CSE1729 - Introduction to Programming October 9, 2021

Problem Set 5

1. In the previous problems set, you wrote generalized versions of the sum and product of terms in a sequence. Show that sum and product are both special cases of a still more general notion called accumulate that combines a collection of terms, using some general accumulation function:

   `(accumulate combiner null-value term a next b)`

   accumulate takes as arguments the same term and range specifications as sum and product, together with a combiner procedure (of two arguments) that specifies how the current term is to be combined with the accumulation of the preceding terms and a null-value that specifies what base value to use when the terms run out.

   (a) Write `accumulate` using simple recursion and show how sum and product can both be defined as simple calls to accumulate.

   (b) In combinatorial mathematics, the Catalan numbers form a sequence of natural numbers that occur in various counting problems, often involving recursively defined objects. The n^{th} Catalan number is given directly in terms of binomial coefficients (recall Lab 2) by

   $$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{2n!}{(n+1)!n!} = \prod_{2}^{n}\frac{n+k}{k} \text{ for } n \geq 0$$

   The first Catalan numbers for $n = 0, 1, 2, 3, \ldots$ are

   $$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, \ldots$$

   Define a SCHEME function, named `(catalan n)`, which uses your `accumulate` function to compute the n^{th} Catalan number using the definition above.

   (c) The Leibniz formula for π, named after Gottfried Leibniz, states that

   $$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

   Use your `accumulate` function to define a new SCHEME function, named `(leibniz-pi` using the definition above.

   (d) Write another version of the accumulate function, named

   `(accumulate-tr combiner null-value term a next b)`

   which implements this same general accumulation function using tail recursion.

   (e) Use your tail-recursive version of accumulate, `accumulate-tr`, to define a SCHEME function, named `(fact n)` which computes the factorial function ($n!$).

   (f) Use your tail-recursive version of accumulate, `accumulate-tr`, to define a SCHEME function, named `(e-to-x x n)` which computes an approximation to e^x using the Maclaurin series as follows:

   $$e^x = \sum_{n=0}^{\infty}\frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

You should start with your implementation of the factorial function using `accumulate-tr` from Part e.

2. (**Pairing function.**) A *pairing function* p is a function that places the natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ into one-to-one correspondence with the set of all *pairs* of natural numbers (usually denoted $\mathbb{N} \times \mathbb{N}$). It is somewhat surprising that such a function should exist at all: it shows that the set of natural numbers has the same "size" as the set of all *pairs* of natural numbers. To be more precise, a pairing function p takes two natural numbers x and y as arguments and returns a single number z with the property that the original pair can always be determined exactly from the value z. (In particular, the function maps no more than one pair to any particular natural number.)

   One pairing function is the following:

   $$p(x, y) = \begin{cases} y^2 + x & \text{if } x \neq \max(x, y) \\ x^2 + x + y & \text{if } x = \max(x, y) \end{cases}$$

   (a) Write a SCHEME function `(encode p)` that computes the pairing function above. (It should take a pair of integers as an argument and return a single integer.) The `max` function evaluates to the larger of its two arguments. An implementation is available as a SCHEME built-in function.

   (b) As mentioned above, this function has the property that if $(x, y) \neq (x', y')$ then $p(x, y) \neq p(x', y')$: it follows that, in principle, the pair (x, y) can be reconstructed from the single value $z = p(x, y)$. In fact, the values x and y can be reconstructed from $z = p(x, y)$ by

   $$u(z) = \begin{cases} (z - \lfloor\sqrt{z}\rfloor^2, \lfloor\sqrt{z}\rfloor) & \text{if } z - \lfloor\sqrt{z}\rfloor^2 < \lfloor\sqrt{z}\rfloor \\ (\lfloor\sqrt{z}\rfloor, z - \lfloor\sqrt{z}\rfloor^2 - \lfloor\sqrt{z}\rfloor) & \text{if } z - \lfloor\sqrt{z}\rfloor^2 \geq \lfloor\sqrt{z}\rfloor \end{cases}$$

   Write a SCHEME function `(decode z)` that takes as an argument an integer z and produces the pair (x, y) for which $p(x, y) = z$. You'll need the `floor` function: `(floor x)` returns the largest integer less than or equal to `x` (that is, it rounds `x` down to the nearest integer).

