```
(defun (sumn-t (n acc)
  :ic (and (natp n) (natp acc))
  :oc (natp (sumn-t n acc))
  (cond (equal n 0) acc)
        (t (sumn-t (- n 1) (+ n acc)))))
```

```
(defunc (sumn* n)
  :ic (natp n)
  :oc (natp (sumn* n))
  (sumn-t n 0))
```

(sumn* n) = (sumn n)

(sumn-t n ~~acc~~ 0) = (+ 0 (sumn n))

≡ {L1}

t

(+ acc (sumn* n)) = (sumn-t n acc) L1

IS for sumn-t

① ¬((natp n) ∧ (natp acc)) ⇒ L1

② (natp n) ∧ (natp acc) ∧ (n = 0) ⇒ L1

③ (natp n) ∧ (natp acc) ∧ (n ≠ 0) ∧ L1|((n (- n 1)) (acc (+ n acc))) ⇒ L1

① contradiction

② C1. (natp n)
C2. (natp acc)
C3. (n = 0)

(sumn-t n acc)
≡ {Def sumn-t, C3}
acc
≡ {~~Def sumn~~ Arithmetic}
(+ acc 0)
≡ {Def sumn, C3}
(+ acc (sumn n))

C1. (natp n)

C2. (natp acc)

C3. (n ≠ 0)

(- n 1) (+ n acc) (+ n acc (sumn (- n 1)))

C4. (natp (- n 1)) ∧ (natp (+ n acc)) ⇒ (sumn-t ~~n acc)~~ = (+ ~~acc (sumn n~~))

C5. (natp (- n 1)) {C3, C1, ~~Arith~~}

C6. (natp (+ n acc)) {C2, C1, Arith}

C7. (sumn-t ~~n acc~~) = (+ (acc ~~(- n 1)~~ (sumn (- n 1)))

(+ acc (sumn n))

≡ {Def sumn, C3}

(+ acc (+ n (sumn (- n 1))))

≡ {Arithmetic}

(+ (+ n acc) (sumn (- n 1)))

≡ {C7}

(sumn-t (- n 1) (+ n acc))

≡ {Def sumn-t, C3}

(sumn-t n acc)

QED

Prove less-t ≡ less

'(5 4 3 2)

[redacted] (2 3 4 5)

```
(defunc less (e lr)
  :ic (and (rationalp e) (lorp lr))
  :oc (lorp (less e lr))
  (cond ((endp lr) lr)
        ((<= e (first lr)) (less e (rest lr)))
        (t (cons (first lr) (less e (rest lr))))))
```

Quiz

Wed to IS.

IS

```
(defunc less* (e lr)
  : ic
  : oc
  (rev (less-t e lr nil)))
```

(rev (less-t e lr nil))

≡ (rev (app (rev (less e lr)) nil))

≡ (rev (rev (less e lr)))

= (less e lr)

L1: (less-t e lr acc)

≡ (app (rev (less e lr)) acc)

Keep value case

C1. (rationalp e)

C2. (lorp lr)

C3. (lorp acc)

~~rationalp~~

~~lorp~~

C4. ¬(endp lr)

C3b. (< (first lr) e)

C5 L1 |((e (rest lr)) (cons (first lr)))

C6. (lorp (rest lr)) {C4, C2}

C7. (lorp (cons (first lr) acc)) {Def lorp, first-rest axioms, C4, C3b, C2}

C8. (less-t (rest lr) (cons (first lr) acc))

= (app (rev (less (rest lr))) (cons (first lr) acc))

Doing obligations 2 & 3

(less-t lr acc)

≡ {Def less-t, C4, C3b}

(less-t (rest lr) (cons (first lr) acc))

≡ {C8}

(app (rev (less (rest lr))) (cons (first lr) acc))

$\equiv$ {Def app}

(app (rev (less (rest l))) (app (list (first lr)) acc))

$\equiv$ {Assoc of app}

(app (app (rev (less (rest l))) (list (first lr))) acc)

$\equiv$ {Def rev, first-rest axiom}

(app (rev (cons (first l) (less (rest l)))) acc)

$\equiv$ {Def less, C3}

(app (rev (less l)) acc)

C1. (lorp acc)

C2. (lorp lr)

C3. (endp lr)

LHS

(less-t lr acc)

$\equiv$ {C3}

acc

RHS

(app (rev (less lr)) acc)

$\equiv$ {C3, Def less}

(app (rev lr) acc)

$\equiv$ {Def rev, C3, Def app}

acc

ssort ssort

(orderedp (ssort l))

Don't have 2 lists

~~(perm l (ssort l))~~

(perm (rev l) l)

Is fe listp ← ~~could use perm~~

ASSUME
perm is trans

(listp (rest l)) ⇒

C1. (listp l)
C2. ¬(endp l)
C3. (perm (rest l) (rev (rest l)))

app (perm x) (perm y)

Lemma here
(perm (app x y) (app y x))
perm is transitive

(perm l (rev l))

≡ {Def perm}

(in (first l) (rev l)) ∧ (perm (rest l) (rev l))

