Logic and Functional Programming

Labwork 3

March 8, 2021

Goal: to practice recursive thinking and the programming principles taught in Lecture 3.

1 Recap

Functions with one or more arguments of a recursive type should be defined by recursion on that argument.

- For list argument(s) there should be a base case when the list is empty, and a recursive case when the list is of the form `(cons v lst)`.
  - the recursive calls should take "simpler" argument values, to ensure termination of computation.
- For numeric argument(s), there should be
  - one or more base cases for small numeric values
  - one or more recursive cases which take simpler (smaller) numeric values.

Exercise set 1

1. Define the function `(joinLists lst1 lst2)` which returns the results of joining lists `lst1` and `lst2`.
   - Example: `(joinLists '(1 2) '(a b))` should return `'(1 2 a b)`.
   - **Suggestion:** implement recursion on the length of `lst1`.
   - Side remark: Racket has the predefined function `(append lst1 ... lstn)` to join an arbitrary number of lists.
2. (LW3 from labwork 2) Define the numeric function `(digit-sum N)` which computes the sum of digits of the decimal representation of `N`. For example, `(digit-sum 726)` should return 15 because $7+2+6=15$. Racket has the following predefined functions for $\mathtt{a}, \mathtt{b} \in \mathbb{N}$:

   `(quotient a b)` ; returns the quotient of dividing `a` by `b`
   `(remainder a b)` ; returns the remainder of dividing `a` by `b`

3. (LW4 from labwork 2) Define `(flatten sl)` which takes as input a nested list of symbols, and returns the list of symbols contained in `sl` in the order in which they occur when `sl` is printed. Intuitively, `flatten` removes all the inner parentheses form its argument. For example:

```
> (flatten '(a b c))
'(a b c)
> (flatten '((a b) c (((d)) e)))
'(a b c d e)
> (flatten '((a) () (b ()) () (c)))
'(a b c)
```

   Note that nested lists of symbols are defined by the grammar

   SL ::= null | (cons s SL) | (cons SL SL)

   where s is a symbol.

2 Recursion by solving a more general problem

Sometimes, the best way to solve a problem is to find a solution to a more general problem, and use that solution to solve the original problem as a special case.

Example

Suppose `von` is a vector of numbers. Define `(vector-sum von)` which returns the sum of elements of `von`, using the functions

- `(vector-length von)`: returns the length (number of elements) of `von`
- `(vector-ref von i)`: returns the `i`-th element of `von`; the elements of `von` are indexed starting from 0.

Instead of defining `(vector-sum von)`, we can define the more general function

`(partial-vector-sum von n)`

which computes the sum of elements with indexes from 0 to `n` in `von`.

Note that `(vector-sum von)` coincides with

`(partial-vector-sum von (vector-length von))`

```
(define (vector-sum von)
   (define (partial-vector-sum von n)
      (if (= n 0)
         0  ; nothing to add
         (+ (partial-vector-sum von (- n 1))
            (vector-ref von (- n 1)))))
   (partial-vector-sum von (vector-length von)))
```

Note that

1. `partial-vector-sum` is tail recursive, therefore it's computation is very fast.

2. The argument `von` of `partial-vector-sum` does not change its value. Since the function `partial-vector-sum` is defined in the body of `vector-sum`, we can optimize the definition of `partial-vector-sum` by eliminating argument `von`:

```
(define (vector-sum von)
   (define (partial-vector-sum n)
      (if (= n 0)
         0  ; nothing to add
         (+ (partial-vector-sum (- n 1))
             (vector-ref von (- n 1)))))
   (partial-vector-sum (vector-length von)))
```

Exercise set 2

1. Let `(rev2 lst1 lst2)` be the function which returns the result of joining the reverse of list `lst1` with list `lst2`. For example, `(rev2 '(3 4) '(2 1))` yields `'(4 3 2 1)`.

