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.

Problem	Points	/out of
1		/ 8
2		/ 18
3		/ 14
Total		/ 40

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`. 8 POINTS

Problem 2 A *sequence* represents a series of values. Sequences may be finite or infinite. In this problem, we'll work with infinite sequences. 18 POINTS

Here are three examples of infinite sequences:

index	0	1	2	3	...
positive integers	1	2	3	4	...
even natural numbers	0	2	4	6	...
lists of `'a`	`'()`	`'(a)`	`'(a a)`	`'(a a a)`	...

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:

- `seq-head`, which consumes a sequence `s` and returns its 0th element.
- `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 ith elements of an infinite sequence `s` forms the ith 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: 14 POINTS

```
;; 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 `fs-length`, which consumes a finite sequence and two natural numbers `lo` and `hi` and produces the length of the finite sequence. Assume that `lo < hi` and that there exists an index `i` in the range `[lo,hi)` such that the element at index `i+1` is `'undef` but the element at index `i` is not.

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

```
> (fs-length even-nats-4to8 0 100)
3
```

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