CS 331 — Mutable Lists Activity

Mattox Beckman

Introduction

One trick with a mutable list is to keep a “last” pointer. This activity will show when it helps and when it doesn’t.

```
(defprotocol MListP
   (first [this])
   (last [this])
   (set-first! [this v])
   (set-last! [this v]))
(deftype List
   [^{:unsynchronized-mutable true} the-first
    ^{:unsynchronized-mutable true} the-last]
   MListP
   (first [this] the-first)
   (last [this] the-last)
   (set-first! [this v] (set! the-first v))
   (set-last! [this v] (set! the-last v))))

(defprotocol MConsP
   (car [this])
   (cdr [this])
   (set-car! [this v])
   (set-cdr! [this v]))
(deftype MCons
     [^{:unsynchronized-mutable true} the-car
      ^{:unsynchronized-mutable true} the-cdr]
   MConsP
   (car [this] the-car)
   (cdr [this] the-cdr)
   (set-car! [this v] (set! the-car v))
   (set-cdr! [this v] (set! the-cdr v)))

(defn mcons [elt xx]
   (Cons. elt xx))
(defn get-last [xx]
   (if (nil? (cdr xx)) xx
       (get-last (cdr xx))))
(defn mlist-aux [& xx]
   (if (empty? xx) nil
       (mcons (first xx) (apply mlist-aux (rest xx)))))
(defn mlist [& xx]
   (let [it (apply mlist-aux xx)]
       (List. it (get-last it)))) ;; For fun: Can you make this more efficient?
(defn insert-front [elt l]
   (set-first! l (mcons elt (first l))))
```

1. Draw a memory diagram showing what happens if we run the following code.

```
(def x (mlist 1 2 3))
(insert-front 10 x)
```

2. Write the code for `insert-end`, that inserts something at the end of the list. It should run in $\mathcal{O}(1)$ time.

3. Suppose we write a `delete-last` function. Does having a `last` pointer help in this case? Why or why not?