Quiz 8

Brian Hou

November 5, 2015

1. Write a tail-recursive procedure `binary-to-decimal` that converts a list of zeros and ones into the corresponding base-10 number.

```
(define (binary-to-decimal bits)

  (define (helper bits so-far)
    (if (null? bits)
        so-far
        (helper (cdr bits)
                (+ (* 2 so-far) (car bits)))))
  (helper bits 0))
```

```
scm> (binary-to-decimal (list))
0
scm> (binary-to-decimal (list 1 0 1))
5
scm> (binary-to-decimal (list 1 1 1 1 0 1))
61
```

2. An **association list** (or alist) is one way of representing key-value mappings that is commonly used in Scheme. An **association** represents a key-value pair; the car of an association is the key and the cdr is the value. An alist is a list of associations. Implement the association list data abstraction.

   The `make-alist` constructor creates an empty association list. (With a Python dictionary, this would be similar to {}.)

   The `get` procedure takes an association list `alist` and a key `k`, finds the first association that has `k` as a key and returns the corresponding value. (In Python, `alist[k]`.)

   The `set` procedure takes an association list `alist`, a key `k`, and a value `v`. It adds the association to the beginning of `alist`. (In Python, `alist[k] = v`.) Note that you don't need to find any old associations that involve `k`, since `get` will return the value from the first matching association.

```
(define (make-alist)                              nil )

(define (get alist k)
  (cond
    ((null? alist)                          'not-found)
    ((eq? (car (car alist)) k) (cdr (car alist)))
    (else                     (get (cdr alist) k)))
                                                      )

(define (set alist k v)  (cons (cons k v) alist) )
```

```
scm> (define alist (make-alist))
scm> (get alist 'unknown-key)
not-found
scm> (define alist (set alist 'a 1))
scm> (get alist 'a)
1
scm> (define alist (set alist 'b 10))
scm> alist
((b . 10) (a . 1))
scm> (define alist (set alist 'a 2))
scm> (list (get alist 'a) (get alist 'b))
(2 10)
```