MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring 2006

Recitation 8 Solutions — 3/3/2006
Data Abstractions

Announcements

Quiz 1 is next tuesday, March 7th, from 7:30-9:30pm.

Recitation notes, including solutions and a pointer to prior year's quiz problems, are appearing shortly at:

`http://people.csail.mit.edu/jastr/6001/spring06/`

From last time

```
(define (make-units C L H)
  (list C L H))
(define get-units-C car)
(define get-units-L cadr)
(define get-units-H caddr)

(define (make-class number units)
  (list number units))
(define get-class-number car)
(define get-class-units cadr)
(define (get-class-total-units class)
  (let ((units (get-class-units class)))
    (+ (get-units-C units)
       (get-units-L units)
       (get-units-H units))))
(define (same-class? c1 c2)
  (= (get-class-number c1) (get-class-number c2)))
```

1. Write a constructor that returns an empty schedule.

   ```
   (define (empty-schedule) '())
   ```

   Order of growth in time, space? $\Theta(1)$ for both.

2. Write a procedure that when given a class and a schedule, returns a new schedule including the new class:

   ```
   (define (add-class class schedule)
           (cons class schedule))
   ```

Order of growth in time, space? $\Theta(1)$ for both.

3. Write a procedure that computes the total number of units in a schedule.

```
(define (total-scheduled-units sched)
  (if (null? sched)
      0
      (+ (get-class-total-units (car sched))
         (total-scheduled-units (cdr sched)))))
```

Order of growth in time, space? $\Theta(n)$ for both.

4. Write a procedure that drops a particular class from a schedule.

```
(define (drop-class sched classnum)
  (cond ((null? sched) nil)
        ((= (get-class-number (car sched)) classnum)
         (drop-class (cdr sched) classnum))
        (else
         (cons (car sched) (drop-class sched classnum)))))
```

Order of growth in time, space? $\Theta(n)$ for both.

5. Implement the freshman credit limit by taking in a schedule, and removing classes until the total number of units is less than max-credits.

```
(define (credit-limit sched max-credits)
  (if (> (total-scheduled-units sched) max-credits)
      (credit-limit (cdr sched) max-credits)
      sched))
```

Order of growth in time, space? $\Theta(n^2)$ time, $\Theta(n)$ space.

HOPs

```
(define (make-student number sched-checker)
  (list number (list) sched-checker))
(define get-student-number car)
(define get-student-schedule cadr)
(define get-student-checker caddr)

(define (update-student-schedule student schedule)
  (if ((get-student-checker student) schedule)
      (list (get-student-number student)
            schedule
            (get-student-checker student))
      "invalid schedule"))
```

6. Finish the call to `make-student` to limit the student to taking at least 1 class.

   ```
   (make-student 575904467 (lambda (sched) (not (null? sched))))
   ```

7. Finish the call to `make-student` to create a first-term freshman (limited to 54 units).

   ```
   (make-student 575904467
                 (lambda (sched)
                   (<= (total-scheduled-units sched) 54)))
   ```

8. Write a procedure that takes a schedule and returns a list of the names of the classes in the schedule. Use `map`.

   ```
   (define (class-names schedule)
     (map get-class-number sched))
   ```

9. Rewrite `drop-class` to use `filter`.

   ```
   (define (drop-class sched classnum)
     (filter (lambda (class) (not (= (get-class-number class) classnum)))
             sched))
   ```

10. Rewrite `total-scheduled-units` to use `map` and `fold-right`.

    ```
    (define (total-scheduled-units sched)
      (fold-right 0 + (map get-class-total-units sched)))
    ```

11. Rewrite `credit-limit` to use `fold-right`.

    ```
    (define (credit-limit sched limit)
      (fold-right
       (lambda (class sched)
         (if (< (total-scheduled-units (add-class class sched)) limit)
             (add-class class sched)
             sched))
       (empty-schedule)
       sched))
    ```

The prior version takes $\Theta(n^2)$ time.

This one only takes $\Theta(n)$ time.

```
(define (credit-limit sched limit)
  (car
   (fold-right
    (lambda (class sched)
     (if (< (+ (get-class-total-units class) (cadr sched)) limit)
         (list (add-class class (car sched))
               (+ (get-class-total-units class) (cadr sched)))
         sched))
    (list (empty-schedule) 0)
    sched)))
```

Micro Quiz

Name:

1. Write a definition of `map`, which takes a procedure and a list, and returns a new list containing the result of applying the procedure to each element of the list.
   Map is of type: $(A \longrightarrow B)$,list$< A >\longrightarrow$list$< B >$.
   Ex: `(map (lambda (x) (+ x 2)) (list 3 5 7))`$\longrightarrow$`(5 7 9)`

```
(define (map proc lst)
  (if (null? lst) '()
      (cons (proc (car lst))
            (cdr lst))))
```

2. Write a definition of `filter`, which takes a predicate and a list and returns a list of all elements for which the predicate returned true.
   Filter is of type: $(A \longrightarrow$boolean$)$,list$< A >\longrightarrow$list$< A >$.
   Ex: `(filter even?  (list 3 5 7))`$\longrightarrow$`()`
   `(filter even?  (list 2 4 5 6))`$\longrightarrow$`(2 4 6)`

```
(define (filter pred lst)
  (cond ((null? lst) '())
        ((pred (car lst))
         (cons (car lst)
               (filter pred (cdr lst))))
        (else (filter pred (cdr lst)))))
```