CS220 Programming Principles

Homework No. 7

Due: December 19, 2018 10:30AM (4x9 points)

Submit your racket file to TA.

Use the given hw07-code.scm file as the starting point of your homework.

Before you submit it, rename it to "hw07-your_student_number.scm."

Part 1 : Exercises

Exercise 1: Do exercise 3.51 of the SICP book, which takes a closer look at delayed evaluation and stream-enumerate-interval.

Exercise 2: Describe the streams produced by the following definitions. Assume that `integers` is the stream of non-negative integers (starting from 1):

```
(define A (stream-cons 1 (scale-stream 2 A)))

(define (mul-streams a b)
  (stream-cons
   (* (stream-first a) (stream-first b))
   (mul-streams (stream-rest a)
                (stream-rest b))))
(define B (stream-cons 1 (mul-streams B integers)))
```

Exercise 3: Given a stream **s** the following procedure returns the stream of all pairs of elements from s:

```
(define (stream-pairs s)
  (if (stream-empty? s)
      empty-stream
      (stream-append
        (stream-map
         (lambda (sn) (list (stream-first s) sn))
         (stream-rest s))
        (stream-pairs (stream-rest s)))))
```

(a) Suppose that integers is the (finite) stream {1, 2, 3, 4, 5}. What is `(stream-pairs s)`?
(b) Give the clearest explanation that you can of how stream-pairs works.
(c) Suppose that s is the stream of positive integers. What are the first few elements of `(stream-pairs s)`? Can you suggest a modification of stream-pairs that would be more appropriate in dealing with infinite streams?

Part 2: Working with Power Series

In section 2.5.3, we saw how polynomials could be represented as lists of terms. In a similar way, we can represent power series, such as

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3 \cdot 2} + \frac{x^4}{4 \cdot 3 \cdot 2} + \cdots$$

$$\cos x = 1 - \frac{x^2}{2} + \frac{x^4}{4 \cdot 3 \cdot 2} - \cdots$$

$$\sin x = x - \frac{x^3}{3 \cdot 2} + \frac{x^5}{5 \cdot 4 \cdot 3 \cdot 2} - \cdots$$

as streams of infinitely many terms. That is, the power series

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

will be represented as the infinite stream whose elements are $\{a_0, a_1, a_2, a_3, \ldots\}$ [1]

Implementing a Power Series as a Stream

We provide two ways to construct a series: `coeffs->series` and `proc->series`. The procedure `coeffs->series` takes a list of initial coefficients and pads it with zeroes to produce a powers series. For example, `(coeff->series '(1 3 4))` produces the power series $1 + 3x + 4x^2 + 0x^3 + 0x^4 + \ldots$.

```
(define (coeffs->series list-of-coeffs)
   (define zeros (stream-cons 0 zeros))
   (define (iter lst)
     (if (null? lst)
         zeros
         (stream-cons (car lst)
                      (iter (cdr lst)))))
   (iter list-of-coeffs))
```

The other constructor, `proc->series`, takes as argument a procedure p of one numeric argument and returns the series

$$p(0) + p(1)x + p(2)x^2 + p(3)x^3 + \ldots$$

In the definition of `proc->series`, the stream `non-neg-integers`, which you will define in this problem set, is the stream of non-negative integers: `{0,1,2,3,…}`.

```
(define (proc->series proc)
   (stream-map proc non-neg-integers))
```

Other Stream Operations

With this representation, we can use basic stream operations such as:

```
(define (add-streams s1 s2)
   (cond ((stream-empty? s1) s2)
         ((stream-empty? s2) s1)
         (else
          (stream-cons (+ (stream-first s1) (stream-rest s2))
                       (add-streams (stream-rest s1)
```

