```
(defunc f (l1 l2)
  :ic (and (listp l1)(listp l2))
  :oc (listp (f l1 l2))
  (cond ((and (endp l1)(endp l2)) nil)
        ((endp l1) (cons (first l2)(f l1 (rest l2))))
        (t         (cons (first l1)(f (rest l1) l2)))))
```

---

1) $\neg(\text{listp } l) \Rightarrow \phi$
2) $(\text{listp } l) \wedge (\text{endp } l) \Rightarrow \phi$
3) $(\text{listp } l) \wedge \neg(\text{endp } l) \wedge (e = (\text{first } l)) \Rightarrow \phi$
4) $(\text{listp } l) \wedge \neg(\text{endp } l) \wedge \neg(e = (\text{first } l)) \wedge \phi_{((l\ (\text{rest } l)))} \Rightarrow \phi$

```
(defunc foo (x_1 ... x_n)
  : input-contract ic
  : output-contract oc
  (cond ((t_1   c_1)
         (t_2   c_0)
          ⋮
         (t_m   c_m)
         (t     c_{m+1})))
```

1) $\neg IC \Rightarrow \varphi$

2) $IC \wedge t_1 \Rightarrow \varphi$

⋮

i) $IC \wedge \neg t_1 \wedge \neg t_2 \wedge \neg t_3 \dots t_i \wedge \bigwedge_{1 \le j \le R_i} \varphi|^{j}_{s_i} \Rightarrow \varphi$

Reverse

1) $\neg((listp\,x) \wedge (listp\,y)) \Rightarrow \varphi$

2) $(listp\,x) \wedge (listp\,y) \wedge (endp\,x) \Rightarrow \varphi$

3) $(listp\,x) \wedge (listp\,y) \wedge \neg(endp\,x) \wedge \varphi|_{((x\ (rest\,x)))} \Rightarrow \varphi$

Each recursive call gives an inductive assumption with the parameters changed accordingly

```
(defunc crapp (x y)
  :ic (and (listp x) (listp y))
  :oc (listp (crapp x y))
  (if (endp x)
      nil
      (cons 'a (crapp (rest x) y))))
```

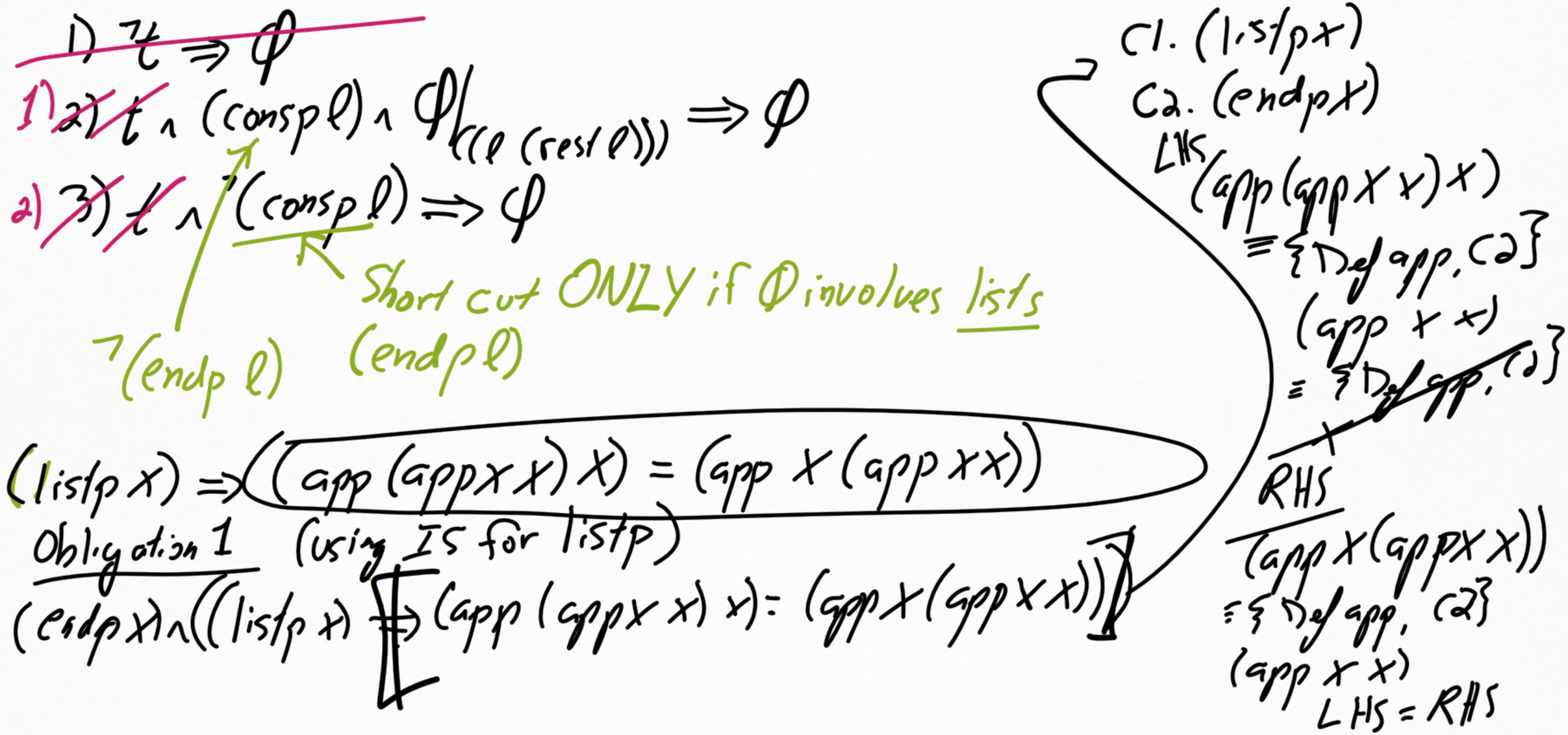
```
(defunc listp (l)
  :ic t
  :oc (boolean p (listp l))
  (if (consp l)
      (listp (rest l))
      (equal l nil)))
```

