- set-car! and set-cdr!
- ring problems
- more set-car!, set-cdr! problems

Dr. Kimberle Koile
compound data mutation

**constructor:**

(cons x y) creates a new pair

**selectors:**

(car p) returns car part of pair
(cdr p) returns cdr part of pair

**mutators:**

(set-car! p new-x) changes car pointer in pair
(set-cdr! p new-y) changes cdr pointer in pair

; Pair,anytype -> undef  --  side-effect only!
How can we tell if two things are equivalent?
-- What do you mean by "equivalent"?

1. **The same object**: test with eq?
   \[(eq? \ a \ b) ==> \ #t\]

2. **Objects that "look" the same**: test with equal?
   \[(equal? \ (\text{list} \ 1 \ 2) \ (\text{list} \ 1 \ 2)) ==> \ #t\]
   \[(eq? \ (\text{list} \ 1 \ 2) \ (\text{list} \ 1 \ 2)) ==> \ #f\]
example 1: pair/list mutation

(define a (list 1 2))
(define b a)
  a ==> (1 2)
  b ==> (1 2)

(set-car! a 10)

b ==> (10 2)

Compare with:

(define a (list 1 2))
(define b (list 1 2))

(set-car! a 10)

b ==> (1 2)
example 2: pair/list mutation

(define x (list 'a 'b))

How is x mutated to achieve the result at right?

And this one?

(set-car! (cdr x) (list 1 2))

1. Eval (cdr x) to get a pair object
2. Change car pointer of that pair object
For the given expressions:
(a) Draw the box and pointer diagram corresponding to the list or pair structure
(b) Write what Scheme prints out after evaluating the last expression in the sequence

1. (define x (cons 7 (list 8 9)))
   (set-car! (cdr x) 10)

a. box and pointer diagram for x
   ![Box and Pointer Diagram for x]

   b. printed result for x
   \[(7 \ 10 \ 9)\]


set-car! and set-cdr! problems

For the given expressions:
(a) Draw the box and pointer diagram corresponding to the list or pair structure
(b) Write what Scheme prints out after evaluating the last expression in the sequence

2.  (define y  '(7))
    (define z  (let ((x (list 'a '(b c) (car y))))
      (set-car! y (cdr x)) .
      (set-cdr x (car (cdr x))) .
      x))

   z

a. box and pointer diagram for x, y and z  

   

b. printed result for z

   (a b c)

instructor notes
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.

3. a. \( x \Rightarrow \) \\
    \( ((3 \ 2 \ 1) \ (2 \ 1) \ (1)) \)

    b. Scheme expression:

    (define x
      (let ((z ' (3 z 1)))
        (list z (cdr z) (cadr z))))

    c. mutation: (set-car! (cdr (second x)) 4)

    \( x \Rightarrow \) \\
    \( ((3 \ 2 \ 4) \ (2 \ 4) \ (4)) \)
Rings are circular structures similar to lists. If we define a ring \( r \):
\[
\text{(define } r \text{ (make-ring \( '(1 \ 2 \ 3 \ 4) \)) )}
\]
the following are true:
\[
\begin{align*}
\text{(nth 0 } r \text{)} & \Rightarrow 1 \\
\text{(nth 1 } r \text{)} & \Rightarrow 2 \\
\vdots \\
\text{(nth 4 } r \text{)} & \Rightarrow 1
\end{align*}
\]

In order to make a ring, we need a procedure \text{last-pair} which returns the last pair in its argument:
\[
\text{(last-pair (list 1 2 3 4))} \Rightarrow (4)
\]

1. Write \text{last-pair}.

\[
\begin{align*}
\text{(define (last-pair } x \text{)} & \\
& \quad \text{(if (null? (cdr } x \text{)) } x \text{ (last-pair (cdr } x \text{)))} \\
\text{or } & \quad \text{(car } x \text{) (error "..." } x \text{ )} \\
& \quad ((\text{not (list? } x \text{)) } x \text{) \text{; pain}} \\
& \quad ((\text{null? (cdr } x \text{) } x) \\
& \quad (\text{else (last-pair (cdr } x \text{))))}
\end{align*}
\]
ring problems

2. Write make-ring!, which takes a list and makes a ring out of it.

(define (make-ring! x)
  (set-cdr! (last-pair x) x)
  x)
)
3. Write the procedure rotate-left, which takes a ring and returns a ring that has been rotated one to the right.
   (define r1 (rotate-left r))
   (nth 0 r1) => 2

   (define (rotate-left ring)
     (car ring))
   )
ring problems

4. What happens if you evaluate (length r) on the above ring?

Write the procedure ring-length, which returns the length of the original list used in constructing the ring. (Hint: Write a helper procedure.)

```
(define (ring-length ring)
  (define (helper n here)
    (if (eq? here ring)
        n
        (helper (+ 1 n) (car here))))
  (helper 1 (car ring)))
```
5. Rotating a ring to the right is harder than rotating to the left. (Why?) Write the procedure rotate-right. (Hint: You might want to use the procedure repeated, which takes a procedure, a number n, and an argument to the procedure, and repeatedly calls the op on the argument n times.)

```
(define (rotate-right ring)
  (repeated rotate-left
    (- (ring-length ring) 1)
    ring))
)
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 =>
   b. Scheme expression:
      ```scheme
      (define x
          (let ((a (cons 9 9)))
              (let* ((b (cons a a))
                     (c (cons b b)))
                  (c)))
      ```
   c. mutation: (set-cdr! (car x) '8)

   x =>
   ```scheme
   (((9) 8) (9) 8)
   ```
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 =>
   (((a b c)) (c))

b. Scheme expression:
   (define x
     (let ((w '(a b c)))
       (cons (list w)
             (cons '() (cddr w)))))

   x =>
   (((a b a)()) a)

c. mutation: (set-cdr! (cddr x) (caaar x))
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.

3. a. x  =>
b. Scheme expression:
   (define x
     (let ((two '2))
       (list (car 1 two)
             (list 1) two)))
c. mutation: (set-car! (caar x) 3)

x =>
   error: (caar x) => 1
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.

4. a.    x  =>

b. Scheme expression:

\[
\text{define } x \\
\text{let } \langle k, '(a) \rangle \\
\text{let } \langle m, \text{list } k \ k \rangle \\
\text{cons } m \ \text{cons (car } m))
\]

4. c.  mutation: (set-cdr! (first x) (second x))

\[
\text{define } x \\
\text{let } \langle k, '(a) \rangle \\
\text{let } \langle m, \text{list } k \ k \rangle \\
\text{cons } m \ \text{cons (car } m))
\]

\[
x \Rightarrow \\
\langle \langle(\text{a} \ (\text{a} \text{) (a)\rangle})
\]
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.

5. a. x =>

b. Scheme expression:

\[
\text{(define } x \\
\text{(let } ((c \text{ (car } '()) '()) \\
\text{(let } ((a \text{ (car } c c)) \\
\text{(set-car! } c d) \\
\text{(set-cdr! } c d) \\
\text{c')}) \text{))}
\]

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

x =>

\[
((())())
\]