6.001 recitation 3/21/07

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

Dr. Kimberle Koile
compound data mutation

**constructor:**

\[(\text{cons } x \ y)\] creates a new pair

**selectors:**

\[(\text{car } p)\] returns car part of pair

\[(\text{cdr } p)\] returns cdr part of pair

**mutators:**

\[(\text{set-car! } p \ \text{new-}x)\] changes car pointer in pair

\[(\text{set-cdr! } p \ \text{new-}y)\] changes cdr pointer in pair

`Pair,\text{anytype} \rightarrow \text{undef} \qquad \text{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?`
   
   \[(\text{eq? } a \ b) \Rightarrow \#t\]

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

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

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

(set-car! a 10)

Compare with:

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

(set-car! a 10)
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! 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

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

  a. box and pointer diagram for x

  b. printed result for x
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

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
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.  

b. Scheme expression:

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

<table>
<thead>
<tr>
<th>a.</th>
<th>x =&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>b.</td>
<td>Scheme expression:</td>
</tr>
<tr>
<td>c.</td>
<td>x =&gt;</td>
</tr>
</tbody>
</table>
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: 
\[
\text{(nth 0 } r \text{) }\Rightarrow 1 \quad \text{(nth 1 } r \text{) }\Rightarrow 2 \quad \ldots \quad \text{(nth 4 } r \text{) }\Rightarrow 1
\]

In order to make a ring, we need a procedure `last-pair` which returns the last pair in its argument:

\[
\text{(last-pair } (\text{list } 1 2 3 4)\text{) }\Rightarrow (4)
\]

1. Write `last-pair`.

\[
\text{(define (last-pair } x \text{)}
\]
2. Write make-ring!, which takes a list and makes a ring out of it.

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

   (define (rotate-left ring) )
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) )
5. Rotating a ring to the right is harder than rotating to the left. (Why?) Write the procedure \textit{rotate-right}. (Hint: You might want to use the procedure \textit{repeated}, which takes a procedure, a number \textit{n}, and an argument to the procedure, and repeatedly calls the \textit{op} on the argument \textit{n} times.)

\begin{verbatim}
(define (rotate-right ring)
)
\end{verbatim}
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:
c. mutation: \( \text{set-cdr! (car x) '}(8) \)
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.  
   
   b. Scheme expression:
   
   c. mutation: (set-cdr! (cddr x) (caar x))
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. [Diagram]

   a. $x =>$

   b. Scheme expression:

   c. mutation: (set-car! (car x) 3)

 x =>
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:

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

| x => |
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. \( \mathbf{x} \Rightarrow \)
b. Scheme expression:
   
   \[
   \text{c. mutation: } (\text{set-car!} \ (\text{cdr} \ \mathbf{x}) \ '())
   \] \[
   (\text{set-cdr!} \ (\text{car} \ \mathbf{x}) \ '())
   \]

\[
\text{x} \Rightarrow
\]