1.) <u>To Prove that</u> $LIST(SEXP_{FG}) \subseteq SEXP_{FG}$

For an S-EXP to be fully general S-Expression it has to satisfy the below:

a.) It should be a symbol, number or Boolean.

b.) A list of values is a value

c.) If v_1 and v_2 are values, $(cons\ v_1 v_2)$ produces a value.

So, $SEXP_{FG}$ is either an atom or a pair of S-Expressions.

To define a list that consumes $SEXP_{FG}$ we must say it happens for each case.

- If $SEXP_{FG}$ is an atom i.e symbol, number, boolean or empty then its judgement form $v \in SEXP_{FG}$ is

$$\frac{v \in SYM}{v \in SEXP_{FG}} \qquad \frac{v \in NUM}{v \in SEXP_{FG}} \qquad \frac{v \in BOOL}{v \in SEXP_{FG}} \qquad \frac{}{'() \in SEXP_{FG}}$$

So, a list of any of the above values is a value. So, a list of $SEXP_{FG}$ is still a subset of $SEXP_{FG}$,

So, $LIST(SEXP_{FG}) \subseteq SEXP_{FG}$

- If $SEXP_{FG}$ is not empty list and we assume it to be a cons cell. then, $SEXP_{FG}$ has either v or vs as the result as (cons v vs) (assumption)

So, $$\frac{v \in SEXP_{FG} \qquad vs \in SEXP_{FG}}{(cons\ v\ vs) \in SEXP_{FG}} \quad — ①$$

$LIST(cons\ v\ vs) = \{(cons\ v\ vs) \mid v \in SEXP_{FG},\ vs \in SEXP_{FG}\}$

$SEXP_{FG} = SYM \cup NUM \cup BOOL \cup \{'()\} \cup \{(cons\ v\ vs) \mid v \in SEXP_{FG},\ vs \in SEXP_{FG}\}$

and $LIST(SEXP_{FG})$ is the smallest set that satisfies.

So, $$\frac{v \in SEXP_{FG} \qquad vs \in LIST(SEXP_{FG})}{(cons\ v\ vs) \in LIST(SEXP_{FG})} \quad — ②$$

From ① & ② we can tell $LIST(SEXP_{FG}) \subseteq SEXP_{FG}$.

3.) Exercise 31 - Pg 203

To Prove that (length (reverse xs)) = (length xs)

Using simple-reverse for this Proof. Pg 104

<u>Base Case</u> : If xs is empty, xs = nil

```
= (length (simple-reverse '()))
```

{ substituting actual parameters in the definition of length function }

```
= (if (null? (simple-reverse '()))
        0
      (+ 1 (length (cdr '()))))
```

{ substituting actual parameters in the definition of simple-reverse f'n }

```
= (if (null?
         (if (null? '())
               '()
               (append (simple-reverse cdr '()) (list1 (car '())))))
        0
      (+ 1 (length (cdr (simple-reverse '())))))
```

{ null?- empty list law }

```
= (if (null?
        (if #t '() (append (simple-reverse cdr '()) list1 (car '()))))
       0
      (+ 1 (length (cdr (simple-reverse '())))))
```

{ if #t law }

```
= (if (null? '())
       0
      (+ 1 (length (cdr (simple-reverse '())))))
```

{ null? empty list law }

```
= (if #t)
       0
      (+ 1 (length (cdr (simple-reverse '())))))
```

{ if #t law }

```
= 0
```

{ Since, length of empty list is zero }

```
= (length '())
```

So, Base Case (xs = nil) is true.

