Programming Languages
CSCI-GA.2110.001 FALL 2020

ML Assignment
Due Sunday, November 29 at 11:55pm

Your assignment is to write in ML, using the SML/NJ system, a series of simple definitions (types and functions) for computing on lists and trees. Obviously, the code should be purely functional. Also, you must not look online for the solutions (nor get the code from any other source).

You should make use of ML's pattern matching facility for defining functions and extracting elements from lists and tuples. For example, do not use the `hd` and `tl` functions to extract the head and tail of a list. Also, if you need to check if a list contains a single element, use a pattern (e.g.,`[x]`) rather than the `length` function.

Finally, be sure to read the section at the bottom of this assignment labeled "Important Notes".

1. Write a function `foo` whose type is

   ```
   ('a -> 'b) -> ('b list -> 'c) -> 'a -> 'c list
   ```

   This is easy to do in a single line.

2. Write a function `bar x`, where `x` is an integer, that returns a function that takes a list `L` of integers and returns a list each of whose elements result from multiplying the corresponding element of `L` by `x`. Do not use the built-in `map` function. Rather, write any recursion that is needed yourself. You should not define any function outside of `bar`.

   Some examples of the use of `bar` (in the SML/NJ system) are:

   ```
   - val g = bar 30 ;
   val g = fn : int list -> int list
   - g [1,2,3,4,5] ;
   val it = [30,60,90,120,150] : int list
   - bar 30 [1,2,3,4,5] ;
   val it = [30,60,90,120,150] : int list
   ```

   `bar` can be written in 4 or 5 lines (or less).

3. In preparation for writing a partition sort function (below), define a function `part` that takes an integer `x` and an integer list `L` as parameters and partitions `L` based on `x`. That is, `part` returns a tuple of two lists, such that the first list contains all elements of `L` less than `x` and the second list contains the elements of `L` that are greater than or equal to `x`.

   An example of the use of `part` is:

   ```
   - part 6 [5,2,8,4,1,9,6,10];
   val it = ([5,2,4,1],[8,9,6,10]) : int list * int list
   ```

   The `part` function can be written in roughly 5 lines (or less).

4. Implement a partition sorting function, `partSort` which uses your `part` function, above, to sort a list `L` of integers, returning a sorted list of the elements of `L`. A partition sort is essentially Quicksort, but without the property of using only a constant amount of space. The algorithm, given a list `L`, is:

- If `L` is empty or contains a single element, return `L` (use pattern-matching for these two cases, not a conditional).
- Otherwise, assuming `L` is of the form `(x::xs)`, partition `xs` into two lists based on `x` by calling your `part` function.
- Recursively call `partSort` on each of the two lists, to sort each of the lists.
- Return a single sorted list containing the elements from the first sorted list, followed by `x`, followed by the elements of the second sorted list.

An example of the use of `partSort` is:

```
- partSort [5,2,9,10,12,4,8,1,19];
val it = [1,2,4,5,8,9,10,12,19] : int list
```

The `partSort` function should be no more than 6 lines.

5. Implement a polymorphic partition sort function, `pSort`, that will sort a list of elements of any type. In order to do this, though, `pSort` must take an additional parameter, the `<` (less-than) operator, that operates on the element type of the list. For example, two uses of `pSort` would be:

```
(* Using the built-in < for comparing integers. The compiler is
   smart enough to figure out which < to use *)
- pSort (op <) [1,9, 3, 6, 7];
val it = [1,3,6,7,9] : int list

(* sorting a list of lists, where the less-than operator compares
   the length of two lists *)
- pSort (fn(a,b) => length a < length b) [[1, 9, 3, 6], [1], [2,4,6], [5,5]];
val it = [[1],[5,5],[2,4,6],[1,9,3,6]] : int list list
```

Note that you will also need to create a helper function for partitioning the list (analogous to the `part` function you wrote above). This time, though, nest the helper function <u>inside</u> the `pSort` function using `let`. Be sure to use `<` as an infix operator inside of `pSort`.

`pSort` can be written in roughly 12 lines, including the nested function.

6. Write a function `reduce` that takes a function `f` and a list `L`, such that if `L` is of the form `[`x_1`, `x_2`, ..., `x_{n-1}`, `x_n`]`, then `reduce` returns the result of `f` x_1 `(f` x_2 `...  (f` x_{n-1} x_n`)...)` For example,

```
- fun g x y = x + y ;
val g = fn : int -> int -> int
- reduce g  [1,2,3,4,5] ;
val it = 15 : int
- reduce (fn x => fn y => x + y) [1,2,3,4,5] ;
val it = 15 : int
```

Note that `f` must be a (curried) function of two arguments, such as you see in the definition of `g` and the lambda expression above. If `L` only has one element, then just that element should be returned (i.e. `f` is never called). If `L` is the empty list, `reduce` should raise an exception (see "Important Notes" below). For example,

```
reduce g [3];
val it = 3 : int
- reduce g [];

uncaught exception reduce_error
  raised at: stdIn:9.1-9.13
```

You don't have to handle the raised exception. Your `reduce` function should be roughly 3 lines.

