CS135 Winter 2020
Tutorial 9: Lambda and Abstract List Functions
Announcements

- Make sure to change your language level to Intermediate Student with lambda.

- Assignment 8 is due **Monday, March 30, 5:00 PM**. Note that this is different from the regular pattern of assignment due dates.

- Similarly, Assignment 9 is due **Friday, April 3rd, 11:59 PM**. It will be shorter than usual.
Debugging Foldr and Lambda

Consider a function `separate` that consumes a value called `sep` and a list. It inserts `sep` between identical items in the list.

```
(separate '(a b c) 'x)  ⇒  '(a b c)
(separate '(a b b c) 'x)  ⇒  '(a b x b c)
```

`;; separate: (listof Any) Any → (listof Any)`

```
(define (separate lst sep)
  (foldr (lambda (itm rr)
     (cond
       [(empty? rr) empty]
       [(equal? itm (first rr)) (cons itm (cons sep rr))]
       [else (list itm rr)])
     0 lst))
```

How many errors are in this function? Correct them.
Debugging Foldr and Lambda

;;; separate: (listof Any) Any → (listof Any)
(define (separate lst sep)
  (foldr (lambda (itm rr)
    (cond
      [(empty? rr) (list itm)]
      [(equal? itm (first rr)) (cons itm (cons sep rr))]
      [else (cons itm rr)]))
  empty lst))
Review: Lambda

This is how lambda is represented in racket:

\[
((\text{lambda } (x_1 \ldots x_n) \text{ expr}) v_1 \ldots v_n) \Rightarrow \text{ expr}'
\]

Where \text{ expr}' is \text{ expr} with all occurrences of \( x_1 \) substituted by \( v_1 \), all occurrences of \( x_2 \) replaced by \( v_2 \), and so on...

For example,

\[
((\text{lambda } (x \ y) (\ast (\text{\texttt{+}} y 4) x)) 5 6) \\
\Rightarrow (\ast (\text{\texttt{+}} 6 4) 5)
\]
Review: Abstract List Functions

Recall the abstract list functions filter, map, foldr, build-list.

These are the contracts:

;; filter: (X → Bool) (listof X) → (listof X)
;; map: (X → Y) (listof X) → (listof Y)
;; foldr: (X Y → Y) Y (listof X) → Y
;; foldl: (X Y → Y) Y (listof X) → Y
;; build-list: Nat (Nat → X) → (listof X)
Warmup: ALF Matchmaker!

Each of the below mini-problems can be solved by a single Racket ALF function call. As a reminder, some of the ALF calls we’ve seen are foldr, foldl, build-list, map, and filter. Match each of these to a problem it would be most useful for among our choices.

A  Given a list of file names, add "temp-" to the start of each file name.
B  Produce a list of numbers counting up to the days Shawerma Plus will reopen.
C  Reverse a list of file names.
D  Given a list of WordTree produce all the WordTree that are just strings.
E  Given a grocery list of symbols, determine if ’toilet-paper occurs more than 3 times in the list.
Warmup: ALF Matchmaker!

Each of the below mini-problems can be solved by a single Racket ALF function call. As a reminder, some of the ALF calls we’ve seen are foldr, foldl, build-list, map, and filter. Match each of these to a problem it would be most useful for among our choices.

(map) Given a list of file names, add "temp-" to the start of each file name.

(build-list) Produce a list of numbers counting up to the days Shawerma Plus will reopen.

(foldr/foldl) Reverse a list of file names.

(filter) Given a list of WordTree produce all the WordTree that are just strings.

