ML 1 – Pattern Matching and Recursion

CS 421 – Fall 2015

Revision 1.2

Assigned September 1, 2015
Due September 8, 2015 – September 10, 2015
Extension None past the allowed lab sign-up time

1 Change Log

1.2 Corrected the termination condition in problem 8 to say "If $x < 1$, then return 0."

1.1 Corrected the description of `pow2` in problem 9; added condition to `approx_exp` in problem 8 to cover when $x < 1$; made all recursive problems 4 points (adding two points to each of problems 7 and 10).

1.0 Initial Release, more instructions on the Computer-Based Testing Facility will follow.

2 Objectives and Background

The purpose of this ML is to help the student master:

- pattern matching
- recursion

3 Done in Computer-Based Testing Facility

Next week, you will be asked to sign up for a time to go to the CBTF, where you will be given a random portion (less than 25%) of this assignment to complete for your grade for assignment. We recommend that you do the whole assignment beforehand to be comfortable with the problems, so you will be able to to the problems efficiently when you go in. To help yourself in that, please read the *Guide for Doing MPs* at

`http://courses.engr.illinois.edu/cs421/mps/index.html`

Be aware that if we have difficulties with the CBTF, this assignment will revert to an MP and will be turned in, in full, in the same manner as MP1.

4 Instructions for Doing the Problems

The problems below have sample executions that suggest how to write answers. You have to use the same function name, but the name of the parameters that follow the function name, if any, need not be duplicated. That is, you are free to choose different names for the arguments to the functions from the ones given in the example execution. We will sometimes use `let rec` to begin the definition of a function that is allowed to use `rec`. You are not required to start your code with `let rec`, and you may use `let rec` when we do not.

For all these problems, you are allowed to write your own auxiliary functions, either internally to the function being defined or externally as separate functions. In fact, you will find it helpful to do so on several problems.

5 Problems

1. (2 pts) Write a function `truth_total : bool * bool -> int` that takes a pair of booleans and returns how many of them are true.

   ```
   let truth_total = ... ;;
   val truth_total : bool * bool -> int = <fun>
   # truth_total (false,false);;
   - : int = 0
   ```

2. (2 pts) Write a function `drop_two : 'a * 'b * 'c * 'd -> 'b * 'd` that takes a 4-tuple and returns a pair of its second and fourth component.

   ```
   let drop_two = ... ;;
   val drop_two : 'a * 'b * 'c * 'd -> 'b * 'd = <fun>
   # drop_two (1,2,3,4);;
   - : int * int = (2, 4)
   ```

3. (2 pts) Write a function `pair_min_max : 'a * 'a -> 'b * 'b -> bool -> 'a * 'b` that, when given two pairs and a boolean, returns a pair where:

   - If the boolean is true, the first element is the minimum of the elements in the first pair and the second element the maximum of the elements in the second pair, and
   - Otherwise, the first element is the maximum of the elements in the first pair and the second element the minimum of the elements in the second pair.

   You may use `max : 'a -> 'a -> 'a` and `min : 'a -> 'a -> 'a` to find the maximum and minimum of two values.

   ```
   let pair_min_max (x,y) (a,b) flip = ...;;
   val pair_min_max : 'a * 'a -> 'b * 'b -> bool -> 'a * 'b = <fun>
   # pair_min_max (1,2) (3,4) true;;
   - : int * int = (1, 4)
   ```

4. (2 pts) Write a function `apply_to_pair : ('a -> 'b -> 'c) -> 'a * 'b -> 'c` that takes a function f of at least two arguments and a pair and returns the value obtained by applying the function to the components of the pair.

   ```
   let apply_to_pair = ... ;;
   val  apply_to_pair :  ('a -> 'b ->  'c) -> 'a * 'b -> 'c = <fun>
   # apply_to_pair (fun x -> fun y -> x + y) (1,2);;
   - : int = 3
   ```

5. (2 pts) Write a function `test_swap : ('a -> bool) -> 'a * 'a -> 'a * 'a` that takes a test function and a pair, applies the test to the first component of the pair, and if the result is `true` returns the pair swapped around. Otherwise, return the pair as given.

```
let test_swap test (a,b) = ...;
val test_swap : ('a -> bool) -> 'a * 'a -> 'a * 'a = <fun>
# test_swap (fun x -> x > 5) (10, 15) ;;
- : int * int = (15, 10)
```

6. (4 pts) Write a function `sum_between : int -> int -> int` that when given integers x and y, regardless of the relative values of x and y, returns the sum of the integers between x and y inclusive.

```
# let sum_between x y = ...
val sum_between : int -> int -> int = <fun>
# sum_between 1 3;;
- : int = 6
```

7. (4 pts) Write a function `is_divis : int -> int -> bool` that takes two numbers, n and m (in that order), and checks whether n is divisible by any of the numbers from 2 to m, inclusive. If $m \leq 1$ or $n \leq 1$, you should return false. You may want to use the infix operation `mod` for testing divisibility.

```
let is_divis = ... ;;
val is_divis : int -> int -> bool = <fun>
# is_divis 4 2;;
- : bool = true
```

8. (4 pts) Write a function `approx_exp : float -> int` that, when applied to a value x, returns the smallest integer n such that $(\Sigma_{k=1}^{n}\frac{1}{k}) \leq x$. If $x < 1$, then return 0. You may want to use `float_of_int : int -> float` to convert an integer into the corresponding float.

```
# let approx_exp x = ...
val approx_exp : float -> int = <fun>
# approx_exp 1.5;;
- : int = 2
```

9. (4 pts) Write a function `pow2 : int -> int` such that, when applied to a value x, returns the largest non-negative integer n such that $2^n \leq x$, if $x \geq 1$, and 0 otherwise. You may want to use the infix operation `/` for integer division.

```
# let rec pow2 x = ...;;
val pow2 : int -> int = <fun>
# pow2 3;;
- : int = 1
```

10. (4 pts) Write a function `sudan : int -> int * int -> int` that, when given a non-negative integer n and a pair of non-negative integers (x, y), does the following:

$$F_n(x,y) = x + y \text{ if } n = 0$$
$$F_n(x,0) = x \text{ if } n > 0$$
$$F_n(x,y) = F_{n-1}(F_n(x,y-1), F_n(x,y-1)+y) \text{ if } n > 0 \text{ and } y > 0$$

```
let sudan n (x,y) = ...;;
val sudan : int -> int * int -> int = <fun>
# sudan 0 (0,0);;
- : int = 0
```