7. Define a polymorphic tree datatype (i.e. `datatype 'a tree = ...`) such that a leaf is labeled with an `'a` and an interior node has a list of children, each of type `'a tree`. That is, each interior node can have an arbitrary number of children, rather than just two (as in a binary tree). For example, your datatype declaration should allow the following tree to be constructed:

```
val myTree = node [node [node [leaf [4,2,14],leaf [9,83,32],leaf [96,123,4]],
                         node [leaf [47,71,82]],node [leaf [19,27,10],
                                                      leaf [111,77,22,66]],
                         leaf [120,42,16]],
                   leaf [83,13]]
```

Your datatype definition should be one line.

8. Define a function `fringe t`, where `t` is an `'a tree` (above), which computes the fringe of `t`. The fringe of a tree is a list of the values at the leaves of a tree. For example,

```
- fringe (node [leaf 1,node [leaf 2,leaf 3],node [node [leaf 4,leaf 5],leaf 6]]) ;
val it = [1,2,3,4,5,6] : int list

- fringe myTree;
val it =
  [[4,2,14],[9,83,32],[96,123,4],[47,71,82],[19,27,10],[111,77,22,66],
   [120,42,16],[83,13]] : int list list
```

You should use your `reduce` function (along with `map`), allowing `fringe` to be written in roughly three lines.

9. Write a polymorphic function `sortTree (op <) t`, where `t` is of type `'a list tree`, that returns a tree with the same structure as `t`, except that the list found at each leaf has been sorted using your `pSort` function, above. For example,

```
(* Again, the compiler is smart enough to figure out that (op <) is the
   integer version of <. *)
- sortTree (op <) myTree ;
val it =
  node
    [node
       [node [leaf [2,4,14],leaf [9,32,83],leaf [4,96,123]],
        node [leaf [47,71,82]],node [leaf [10,19,27],leaf [22,66,77,111]],
        leaf [16,42,120]],leaf [13,83]] : int list tree
```

10. Suppose that a set is represented in ML as a list. Define the function `powerSet L`, where `L` is of type `'a list`, that returns the power set of `L`. That is, it returns the list of all possible sub-lists of `L`. For example,

    ```
    - powerSet [1,2,3] ;
    val it = [[1,2,3],[1,2],[1,3],[1],[2,3],[2],[3],[]] : int list list
    ```

    This function can be written in around 5 lines or so. Recursive thinking is particularly important here. Hints: The base case, when `L` is the empty list, returns `[[]]`, the list containing the empty list. Think about the relationship between the power set of `L` and the power set of the tail (cdr) of `L`.

Important Notes

- You should put your code in a file with a `.sml` extension. To load a file containing ML code into the SML/NJ system, type

  ```
  use "filename.sml";
  ```

  When you are finished with the assignment, upload just the file containing your definitions. Be sure to use exactly the same function and type names as specified above.

- Put the following lines at the top of your file, to tell the SML/NJ system the maximum depth of a datatype to print and the maximum length of a list to print.

  ```
  Control.Print.printDepth := 100;
  Control.Print.printLength := 100;
  ```

  If you don't put these lines in your file, the system will only print a limited number of elements of a list, or to a limited depth in a datatype (such as a tree), after which it prints `#` to save space.

  Important: You need to include the semicolons in the two above lines. However, these are the only semicolons that should appear in your file. You should not have any other semicolons in your code.

- To define an exception in ML (in its simplest form), you write
  `exception` *exception_name*

  To raise the exception, you write

  `raise` *exception_name*

  For example,

  ```
  exception DivByZero

  fun myMod x 1 = raise DivByZero
   |  myMod x y = x mod (y-1)
  ```

  See the ML online references for defining exceptions that take parameters and for handling exceptions (neither of which you need to do for this assignment).

- To declare a function that takes a function as a parameter, such that the parameter is used as an infix operator, use the `(op ...)` syntax as shown below.

```
- fun f (op <) x y = if x < y then "less" else "not less";
val f = fn : ('a * 'b -> bool) -> 'a -> 'b -> string

- (* passing a lambda expression as the < operator *)
f (fn(a,b) => (a*a) < (b*b)) ~4 3;
val it = "not less" : string
```

Note that negative numbers are written using `~`, not `-`.

- Be mindful of the difference between a function that takes two parameters, e.g.

  ```
  fun f x y = x + y
  ```

  and a function that takes a single parameter that is a tuple with two elements, e.g.

  ```
  fun g (x,y) = x + y
  ```

  Any function used as an infix operator, e.g. passed as a parameter to a function such as

  ```
  fun h (op <) x y = x < y
  ```

  must take a tuple as a parameter.

- Here is a silly one: Do <u>not</u> use `(a::as)` as a pattern for matching a list. Why? Because `as` is a keyword used in ML, for a feature we did not discuss. For example, the following, perfectly reasonable, definition of `length` generates a compiler error.

  ```
  -   fun length [] = 0
       |  length (a::as) = 1 + length as
  ;
  = = stdIn:2.20-2.23 Error: syntax error: deleting  RPAREN EQUALOP
  ```

- Do not copy-and-paste the sample code from this PDF document into your program or into the SML system, since some symbols may be stored differently in PDF than in plain text. I have provided a plain text version of the sample code along with this PDF.