[1] In this representation, all streams are infinite: a finite polynomial will be represented as a stream with an infinite number of trailing zeroes.

```
                                              (stream-rest s2))))))
(define (scale-stream c stream)
  (stream-map (lambda (x) (* x c)) stream))
```

to implement series operations, and thus arrive at a complete power series data abstraction:

```
(define add-series add-streams)

(define scale-series scale-stream)

(define (negate-series s)
  (scale-series -1 s))

(define (subtract-series s1 s2)
  (add-series s1 (negate-series s2)))

(define (show-series s nterms)
  (if (= nterms 0)
    'done
    (begin (write-line (stream-first s))
           (show-series (stream-rest s) (- nterms 1)))))

(define (series-coeff s n)
  (stream-ref s n))
```

You will find the `show-series` procedure helpful because it allows you to examine the series that you will generate in this problem set. You can also examine an individual coefficient (of x^n) in a series using `series-coeff`.

Note: After loading the code for this problem set, you will find that Scheme's basic arithmetic operations +, -, *, and / will work with rational numbers. For instance, (/ 3 4) will produce 3/4 rather than .75. You'll find this useful in doing the exercises below.

Problem 1: Load the code for homework 7 (file hw07-code.scm). Define the stream non-neg-integers. Show how to define the series:

$$S_1 = 1 + x + x^2 + x^3 + \ldots$$
$$S_2 = 1 + 2x + 3x^2 + 4x^3 + \ldots$$

Turn in your definitions and a couple of coefficient printouts to demonstrate that they work.

Problem 2: Complete the definition of the following procedure, which multiplies two series:

```
(define (mul-series s1 s2)
  (stream-cons < ??E1?? >
          (add-series < ??E2?? >
                  < ??E3?? > )))
```

To test your procedure, demonstrate that the product of S_1 (from problem 1) and S_1 is S_2. What is the coefficient of x^{10} in the product of S_2 and S_2? Turn in your definition of `mul-series`. (Optional: Give a general formula for the coefficient of x^n in the product of S_2 and S_2.)

Inverting a power series

Let S be a power series whose constant term is 1. We'll call such a power series a "unit power series." Suppose we want to find the *inverse* of S, 1/S, which is the power series X such that $S \bullet X = 1$. Write $S = 1 + S_R$ where S_R is the rest of S after the constant term. Then we can solve for X as follows:

$$S \bullet X = 1$$
$$(1 + S_R) \bullet X = 1$$

$$X + S_R \bullet X = 1$$
$$X = 1 - S_R \bullet X$$

In other words, X is the power series whose constant term is 1 and whose rest is given by the negative of S_R times X.

Problem 3: Use this idea to write a procedure `invert-unit-series` that computes 1/S for a unit power series S. You will need to use `mul-series`. To test your procedure, invert the series S_1 (from problem 1) and show that you get the series 1 - x. (Convince yourself that this is the correct answer.) Turn in a listing of your procedure. This is a very short procedure, but it is very clever. In fact, to someone looking at it for the first time, it may seem that it can't work-that it must go into an infinite loop. Write a few sentences of explanation explaining why the procedure does in fact work, and does not go into a loop.

Problem 4: Write a procedure `div-series` that divides two power series. `div-series` should work for any two series, provided that the denominator series begins with a non-zero constant term. (If the denominator has a zero constant term, then `div-series` should signal an error.) You may find it useful to reuse some answer from a previous exercise in this problem set. Turn in a listing of your procedure along with three or four well-chosen test cases (and demonstrate why the answers given by your division are indeed the correct answers).

Problem 5: Define a procedure `integrate-series-tail` that, given a series

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

returns the integral of the series (except for the constant term)

$$a_0 x + \frac{1}{2} a_1 x^2 + \frac{1}{3} a_2 x^3 + \frac{1}{4} a_3 x^4 + \cdots$$

Turn in a listing of your procedure and demonstrate that it works by computing `integrate-series-tail` of the series S_2 from problem 1.

Problem 6: Demonstrate that you can generate the series for e^x as

```
(define exp-series
  (stream-cons 1 (integrate-series-tail exp-series)))
```

Explain the reasoning behind this definition. Show how to generate the series for sine and cosine, in a similar way, as a pair of mutually recursive definitions.