Higher-order List Functions in Racket

CS251 Programming Languages
Fall 2017, Lyn Turbak
Department of Computer Science
Wellesley College

Higher-order List Functions

A function is **higher-order** if it takes another function as an input and/or returns another function as a result. E.g. `app-3-5`, `make-linear-function`, `flip2`.

We will now study **higher-order list functions** that capture the recursive list processing patterns we have seen.

Recall the List Mapping Pattern

\[(\text{map}\,F\,(\text{list}\ v1\ v2\ \ldots\ vn))\]

\[
\begin{array}{c}
  v1 \quad v2 \quad \cdots \quad vn \\
  \downarrow \quad \downarrow \quad \cdots \quad \downarrow \\
  F\ (F\ v1) \quad F\ (F\ v2) \quad F\ (F\ vn) \\
\end{array}
\]

\[
\begin{array}{c}
  \vdash \\
  \downarrow \\
  \vdash \\
\end{array}
\]

\[
\begin{array}{c}
  \text{(define (mapF xs)} \\
  \quad (\text{if (null? xs)} \\
  \quad \quad \text{null} \\
  \quad \quad (\text{cons (F (first xs)} \\
  \quad \quad \quad (\text{mapF (rest xs)}))))}
  \end{array}
\]

Express Mapping via Higher-order my-map

\[
\begin{array}{c}
  \text{(define (my-map f xs)} \\
  \quad (\text{if (null? xs)} \\
  \quad \quad \text{null} \\
  \quad \quad (\text{cons f (first xs)}) \\
  \quad \quad (\text{my-map f (rest xs)}))))
  \end{array}
\]
my-map Examples

> (my-map (λ (x) (* 2 x)) (list 7 2 4))

> (my-map first (list (list 2 3) (list 4) (list 5 6 7)))

> (my-map (make-linear-function 4 7) (list 0 1 2 3))

> (my-map app-3-5 (list sub2 + avg pow (flip pow) make-linear-function))

Your turn

(map-scale n nums) returns a list that results from scaling each number in nums by n.

> (map-scale 3 (list 7 2 4))
'(21 6 12)

> (map-scale 6 (range 0 5))
'(0 6 12 18 24)

Currying

A curried binary function takes one argument at a time.

(define (curry2 binop)
  (λ (x) (λ (y) (binop x y))))

(define curried-mul (curry2 *))

> ((curried-mul 5) 4)
30

> (my-map (curried-mul 3) (list 1 2 3))

> (my-map ((curry2 pow) 4) (list 1 2 3))

> (my-map ((curry2 (flip2 pow)) 4) (list 1 2 3))

> (define lol (list (list 2 3) (list 4) (list 5 6 7)))

> (map ((curry2 cons) 8) lol)

> (map (??? 8) lol)
'(2 3 8 (4 8) (5 6 7 8))

Mapping with binary functions

(define (my-map2 binop xs ys)
  (if (not (= (length xs) (length ys)))
      (error "my-map2 requires same-length lists")
      (if (or (null? xs) (null? ys))
          null
          (cons (binop (first xs) (first ys))
                (my-map2 binop (rest xs) (rest ys))))))

> (my-map2 pow (list 2 3 5) (list 6 4 2))
'(64 81 25)

> (my-map2 cons (list 2 3 5) (list 6 4 2))
'((2 . 6) (3 . 4) (5 . 2))

> (my-map2 cons (list 2 3 4 5) (list 6 4 2))
ERROR: my-map2 requires same-length lists
Built-in Racket \texttt{map} Function
Maps over Any Number of Lists

\begin{verbatim}
> (map (λ (x) (* x 2)) (range 1 5))
'(2 4 6 8)
> (map pow (list 2 3 5) (list 6 4 2))
'(64 81 25)
> (map (λ (a b x) (+ (* a x) b))
  (list 2 3 5) (list 6 4 2) (list 0 1 2))
'(6 7 12)
> (map pow (list 2 3 4 5) (list 6 4 2))
ERROR: map: all lists must have same size;
arguments were: #<procedure:pow> '(2 3 4 5) '(6 4 2)
\end{verbatim}

Recall the List Filtering Pattern

\begin{verbatim}
(filterP (list v1 v2 ... vn))
\end{verbatim}

\begin{verbatim}
(define (filterP xs)
  (if (null? xs)
      null
      (if (P (first xs))
          (cons (first xs) (filterP (rest xs)))
          (filterP (rest xs)))))
\end{verbatim}