Generic Lists

- app (extra var)
- rev
- tser
- len
- in
- del-all
- sum-nl
- count-all

} Low

- listp


1) ¬t ⇒ φ
1) 2) t ∧ (consp l) ∧ φ|((l (rest l))) ⇒ φ
2) 3) t ∧ ¬(consp l) ⇒ φ
Short cut ONLY if φ involves lists
¬(endp l)
(endp l)
(listp x) ⇒ ((app (app x x) x) = (app x (app x x)))
(using IS for listp)
Obligation 1
(endp x) ∧ ((listp x) ⇒ (app (app x x) x) = (app x (app x x)))
C1. (listp x)
C2. (endp x)
LHS
(app (app x x) x)
≡ {Def app, C2}
(app x x)
≡ {Def app, C2}
x
RHS
(app x (app x x))
≡ {Def app, C2}
(app x x)
LHS = RHS

$\neg(endp\ x) \Rightarrow \phi|_{((x\ (rest\ x)))}$ $\quad (listp\ x) \Rightarrow ((app\ x\ (app\ x\ x)) = (app\ (app\ x\ x)\ x))$

C1. $(listp\ x)$

C2. $\neg(endp\ x)$

C3. $(listp\ (rest\ x)) \Rightarrow (app\ (rest\ x)\ (app\ (rest\ x)\ (rest\ x)) = (app\ (app\ (rest\ x)\ (rest\ x))\ (rest\ x)))$

C4. $(listp\ (rest\ x))$ {C1, C2. Def listp}

C5. α {C4, C3, MP}

$(app\ (first\ x)\ (app\ x\ x))$.

$\equiv$ {Def app, C2}

$(cons\ (first\ x)\ (app\ (rest\ x)\ (app\ x\ x)))$

α

$\equiv$ I need to generalize

The MONSTER

$(listp\ l) \Rightarrow (rev(rev\ l)) = l$

Using the I.S for listp

1) $(endp\ l) \Rightarrow \phi$
2) $\neg(endp\ l) \wedge \phi|_{((l\ (rest\ l)))} \Rightarrow \phi$

consp in reality

Obligation 1

C1. $(listp\ l)$
C2. $(endp\ l)$
$(rev(rev\ l))$

$\equiv \{C2, Def\ rev\}$
$(rev\ l)$
$\equiv \{C2, Def\ rev\}$
l

Obligation 2

C1. $(listp\ l)$
C2. $\neg(endp\ l)$
C3. $(listp(rest\ l)) \Rightarrow (rev(rev(rest\ l))) = (rest\ l)$
C4. $(listp\ (rest\ l))$ $\{C2, C1, Def\ listp\}$
C5. $(rev\ (rev(rest\ l))) = (rest\ l)$ $\{C3, C4, MP\}$
$(rev\ (rev\ l))$
$\equiv \{Def\ rev, C2\}$

$(rev\ (app\ (rev\ (rest\ l))\ (list\ (first\ l))))$

$\equiv$ { L1 / $((a\ (rev\ (rest\ l)))\ (b\ (list\ (first\ l))))$ }

$(app\ (rev\ (list\ (first\ l)))\ (rev\ (rev\ (rest\ l))))$

$=$ {c5}

$(app\ (rev\ (list\ (first\ l)))\ (rest\ l))$

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

$(app\ (app\ (rev\ nil)\ (list\ (first\ l)))\ (rest\ l))$

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

$(app\ (list\ (first\ l))\ (rest\ l))$

$=$ {Def list, consp axiom, Def endp, Def app}

$(cons\ (first\ l)\ (app\ nil\ (rest\ l)))$

$\equiv (cons\ (first\ l)\ (rest\ l))$

{first-rest axioms}

$\square$

$(app\ (rev\ b)\ (rev\ a))$

$a = '(1\ 2\ 3)$

$b = '(a\ b\ c)$

$(rev\ (app\ a\ b)) = '(c\ b\ a\ 3\ 2\ 1)$

L1: $(listp\ a) \wedge (listp\ b) \Rightarrow (rev\ (app\ a\ b)) = (app\ (rev\ b)\ (rev\ a))$

Proving like a pro

IS for listp

C1. (listp a)
C2. (listp b)
C3. ¬(endp a)
C4. (listp (resta a)) ∧ (listp b)
⇒ (rev (app (resta a) b)) = (app (rev b) (rev (resta a)))
C5. (listp (resta a)) {C3, C1}
C6. α {C4, C5, C2, MP}

(rev (app a b))
≡ {Def app, C3}
(rev (cons (first a) (app (resta a) b)))
≡ {Def rev, cons axiom, Def endp}
(app (rev (app (resta a) b)) (list (first a)))
≡ {C6 ⇐ C6 | (a (resta a))}
(app (app (rev b) (rev (resta a))) (list (first a)))
≡ {Assoc of app}
(app (rev b) (app (rev (resta a)) (list (first a))))
≡ {Def rev}
(app (rev b) (rev a))

Obl
C1. (listp a)
C2. (listp b)
C3. (endp a)

LHS
(rev (app a b))
≡ {C3, Def app}
(rev b)

RHS
(app (rev b) (rev a))
≡ {L2, C3}
(rev b) ✓

(listp x) ⇒ ((app x nil) = x)

First row: Prove using IS for app'

Rows 2 to last: Prove using listp

Last row: Prove like a pro.