Lists defined inductively

$\mathit{LIST}(A)$ is the smallest set satisfying this equation:

$$\mathrm{LIST}(A) = \{\texttt{'()}\} \cup \{\, \texttt{(cons}\ a\ as\texttt{)} \mid a \in A, as \in \mathrm{LIST}(A)\}$$

Equivalently, $\mathit{LIST}(A)$ is defined by these rules:

$$\frac{}{\texttt{'()} \in \mathit{List}(A)}\ \textbf{(Empty)}$$

$$\frac{a \in A \qquad as \in \mathit{List}(A)}{\texttt{(cons}\ a\ as\texttt{)} \in \mathit{List}(A)}\ \textbf{(Cons)}$$

One more inductive definition

A list of A is one of:

- The empty list `'()`
- `(cons a as)`, where `a` is an A and `as` is a list of A

Lists generalized: S-expressions

An ordinary S-expression is one of:

- An atom (symbol, number, Boolean)
- A list of ordinary S-expressions

Can write literally in source, with quote

μScheme vs Impcore

New abstract syntax:

- LET (keyword, names, expressions, body)
- LAMBDAX (formals, body)
- APPLY (exp, actuals)

New concrete syntax for LITERAL:

`(quote` *S-expression*`)`

`'` *S-expression*

Equations and function for append

```
(append '()          ys) == ys

(append (cons z zs) ys) == (cons z (append zs ys))


(define append (xs ys)

  (if (null? xs)

      ys

      (cons (car xs) (append (cdr xs) ys))))
```

Naive list reversal

```
(define reverse (xs)
   (if (null? xs)
       ’()
       (append (reverse (cdr xs))
               (list1 (car xs)))))
```

Reversal by accumulating parameters

```
(define revapp (xs ys)
   ; return (append (reverse xs) ys)
   (if (null? xs)
       ys
       (revapp (cdr xs)
               (cons (car xs) ys))))

(define reverse (xs) (revapp xs ’()))
```

A-list example

```
-> (find 'Building
         '((Course 105) (Building Barnum)
           (Instructor Ramsey)))
Barnum
-> (val nr (bind 'Office 'Halligan-222
           (bind 'Courses '(105 150TW)
           (bind 'Email 'comp105-grades '()))))
((Email comp105-grades)
 (Courses (105 150TW))
 (Office Halligan-222))
-> (find 'Office nr)
Halligan-222
-> (find 'Favorite-food nr)
()
```

Laws of association lists

```
(find k (bind k  v a-l)) = v
(find k (bind k′ v a-l)) = (find k a-l), provided k != k′
(find k ′()) =  ′() --- bogus!
```