

## Class 41: Deep Recursion

**Held:** Tuesday, 15 April 2008

**Summary:** We begin our exploration of non-list structures that one can build with pairs (aka “cons cells”). In particular, we consider a recursive data structure typically called a *tree*. We then explore how one writes recursive procedures that work with trees.

### Related Pages:

- EBoard.
- Lab: Deep Recursion.
- Reading: Deep Recursion.

### Notes:

- I will continue to reserve time at the start of class for discussion of campus issues.
- We will have visitors in class today and tomorrow.
- Are there any final questions on Assignment 9?
- EC for Gene Gaub’s concert Thursday at 11 a.m. in Herrick.
- EC for Thursday afternoon’s CS Thursday extra.
- Reading for tomorrow: Association Lists.

### Overview:

- Lists, revisited.
- Trees, introduced.
- Deep recursion, considered.
- Playing with color trees.

## Review: What is a List?

- As you may have noticed from our work to date, lists are an essential part of Scheme programming.
- However, we have thought about lists in a number of different ways.
- When we see a list, we think of it as a group of data in order.
- When we build or use a list, we think of it in terms of the key operations, `cons`, `car`, `cdr`, `null`, and `null?`.
- We can also think of a list through its recursive definition. A list is either
  - The empty list or
  - Cons of a value and a list.
- Note that our procedural recursion typically uses this recursive data definition.

```
(define recursive-list-proc
  (lambda (lst)
    (if (null? lst)
        base-case
        (combine (car lst) (recursive-list-proc (cdr lst))))))
```

## Complicating Structures: Nested Lists and Trees

- There are many other interesting ways to combine values.
- As you've seen, lists can certainly contain other lists. We often refer to such lists as *nested lists*.
- How do we define nested lists? We might limit our definition above slightly
  - A nested list is a list in which one of the component values is a list.
- Note that it will be harder to come up with a nice recursive definition, since some component values will be lists, and some will not.
- There are also other interesting things we can do with cons cells. In particular, if we don't limit the last value to null, we can define what is traditionally called a *tree*. Here's one form of tree.
  - Any simple value (string, symbol, procedure, number, character, or boolean) is a tree.
  - Cons of any two trees is a tree.
- Computer scientists have found a wide variety of applications for trees and tree-like structures.

## Complicating Procedures: Deep Recursion

- Note that we may recurse differently over trees and nested lists.
- In particular, if the car of a list or pair is a list (or any pair structure), you might want to recurse over that structure, too.
- And if the car list has its own element list, we'd like to recurse on that list, too.
- Hence, in *deep recursion*, you recurse over both car and cdr of pairs.
- The typical form is

```
(define recursive-tree-proc
  (lambda (tree)
    (if (pair? tree)
        (combine (recursive-tree-proc (car tree))
                 (recursive-tree-proc (cdr tree)))
        (base-case tree))))
```

- Does this interesting theoretical approach have any practical implications?
  - That is, "Are there ways we'd want to use this?"
- There are certainly many cases in which it helps to have a helper of this form.
- Note that when writing list-recursive (and number-recursive) procedures, we often wrote a helper which included something that accumulated results (we typically called that things *so-far*).
  - It is much more difficult to use this kind of helper for deep recursive procedures.
  - We will, nonetheless, still use husk-and-kernel techniques to check preconditions.

## An Example: Searching in Color Trees

- We'll consider an example related to images.
- Let's suppose we build a tree whose leaves are colors. (RGB colors, color names, whatever.)
- One thing we might ask is whether a color appears in the tree.
- We start with the template for deep recursion.

```
(define recursive-tree-proc
  (lambda (tree)
    (if (pair? tree)
        (combine (recursive-tree-proc (car tree))
                 (recursive-tree-proc (cdr tree)))
        (base-case tree))))
```

- We need to add another parameter to the procedure, `color`.
- What should the base case be?
- Suppose we know whether the color is in the car of the tree and whether the color is in the cdr of the tree. How do we decide if it's in the tree?

## Color Trees, Revisited

- We can also use color trees to represent images by decomposing the image into portions and choosing a color that represents the portion.

## Lab

- Do the lab on deep recursion.

---

Copyright © 2007-8 Janet Davis, Matthew Kluber, and Samuel A. Rebelsky. (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.