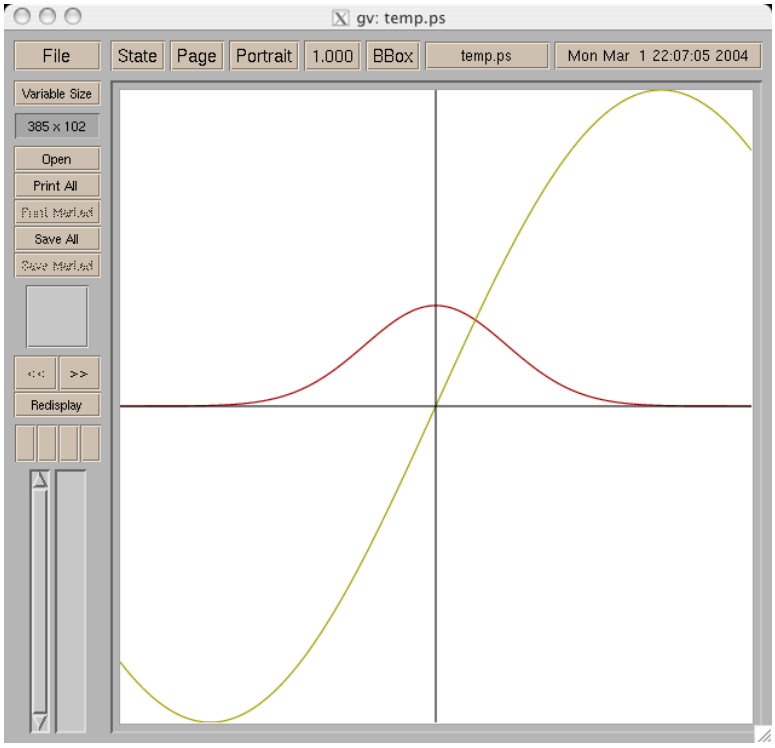
The ML function `plotExprs` should handle singularities gracefully and still produce an appropriate plot. Hint: we suggest two approaches: one is building into the evaluator certain knowledge of the behavior of elementary real functions, such as, e.g., that the function ln is defined only for positive arguments; and the other is relying on the behavior of SML Basis Library functions from the structure `Real` and inspecting their results.

The drawing area of the resulting PostScript program should be *consistent* for functions with arbitrary domains and ranges; to make this precise, we insist that the bounding box in the generated PostScript must be 0 0 500 500, and the plots should fully use the drawing rectangle.

Usage example: to plot the mathematical functions $x \mapsto e^{-x^2}$ and $t \mapsto A \sin \omega t$ with parameter values $A = \pi$ and $\omega = 0.7$ on the interval $[-\pi, \pi]$, invoke

```
plotExprs {
           fies =
             [
              {
               func = Exp (Neg (Power (Var "x", IntNum 2))),
               indep = "x",
               env = emptyEnv
              },
              {
               func = Mul (Var "A", Sin (Mul (Var "omega", Var "t"))),
               indep = "t",
               env = extendEnv (extendEnv emptyEnv "A" Math.pi) "omega" 0.7
              }
             ],
           interval = {lower= ~Math.pi, upper= Math.pi},
           numPoints = 1000
          }
```

The result should look like:



5.2 Programming language semantics [C; K.1.1; K.1.3; K.1.4; K.2.5; K.2.7; K.3.1] (50pts)

The goal is to add new features to the language PCF discussed in class. For each new feature, do the following: extend the static and dynamic semantics to treat the new syntactic forms; extend the scanner and parser as needed; extend the type checker and the interpreter; write programs in the extended language; and test.

Carefully design the various modules you will need for this exercise. The code provided to you is in the ML core language, and it must be reorganized to take advantage of ML modules. Use your experience from previous CS classes to come up with a good design.

The code must be built using the SML/NJ Compilation Manager (instead of manually loading modules using `use`).

5.2.1 No new features (10pts)

Rewrite the provided code to use ML modules.

5.2.2 Let expressions (10pts)

We extend the syntax of expressions by allowing a let-binding:

$e \longrightarrow$ `let val` x `=` e_1 `in` e_2 `end`

The intended meaning is just like the `let` in ML (but there is no polymorphism involved).

5.2.3 List primitives (20pts)

We extend the syntax of types as follows:

$\tau \longrightarrow$ `list(` τ `)`

We extend the syntax of expressions as follows:

$e \longrightarrow$ `nil(` τ `)`

$e \longrightarrow$ `cons(` τ `,` e_1 `,` e_2 `)`

$e \longrightarrow$ `null(` τ `,` e `)`

$e \longrightarrow$ `hd(` τ `,` e `)`

$e \longrightarrow$ `tl(` τ `,` e `)`

Thus we can write, for instance:

```
fun len (l: int list): int is
   if int null(int,l) then 0 else +(1,len(tl(int,l))) fi
end
```

The intended meaning is that these list primitives implement homogeneous lists, as in ML. The argument τ to the primitives is always the list element type. In some sense, the primitives are arbitrarily overloaded with respect to the list element type, but there is no polymorphism involved.

5.2.4 Together (10pts)

Both let expressions and list primitives are added.

5.2.5 Polymorphism (100pts extra credit)

Extend the syntax of types as follows:

$\tau \longrightarrow \alpha$

5.2.6 Compilation (100pts extra credit)

Translate PCF to the list stack machine code from Homework 4.3.1.

How to turn in

Turn in your code by running

~jackmp/cs451TA/handin your-file

on a regular UNM CS machine. You should use whatever filename is appropriate in place of your-file.

Include the following statement with your submission, signed and dated:
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.