Homework 2 — ML core language — assigned Sunday 6 February — due Wednesday 16 February

Reading assignment

Read Chapters 1, 2, and 3 of *ML for the Working Programmer*.

2.1 Using lists for arithmetic: writing recursive functions over lists (30pts)

Numerals can be represented as lists of integers. For instance, decimal numerals can be expressed as lists of integers from 0 to 9. In this representation, the integer 12345678901234567890 would be represented as the ML list `[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]: int list`.

Write the following functions:

- (10pts) `makeLongInt: int -> int -> int list`, such that `makeLongInt` *r n* computes the list representation of the integer *n* in radix *r*. You can assume that $n \geq 0$, and that $r > 1$. For example, `makeLongInt 10 123` should evaluate to `[1,2,3]`.
- (10pts) `evaluateLongInt: int -> int list -> int`, such that `evaluateLongInt` *r l* computes an integer corresponding to the value of list *l*, which uses radix *r*. You can assume that *l* is a valid list for radix *r*, and the value of the list is small enough to fit into an ML `int`. For example, `evaluateLongInt 10 [1,2,3]` should evaluate to `123: int`.
- (10pts) `addLongInts: int -> (int list * int list) -> int list`, such that `addLongInts` $r\ (a,b)$ computes the sum of the nonnegative integers given by lists *a* and *b*; all three lists use radix *r*. For example, `addLongInts 10 ([1,2,3], [1])` should evaluate to `[1,2,4]: int list`. Important: the argument lists *a* and *b* can represent arbitrarily large integers, unlike ML's built-in `int` type.

Carefully state all preconditions for all function arguments. Make sure that your tests cover the space of permissible argument values well.

2.2 Drawing: writing recursive functions over lists; manipulating strings (15pts)

In this exercise, we develop some simple tools for drawing.

A drawing is just a line drawing consisting of some number of polygons. A polygon is given as a list of vertices, and a vertex is simply a pair of real numbers for the *x* and *y* coordinates.

For instance,

```
[[(100.0,100.0),(100.0,200.0),(200.0,100.0)],
[(150.0,150.0),(150.0,200.0),(200.0,200.0),(200.0,150.0)]]
```

is an internal representation in ML of a drawing consisting of a triangle and a square.

Your task is to convert such a representation of a drawing into a simple page description in the PostScript language. Specifically, you are to write an ML function `makeCommand: (real * real) list list -> string`.

The result returned by `makeCommand` is an ML value of type string, which must contain valid PostScript commands for drawing the given polygons. When the SML/NJ top level prints this result, paste it by hand into a file *result.ps*, and then display it by running GhostScript: `gs result.ps`.

For instance, the expression

```
makeCommand [[(100.0,100.0),(100.0,200.0),(200.0,100.0)],
[(150.0,150.0),(150.0,200.0),(200.0,200.0),(200.0,150.0)]]
```

should evaluate to the string:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 100.0 100.0 200.0 200.0

100.0 100.0 moveto
100.0 200.0 lineto
200.0 100.0 lineto
closepath
stroke

150.0 150.0 moveto
150.0 200.0 lineto
200.0 200.0 lineto
200.0 150.0 lineto
closepath
stroke

showpage
%%EOF
```

which will be displayed as in Figure 1.

Note that the bounding box is the smallest upright rectangle such that no points of the drawing lie outside it; it is specified by giving the coordinates of its lower left and upper right corners, in our example $(100.0, 100.0)$ and $(200.0, 200.0)$.

The example shown here entirely suffices as a pattern to follow; however, if you would like to learn more about the PostScript language you can follow the links on the course web page.

Hint: you need to learn how to control the top level of SML/NJ so that it does not truncate very long strings when it prints them.

Optional: learn how to do file output in ML, and produce the output file directly from ML instead of cutting and pasting. Implement a function `dumpStringToFile: string * string -> unit`, which takes two strings; the first is the name of the file to be written, and the second is the string that is to be dumped to the file.

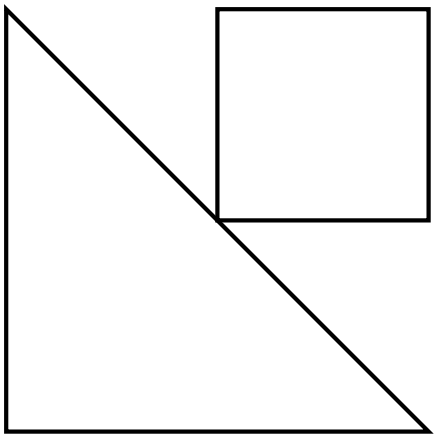

Figure 1: A triangle and a square.

2.3 Using lists for sets: writing recursive functions over lists (35pts)

Let us use the ML type `int list` to represent sets of integers. In this representation, there must be no duplicates in the list, and the order of the list elements is immaterial.

- (5pts) Write an ML function `union: int list * int list -> int list` that takes two sets and returns their union.
- (5pts) Write an ML function `intersection: int list * int list -> int list` that takes two sets and returns their intersection.
- (5pts) Write an ML function `difference: int list * int list -> int list` that takes two sets and returns their set difference.
- (5pts) Write an ML function `equal: int list * int list -> bool` that takes two sets and returns `true` if and only if the two sets are equal.
- (15pts) Write an ML function `powerset: int list -> int list list` that takes a set S and returns its powerset 2^S. (The powerset 2^S of a set S (sometimes written $P(S)$) is the set of all subsets of S.) Note that the result uses the ML type `int list list` to represent sets of sets of integers.

2.4 Using lists for text (20pts)

Write a function `split_into_words` to split text into words. Spaces, tabs, and new lines are word separators.

The function you write will replace the comment in the code fragment below, to make the whole work.

```
local
   fun acceptfile (fileName: string) : string =
      let
```

```
         val f = TextIO.openIn fileName
         val s = TextIO.inputAll f
         val _ = TextIO.closeIn f
      in
         s
      end
   type word = char list
   type sentence = char list
   (* your code goes here *)
   fun split_into_words (s: sentence): word list =
   (* begin your code *)

   (* end your code *)
in
   fun main (filename: string) =
      let
         val s = acceptfile filename
         val cl = explode s
         val wl = split_into_words cl
      in
         wl
      end
end
```

The declaration of `split_into_words` should be expressed in terms of the language primitives (i.e., without the use of library functions).

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.