(del (first l))

≡ {Def rev, C2}

(in (first l) (app (rev (rest l)) (list (first l)))) ∧ (perm (rest l) (rev l))

(del (first l))

≡ {Φ_{in_app}}

(in (first l) (rev (rest l))) ∨ (in (first l) (list (first l)))) ∧ (perm (rest l) (rev l))

≡ {Def list, Def endp, cons pair, Def in, PL}

(perm (rest l) (del (first l) (rev l)))

{Trans perm, $\Phi_{perm\ app}$}

≡ {Def rev, C2}

(perm (rest l) (del (first l) (app (rev (rest l)) (list (first l)))))

≡ {??} (perm (del e (app x y)) (del e (app y x))) (perm (rev (rest l)) (rest l))

(in (first l) (rev l)) —

(perm (rest l) (del (first l) (app (rev (rest l)) (list (first l)))))

≡ (perm (rest l) (del (first l) (app (list (first l)) (rev (rest l)))))

Lemma we have

(perm x y) ⇒ (perm (del e x) (del e y))

using I.S. for perm

≡ (perm (rest l)) (rev (rest l)))

≡ {Ind Assumption}

t

$\phi_{perm.app}$: (listp x) ∧ (listp y) ⇒ (perm (app x y) (app y x))

Direct ~~proof~~? Listp x ≠ empty

(in (first (app x y)) (app y x)) ∧ (perm (app (rest x) y) (del (first x) (app y ~~(rest x)~~ x)))

C? (perm (app (rest x) y) (app y (rest x))) ≡

(in (first x) (app y x))

(in (first x) y) ∨ (in (first x) x))

(perm (app (rest x) y) (del

~~(del e l)~~ ∧ (not (in e l))

⇒ ((del e l) = l)

L: (del e (app x y))

≡ (app (del e x) y) if e ∈ x

or

= (app x (del e y))

(dele (app x y))

(perm (dele (app x y)) (app x (dele y)))

~~(in e x)~~ ∧ (in e y)

= (perm (app (del e x) y) (app x (del e y)))

Lets me use del in x

I.S. for perm

② (endp x) ⇒ contradiction

③ (first x) = e) (perm (app (rest x) y) (app x (dele y))

(del ey) (del e

4

(in e l) ⇒ (perm (cons e (dele l))

this can be the app of the list

(perm (dele (app x (cons e y)))

(app x y)

(perm (app x e y) (cons e (app x y)))

(perm (dele (app x y)) (app (del e x) y)

(perm x y) = (perm (rest x) (del e y))

AB

perm

(in e y) ⇒ (app x y) = (e (app x (dele y)))

$(\text{listp } x) \wedge (\text{listp } y) \Rightarrow \overset{\text{perm}}{(}\text{del } e\ (\text{app } x\ (\text{cons } e\ y))) \neq (\text{app } x\ y))$

Using I.S. for ~~perm~~ in?

① $\neg IC \Rightarrow \varphi$

② $IC \wedge (\text{endp } x) \Rightarrow \varphi$

~~③ $IC \wedge \neg(\text{endp } x) \wedge (e = (\text{first } x)) \Rightarrow \varphi$~~

④ $IC \wedge \neg(\text{endp } x) \wedge \cancel{(e \neq (\text{first } x))} \wedge \varphi_{((x\ (\text{rest } x))} \Rightarrow \varphi$

cases

③ C1. (listp x)

C2. (listp y)

C3. $\neg$(endp x)

C4. (e = (first x))

$IC \Rightarrow$

C5. (perm (del e (app (rest x) (cons e y))) (app (rest x) y)

C6 (listp (rest x))

C7. (perm (del e (app (rest x) (cons e y))) (app (rest x) y))

~~Just review?~~

Not equal e

(perm (del e (app x (cons e y))) (app x y)

≡ {Def of del}

(perm (cons (first x) (del e (app (rest x) (cons e y))) (del (first x) (app x y))

= (perm ~~(cons (firs~~ (del e (app (rest x) (cons e y))) (app (rest x) y))

= {C7}

t

{Def del, C8}

(perm (app (rest x) (cons e y))) (app x y))

= (perm (app x y)) (app (rest x) (cons e y))

→ Def perm

(perm (app (rest x) y)

(del e (app (rest x) (cons e y)))

= {symm of perm}

(perm l (rev l)) C2 I.S for listp
{Def perm, Def rev, PL}
≡ (~~perm~~ in (first l) (app (rev (rest l)) (list (first l)))
∧ (perm (rest l) (~~app~~ del (first l) (app (rev (rest l)) (list (first l))) Context

≡ { ϕ_{app-in} ~~PL~~ }
(in (first l) (rev (rest l))) ∨ (in (first l) (list (first l)))
∧ (perm (rest l) (del (first l) (app (rev (rest l) (list (first l))))

≡ { PL, Def in, ~~def app~~ first-rest axiom }
(perm (rest l) (del (first l) (app (rev (rest l)) (list (first l)))))

crap
We need a lemma

≡ { $\phi_{del-app}$ |((x (app (rev (rest l)))
(y nil)
(e (first l))) }

~~Pe~~ $\phi_{perm\ trans}$

(perm (rest l) (app (rev (rest l)) nil))

