CSE1729 - Introduction to Programming October 11, 2018

Problem Set 6

1. **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, introduced by Hopcroft and Ullman [1], is the following:

   $$p(x, y) = \frac{1}{2}(x + y - 2)(x + y - 1) + x\,.$$

   (a) Write a SCHEME function `encode` that computes the pairing function above. (It should take a pair of numbers as its one argument and return a single number.)

   (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 first computing the quantities

   $$w = \left\lfloor \sqrt{2z} - \frac{1}{2} \right\rfloor, \quad \text{and}$$
   $$t = \frac{w^2 + w}{2}\,.$$

   Then $x = z - t$ and $y = w - x + 2$.

   Write a SCHEME function `decode` 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).

   Hint: You may wish to use the `let*` form for this problem. `let*` has the form

```
(let* ((x1 <expr1>)
       (x2 <expr2>)
       ...
       (xk <exprk>))
  <let-expr>)
```

[1] p. 169, *Introduction to Automata Theory, Languages and Computation*, 1979.

The form is a simple method for writing "nested `lets`." It is evaluated, informally, as follows. Starting with the external environment, `expr1` is evaluated and the variable `x1` is immediately bound to the resulting value. Following this, `<expr2>` is evaluated in the resulting environment and the variable `x2` is bound to the value. This continues for the remaining variable/expression pairs. Finally, `<let-expr>` is evaluated in the resulting environment and its value is returned. Note, for example, that `x1` may appear in `<expr2>`. (Recall that in a regular `let` expression, the `<expri>` are all evaluated in the external environment.)

2. Define a SCHEME function, named `(collapse l)`, that takes a SCHEME list, which may have lists or lists of lists as elements, as a parameter. Your function should return a list with all of the same atomic elements as the original list in the same order, but all in a single list on the same "level." For example,

   ```
   >(collapse (list (list 1 2 3) (list 4 5 6 (list 7 8 9))))
   (1 2 3 4 5 6 7 8 9)
   ```

   You may find the SCHEME procedure `pair?` useful. It takes one argument and evaluates to true if that argument is a SCHEME pair (e.g. a list) and false otherwise.

3. [SICP EXERCISE 2.27] Produce a `(deep-reverse l)` procedure that takes a list, `l`, as argument and returns as its value the list with its elements reversed and with all sublists deep-reversed as well. For example,

   ```
   (define x (list (list 1 2) (list 3 4)))
    x
   ((1 2) (3 4))
   (reverse x)
   ((3 4) (1 2))
   (deep-reverse x)
   ((4 3) (2 1))
   ```

4. **Truncatable Primes**

   (a) Define a function named `(explode x)` which converts any positive integer x to a list of single-digit integers (use division by 10 and the floor function). For instance, the expression `(explode 12345)` would return `'(1 2 3 4 5)`. You may only use `cons`, `car`, `cdr`, functions you wrote in lab or are contained in the lecture slides. You may not use string or char functions.

   (b) Define a SCHEME function named `(implode l)` which takes a list of digits in base 10 and converts them to an integer (the inverse operation of `explode`). You may only use `cons`, `car`, `cdr`, functions you wrote in lab or are contained in the lecture slides. You may not use string or char functions.

   (c) **Left Truncatable Primes** In number theory, a left-truncatable prime is a prime number which, in a given base, contains no 0, and if the leading ("left") digit is successively removed, then all resulting numbers are prime. For example, 9137, since 9137, 137, 37 and 7 are all prime.

i. Define a SCHEME procedure, named `(left-truncatable-prime?  p)`, that takes one integer argument, `p`, and evaluates to true (`#t`) if the integer `p` is a left-truncatable prime and false (`#f`) otherwise.
ii. Define a SCHEME procedure, named `(nth-left-trunc-prime n)`, that takes one argument, `n`, and uses the `find` function you write in Lab 7 and `(left-truncatable-prime?  p)` to return the n^{th} left-truncatable prime number.

(d) **Right Truncatable Primes** A right-truncatable prime is a prime which remains prime when the last ("right") digit is successively removed. 7393 is an example of a right-truncatable prime, since 7393, 739, 73, 7 are all prime.

i. Define a SCHEME procedure, named `(right-truncatable-prime?  p)`, that takes one integer argument, `p`, and evaluates to true (`#t`) if the integer `p` is a right-truncatable prime and false (`#f`) otherwise.
ii. Define a SCHEME procedure, named `(nth-right-trunc-prime n)`, that takes one argument, `n`, and uses the `find` function you write in Lab 7 and `(right-truncatable-prime?  p)` to return the n^{th} right-truncatable prime number.

(e) **Two-Sided Primes** There are 15 primes which are both left-truncatable and right-truncatable.

i. Define a SCHEME procedure, named `(two-sided-prime?  p)`, that takes one integer argument, `p`, and evaluates to true (`#t`) if the integer `p` is both a left-truncatable prime and a right-truncatable prime, and false (`#f`) otherwise.
ii. Define a SCHEME procedure, named `(nth-two-sided-prime n)`, that takes one argument, `n`, and uses the `find` function you write in Lab 7 and `(two-sided-prime?  p)` to return the n^{th} two-sided prime number.

5. **Mergesort**

(a) Define a SCHEME function `(split l)` which takes a list ℓ as an argument and "splits it" into two lists ℓ_1 and ℓ_2 of roughly the same size. (In particular, the sizes of ℓ_1 and ℓ_2 should differ by no more than one.) Since the function must return two lists, they can be returned as a pair.

Hint: There are two natural ways to do this. The first is to compute the length of the list and then peel off the right number of elements from the front of the list. The second traverses the list and places "even" elements in one list and "odd" ones in another.

(b) Define a SCHEME function, `(merge l1 l2)`, which takes two lists ℓ_1 and ℓ_2 as arguments. Assuming that each of ℓ_1 and ℓ_2 are *sorted* lists of integers (in increasing order, say), `merge` must return the sorted list containing all elements of ℓ_1 and ℓ_2.

To carry out the merge, observe that since ℓ_1 and ℓ_2 are already sorted, it is easy to find the smallest element among all those in ℓ_1 and ℓ_2: it is simply the smaller of the first elements of ℓ_1 and ℓ_2. Removing this smallest element from whichever of the two lists it came from, we can recurse on the resulting two lists (which are still sorted), and place this smallest element at the beginning of the result.

(c) Put these two ideas together to define the SCHEME procedure `(mergesort l)`, a famous sorting procedure. `mergesort`, when called on a list ℓ, begins by splitting it into two pieces ℓ_1 and ℓ_2. Each piece is then sorted (using `mergesort`!) and the results are merged.