(foldl/foldr) Given a grocery list of symbols, determine if 'toilet-paper occurs more than 3 times in the list.
Problem 1: Abstract List Tasks 1/2

You may not use explicit recursion in any of this week’s tutorial problems. Use abstract list functions to complete the following tasks:

Sum the numbers in a list:
\[(\text{sum-list} \ (\text{\texttt{'1 2 3 4 5 6}})) \Rightarrow 21\]

Double each number in a list:
\[(\text{double-list} \ (\text{\texttt{'1 2 3 4 5 6}})) \Rightarrow \text{\texttt{'2 4 6 8 10 12}}\]
Problem 1: Solutions 1/2

;; Sum the numbers in a list.
(define (sum-list lst)
  (foldr + 0 lst))

;; Double each number in a list.
(define (double-list lst)
  (map (lambda (x) (* x 2)) lst))
Problem 1: Abstract List Tasks 2/2

Use abstract list functions to complete the following tasks:

Keep all the numbers in a list that are divisible by 3:

\( \text{(keep-triple '(1 2 3 4 5 6)) } \Rightarrow '(3 6) \)

Create a list of odd numbers from 1 to n:

\( \text{(odd-list 6)} \Rightarrow '(1 3 5) \)
Problem 1: Solutions 2/2

;; Keep all the numbers in a list that are divisible by 3.
(define (keep-triple lst)
  (filter (lambda (x) (= 0 (remainder x 3))) lst))

;; Create a list of odd numbers from 1 to n.
(define (odd-list n)
  (filter odd? (build-list n add1)))
Problem 2: make-posns

Write a function `make-posns` which consumes a list of x values and a lists of y values. Both lists are of equal length and produces a list of posns, where the $i^{th}$ posn consists of the $i^{th}$ element in the first list and the $i^{th}$ element in the second list.

**Hint:** Remember that `map` can consume multiple lists.
Problem 2: make-posns Example

(check-expect (make-posns (list 1 2 3 4 5))
    (list 6 7 8 9 10))

    (list (make-posn 1 6)
        (make-posn 2 7)
        (make-posn 3 8)
        (make-posn 4 9)
        (make-posn 5 10)))
Problem 2: Solution

;; make-posns: (listof Num) (listof Num) → (listof Posn)
;; requires: list1 and list2 are of equal length
(define (make-posns list1 list2)
  (map (lambda (x y) (make-posn x y)) list1 list2))

Note that for make-posns, we do not need a redundant lambda function and we can improve the code as follows:

(define (make-posns list1 list2)
  (map make-posn list1 list2))
Problem 3: multi-odds-to

Write a function multi-odds-to which consumes a natural number and produces the product of all positive odd numbers that are less or equal to n.

(check-expect (multi-odds-to 5) 15)
Problem 3: Solution

;; multi-odds-to: Nat → Nat

(define (multi-odds-to n)
  (foldr ∗ 1 (filter odd? (build-list n add1)))))
Problem 4: map-lofn

Write a function `map-lofn` which consumes a `(listof Any)` and a list of functions. The functions in the consumed list will have the contract `Num → Any`. `map-lofn` produces a list of lists, where each sublist contains the result after applying each function from the consumed list to each number in the consumed `(listof Any)`.

```
(check-expect (map-lofn (list 3.5 'four 18 "q" 0)
                         (list sqr add1 zero?))
               (list (list 12.25 324 0)
                      (list 4.5 19 1)
                      (list false false true)))
```
Problem 4: Solution

;; map-lofn: (listof Any) (listof (Num → Any)) → (listof (listof Any))
(define (map-lofn loany lofn)
  (local
    [(define lonums (filter number? loany))]
    (map (lambda (fn) (map fn lonums)) lofn)))
Reading: Stepping with Lambda

$((\text{lambda} \ (x_1 \ldots \ x_n) \ \text{expr}) \ v_1 \ldots \ v_n) \Rightarrow \text{expr}'$

Provide a step-by-step evaluation of the following code:

$\Rightarrow ((\text{lambda} \ (x \ y \ z) \ (\ast \ x \ 5)) \ 9 \ (+ \ 1 \ 2) \ 8)$
Reading: Stepping with Lambda

\((\text{lambda} \ (x_1 \ldots \ x_n) \ expr) \ v_1 \ldots \ v_n) \Rightarrow expr'\)

