

Class 43: Higher-Order Procedures, Revisited

Held: Tuesday, 17 November 2009

Summary: We visit the topic of *higher-order procedures*, one of the most important techniques in languages like Scheme. Higher-order procedures are procedures, like `map`, that take other procedures as parameters, return other procedures as values, or both.

Related Pages:

- EBoard.
- Lab: Design Patterns and Higher-Order Procedures.
- Reading: Design Patterns and Higher-Order Procedures.

Notes:

- Reading for Wednesday: Searching.
- I'll reserve some time at the start of class for questions and comments on the project proposals.
- EC for attending the swim meet on Friday.
- Just Q&A prior to lab.

Overview:

- Elegance.
- Procedures as parameters.
- Procedures as return values.
- Writing `map`.
- Writing `all?`.

Background: Guiding Principles

- *Write less, not more*
- *Refactor*
- *Name appropriately*
 - Good names for things that need names
 - No names for things that don't need names

Background: A Related Philosophy

The following is variant of something John Stone says ...

- The first time you read a new procedure structure (such as recursion over a list), you learn something.
- The second time you read the same structure, you learn something else.
- The third time, you learn a bit more.
- After that, reading doesn't give much benefit.
- The first time you write the same structure, you learn something more about that structure
- The second time, you learn even more.
- The third time, you learn a bit more.
- After that, there's no benefit.
- So ... extract the common code so you don't have to write it again. d yes, you learn something

Two Motivating Examples

- `all-real?` and `all-integer?`
- `add-5-to-each` and `multiply-each-by-5`

Procedures as Parameters

- We've been writing it a lot.
- Useful
- Concise
- Supports refactoring

Procedures as Return Values

- Another way to create procedures (anonymous and named).
- Strategy: Write procedures that return new procedures.
- These procedures can take plain values as parameters:

```
(define redder
  (lambda (amt)
    (lambda (color)
      (rgb ...))))
```

- How to think about this:
 - a procedure that takes *amt* as a parameter,
 - returns a new procedure that takes *color* as a parameter
- Can also take procedures as parameters
- One favorite: `compose`

```
(define compose
  (lambda (f g)
    (lambda (x)
      (f (g x)))))
```

- Examples
 - sine of square root of x: `(compose sin sqrt)`

- last element of a list: `(compose car reverse)`
- Another: `left-section`

```
(define left-section
  (lambda (func left)
    (lambda (right)
      (func left right))))
(define l-s left-section)
```
- Examples:
 - add two: `(l-s + 2)`
 - double: `(l-s * 2)`
- Not mentioned in the reading, but there's a corresponding `right-section`

```
(define right-section
  (lambda (func right)
    (lambda (left)
      (func left right))))
(define r-s right-section)
```
- If we were confident with this procedure, we could use it in the exam


```
(define smokes? (r-s vector-ref 3))
```

Encapsulating Control

- Possible for complex common code, too (particularly control).
- `map` is the standard example.

```
(define map
  (lambda (fun lst)
    (if (null? lst)
        null
        (cons (fun (car lst))
                (map fun (cdr lst))))))
```

- Another issue: Checking the type of elements in a list

```
(define all-numbers?
  (lambda (lst)
    (or (null? lst)
        (and (pair? lst)
              (number? (car lst))
              (all-numbers? (cdr lst))))))
(define all-symbols?
  (lambda (lst)
    (or (null? lst)
        (and (pair? lst)
              (symbol? (car lst))
              (all-symbols? (cdr lst))))))
```

- Common code

```
(define all
  (lambda (test? lst)
    (or (null? lst)
        (and (pair? lst)
              (test? (car lst))
              (all test? (cdr lst))))))
```

Concluding Comments

- Yes, skilled Scheme programmers write this way.
 - It's quick.
 - It's clear (at least to skilled Schemers).
 - It reduces mistakes.
- Such control The ability to encapsulate control in this way is fairly unique to Scheme,
- It's one of the reasons we love it at Grinnell.
 - Or at least a reason I love it.

Copyright © 2007-9 Janet Davis, Matthew Kluber, Samuel A. Rebelsky, and Jerod Weinman. (Selected materials copyright by John David Stone and Henry Walker and used by permission.) This material is based upon work partially supported by the National Science Foundation under Grant No. CCLI-0633090. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.