CS 135 Winter 2017
Tutorial 10: More about abstract list functions
Reminders

• Assignment 10 is due Monday, April 3, 2017 at 9:00 pm.

• There will be an exam help session, the time will be posted on Piazza.
Review: foldl

Recall the implementation of the abstract list function \texttt{my-foldl} as shown in slide 46 Module 11.

\texttt{;; (my-foldl \texttt{combine} \texttt{acc} \texttt{lst0}) applies the \texttt{combine} function to each element in \texttt{lst0} from left to right and \texttt{acc}, storing the result of each step in \texttt{acc}.}

\texttt{;; my-foldl: (X Y \rightarrow Y) Y (listof X) \rightarrow Y}

\texttt{(define \texttt{(my-foldl \texttt{combine} \texttt{acc} \texttt{lst0})}}

\texttt{  (local \[(define \texttt{(foldl-acc \texttt{lst} \texttt{acc})}}}

\texttt{    (cond \[(\texttt{empty? \texttt{lst}) \texttt{acc}]

\texttt{      \[\texttt{else \texttt{(foldl-acc (rest \texttt{lst})}}]

\texttt{       \texttt{(combine \texttt{(first \texttt{lst}) \texttt{acc}))})}}

\texttt{     \texttt{)])})]

\texttt{  \texttt{(foldl-acc \texttt{lst0} \texttt{acc})})}
Review: foldl

;; my-foldr: (X Y → Y) Y (listof X) → Y
(define (my-foldr combine base lst)
  (cond
   [(empty? lst) base]
   [else (combine (first lst)
                 (my-foldr combine base (rest lst))))])

;; my-foldl: (X Y → Y) Y (listof X) → Y
(define (my-foldl combine acc lst0)
  (local [(define (foldl-acc lst acc)
             (cond
              [(empty? lst) acc]
              [else (foldl-acc (rest lst)
                               (combine (first lst) acc))])]
          (foldl-acc lst0 acc)))
Review: foldl

Unlike foldr, the combine function applied at every step does not take an element and the result of recursive call of the rest of the list. The function given to foldl takes an element and the partial result computed by processing all prior elements.

What is the difference between these calls?

(foldr cons empty lst)
(foldl cons empty lst)
Clicker Question - foldl

What is the value produced by the following expression?

(foldl / 2 (list 6 3 4 2))

A  error: / expected number, received list
B  1/8
C  1/2
D  8
E  (list 1/3 2/3 1/2 1)
Group Problem - take

Write a function `take` that consumes a list and a natural number `n`, and produces a list containing only the first `n` elements of the list, or the entire list if it contains fewer than `n` elements using `foldl`.

;; Examples:

(check-expect (take empty 2) empty)
(check-expect (take (list 'b 'a 's 'e) 3) (list 'b 'a 's))
(check-expect (take (list 1 1 2) 4) (list 1 1 2))
Review: build-list

Recall the implementation of the abstract list function `my-build-list` as shown in Slide 88 of Module 10,

```
(define (my-build-list n f)
  (local
    [(define (list-from i)
        (cond [(>= i n) empty]
              [else (cons (f i) (list-from (add1 i)))]))]
  (list-from 0)))
```
Group Problem - diagonal

Write a function `diagonal` using `build-list` that consumes a `Nat` and produces a square matrix, a `(listof (listof Nat))`, that contains 1s on the main diagonal and 0s everywhere else. The mult-table example in class (module 10 slide 89) may be useful.

(check-expect (diagonal 3)
  (list (list 1 0 0)
        (list 0 1 0)
        (list 0 0 1)))
Clicker Question - Choosing ALFs

Consider the purpose and contract of some function, \texttt{n-in-a-row}. Which abstract list function would be most useful when implementing \texttt{n-in-a-row}?

\((\text{n-in-a-row } n \text{ desired } \text{lochar})\) determines if at least \(n\) consecutive occurrences of the consumed character, \texttt{desired}, appear in the list \texttt{lochar}

\(\text{n-in-a-row}: \text{Nat Char (listof Char)} \rightarrow \text{Bool}\)

\text{Requires: } n > 0

A \ foldr
B \ map
C \ filter
D \ build-list
Group Problem - find-card

(define-struct card (suit value))

;; A Card is a (make-card Sym Nat)

;; requires: the suit of a Card must be one of
;;   'Spades, 'Clubs, 'Hearts, 'Diamonds and
;; the value of a Card is between 1 and 13 inclusive

Given the data definition above. Write a function find-card that consumes a (listof Card) and a Card, and produces a number indicating the position of the first card that is equivalent to the given card in the list. If no card is found then find-card produces false. You may not use explicit recursion. You may use the built-in abstract list functions.