≡ { $\phi_{app-x\ nil}$ }

(perm (rest l) (rev (rest l)))

≡ { C6 }

t

ϕ_{del_app} : (listp x) ∧ (listp y) ⇒ (del e (app ~~x~~ (cons e y)) (app x y))

(see previous)

More general

(perm (cons e (app x y)) (app x (cons e y)))

ϕ_{perm_app} : (listp x) ∧ (listp y) ⇒ (perm (app x y) (app y x))

Is it perm? ✗

(perm (app (rest x) (del (first x) y)) (app (del (first x) y) (rest x)))

Not sure I can delete element

(perm (app x y) (app y x))

≡ (~~perm (app (rest x) y) (del (first x) (app y x))~~)

Ind Assupts (perm (app (rest x) y) (app y (rest x)))

(perm (app (rest x) y) (del (first x) (app y x)))

≡ { $\phi_{perm\text{-}app\text{-}delAR}$ & $\phi_{perm\text{-}trans}$ }

(perm (app (rest x) y) (app y (rest x)))

≡ {C8? Ind Assuption}

t

$\varphi_{\text{app-front}}$: (listp x) ∧ (listp y) ⇒ (perm (app X (cons e Y))
(cons e (app X Y)))

endpx : (cons e Y) = (app nil (cons e Y))

~~if first~~

(perm (app x (cons e Y))) (cons e (app X Y))

≡ {Def perm} e = (first x)

(in (first x) (cons e (app x y)) ∧ (perm (app (rest x) (cons e y)) ≡
{first rest app C4..} (del (first x) (cons e (app x y))
= (perm (app (rest x) (cons e y))
≡ (cons e (app (rest x) y))

C1 (listp x)
C2 (listp y)
C3 (e = (first x)
C4 (endp x)

~~(perm (app x (cons y)) (cons e (app x y))~~
~~≡ (perm (app (rest x) (cons e y)) (app x y)~~
~~≡ (perm (app (app (rest x) (first x)) y) (app x y)~~ or (cons e (app (rest x) y))

??

~~Case~~

del-all as the I.S.

Ind Assumption (perm (app (rest x) (cons e y)) (cons (app (rest x) y))

(perm (app (rest x) (cons e y) (del e (cons e (app (rest x) y)))
if e = (first x) ~~then~~
(app x y) = (cons e (app (rest x) y)

(perm (app x (cons e y)) (cons e (app x y)))

C1. (listp x)
C2. (listp y)
C3. (endp x)

(app x (cons e y))
≡ {Def app, C3}
(cons e y)
≡ {Def app, C3}
(cons e (app x y))

C1. (listp x)
C2. (listp y)
C3. ~~¬~~(endp x)
C4. (listp (rest x)) ∧ (listp y) ⇒ (perm (app (rest x) (cons e y)) (cons e (app (rest x) y)))

C5. (listp (rest x))
C6. (perm (app (rest x) (cons e y)) (cons e (app (rest x) y))) {C4, C5, C2, MP}
C7. (in (first x) (app x y)) {Def in, first-rest axioms, Def app}
C8. (in (first x) (cons e (app x y))) {C7, Def in, consp axioms, Def endp, PL}

(perm (app x (cons e y)) (cons e (app x y)))
≡ {Def perm, C3, C8, PL, Def app}
(perm (~~cons (first~~ (app (rest x) (cons e y)) (del (first x) (cons e (app x y)))
Case 1: e = (first x)
≡ {Def del}
(perm (app (rest x) (cons e y)) (app x y))
≡ {Def app, e = (first x)}
(cons e (app (rest x) y))
(perm (app (rest x) (cons e y)) (cons e (app (rest x) y))
≡ {C8}
t

Case 2 e ≠ (first x)
≡ {Def del, first-rest axios, def app}
(Cons e (app (rest x) y))

ssort-t = ssort

~~[illegible]-t~~ del-~~[illegible]~~-t = del

```
(defunc del* ~~[illegible]~~(e x)
  :ic (listp x)
  :oc (listp (del* e x))
  (rev (del-t e x nil)))

(defunc del-t (e x acc)
  :ic (and (listp x)(listp acc))
  :oc (listp (del-t e x acc))
  (cond ((endp x) ~~[illegible]~~ acc))
        ((equal e (first x)) (~~[illegible]~~ app (rev (rest x)) acc)))
        (t (del-t e (rest x) (cons (first x) acc)))))
```

L1: (del-t e x acc) = (app (rev (del e x)) acc)

c1. (listp x)

c2. (listp acc)

c3. ¬(endp x)

c4. (e = (first x))

(del-t e x acc)

= { Def del-t, c3, c4 }

(app (rev (rest x)) acc)

= { Def del, c4, c3 }

(app (rev (del e ~~[illegible]~~ x)) acc)

a b c d e

acc = '(b a)

(del c x) = '(d e)

del-t (~~[illegible]~~ (e d) ba)