CS 2SC3 and SE 2S03 Fall 2008

Programming Exercise 5

Instructor: William M. Farmer

Revised: 26 November 2008

Files due: 4 December 2008

The purpose of this programming exercise is to learn how to build procedures using recursion and higher-order procedures.

Background

Let the *iterator* on a set A be the 3-ary higher-order function

$$i : (A \to A), \mathbf{N}, A \to A$$

defined by:

$$i(f, n, a) = \begin{cases} \text{undefined} & \text{if } n < 0 \\ a & \text{if } n = 0 \\ f^n(a) & \text{if } n > 0 \end{cases}$$

$f^n(a)$ is the result of iteratively applying the function f n times to the value a. For instance, $f^3(a) = f(f(f(a)))$.

As an example, let i be the iterator on $\mathbf{N}$ and suc be the successor function $(\lambda\, x : \mathbf{N} \,.\, x + 1)$. Then

$$i(\mathsf{suc}, n, m) = m + n.$$

That is, addition on $\mathbf{N}$ is defined by applying the iterator on $\mathbf{N}$ to the successor function suc. For instance,

$$i(\mathsf{suc}, 3, 2) = \mathsf{suc}(\mathsf{suc}(\mathsf{suc}(2))) = ((2 + 1) + 1) + 1 = 5.$$

Part A

Write an OCaml program that satisfies the requirements listed below. Put your iterator code in a file named `iterator5.ml`, your test code in a file named `iterator_test5.ml`, and your log book in a file named `log.txt`. Put all three of these files into a directory named `ex-5-a`. Using subversion, import this directory into your directory in the course subversion repository. Your files must be submitted no later than **10:30 a.m. on Thursday, December 4, 2008.**

Part B

There is no Part B.

Program Requirements

1. The programs includes the following three implementations of an iterator on an arbitrary set:

   (a) `iterator_for : ('a -> 'a) -> int -> 'a -> 'a` that is implemented using a for loop.

   (b) `iterator_nontail : ('a -> 'a) -> int -> 'a -> 'a` that is implemented using nontail recursion.

   (c) `iterator_tail : ('a -> 'a) -> int -> 'a -> 'a` that is implemented using tail recursion.

2. An exception `Negative_argument` is raised when one of the iterators above is applied to arguments f, n, a where n is a negative integer.

3. The program includes a function

   ```
   plus_maker :
      ((int -> int) -> int -> int -> int)
        -> int -> int -> int
   ```

   that builds, when given one of the iterators above, the addition function on `int`. `plus_maker` defines the addition function by iterating the successor function as shown in Background section above.

4. The program includes three versions of the addition function — named `plus_for`, `plus_nontail`, and `plus_tail` — build by applying `plus_maker` to `iterator_for`, `iterator_nontail`, and `iterator_tail`, respectively.

5. The program includes a function

   ```
   times_maker :
      ((int -> int) -> int -> int -> int)
        -> int -> int -> int
   ```

   that builds, when given one of the iterators above, the multiplication function on `int`. `times_maker` defines the multiplication function by iterating the addition function defined above.

6. The program includes three versions of the multiplication function — named `times_for`, `times_nontail`, and `times_tail` — build by applying `times_maker` to `iterator_for`, `iterator_nontail`, and `iterator_tail`, respectively.

7. The program includes a function

```
exp_maker :
   ((int -> int) -> int -> int -> int)
     -> int -> int -> int
```

that builds, when given one of the iterators above, the exponentiation function on `int`. `exp_maker` defines the exponentiation function by iterating the multiplication function defined above.

8. The program includes three versions of the multiplication function — named `exp_for`, `exp_nontail`, and `exp_tail` — build by applying `exp_maker` to `iterator_for`, `iterator_nontail`, and `iterator_tail`, respectively.

9. The program tests the implementation of the iterators and the plus, times, and exponention makers by verifying that:

   (a) Each of `plus_for`, `plus_nontail`, and `plus_tail` is equal to the addition function on `int`.

   (b) Each of `times_for`, `times_nontail`, and `times_tail` is equal to the multiplication function on `int`.

   (c) Each of `exp_for`, `exp_nontail`, and `exp_tail` is equal to the exponentiation function on `int`.

Notes:

1. Please put your name and MAC ID at the top of each of your files.

2. Your programs must be your own work.

3. Your programs must compile and execute correctly on either mills or moore to receive full marks.

4. Files submitted late will receive no marks.