CS 2500 Exam 2 HONORS SUPPLEMENT – Fall 2013

Your Name: ____________________________

Instructor: ____________________________

• This supplement to Exam 2 is intended for students enrolled in the Honors section of 2500.

• See the instructions on the regular exam, but keep in mind that specific instructions on any given problem override the general instructions on the regular exam. Also, you may use lambda or local as needed.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>out of</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>40</td>
</tr>
</tbody>
</table>

*Good luck!*
Problem 1  Design the function `concat`, which consumes a list of lists and appends them all to produce a single list. Give `concat` its most general signature and define it using a loop function. You may not use `append` or `apply`. 
Problem 2 A sequence represents a series of values. Sequences may be finite or infinite. In this problem, we’ll work with infinite sequences.

Here are three examples of infinite sequences:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive integers</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>even natural numbers</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>lists of 'a</td>
<td>()</td>
<td>'(a)</td>
<td>'(a a)</td>
<td>'(a a a)</td>
<td>...</td>
</tr>
</tbody>
</table>

Here is a data definition for representing infinite sequences:

;;; A [Sequence X] is a [Natural -> X]
;;; interpretation: when the function is applied to an
;;; index (a Natural), it gives back the element at
;;; that index.

Here is an example of a [Sequence Natural], the even natural numbers:

(define even-nats (lambda (i) (* 2 i)))

Here is a convenient function for producing a list with the first \( n \) elements of an infinite sequence:

;;; seq->listn : [Sequence X] Natural -> [List X]
;;; Build a list with the first \( n \) elements of the
;;; sequence \( s \)
(define (seq->listn s n)
  (map s (build-list n (lambda (x) x))))

For example,

> (seq->listn even-nats 10)
(list 0 2 4 6 8 10 12 14 16 18)

You may use even-nats and seq->listn for tests, but they should not be used otherwise.
(a) (8 pts) Design the following functions:

- \texttt{seq-head}, which consumes a sequence \( s \) and returns its 0th element.
- \texttt{seq-rest}, which consumes a sequence \( s \) and returns a sequence with all but the 0th element of \( s \).
(b) (10 pts) A *series* for a sequence \( s \) gives the sums of the elements in \( s \). More precisely, adding the 0th through \( i \)th elements of an infinite sequence \( s \) forms the \( i \)th element of another infinite sequence, called a series.

For example, the series for the sequence of positive integers 1, 2, 3, 4, ... is: 1, 3, 6, 10, ... .

Design the function `seq->series`, which consumes a `[Sequence X]`, and a function for adding \( X \)s (with signature `[X X -> X]`), and produces a series for the given sequence.
Problem 3  Consider the following data definition for finite sequences:

```haskell
;; A [Maybe X] is one of:
;;  - 'undef
;;  - X

;; A [FiniteSeq X] is a [Sequence [Maybe X]]
;; Constraint: there exists some index i>0 such that
;;  - no elements at indices [0,i) equal 'undef
;;  - all elements at indices >= i equal 'undef
```

Informally, the above data definition allows us to represent a finite sequence 1, 2, 3 as the infinite sequence 1, 2, 3, 'undef, 'undef, 'undef, ...

(a) (2 pts) Define even-nats-4to8, an instance of [FiniteSeq Natural] that represents the sequence of even natural numbers in the range [4,8]—that is, the finite sequence 4, 6, 8.
(b) (12 pts) Design the function \texttt{fs-length}, which consumes a finite sequence and two natural numbers \texttt{lo} and \texttt{hi} and produces the length of the finite sequence. Assume that \texttt{lo} < \texttt{hi} and that there exists an index \texttt{i} in the range \([lo, hi)\) such that the element at index \texttt{i+1} is ’\texttt{undef} but the element at index \texttt{i} is not.

For example, for the finite sequence \texttt{even-nats-4to8} that you defined in part (a):

\begin{verbatim}
  > (fs-length even-nats-4to8 0 100)
  3
\end{verbatim}

To get credit for this problem, you will need to use an efficient generative recursion design.