# Recursive Programming in Lisp L2

Robert Krisztian Sandor, group 927

3.12.2015

**6**. a. Write a function to return the product of all numerical atoms in a list, at any level.
b. Write a function to sort a linear list with keeping the double values.
c. Write a function to return the union of two sets.
d. Write a function to reverse a list.

**Mathematical Models**

**a.**

$$product(l_1, ..., l_n) = \begin{cases} 1 & if\ n = 0 \\ product(l_1) * product(l_2, ..., l_n) & if\ l_1\ is\ a\ list \\ l_1 * product(l_2, ..., l_n) & if\ l_1\ is\ a\ number \\ 1 & otherwise \end{cases}$$

**b.**

$$listl(x, l_1, ..., l_n) = \begin{cases} emptylist & if\ n = 0 \\ l_1 \cup listl(x, l_2, ..., l_n) & if\ l_1 < x \\ listl(x, l_2, ..., l_n) & otherwise \end{cases}$$

$$listge(x, l_1, ..., l_n) = \begin{cases} emptylist & if\ n = 0 \\ l_1 \cup listge(x, l_2, ..., l_n) & if\ l_1 >= x \\ listge(x, l_2, ..., l_n) & otherwise \end{cases}$$

$$qsort(l_1, ..., l_n) = \begin{cases} emptylist & if\ n = 0 \\ qsort(listl(l_1, l_2, ..., l_n)) \cup (l_1) \cup qsort(listge(l_1, l_2, ..., l_n)) \end{cases}$$

**c.**

$$exists(e, l_1, ..., l_n) = \begin{cases} false & if\ n = 0 \\ true & if\ e = l_1 \\ exists(e, l_2, ..., l_n) & otherwise \end{cases}$$

$$mergesets(l_1, ..., l_n, k_1, ..., k_m) = \begin{cases} (l_1, ..., l_n) & if\ m = 0 \\ (k_1, ..., k_m) & if\ n = 0 \\ mergesets(l_1, ..., l_n, k_2, ..., k_m) & if\ exists(k_1, l_1, ..., l_n) \\ k_1 \cup mergesets(l_1, ..., l_n, k_2, ..., k_m) & otherwise \end{cases}$$

**d.**

$$reverselist(l_1, ..., l_n) = \begin{cases} emptylist & if\ n = 0 \\ reverselist(l_2, ..., l_n) \cup l_1 & otherwise \end{cases}$$

**Meaning of predicates. Flow models. Source Code**

```
; a
; product_list(l : List)
; l - list of any elements
(defun product_list (l)
  (cond
    ((null l) 1)
    ((numberp (car l)) (* (car l) (product_list (cdr l))))
    ((listp (car l)) (* (product_list (car l)) (product_list (cdr l))))
    (t 1)
    )
  )

; b
; qsort(l : List)
; l - list of numerical atoms
(defun qsort (l)
  (cond
    ((null l) nil)
    (t (append
        (qsort (listl (car l) (cdr l)))
        (cons (car l) nil)
        (qsort (listge (car l) (cdr l)))
        ))
    )
  )
```

```
; listl(e : Integer, l : List)
; e - number to compare to
; l - list of numerical atoms
(defun listl (e l)
  (cond
    ((or (null e) (null l)) nil)
    ((< e (car l)) (listl e (cdr l)))
    (t (cons (car l) (listl e (cdr l))))
    )
  )

; listl(e : Integer, l : List)
; e - number to compare to
; l - list of numerical atoms
(defun listge (e l)
  (cond
    ((or (null e) (null l)) nil)
    ((>= e (car l)) (listge e (cdr l)))
    (t (cons (car l) (listge e (cdr l))))
    )
  )

; c
; exists(e : Atom, l : List)
; e - Atom to be searched for
; l - list to search in
(defun exists (e l)
  (cond
    ((or (null e) (null l)) nil)
    ((eq (car l) e) t)
    (t (exists e (cdr l)))
    )
  )

; merge_sets(a : List, b : List)
; a, b - sets of elements
(defun merge_sets (a b)
  (cond
    ((null a) b)
    ((null b) a)
    ((exists (car b) a) (merge_sets a (cdr b)))
    (t (cons (car b) (merge_sets a (cdr b))))
    )
  )

; d
; reverse_list(l : List)
; l - list of any elements
(defun reverse_list (l)
  (cond
    ((null l) nil)
    (t (append (reverse_list (cdr l)) (list (car l))))
    )
  )
```

### Examples

```
[4]> (product_list `(1 2 (3 (4))))
24

[5]> (qsort `(6 3 2 4 5 1))
(1 2 3 4 5 6)

[6]> (merge_sets `(7 6 5 4 3) `(2 3 4 5 6 7 8 9))
(2 8 9 7 6 5 4 3)

[7]> (reverse_list `(1 2 3 4 5 6))
(6 5 4 3 2 1)
```