3. In lecture, we proposed representing complex numbers, numbers of the form $(a + bi)$, using pairs to store both the real part and the imaginary part. We presented convenience functions for creating complex numbers stored in SCHEME pairs as well as some functions which operate on complex numbers (e.g. `add-complex`, `mult-complex`). In this question, we will expand on the functions we can apply to complex numbers.

   (a) The subtraction of two complex numbers, $(a + bi)$ and $(c + di)$, in component form is given by:

   $$(a, b) - (c, d) = (a - c, b - d)$$

   Define a SCHEME function, named `(sub-complex c d)`, which accepts two complex numbers represented by pairs and returns a complex number representing their difference.

   (b) The division of two complex numbers, $(a + bi)$ and $(c + di)$, in component form is given by:

   $$\frac{(a, b)}{(c, d)} = \left(\frac{ac + bd}{c^2 + d^2}, \frac{bc - ad}{c^2 + d^2}\right)$$

   Define a SCHEME function, named `(div-complex c d)`, which accepts two complex numbers represented by pairs and returns a complex number representing the first divided by the second.

4. Vieta's formulas relate the coefficients of a polynomial to sums and products of its roots. Recall from algebra, a polynomial of degree n:

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

(where coefficients a_i may be real or complex and $a_n \neq 0$) has n complex roots by the fundamental theorem of algebra.

   (a) Vieta's Formulas for quadratic equations: Given a function $f(x) = ax^2+bx+c$, if the equation $f(x) = 0$ has roots r_1 and r_2, then

$$\begin{aligned} r_1 + r_2 &= -\frac{b}{a} \\ r_1 r_2 &= \frac{c}{a}. \end{aligned}$$

      i. Define a SCHEME function, named (`sum-quadratic-roots a b c`) which computes the sum of the roots of the quadratic equation defined by $f(x) = ax^2 + bx + c$ using Vieta's formula as defined above. The coefficients may be complex numbers. So, your function must accept a complex numbers for each of the arguments `a`, `b`, and `c`.

      ii. Define a SCHEME function, named (`prod-quadratic-roots a b c`) which computes the product of the roots of the quadratic equation defined by $f(x) = ax^2 + bx + c$ using Vieta's formula as defined above. Again, your function must accept a complex numbers for each of the arguments `a`, `b`, and `c`.

   (b) Vieta's Formulas for cubic equations: Given an equation of the form $ax^3 + bx^2 + cx + d = 0$, where a, b, c, and d are complex numbers and a is non-zero. By the fundamental theorem of algebra, a cubic equation always has 3 roots (some may be equal). For such a cubic equation, let p, q, and r be its roots, then the following relations hold

$$\begin{aligned} p+q+r &= -\frac{b}{a}, \\ pq+qr+rp &= \frac{c}{a}, \\ pqr &= -\frac{d}{a} \end{aligned}$$

      i. Define a SCHEME function, named (`sum-cubic-roots a b c d`) which computes the sum of the roots of the cubic equation defined by $ax^3 + bx^2 + cx + d = 0$ using Vieta's formula as defined above.

      ii. Define a SCHEME function, named (`sum-pairs-cubic-roots a b c d`) which computes the following sum of pairs of the roots of the cubic equation defined by $ax^3 + bx^2 + cx + d = 0$ using Vieta's formula as defined above. Your function should compute the sum $pq + qr + rp$ as defined above.

      iii. Define a SCHEME function, named (`prod-cubic-roots a b c d`) which computes the product of the roots of the cubic equation defined by $ax^3 + bx^2 + cx + d = 0$ using Vieta's formula as defined above.