Homework 4: Functional programming

Due: Wednesday, Feb. 25.

This homework can either be done individually (which I suggest) or in groups. At this point in the course, it seemed best to still do some smaller problems. We'll do some larger functional programs later, when we do semantics.

For all SML programs, you must run your code with SML. You must also provide evidence that your program is correct (for example, test cases). Hand in a printout of your code and your testing.

You should program in a purely functional style (for example, don't use **ref**).

1. Consider the following higher-order function.

```
fun iter n f a = if n=0 then a else f(iter(n-1) f a);
```

   (a) (5 points) Given a function f, what is (iter 0 f) equal to?

   (b) (10 points) Define a function `power: int -> int -> int` such that (power n k) $= n^k$ by using `iter`, and without using recursion.

   (c) (10 points, extra credit; applications—number theory, semantics, programming) Using only `iter`, small numeric constants, function definitions, and an increment function, define Ackerman's function.

2. Consider the following higher-order function.

```
fun itlist [] f b = b
  | itlist (a::lis) f b = f (a, itlist lis f b);
```

   (a) (5 points) Define the `length: 'a list -> int` function using `itlist`, and without using recursion.

   (b) (10 points) Using `itlist`, define nonrecursively the function `flat: 'a list list -> 'a list` that appends all the lists in its argument in order. For example,

   `flat([[1,2],[3],[4,5,6],[]])`

   returns `[1,2,3,4,5,6]`.

   (c) (15 points, extra credit; programming) Using `itlist`, define nonrecursively the function `permut: 'a list -> 'a list list` that returns a list of all permutations of its argument. For example, `permut([1,2])` returns `[[1,2],[2,1]]`.

   (d) (20 points, extra credit; programming) Using functions like `iter` and `itlist`, define a function `prodlis: real list * real list -> (int*int -> real)` such that

   prodlis([$a_0, a_1, \ldots, a_{n-1}$],[$b_0, b_1, \ldots, b_{k-1}$])

   is a function f, such that for all $0 \le i < n$, $0 \le j < k$, $f(i,j) = a_i \cdot b_j$.

3. (10 points, extra credit; applications—software engineering, semantics) Formally prove one of the above programs you have written is correct. See chapter 6 of Paulson's book.

4. (15 points, extra credit; programming) John Backus was the principal scientist on the team that invented FORTRAN. In 1978 the Association for Computing Machinery gave him its highest award, the Turing Award. In his influential Turing Award lecture, "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs" (*Communications of the ACM*, Volume 21, Number 8, August 1978, pages 613–641), he gives reasons for abandoning languages like FORTRAN and in favor of functional programming.

   Read this paper (it's in the library, but don't take it away, so that others can read it too). Code up enough of Backus's primitives to define matrix multiplication in his style.

5. Consider the following SML datatype declaration.

```
datatype 'a bush = EmptyBush
  | FlatBush of 'a * ('a bush list)
  | BushyBush of ('a bush list) * ('a bush list);
```

   (a) (10 points) Define a function `sum_bush: int bush -> int` that takes a bush of integers and returns their sum. For example, `sum_bush(EmptyBush)` returns 0, and

   ```
   sum_bush(FlatBush(3, [FlatBush(4,[EmptyBush])]))
   ```

   returns 7. A more complex example is:

   ```
   sum_bush(FlatBush(3, [EmptyBush,
                          BushyBush([FlatBush(4,[EmptyBush])],
                                    [BushyBush([EmptyBush],[EmptyBush])]),
                          FlatBush(5, [FlatBush(3, [])])]));
   ```

   this example returns 15.

   (b) (10 points) Define a function `bush_subst_all: ''a * ''a * ''a bush -> ''a bush` that takes two elements of an equality type, `new`, and `old` (with `new` being the first of the three arguments), and a bush, and returns a bush that is just like the argument bush, but with each occurrence of `old` replaced by `new`.

   (c) (20 points) Define a function `bush_wacker` that is an abstraction of the above two examples, and show how you can use `bush_wacker` to define both `sum_bush`, `bush_subst_all`, and a function `prod_bush` that returns the product of all the numbers in an `int bush`. That is, `bush_wacker` should be a functional such that for some arguments you can write the following definitions:

   ```
   val sum_bush = bush_wacker ...;
   val bush_subst_all = bush_wacker ...;
   ```

6. (10 points) Do exercise 5.27 in Paulson's book.

7. (10 points) Do exercise 5.30 in Paulson's book.

8. (5 (or more) points each, extra credit only) Do whatever problems interest you from Paulson's book, chapters 5-8 (inclusive). You will receive 5 points for each. If you want to do a problem that you think is worth more than 5 points of extra credit, suggest a number of points to Gary. (You can use `ref` for appropriate problems in chapter 8.)