   Note that, if `lst1` is not empty, then `(rev2 lst1 lst2)` returns the same result as

   ```
   (rev2 (cdr lst1) (cons (car lst1) lst2))
   ```

   (a) Write a tail recursive definition of `rev2`.

   (b) Define the function `(revList lst)` which computes the reverse of list `lst` as a special case of using the function `rev2`.

2. Let `(f2 a b c)` the function which computes $\mathtt{c} \cdot \mathtt{a}^{\mathtt{b}}$ when $\mathtt{a}, \mathtt{b}, \mathtt{c}$ are non-negative integers. Note that

$$\mathtt{c} \cdot \mathtt{a}^{\mathtt{b}} = \begin{cases} \mathtt{c} & \text{if } \mathtt{b} = 0, \\ \mathtt{c} \cdot (\mathtt{a}^2)^{\mathtt{b}/2} & \text{if } \mathtt{b} > 0 \text{ is even,} \\ (\mathtt{c} \cdot \mathtt{a}) \cdot (\mathtt{a}^2)^{(\mathtt{b}-1)/2} & \text{if } \mathtt{b} \text{ is odd.} \end{cases}$$

   (a) Write a tail recursive definition of `f2`.

   (b) Define the function `(power a b)` which computes $\mathtt{a}^{\mathtt{b}}$ for $\mathtt{a}, \mathtt{b} \in \mathbb{N}$ as a special case of using the function `f2`.

3. Newton discovered the following method to compute $\sqrt{a}$ when a is a non-negative number: $\sqrt{a}$ is the limit of the sequence of numbers $(x_n)_{n\in\mathbb{N}}$ where

$$x_0 = 1.0, \quad x_{n+1} = (x_n + a/x_n)/2 \quad \text{for all } n \in \mathbb{N}.$$

   (a) Define the function `(improve xn a)` which takes as inputs the values of of x_n and a and returns the value of x_{n+1}.

(b) x a *good enough* approximation of $\sqrt{a}$ if $|x^2 - a| \leq 0.000001$. Define the boolean function `(good? x a)` which returns `#t` if the value of `x` is a good enough approximation of $\sqrt{a}$, and `#f` otherwise.

(c) Newton's method finds a good approximation of $\sqrt{a}$ starting from a number `x`, as follows: If `x` is a good approximation of $\sqrt{a}$ it returns `x`, otherwise it returns a good approximation od $\sqrt{a}$ starting from `(improve x a)`.

Write a tail recursive definition of the function `(newton2 a x)` which uses Newton's method to compute a good approximation of $\sqrt{\texttt{a}}$ starting from `x`.

4. Newton discovered the following method to compute $\sqrt[3]{a}$ when $a \in \mathbb{R}$: $\sqrt[3]{a}$ is the limit of the sequence of numbers $(x_n)_{n\in\mathbb{N}}$ where

$$x_0 = 1.0, \quad x_{n+1} = (2 \cdot x_n + a/x_n^2)/3 \quad \text{for all } n \in \mathbb{N}.$$

(a) Define the function `(improve3 xn a)` which takes as inputs the values of of x_n and a and returns the value of x_{n+1}.

(b) x a *good enough* approximation of $\sqrt[3]{a}$ if $|x^3 - a| \leq 0.000001$. Define the boolean function `(good3? x a)` which returns `#t` if the value of `x` is a good enough approximation of $\sqrt[3]{a}$, and `#f` otherwise.

(c) Newton's method finds a good approximation of $\sqrt{a}$ starting from a number `x`, as follows: If `x` is a good approximation of $\sqrt[3]{a}$ it returns `x`, otherwise it returns a good approximation od $\sqrt{a}$ starting from `(improve3 x a)`.

Write a tail recursive definition of the function `(newton3 a x)` which uses Newton's method to compute a good approximation of $\sqrt[3]{\texttt{a}}$ starting from `x`.

