from last time: set-car!, set-cdr!
- trees

Dr. Kimberle Koile
more set-car! and set-cdr! problems

For the box & pointer diagram:
(a) Write what Scheme prints out for the structure (if it can)
(b) Write a Scheme expression that makes the structure (if an error, describe it)
(c) Draw the structure that results from the mutation, and its printed representation.

1. a. \( x \Rightarrow \)

b. Scheme expression:

\[
\begin{align*}
&\text{(define } x \\
&\quad (\text{let } ((a \ '9)) \ 1. \\
&\quad \text{let*} \\
&\quad \quad (\text{let*} (((b \ (\text{cons} \ a \ a)) \ 2. \\
&\quad \quad \text{cons} \ b \ b)))) \ 3. \\
&\quad \text{con live anything}
\end{align*}
\]

(c. mutation: (set-cdr! (car x) '8))

\[
\begin{align*}
x \Rightarrow (\text{(cons} \ (\text{car} \ x) \ (\text{cdr} \ x)) \\
\text{a. } x \Rightarrow \frac{(\text{(cons} \ (\text{car} \ x) \ (\text{cdr} \ x)) \ 3 \ \text{sets}}{\text{start with empty}}
\end{align*}
\]
more set-car! and set-cdr! problems

For the box & pointer diagram:
(a) Write what Scheme prints out for the structure (if it can)
(b) Write a Scheme expression that makes the structure (if an error, describe it)
(c) Draw the structure that results from the mutation, and its printed representation.

2.

a. \[ x \Rightarrow \]

\[ (((a\ b\ c))\ (c)) \]

b. Scheme expression:

\[ (define\ x\ (let\ ((w\ (\ a\ b\ c)))\ (\ car\ (\ list\ w)\ (\ car\ (\ car\ (\ cdr\ w)))))) \]

\[ x \Rightarrow \]

\[ (((a\ ba))\ (())\ a) \]

c. mutation: \( (set-cdr!\ (cddr\ x)\ (caar\ x)) \)
• A tree is a nested list; each node is a list of the children of that node
• A child is either another tree or a leaf node
  – A child that is a tree is called a *subtree*
  – A leaf node is anything that is not a pair (i.e., a symbol or a self-evaluating value).
1. Draw a box-and-pointer structure for the following tree. How does the interpreter print this structure?

Tree representation

Box & pointer

Printed representation: 

```
(((1 2) 3) 4 (5 6))
```
2a. Draw the interpretation of this list as a tree structure: \(((1 \ 2) \ 3) \ (4 \ (5 \ 6) \ 7) \ (8 \ 9 \ 10)\)

2b. Draw the box-and-pointer diagram.
counting leaves

(define (countleaves tree)
  (cond ((null? tree) 0)
        ((leaf? tree) 1)
        (else (+ (countleaves (car tree))
                  (countleaves (cdr tree))))))

(define (leaf? x)
  (not (pair? x)))

(1 ((2 3) 4) 5 6)
doubling a tree: version 1

(define (countleaves tree)
  (cond ((null? tree) 0)
        ((leaf? tree) 1)
        (else (+ (countleaves (car tree))
                  (countleaves (cdr tree))))))

(define (leaf? x)
  (not (pair? x)))

(define (double-tree tree)
  (cond ((null? tree) '())
        ((leaf? tree) (x 2 tree))
        (else (cons (double-tree (car tree))
                    (double-tree (cdr tree))))))

(1 ((2 3) 4) 5 6) => (2 ((4 6) 8) 10 12)
doubling a tree: version 2, map

v. 1 (define (double-tree tree)
    (cond ((null? tree) ‘())
          ((leaf? tree) double-tree)
          (else (cons (double-tree (car tree)) (double-tree (cdr tree))))
    )))

v. 2 (define (double-tree tree)
    (if (leaf? tree)
        (cond ((null? tree) ‘())
              ((leaf? tree) double-tree)
              (else (double-tree (car tree))
                    (map double-tree tree))))
    )))

(1 (((2 3) 4) 5 6)) =>
(2 (((4 6) 8) 10 12)
doubling a tree: version 3, map-tree

v. 1 (define (double-tree tree)
     (cond ((null? tree) ‘())
           ((leaf? tree)                              
             (x 2 tree)
           (else ( cons (double-tree (car tree))
                       (double-tree (cdr tree)))))))

(define (map-tree proc tree)
     (if (leaf? tree)
         (proc tree)
         (map-tree proc tree)))

(define (double x)
     (* 2 x))

v. 2 (define (double-tree tree)
     (if (leaf? tree)
         (x 2 tree)
         (map double-tree tree)))

(define (double-tree tree)
     (map (λ (tree) (map-tree proc tree))
          tree))

Why need λ?
fib arg mismatch: proc passed to map
takes 1 arg, not 2

v. 3 (define (double-tree tree)
     (map-tree double-tree))