CS 135 Winter 2017

Tutorial 9: Abstract List Functions
Review: Lambda

\(((\text{lambda} \ (x_1 \ldots \ x_n) \ \text{exp}) \ v_1 \ldots \ v_n) => \exp'\)

where \(\exp'\) is \(\exp\) with all occurrences of \(x_1\) replaced by \(v_1\), all occurrences of \(x_2\) replaced by \(v_2\), and so on.

As an example:

\(((\text{lambda} \ (x \ y) \ (* \ (+ \ y \ 4) \ x)) \ 5 \ 6) => (* \ (+ \ 6 \ 4) \ 5)\)
Group Problem - Stepping with Lambda

In a text editor or on paper, do a full trace of the following code.

(((lambda (x y) (lambda (x) (* x y))) 5 6) 10)
Review: foldr

Recall the implementation of the abstract list function `my-foldr` as shown in Slide 73 of Module 10,

```scheme
(define (my-foldr combine base lst)
  (cond
    [(empty? lst) base]
    [else (combine (first lst)
      (my-foldr combine base (rest lst)))]))
```
Clicker Question - Maximum of a list

\[
(\text{define } (\text{my-foldr} \ \text{combine} \ \text{base} \ \text{lst})
\begin{align*}
&(\text{cond} \\
&\quad [(\text{empty?} \ \text{lst}) \ \text{base}] \\
&\quad [\text{else} \ \text{combine} \ \text{(first} \ \text{lst}) \\
&\quad \quad (\text{my-foldr} \ \text{combine} \ \text{base} \ \text{(rest} \ \text{lst})))])])
\end{align*}
\]

Using the \text{my-foldr} function defined above, which of the following expressions can be used to find the maximum value of a list of numbers \text{lon}?

A \ (\text{my-foldr} \ \text{max} \ \text{empty} \ \text{lon})

B \ (\text{my-foldr} \ \text{max} \ \text{(first} \ \text{lon}) \ \text{(rest} \ \text{lon}))

C \ (\text{my-foldr} \ \lambda(x \ y) \ (x \ y) \ \text{empty} \ \text{lon})

D \ (\text{my-foldr} \ \lambda(x \ y) \ (x \ y) \ \text{true} \ \text{lon})

E \ (\text{my-foldr} \ \lambda(x \ y) \ (x \ y) \ \text{(first} \ \text{lon}) \ \text{(rest} \ \text{lon}))
Clicker Question - Maximum of a list

(define (my-foldr combine base lst)
  (cond
    [(empty? lst) base]
    [else (combine (first lst)
      (my-foldr combine base (rest lst))))]]
)

Using the my-foldr function defined above, which of the following expressions can be used to find the maximum value of a list of numbers lon?

A (my-foldr max empty lon)
B (my-foldr max (first lon) (rest lon))
C (my-foldr (lambda (x y) (> x y)) empty lon)
D (my-foldr (lambda (x y) (> x y)) true lon)
E (my-foldr (lambda (x y) (> x y)) (first lon) (rest lon))
Group Problem - Stepping with foldr

In a text editor or on paper, do a full trace of the following code.

(foldr (lambda (x y) (cond [(even? x) (cons x y)] [else y])) empty '(1 1 2 3 5 8))
Group Problem - my-filter

Write a function `my-filter` which implements the built-in `filter` function using `foldr` and no explicit recursion.

```scheme
(check-expect (my-filter positive? (list 3 -1 24 0 6))
  (list 3 24 6))
```
Group Problem - within-radius

Write a function `within-radius` that consumes a listof Posn and a radius, and produces the listof Posn that are within the distance of the radius (inclusive) from the origin.

```scheme
(check-expect (within-radius (list (make-posn 3 4) (make-posn 5 12) (make-posn 6 8)) 6)
             (list (make-posn 3 4)))
```
Review: map

It was shown in the lecture that the function `map` works as follows.

\[(\text{map } f \ (\text{list } x_1 \ x_2 \ldots \ x_n)) \Rightarrow (\text{list } (f \ x_1) \ (f \ x_2) \ldots \ (f \ x_n))\]

However, `map` can also consume more than one list, for example:

\[(\text{map } f \ (\text{list } x_1 \ x_2 \ldots \ x_n) \ (\text{list } y_1 \ y_2 \ldots \ y_n)) \Rightarrow (\text{list } (f \ x_1 \ y_1) \ (f \ x_2 \ y_2) \ldots \ (f \ x_n \ y_n))\]
Group Problem - map-lofn

Write a function map-lofn which consumes an argument of type Any and a list of functions and produces a list of Any by applying each function from the list to the first argument.

;; (map-lofn item lofn) consumes an item of any type and a list of functions lofn, and produces a list containing the output of each function in lofn applied to item
;; map-lofn: X (listof (X → Any)) → (listof Any)
Group Problem - lookup-al

Recall the function lookup-al used to find the value associated with some key in an association list,

;; lookup-al: Num AL → (anyof Str false)
(define (lookup-al k alst)
  (cond
   [(empty? alst) false]
   [(= k (first (first alst))) (second (first alst))]
   [else (lookup-al k (rest alst))]))

Rewrite the function lookup-al without using explicit recursion. That is, nowhere in your function should you directly or indirectly call lookup-al. Consider which abstract list functions may be helpful. Any constants or helper functions should be defined locally.
Group Problem - only-using-letters? (Optional)

Write a function `only-using-letters?` which consumes a list of Chars, `loc`, and a String, `message`. Your function will produce `true` if the `message` is made up of only characters from `loc` and false otherwise. Any constants or helper functions should be defined locally. You may not use any explicit recursion.

(check-expect (only-using-letters? (list #\a #\b #\c) "abacaba") true)
(check-expect (only-using-letters? (list #\a #\b #\c) "abacdaba") false)