The last racket...

- set-equal redux
- code-review
- quiz
- ReasonML intro
Announcement

• Flu shots today in Sayles Hall
• Effectiveness substantially reduced if you didn't get a good night's sleep
Last time: set equality

- Represent sets as lists of items, with no repetitions.
- Order doesn't matter
- Ordinary list-equality *isn't* a test of set-equality!
  - (list 1 2) (list 2 1): different lists, same set!
- Sets B and C are equal if
  1. every member of B is a member of C ("B is a subset of C")
  2. every member of C is a member of B ("C is a subset of B")

(define (set-equal? b c) (and (subset? b c) (subset? c b))
Subset?

- is b a subset of c?
  - Is each element of b an element of c?

(define (subset? b c)
  (foldr (lambda (item result) (and (member? item c) result))
         true b))
What was wrong with that?

• suppose s1 is (list (list 1 2) (list 1)) represents a set whose elements are \{1,2\} and \{1\} (i.e., its elements are SETS!)
• suppose s2 is (list (list 1) (list 2 1))

Are these "set-equal?"
• NO! because the first item in s1 isn't "racket equal" to any element of s2, even though it is equal, as a set, to the second element of s2.
Rewrite member to use an "equality testing" proc

;; s-member? : 'a * ('a list) * ('a * 'a -> bool) -> bool
;; test whether an item is equal to a member of the set
;; s, using the specified equality test

(define (s-member? item s eqtest)
  (cond
   [(empty? s) false]
   [(cons? s) (or (eqtest item (first s))
                   [(cons s) (s-member? item (rest s) eqtest))]]))
Rewrite subset!

• is b a subset of c?
  • Is each element of b an element of c?

(define (s-subset? b c eqtest)
  (foldr (lambda (item result)
        (and (s-member? item c eqtest) result))
         true b))
Rewrite set-equal? !

(define (s-set-equal? b c eqtest)
  (and (s-subset b c eqtest) (s-subset c b eqtest)))
Use this to define particular set-equality tests

\[(\text{define } (\text{s-set-equal? } b \ c \ \text{eqtest})\]
\[\quad \text{(and } (\text{s-subset } b \ c \ \text{eqtest}) (\text{s-subset } c \ b \ \text{eqtest}))\]

- equality-testing for *sets of integers*:
  \[(\text{define } (\text{int-set-equal? } b \ c)\]
  \[\quad (\text{s-set-equal? } b \ c =)\]

- equality-testing for *sets of integer-sets*:
  \[(\text{define } (\text{int-set-set-equal? } b \ c)\]
  \[\quad (\text{s-set-equal? } b \ c \ \text{int-set-equal})\]

- Use these last two for your subsets check-expects!
Testing :"subsets"

• Using this to test subsets, i.e., compare two sets of int-sets:
  (check-expect
   (int-set-set-equal?
    (subsets (list 1))
    (list (list 1) empty)) true)
(define (flip alop)
  (cond
    [(empty? alop) empty]
    [(cons? alop)
      (cons (list (second (first alop)) (first (first alop)))
            (flip (rest alop)))]))
Improved (maybe)

(define (flip alop)
  (cond
    [(empty? alop) empty]
    [(cons? alop)
     (let ((pair (first alop)))
       (cons (list (second pair) (first pair))
             (flip (rest alop))))])))
(define (my-member? item aloi)
  (cond
    [(empty? aloi) false]
    [(and (cons? aloi) (= item (first aloi))) true]
    [(and (cons? aloi) (not (= item (first aloi))))
      (my-member? item (rest aloi))])))

• Problem: code follows both structure and content of data; should be just structure
Improved (1)

(define (my-member? item aloi)
   (cond
      [(empty? aloi) false]
      [(cons? aloi) (if (= item (first aloi))
                      true
                      (my-member? item (rest aloi)))]))

Problem: if-expression with a boolean return value – ick!
Improved (2)

(define (my-member? item aloi)
  (cond
   [(empty? aloi) false]
   [(cons? aloi) (or (= item (first aloi))
                   (my-member? item (rest aloi)))]))
Quiz setup: A silly procedure

> (define (silly alon)
  (cond
    [(empty? alon) (list 1)]
    [(cons? alon) (append (silly (rest alon))
                            (silly (rest alon)))]))

> (silly empty)
(list 1)

> (silly (list 3))
(list 1 1)

> (silly (list 4 5))
(list 1 1 1 1)

> (silly (list 1 3 6))
(list 1 1 1 1 1 1 1 1)
Quiz
Solution
On to ReasonML!