CSE1729 - Introduction to Principles of Programming — September 28, 2014

Problem Set 4

Note: Whenever we use the terms iterative process or iterative form, we are talking about tail-recursive code. When we use the terms recursive process or recursive form we are talking about code that is not tail-recursive. Both iterative and recursive processes are found in recursive functions.

1. Abstracting the summation of a series
   (a) Consider the harmonic numbers $H_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$. Last week you wrote a recursive SCHEME function (named `harmonic`) which, given a number n, computes H_n. Revise your `harmonic` function, keeping the name `(harmonic n)`, to take advantage of the `sum` function seen in the textbook (Section 1.3.1) and shown below:

```
(define (sum term a next b)
    (if (> a b)
        0
        (+ (term a)
           (sum term (next a) next b))))
```

   Of course, your new and improved definition of harmonic should not be recursive itself and should rely on `sum` to do the hard work.
   (b) The above definition of sum is a recursive process. Write an iterative version `sum-i` that solves the same problems as `sum` in an iterative fashion. Demonstrate that it works by using it to define `harmonic-i`.
   (c) Show that your harmonic functions work for 1, 50, and 100.

2. Abstracting the product of a series
   (a) The above function `sum` is an abstraction of the sigma notation for summation:

$$\sum_{n=a}^{b} f(n) = f(a) + \cdots + f(b)$$

   Write a function `(product term a next b)` that abstracts the pi notation for the product of a series. You should name your function `(product-i term a next b)` if you choose to write an iterative version of the function.

$$\prod_{n=a}^{b} f(n) = f(a) \times \cdots \times f(b)$$

   (b) If your `product` function is in a recursive form, write a second one that is in the iterative form named `(product-i term a next b)`, and vice-versa (that is, you should write 2 versions of `product`, one recursive and one iterative while following the naming convention).
   (c) Use your product functions to define an function `(pi-approx n)` that returns an approximation of π using the first n terms of the following:

$$\frac{\pi}{4} = \frac{2 \times 4 \times 4 \times 6 \times 6 \times 8 \cdots}{3 \times 3 \times 5 \times 5 \times 7 \times 7 \cdots}$$

   You will need to define appropriate functions for `term` and `next`.
   (d) Show that your pi-approx function works for 1, 100, and 1000. What do these tests demonstrate to you?

3. SICP Exercise 1.37 - As you will recall from lab, an infinite continued fraction is an expression of the form

$$f = \cfrac{N_1}{D_1 + \cfrac{N_2}{\ddots + \frac{N_k}{D_k}}}$$

As an example, one can show that the infinite continued fraction expansion with the N_i and the D_i all equal to 1 produces $\frac{1}{\phi}$, where ϕ is the golden ratio (described in section 1.2.2). One way to approximate an infinite continued fraction is to truncate the expansion after a given number of terms. Such a truncation – a so-called k-term finite continued fraction – has the form

$$f = \cfrac{N_1}{D_1 + \cfrac{N_2}{\ddots + \cfrac{N_k}{D_k}}}$$

Suppose that n and d are procedures of one argument (the term index i) that return the N_i and D_i of the terms of the continued fraction.

(a) Define a procedure `cont-frac` such that evaluating `(cont-frac n d k)` computes the value of the k-term finite continued fraction. Check your procedure by approximating $\frac{1}{\phi}$ using

```
(cont-frac (lambda (i) 1.0)
 (lambda (i) 1.0)
 k)
```

for successive values of k.

(b) SICP Exercise 1.38 - In 1737, the Swiss mathematician Leonhard Euler published a memoir *De Fractionibus Continuis*, which included a continued fraction expansion for $e-2$, where e is the base of the natural logarithms. In this fraction, the N_i are all 1, and the D_i are successively $1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, \ldots$. Write a program that uses your `cont-frac` procedure from exercise 1.37 to approximate e, based on Euler's expansion. Name your function `(e-approx k)`.

(c) SICP Exercise 1.39 - A continued fraction representation of the tangent function was published in 1770 by the German mathematician J.H. Lambert:

$$tan(x) = \cfrac{x}{1 - \cfrac{x^2}{3 - \cfrac{x^2}{5 - \ddots}}}$$

where x is in radians. Define a procedure `(tan-cf x k)` that computes an approximation to the tangent function based on Lambert's formula. `k` specifies the number of terms to compute, as in exercise 1.37.
Note: you should define your own constant for π in R5RS. `(define pi 3.14159)` should be sufficient.