Inductive Case xs ≠ nil

Consider xs = (cons y ys)

```
  (length (simple-reverse xs))
                                    { substituting xs = (cons y ys) }
= (length (simple-reverse (cons y ys)))
                                    { substitute actual parameters into the
                                      definition of simple-reverse function }
= (length (if (null? (cons y ys))
              (cons y ys)
              (append (simple-reverse (cdr (cons y ys))) (list1 (car (cons y ys))))))
                                    { null? cons law }
= (length (if #f
              (cons y ys)
              (append (simple-reverse (cdr (cons y ys))) (list1 (car (cons y ys))))))
                                    { if #f law }
= (length (append (simple-reverse (cdr (cons y ys))) (list1 (car (cons y ys)))))
                                    { car - cons law }
= (length
     (append (simple-reverse (cdr (cons y ys))) (list1 (y))))
                                    { cdr - cons law }
= (length
     (append (simple-reverse (ys)) (list1 (y))))
                                    { Pg 102  list1 (x) = (cons x '()) }
= (length
     (append (simple-reverse (ys)) (cons y '())))
                                    { Pg 117 (length (append xs ys)) = (+ (length xs) (length ys)) }
= (+ (length (simple-reverse ys)) (length (cons y '())))
                                    { substitute parameter into the definition of length }
= (+ (length (simple-reverse ys))
     (if (null? (cons y '()))
         0
         (+ 1 (length (cdr (cons y '()))))))
```

```
                              { cdr-cons law }
=  (+   (length (simple-reverse ys))
        (if (null? (cons y '()))
            0
            (+ 1 (length '()))))

                              { null? cons law }
=  (+   (length (simple-reverse ys))
        (if #f
            0
            (+ 1 (length '()))))
                              { if #f law }
=  (+   (length (simple-reverse ys))
        (+ 1 (length '())))
                              { induction-hypothesis, (length (reverse ys)) = (length ys) }
=  (+   (length ys)
        (+ 1 (length '())))
                              { substitute parameters into the definition of length function }
=  (+   (length ys)
        (+ 1 (if (null? '()) 0 (+ 1 (length (cdr '()))))))
                              { null? empty law }
=  (+   (length ys)
        (+ 1 (if #t 0 (+ 1 (length (cdr '()))))))
                              { if #t law }
=  (+   (length ys)
        (+ 1 0))
                                        { addition is commutative }
=  (+   (length ys) 1)          =  (+ 1 (length ys))
                              { Pg 102 (length (cons v vs)) = (+ 1 (length vs)) }
=  (length (cons y ys))   =  (length xs)   { By assumption xs = (cons y ys) }
      So, (length (reverse xs)) = (length xs)
```

A.) 1.) To Prove that (cdr (cons x xs)) == xs

Using the style of Operational Semantics of List Operations, Impcore Style

$$\frac{\langle cdr, \xi_0, \phi, \rho_0\rangle \Downarrow \langle PRIMITIVE(cdr), \xi_1, \phi, \rho_1\rangle \qquad \dfrac{\begin{array}{c}\langle CONS, \xi_1, \phi, \rho_1\rangle \Downarrow \langle PRIMITIVE(CONS), \xi_2, \phi, \rho_2\rangle \\ \langle x, \xi_2, \phi, \rho_2\rangle \Downarrow \langle v_1, \xi_3, \phi, \rho_3\rangle \\ \langle xs, \xi_3, \phi, \rho_2\rangle \Downarrow \langle v_2, \xi_4, \phi, \rho_4\rangle * \\ (x \text{ evaluates to } v_1) \\ (xs \text{ evaluates to } v_2)\end{array}}{\langle APPLY(CONS, x, xs), \xi_1, \phi, \rho_1\rangle \Downarrow \langle CONS\langle v_1, v_2\rangle, \xi_4, \phi, \rho_4\rangle}}{\langle APPLY(cdr, (cons\ x\ xs)), \xi_0, \phi, \rho_0\rangle \Downarrow \langle v_2, \xi_4, \phi, \rho_4\rangle}$$

(when e_2 i.e xs evaluates the result is $\langle v_2, \xi_4, \phi, \rho_4\rangle$) *

So,

(cdr (cons x xs)) == xs

Scanned by CamScanner

A.) 2.) Given that e_1 & e_2 are arbitrary expressions where the

- evaluation of e_1 terminates
- evaluation of e_2 terminates
- evaluation of (cdr (cons e1 e2)) terminates
- evaluation of (cdr (cons e1 e2)) == e_2

We consider the cases of e_1 & e_2 to be SYMBOL, NUMBER, BOOLEAN, EMPTY in combinations.

