Programming Paradigms Fall 2022 — Problem Sets

by Nikolai Kudasov

September 19, 2022

1 Problem set №3

1. Implement the following functions over the list of binary digits in Racket using higher-order functions (`apply`, `map`, `andmap`, `ormap`, `filter`, `foldl`) and **without explicit recursion**.

   (a) Convert into a decimal number:

   ```
   (binary-to-decimal '(1 0 1 1 0)) ; ==> 22
   ```

   (b) Remove leading zeros:

   ```
   (remove-leading-zeros '(0 0 0 1 0 1 1 0)) ; ==> '(1 0 1 1 0)
   ```

   (c) Count zeros in a binary string (not counting leading zeros):

   ```
   (count-zeros '(0 0 0 1 0 1 1 0)) ; ==> 2
   ```

   (d) Group consecutive digits into lists:

   ```
   (group-consecutive '(0 0 0 1 0 1 1 0)) ; ==> '((0 0 0) (1) (0) (1 1) (0))
   ```

   (e) Encode a binary string by removing leading zeros and replacing each consecutive substring of digits with its length. For example, `'(0 0 0 1 1 0 1 1 1 0 0)` has some leading zeros, then 2 ones, then 1 zero, then 3 ones, then 2 zeros, so it should be encoded as `'(2 1 3 2)`:

   ```
   (encode-with-lengths '(0 0 0 1 1 0 1 1 1 0 0)) ; ==> '(2 1 3 2)
   ```

   (f) Decode a binary string from an encoded representation from the previous exercise:

   ```
   (decode-with-lengths '(2 1 3 2)) ; ==> '(1 1 0 1 1 1 0 0)
   ```

2. Consider the following sample definition:

```
(define employees
  '(("John" "Malkovich" . 29)
    ("Anna" "Petrova"   . 22)
    ("Ivan" "Ivanov"    . 23)
    ("Anna" "Karenina"  . 40)))
```

Recall that `'("Anna" "Petrova" . 22)` is equivalent to `(cons "Anna" (cons "Petrova" 22))`. This value is a pair where first element is the first name of an employee and second element is a pair of last name and age.

(a) Implement a function `fullname` that takes employee and returns their full name as a pair of first and last name:

```
(fullname '("John" "Malkovich" . 29))
; '("John" . "Malkovich")
```

(b) Using higher-order functions (`map`, `ormap`, `andmap`, `filter`, `foldl`) and without explicit recursion, write down an expression that computes a list of entries from `employees` where employee's first name is `"Anna"`.

(c) Using higher-order functions (`map`, `ormap`, `andmap`, `filter`, `foldl`) and without explicit recursion, implement a function `employees-over-25` that computes a list of full names of employees whose age is greater than 25 given a list of employee entries as input:

```
(employees-over-25 employees)
; '(("John" . "Malkovich") ("Anna" . "Karenina"))
```

3. Consider the following definitions:

```
(define (remove-odd values) (filter even? values))
(define (sum-even values)
  (cond
    [(empty? values) 0]
    [(even? (first values))
     (+ (first values) (sum (rest values)))]
    [else
     (sum (rest values))]))
```

Using the Substitution Model, we can prove that for any valid list of numbers `values`, the following two expressions are equivalent:

- `(apply + (remove-odd values))`
- `(sum-even values)`

Indeed, when `values` is an empty list we get

```
(apply + (remove-odd '()))
= (apply + (filter even? '()))  ; by definition of remove-odd
= (apply + '())                 ; by definition of filter
= 0                             ; by definition of apply
= (sum-even '())                ; by definition of sum-even (inverted)
```

Complete the proof for the case when `values` is not empty:

```
(apply + (remove-odd (cons x xs)))
= ... ; <- your proof as a sequence of equalities goes here
= (sum-even (cons x xs))
```

In addition to regular Substitution Model, you can use the following equivalences:

(a) `(apply + (remove-odd xs))` ≡ `(sum-even xs)` (inductive hypothesis)

(b) for all `p`, `y`, `ys`, the following expressions are equivalent:

- `(filter p (cons y ys))`
- ```
  (cond
    [(p y) (cons y (filter p ys))]
    [else (filter p ys)])
  ```

(c) for all `y`, `ys`, `(apply + (cons y ys))` ≡ `(+ y (apply + ys))`

(d) for all `f`, `c1`, `c2`, `e1`, `e2`, the following expressions are equivalent:

- `(f (cond [c1 e1] [c2 e2]))`
- `(cond [c1 (f e1)] [c2 (f e2)])`
  ```