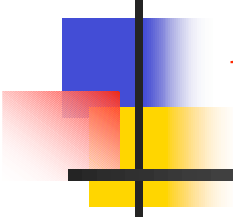


# Generative (Non-structural) Recursion



---

Corky Cartwright  
Department of Computer Science  
Rice University



# The Recipe Until Now

---

- Data analysis and design including generic templates
- For each function in the design (visible interface)
  - Contract, purpose
  - Examples (stated as tests)
  - Template Instantiation
    - Precisely followed the structure of the data we consume
    - Using this template, we can do "almost everything"
  - Testing



# Structural Recursion

---

- Is the best problem-solving strategy
  - For the vast majority of functions over recursive data.
  - Yields satisfactory efficiency in most cases.
- Cannot, in principle, compute all computable functions
- Is ill-suited to an important class of problems that technically can be solved using structural recursion but can be solved more cleanly and efficiently using non-structural methods.



# Non-structural Functional Programs

---

- Best explained by presenting some examples before discussing the general template.

Problem: efficiently sort a list of numbers

Good solutions: merge-sort, quick-sort



# Merge Sort

---

- Not going to present the actual program because it will be an exercise on the next homework assignment.
- Idea:
  - Base case: list of length 0 or 1
  - Inductive case:
    - split the list into two (almost) equal parts
    - sort each part
    - merge the two results

Why non-structural?



# Quick Sort

---

- Invented by C.A.R. ("Tony") Hoare
- Functional version is derived from the imperative (destructive) algorithm; less efficient but still works very well
- Idea:
  - Base case: list of length 0 or 1
  - Inductive case:
    - partition the list into the singleton list containing first, the list of all items  $\leq$  first, and the list of all items  $>$  first
    - sort the the lists of lesser and greater items
    - return (sorted lesser) + (first) + (sorted greater) where + means list concatenation (append)



# Quicksort Breaks Structural Template

---

```
(define (qsort l)
  (cond [(empty? l) empty]
        [else
         (local ((define pivot (first l))
                  (define other (rest l)))
          (append
            (qsort [filter (lambda (x) (<= x pivot)) other])
            (list pivot)
            (qsort [filter (lambda (x) (> x pivot)) other]))]))])
```



# Quicksort Still Terminates

---

```
(define (qsort l)
  (cond [(empty? l) empty]
        [else
         (local ((define pivot (first l))
                  (define other (rest l)))
           (append
            (qsort [filter (lambda (x) (<= x pivot)) other])
            (list pivot)
            (qsort [filter (lambda (x) (> x pivot)) other])))]))
```

Why?





# Not so quick sort

---

```
(define (qsort l)
  (cond [(empty? l) empty]
        [else
         (local ((define pivot (first l))
                 (define other l))
          (append
           (qsort [filter (lambda (x) (<= x pivot)) other])
           (list pivot)
           (qsort [filter (lambda (x) (> x pivot)) other]))]))])
```



# A More General Recipe

---

- Data analysis and design
- Contract, purpose, header
- Examples
- Template Instantiation
  - **A bit more flexible than before (non-structural)**
- **Explicit termination argument**
- Testing



# Generative Template

---

```
(define (generative-recursive-fun problem)
  (cond
    [(trivially-solvable? problem)
     (determine-solution problem)]
    [else
     (combine-solutions
      ... problem ...
      (generative-recursive-fun (generate-problem-1 problem))
      ...
      (generative-recursive-fun (generate-problem-n problem))))]))
```



# Sample termination argument

---

- Quicksort terminates because each recursive call (qsort l) reduces the metric (length l). In particular, both `[filter (lambda (x) (<= x pivot)) other]` and `[filter (lambda (x) (> x pivot)) other]` are sublists of other which is shorter than l
- Without such an argument a non-structural program must be considered incomplete.



## General framework for proving termination

---

- Devise a metric (a size function) with some familiar structural type as the output (usually `nat`) for the problem and show that each recursive call involves a smaller problem than the original one.
- In pathological cases, this ordering may require the use of lexicographic ordering on  $n$ -tuples (or unbounded sequences) of data values. These pathologies are *rare* in practice. Not a single occurrence in DrJava code base.



# Why Generative Recursion?

---

- What if we can choose between
  - a structural solution and
  - a generative solution?
- Often, the second is much faster
  - Sorting
  - Simpler example from book: greatest-common-divisor (GCD)  $\text{gcd}(6,9)=3$ ,  $\text{gcd}(99, 18) = 9$ , etc.  
structural version so brain-damaged I could not follow the narrative. I had to infer what the code did.  
Rant: local function in book often have no contracts!
  - Even better example: searching an ordered list (but not functional!)



# Are all data types structural?

---

- Surprisingly controversial question.
- Book says no.
- Walid Taha said no in Comp 210.
- I say yes! Why? Every computational representation uses inductively defined trees. Even real numbers? Floating point.
- Question: is the structural ordering always useful in proving properties of a type? What about rationals? Floating point numbers?
- What about infinite streams and trees? How do we define the domain of functions  $A \rightarrow B$ ? The naive answer is non-structural. Use computable subset of set theoretic definition of  $A \rightarrow B$ . There is a much better structural answer but WAY beyond scope of this course. Material sometimes covered in Comp 311.



# Some Algorithm Families

---

- Sorting and Searching
- Mathematical iteration: bisection, Newton's method.
- Backtracking (traversing a maze, 8 queens)
- Dynamic Programming (with Java)





# Termination Argument

---

- ; If we start with an interval  $S$  wide, then
- ; we only need a limited number of steps
- ; to reach an interval  $R$  wide. In particular,
- ; the intervals will proceed as  $S, S/2, S/4,$
- ; ..., and will reach size smaller than  $R$  in
- ;  $\log_2 (R/S)$  steps.



# The Tradeoff (if we can chose)

---

- How do we chose between
  - a structural solution and
  - a generative solution?
- Speed vs. clarity
- Chapter 26 has a very nice example
  - Greatest-common-divisor (GCD)
  - $\text{gcd}(6,9)=3$ ,  $\text{gcd}(99, 18) = 9$ , etc.



## For Next Class

---

- Homework due Monday
- Continue Reading:
  - Ch 25-28: Non-structural recursion.
- Start on next homework assignment
  - (mergesort I on) (Problem 26.1.2 but top-down rather than bottom-up version of mergesort)