Oops, I made a mistake. Won't be the last time.

I said I thought the system append procedure was done more efficiently than the recursive append we wrote (which rebuilt the list). That is wrong. Consider this example:

```
(define a '(1 2 3))
(append (cdr a) '(4 5))
```

The append call certainly creates a list (2 3 4 5). However, if it did this by changing the tail pointer of the list (1 2 3) to point at the list (4 5), then list a would change to (1 2 3 4 5). That isn't the case; a remains (1 2 3).

A few more examples on flat lists:

lat = list of atoms

(remove-numbers lat) removes all of the numbers from lat

We did this as

```
(define remove-numbers (lambda (lat)
    (cond
        [(null? lat) null]
        [(number? (car lat)) (remove-numbers (cdr lat))]
        [else (cons (car lat) (remove-numbers (cdr lat)))]))
```

Now do:

(remove-stuff pred lat) removes any element from lat that satisfies pred.

(rev lat) reverses lat.