Announcements

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

• Assignment 9 is due Tuesday, November 26, 9:00 PM.

• The times and locations of office hours are posted on the “Office and Consulting Hours” pages of the course website. Please email us at “cs135@uwaterloo.ca” to set up an appointment outside of these hours.

• This tutorial is posted on the course website.

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, the next step here would be:

\[((\text{lambda} \ (x \ y) \ (* \ (+ \ y \ 4) \ x)) \ 5 \ 6)\]

\[\Rightarrow (* \ (+ \ 6 \ 4) \ 5)\]
Stepping Problem 1: 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:

\[
\Rightarrow ((\lambda (x \ y \ z) (\ast \ x \ 5)) \ 9 \ (\#\ 1 \ 2) \ 8)
\]

Stepping Problem 2: 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:

\[
\Rightarrow (((\lambda (x \ y) (\lambda (x) (\ast \ x \ y))) \ 5 \ 6) \ 10)
\]

Review: Abstract List Functions

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

These are the contracts:

\[
\text{filter: } (X \rightarrow \text{Bool}) \ (\text{listof } X) \rightarrow (\text{listof } X)
\]
\[
\text{map: } (X \rightarrow Y) \ (\text{listof } X) \rightarrow (\text{listof } Y)
\]
\[
\text{foldr: } (X \ Y \rightarrow Y) \ Y \ (\text{listof } X) \rightarrow Y
\]
\[
\text{foldl: } (X \ Y \rightarrow Y) \ Y \ (\text{listof } X) \rightarrow Y
\]
\[
\text{build-list: } \text{Nat} \ (\text{Nat} \rightarrow X) \rightarrow (\text{listof } X)
\]
Problems 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:
\( (\text{sum-list } \langle 1 \ 2 \ 3 \ 4 \ 5 \ 6 \rangle) \Rightarrow 21 \)

Double each number in a list:
\( (\text{double-list } \langle 1 \ 2 \ 3 \ 4 \ 5 \ 6 \rangle) \Rightarrow \langle 2 \ 4 \ 6 \ 8 \ 10 \ 12 \rangle \)

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

Create a list of odd numbers from 1 to n:
\( (\text{odd-list } 6) \Rightarrow \langle 1 \ 3 \ 5 \rangle \)

Clicker Question 1: Choosing ALFs
Consider the purpose and contract of the function \texttt{3-in-a-row}.
Which ALF would be the most useful when implementing \texttt{3-in-a-row}?

\texttt{;; (3-in-a-row desired lochar) Determines if at least three}
\texttt{;; consecutive occurrences of the desired character appear in lochar.}
\texttt{;; 3-in-a-row: Char (listof Char) \rightarrow Bool}

A foldr
B map
C filter
D build-list
Review: Stepping Through ALFs
We evaluate abstract list functions in one step as long as the
arguments are in the simplest form.
Consider the following example:

```
(foldr (lambda (item result)
    (cond [(odd? item) (cons item result)]
          [else result)]) empty '(1 1 2 3 5 8))
⇒ '(1 1 3 5)
```

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 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)

Exam Practice Problem 1: Increasing Lists
Write a function called increasing-lists that consumes a positive integer n and produces a list of n lists of natural numbers, where the i\textsuperscript{th} list contains the first i + 1 natural numbers.

(increasing-lists 1) ⇒ '((0))
(increasing-lists 4) ⇒

Exam Practice Problem 2: 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))