Recursión e Inducción en Acl2

Consideremos las siguientes definiciones Lisp:

```
(defun append (l1 l2)
  (if (consp l1)
      (cons (car l1) (append (cdr l1) l2))
      l2))

(defun reverse (l1)
  (if (consp l1)
      (append (reverse (cdr l1)) (cons (car l1) nil))
      nil))

(defun pertenece (x l)
  (cond ((not (consp l)) nil)
        ((equal x (car l)) t)
        (t (pertenece x (cdr l)))))

(defun swaptree (l)
  (if (consp l)
      (cons (swaptree (cdr l))
            (swaptree (car l)))
      l))
```

Demostrar utilizando inducción y simplificación las siguientes conjeturas:

Ejercicio 1

La función `append` es asociativa:

```
(equal (append (append l1 l2) l3)
       (append l1 (append l2 l3)))
```

Ejercicio 2

El efecto de `append` sobre `reverse` y `nil`

```
(equal (append (reverse l) nil)
       (reverse l))
```

En este ejercicio será necesario utilizar el resultado del ejercicio anterior como regla de simplificación.

Ejercicio 3

El efecto de `reverse` sobre `append`

```
(equal (reverse (append l1 l2))
       (append (reverse l2) (reverse l1)))
```

En este ejercicio será necesario utilizar los resultados de los ejercicios anteriores como reglas de simplificación.

Ejercicio 4

Caracterización de la pertenencia al resultado de `append`

```
(equal (pertenece e (append l1 l2))
       (or (pertenece e l1)
           (pertenece e l2)))
```

Ejercicio 5

La función `swaptree` es reversible

```
(equal (swaptree (swaptree l))
       l)
```