e_1 ↓ / e_2 →	SYM	BOOL	NUM	'()
SYM	(SYM . SYM)	(SYM . BOOL)	(SYM . NUM)	(SYM)
BOOL	(BOOL . SYM)	(BOOL . BOOL)	(BOOL . NUM)	(BOOL)
NUM	(NUM . SYM)	(NUM . BOOL)	(NUM . NUM)	(NUM)
'()	(() . SYM)	(() . BOOL)	(() . NUM)	(())

The values in boxes are results of (cons e_1 e_2)

From Pg 99, If V_1 & V_2 are values then (cons V_1 V_2) is also a value.

I plan to do (cdr (cons e_1 e_2)) of the above results then,

(cdr (cons SYM_1, SYM_2)) = SYM_2 ⊙
(cdr (cons SYM BOOL)) = BOOL ■
(cdr (cons SYM NUM)) = NUM #
(cdr (cons SYM '())) = '() ★
(cdr (cons BOOL SYM)) = SYM ⊙
(cdr (cons $BOOL_1$, $BOOL_2$)) = $BOOL_2$ ■
(cdr (cons BOOL NUM)) = NUM #
(cdr (cons BOOL '())) = '() ★

(cdr (cons NUM SYM)) = SYM ⊙
(cdr (cons NUM BOOL)) = BOOL ■
(cdr (cons NUM_1, NUM_2)) = NUM_2 #
(cdr (cons NUM '())) = '() ★
(cdr (cons '() SYM)) = SYM ⊙
(cdr (cons '() BOOL)) = BOOL ■
(cdr (cons '() NUM)) = NUM #
(cdr (cons '() '())) = '() ★

We can observe that if

★ If e_2 is '() then no matter what is e_1 it always results in '()
⊙ If e_2 is SYM then whatever e_1 is it still evaluates to e_2
■ If e_2 is BOOL then whatever e_1 is it results in e_2
If e_2 is NUM then e_1 can be anything but e_2 is always the result

So, from this analysis we can confirm that the evaluation of e_1 terminates, e_2 terminates, (cdr (cons e_1 e_2)) terminates then the evaluation of (cdr (cons e_1 e_2)) == e_2

Effectively we have 4 cases, when e_2 = SYM | BOOL | NUM | EMPTY

So, e_1 = SYM / BOOL / NUM / EMPTY (It doesn't matter)

If e_2 is SYM then it evaluates to SYM

If e_2 is BOOL then it evaluates to BOOL

If e_2 is NUM then it evaluates to NUM

If e_2 is '() then it evaluates to '()

So, Operational semantics in Impcore Style,

$$\frac{\begin{array}{c} \langle CONS, \xi_1, \phi, \rho_1\rangle \Downarrow \langle PRIMITIVE(CONS), \xi_2, \phi, \rho_2\rangle \\ \langle e_1, \xi_2, \phi, \rho_2\rangle \Downarrow \langle v_1, \xi_3, \phi, \rho_3\rangle \\ \langle e_2, \xi_3, \phi, \rho_3\rangle \Downarrow \langle v_2, \xi_4, \phi, \rho_4\rangle \\ e_1 \text{ evaluates to } v_1 \\ e_2 \text{ evaluates to } v_2 \end{array}}{\langle cdr, \xi_0, \phi, \rho_0\rangle \Downarrow \langle PRIMITIVE(cdr), \xi_1, \phi, \rho_1\rangle \Downarrow \langle APPLY(CONS, e_1, e_2), \xi_1, \phi, \rho_1\rangle \Downarrow \langle CONS\langle v_1, v_2\rangle, \xi_4, \phi, \rho_4\rangle}$$

$$\overline{\langle APPLY(cdr, cons(e_1, e_2)), \xi_0, \phi, \rho_0\rangle \Downarrow \langle v_2, \xi_4, \phi, \rho_4\rangle}$$

So, clearly

(cdr (cons e_1 e_2)) == e_2 (as e_2 evaluates to $\langle v_2, \xi_4, \phi, \rho_4\rangle$)