4. Recall the definition of the derivative of a function from calculus (you will not need to know any calculus to solve this problem):

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

By choosing very small values for h in the above equation, we can get a good approximation of $f'(x)$.

(a) Write a function (call it `der` for *derivative*) in SCHEME that takes a function f and a value for h as formal parameters so that `(der f h)` returns the function g defined by the rule

$$g(x) = \frac{f(x+h) - f(x)}{h}.$$

(As mentioned above, for small `h`, g is a good approximation for the derivative of f. Important note: Your function should take a *function* and a number `h` as arguments, and return a *function*.)

(b) Compare the derivative of $\sin(x)$, as computed by your function with `h` $= .5$, with the function $\cos(x)$ over at least 4 values $(0, \pi/2, \pi, 3\pi/4)$.

(c) Write a function `(fun x)` that computes $3x^2 - 2x + 7$. As with the previous, compare your calculated derivative with $6x - 2$ for a few values of x.

5. SICP Expercise 1.43 - If f is a numerical function and n is a positive integer, then we can form the n^{th} repeated application of f, which is defined to be the function whose value at x is $f(f(...(f(x))...))$. For example, if f is the function $x \mapsto x+1$, then the n^{th} repeated application of f is the function $x \mapsto x+n$. If f is the operation of squaring a number, then the n^{th} repeated application of f is the function that raises its argument to the 2^n th power. Write a procedure named `(repeated f n)`, that takes as inputs a procedure that computes f and a positive integer n and returns the procedure that computes the n^{th} repeated application of f. Your procedure should be able to be used as follows:

```
((repeated square 2) 5)
625
```

You may want to take advantage of the `compose` function you wrote in lab.

6. **Here's an extra practice problem which will not be graded (though you should give it a try).**
While Sam is working on a CSE1729 Problem Set and listening to some tunes in the computer lab, Sam gets Morgan's number. Sam has trouble remembering telephone numbers, so Sam wants to write it down. But Sam wants to make sure that if the paper with the number on it gets lost, nobody picks it up and calls Morgan. So, instead of writing the number, Sam breaks the number abc-defg into a 3 digit number abc and a 4 digit number $defg$, and writes the fraction $\frac{abc}{defg} = 0.16330759088341$ on the paper. Noting that all of the individual digits in both parts of the phone number are unique.

Sure enough, Sam forgets the number, but finds the paper, and recalls an algorithm using continued fractions to find a rational approximation for 0.16330759088341. The numerator will always be one. The integers for the denominator can be obtained by:

1. Record the integer part (before the decimal point) as the next value in the continued fraction list. (For an input value less than 1, this will be 0.)
2. Subtract the whole from the to get a value less than 1.
3. If the result is 0, stop
otherwise compute $\frac{1}{x}$
4. Start again with the new number which is larger than 1

Since Sam doesn't know how many times this process will need to be repeated, and having just written a generalized function for computing continued fractions, Sam decides to make good use of this code. Having finished the problem set earlier, you can get Morgan's number as well.

(a) Write a SCHEME function named `(fraction-approx a k)` that will carry out the process outlined above `k` times to obtain the k^{th} integer in the denominator. So, if `a` is the decimal for which we want to compute a rational approximation, our function should:

1. Compute $\frac{1}{a}$
2. Find the integer part of this quantity by using the `floor` function. You may need to use the `inexact->exact` function to convert this value to an integer to ensure your final function returns a fraction instead of a decimal result.
3. If this is the k^{th} integer for the denominator in the continued fraction, return this value.
Otherwise, subtract this integer value from `a` to obtain the decimal to pass in to the next recursive call.
4. Call the function recursively with the new value for `a`.

(b) Write an expression, `(define morgan (...))`, that will use your `fraction-approx` function to compute the k-term finite continued fraction which has 1 as the numerator for every term (N_k) and uses `fraction-approx` starting with the value 0.16330759088341 for `a` for the denominator. Experiment with this function, increasing `k` (the number of terms), until you obtain a fraction of the form $\frac{abc}{defg}$. That is, you're looking for a fraction composed of a three digit numerator and a four digit denominator in which none of the digits are the same.

Submitting Solutions

Once you have finished:

1. Save your work to a file using the appropriate filename, problemSet4.rkt.
2. Submit your solution file for grading via the MIMIR site.