Express Filtering via Higher-order \texttt{my-filter}

\begin{verbatim}
(define (my-filter pred xs)
  (if (null? xs)
      null
      (if (pred (first xs))
          (cons (first xs)
            (my-filter pred (rest xs))))
      (my-filter pred (rest xs))))
\end{verbatim}

Built-in Racket \texttt{filter} function acts just like \texttt{my-filter}

\begin{verbatim}
> (filter (λ (x) (> x 0)) (list 7 -2 -4 8 5))
> (filter (λ (n) (= 0 (remainder n 2)))
  (list 7 -2 -4 8 5))
> (filter (λ (xs) (>= (len xs) 2))
  (list (list 2 3) (list 4) (list 5 6 7))
> (filter number?
    (list 17 #t 3.141 "a" (list 1 2) 3/4 5+6i))
> (filter (lambda (binop) (>= (app-3-5 binop)
                        (app-3-5 (flip2 binop))))
        (list sub2 + avg pow (flip2 pow)))
\end{verbatim}
Recall the Recursive List Accumulation Pattern

\[
\text{recf} (\text{list } v_1 v_2 \ldots v_n)
\]

\[
\begin{array}{c}
\text{combine} \\
\text{combine} \\
\text{nullval}
\end{array}
\]

\[
(\text{define (rec-accum } xs)
(\text{if (null? } xs)
nullval
(\text{combine} (\text{first } xs)
(\text{rec-accum} (\text{rest } xs)))))
\]

Express Recursive List Accumulation via Higher-order \text{my-foldr}

\[
(\text{define (my-foldr } \text{combine } \text{nullval} \text{ xs})
(\text{if (null? } \text{xs})
nullval
(\text{combine} (\text{first } \text{xs})
(\text{my-foldr} \text{combine}
nullval
(\text{rest } \text{xs})))))
\]

\textbf{my-foldr Examples}

\[
\begin{align*}
&> (\text{my-foldr} + 0 \text{ (list 7 2 4)}) \\
&> (\text{my-foldr} * 1 \text{ (list 7 2 4)}) \\
&> (\text{my-foldr} - 0 \text{ (list 7 2 4)}) \\
&> (\text{my-foldr} \text{ min } +\text{inf}.0 \text{ (list 7 2 4)}) \\
&> (\text{my-foldr} \text{ max } -\text{inf}.0 \text{ (list 7 2 4)}) \\
&> (\text{my-foldr} \text{ cons } \text{list } 8 \text{ (list 7 2 4)}) \\
&> (\text{my-foldr} \text{ append null}
\text{ (list (list 2 3) (list 4) (list 5 6 7))})
\end{align*}
\]

\textbf{More my-foldr Examples}

\[
\begin{align*}
&> (\text{my-foldr} (\lambda (a \ b) \ (\text{and } a \ b)) \ #t \ (\text{list } \#t \ \#t \ \#t)) \\
&> (\text{my-foldr} (\lambda (a \ b) \ (\text{and } a \ b)) \ #t \ (\text{list } \#t \ \#f \ \#t)) \\
&> (\text{my-foldr} (\lambda (a \ b) \ (\text{or } a \ b)) \ #f \ (\text{list } \#t \ \#f \ \#t)) \\
&> (\text{my-foldr} (\lambda (a \ b) \ (\text{or } a \ b)) \ #f \ (\text{list } \#f \ \#f \ \#f)) \\

&;; \text{This doesn’t work. Why not?} \\
&> (\text{my-foldr} \text{ and } \#t \ (\text{list } \#t \ \#t \ \#t))
\end{align*}
\]
Mapping & Filtering in terms of \texttt{my-foldr}

\begin{verbatim}
(define (my-map f xs)
  (my-foldr ???
    ???
    xs))

(define (my-filter pred xs)
  (my-foldr ???
    ???
    xs))
\end{verbatim}

Built-in Racket \texttt{foldr} Function
Folds over Any Number of Lists

\begin{verbatim}
> (foldr + 0 (list 7 2 4))
13
> (foldr (lambda (a b sum) (+ (* a b) sum))
  0
  (list 2 3 4)
  (list 5 6 7))
56
> (foldr (lambda (a b sum) (+ (* a b) sum))
  0
  (list 1 2 3 4)
  (list 5 6 7))
ERROR: foldr: given list does not have the same size as the first list: '(5 6 7)
\end{verbatim}