Lists defined inductively

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

\[
\text{LIST}(A) = \{ \text{'()} \} \cup \{ (\text{cons } a \text{ as}) \mid a \in A, as \in \text{LIST}(A) \}
\]

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

- \( \text{'()} \in \text{List}(A) \) \hspace{2cm} (\text{EMPTY})
- \( a \in A \quad as \in \text{List}(A) \)
  \( (\text{cons } a \text{ as}) \in \text{List}(A) \) \hspace{1cm} (\text{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
\(\mu\)Scheme vs Impcore

New abstract syntax:
- LET (keyword, names, expressions, body)
- LAMBDA X (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

\[(\text{find } k \ (\text{bind } k \ v \ a-l)) = v\]
\[(\text{find } k \ (\text{bind } k' \ v \ a-l)) = (\text{find } k \ a-l), \text{ provided } k \neq k'\]
\[(\text{find } k \ '()) = '() --- bogus!\]