Provide a step-by-step evaluation of the following code:

\[
\Rightarrow ((\text{lambda} \ (x \s y \ z) \ (\ast \ x \ 5)) \ 9 \ (\mathbin{+} \ 1 \ 2) \ 8)
\Rightarrow ((\text{lambda} \ (x \s y \ z) \ (\ast \ x \ 5)) \ 9 \ 3 \ 8)
\]
Reading: Stepping with Lambda

\[ ((\text{lambda} \ (x_1 \ldots \ x_n) \ \text{expr}) \ v_1 \ldots \ v_n) \Rightarrow \text{expr}' \]

Provide a step-by-step evaluation of the following code:

\[ \Rightarrow ((\text{lambda} \ (x \ y \ z) \ (\ast \ x \ 5)) \ 9 \ (+ \ 1 \ 2) \ 8) \]
\[ \Rightarrow ((\text{lambda} \ (x \ y \ z) \ (\ast \ x \ 5)) \ 9 \ 3 \ 8) \]
\[ \Rightarrow (\ast \ 9 \ 5) \]
Reading: Stepping with Lambda

\((\text{lambda} \ (x_1 \ldots \ x_n) \ \text{expr}) \ v_1 \ldots \ v_n) \Rightarrow \text{expr'}\)

Provide a step-by-step evaluation of the following code:

\[
\Rightarrow (\text{lambda} \ (x \ y \ z) \ (\times \ x \ 5)) \ 9 \ (\text{+} \ 1 \ 2) \ 8) \\
\Rightarrow (\text{lambda} \ (x \ y \ z) \ (\times \ x \ 5)) \ 9 \ 3 \ 8) \\
\Rightarrow (\times \ 9 \ 5) \\
\Rightarrow 45
\]
Reading: Stepping with Nested Lambda

\[ ((\text{lambda} \ (x_1 \ldots \ x_n) \ \text{expr}) \ v_1 \ldots \ v_n) \Rightarrow \text{expr}' \]

Provide a step-by-step evaluation of the following code:

\[ \Rightarrow (((\text{lambda} \ (x \ y) \ (\text{lambda} \ (x) \ (\ast \ x \ y))) \ 5 \ 6) \ 10) \]
Reading: Stepping with Nested Lambda.

\(((\text{lambda} \ (x_1 \ldots \ x_n) \ \text{expr}) \ v_1 \ldots \ v_n) \Rightarrow \text{expr}'\)

Provide a step-by-step evaluation of the following code:

\[\Rightarrow (((\text{lambda} \ (x \ y) \ (\text{lambda} \ (x) \ (* \ x \ y))) \ 5 \ 6) \ 10)\]
\[\Rightarrow ((\text{lambda} \ (x) \ (* \ x \ 6)) \ 10)\]
Reading: Stepping with Nested Lambda

\[
((\lambda (x_1 \ldots x_n) \text{expr}) v_1 \ldots v_n) \Rightarrow \text{expr}'
\]

Provide a step-by-step evaluation of the following code:

\[
\Rightarrow (((\lambda (x \ y) (\lambda (x) (\times x \ y))) 5 \ 6) 10)
\]
\[
\Rightarrow ((\lambda (x) (\times x \ 6)) \ 10)
\]
\[
\Rightarrow (\times 10 \ 6)
\]
Reading: Stepping with Nested Lambda

\[ ((\text{lambda } (x_1 \ldots x_n) \text{ expr}) v_1 \ldots v_n) \Rightarrow \text{ expr}' \]

Provide a step-by-step evaluation of the following code:

\[ \Rightarrow (((\text{lambda } (x \ y) (\text{lambda } (x) (* \ x \ y))) 5 \ 6) \ 10) \]
\[ \Rightarrow ((\text{lambda } (x) (* \ x \ 6)) \ 10) \]
\[ \Rightarrow (* \ 10 \ 6) \]
\[ \Rightarrow 60 \]