5. Write a tail-recursive definition of the function `(eval v lst)` which takes as input a list of numbers

`lst = '(`a_0 a_1 `...` a_n`)`

and computes the value of $a_0 + a_1 \cdot \texttt{v} + \ldots + a_n \cdot \texttt{v}^n$.

6. Consider lists of symbols defined by the grammar

`SL ::= null | (cons` s `SL) | (cons SL SL)`

where s is a symbol. Write a tail recursive definition of the function `(flattenTR sl)` which computes the reverse of the flattened form of a list of symbols `sl`.

7. A rational number is a number whose value coincides with $\frac{\texttt{a}}{\texttt{b}}$ where $\texttt{a}, \texttt{b} \in \mathbb{Z}$ and $\texttt{b} \neq 0$. Suppose we choose to represent every rational number $\frac{\texttt{a}}{\texttt{b}}$ as a pair `(cons a b)` where $\texttt{a}, \texttt{b} \in \mathbb{Z}$. Thus, we consider the following BNF for rational numbers

$\langle$rat$\rangle$`::=(cons` $\langle$integer$\rangle$ $\langle$integer$\rangle$`)`

Define the following operations:

(a) `(rat? q)`, which recognizes if $\texttt{q} \in \langle\text{rat}\rangle$.

(b) `(qsum q r)`, `(qdif q r)`, `(qmul q r)`, and `(qdiv q r)`, which take as inputs two values $\texttt{q}, \texttt{r} \in \langle\text{rat}\rangle$ and compute their rational sum, difference, product, and division.

(c) `(rat-eq? q1 q2)`, which returns `#t` if `q` and `r` represent the same rational number, and `#f` otherwise.

(d) `(simplify q)`, which returns $\texttt{r} \in \langle\text{rat}\rangle$ such that `q` and `r` represent the same rational number, and `r=(cons a b)` where `a`, `b` are relatiely prime. To define this function, you can use the predefined function `(gcd u v)` which returns the greatest common divisor of two integers $\texttt{u}, \texttt{v} \in \mathbb{Z}$.

8. A complex number is a number whose value coincides with $\texttt{a} + \texttt{i} \cdot \texttt{b}$ where `a`, `b` are floating-point numbers and $\texttt{i} \in \mathbb{C}$ is the imaginary unit that satisfies the equation $\texttt{i}^2 = -1$. Suppose we choose to represent such a complex number as a pair `(cons a b)`. Thus, we consider the following BNF for complex numbers

   ⟨cplx⟩`::=(cons` ⟨real⟩ ⟨real⟩`)`

   Define the following operations:

   (a) `(cplx? c)`, which recognizes if $\texttt{c} \in \langle\text{cplx}\rangle$.

   (b) `(abs-value c)`, which computes the absolute value of $\texttt{c} \in \langle\text{cplx}\rangle$. Remember that the absolute value of a complex number $z = a + \texttt{i} \cdot b$ is $|z| = \sqrt{a^2 + b^2}$.

   (c) `(rdiv c r)` which takes as inputs $\texttt{c} \in \langle\text{cplx}\rangle$ and $\texttt{r} \in \langle\texttt{real}\rangle$, $\texttt{r} \neq \texttt{0}$, and returns the value from ⟨cplx⟩ corresponding to the division of `c` by `r`.

   (d) `(csum q r)`, `(cdif q r)`, `(cmul q r)`, and `(cdiv q r)`, which take as inputs two values $\texttt{q}, \texttt{r} \in \langle\text{cplx}\rangle$ and compute their complex sum, difference, product, and division.

9. Consider sets represented by lists without repeating elements, and let `A`, `B` be two such sets. Define the following operations:

   (a) `(union A B)`, which computes the set union of `A` and `B`.

   (b) `(difference A B)`, which computes the set difference of `A` and `B`.

   You can use the predefined function `(member v lst)` which returns `#f` if `v` is not equal to any element of list `lst`, and true otherwise.