A NEW FORM OF RECURSION

COMP 210 – 28 OCT 2005
SCHEME: THE STORY SO FAR

- Values
  - simple (numbers, symbols, empty)
  - compound (structures, lists)
  - functions (lambda)

- Language
  - Ways to work with values
  - define, cond, local
  - primitive functions
WHAT CAN WE DO?

Simple math of the $f(x) = x^2$ variety

$$(\text{define } (f \ x) \ (\ast \ x \ x))$$

Structural recursion

$$\text{;; A list is empty or (cons X list)}$$

$$\text{;; f : } [X] \rightarrow ?$$

$$\text{(define } (f \ \text{a-list}) \quad \text{;; f : [X] } \rightarrow ?$$
$$\text{(cond}$$
$$\quad [(\text{empty? a-list}) \cdots]$$
$$\quad [\text{else ... } (\text{first a-list}) \cdots (f \cdots (\text{rest a-list}) \cdots)])$$
SO CAN WE DO ANYTHING?

▷ Yes, we can compute anything!

▷ (As long as it’s simple math.)

▷ (Or walking down a list.)

▷ (Or a family tree.)

▷ (Or counting natural numbers down to ‘Zero.)

▷ Is there nothing else?
WHAT ABOUT GENIUS

INSPIRATION

CLEVERNESS

(ETC.)
And consider new types of computation

...which do not fit “simple math” or “structural recursion”.
PROBLEM #1: PHYSICS

» Can we figure out where an airborne object will hit the ground?

» High school physics: modeling motion

\[ x_{n+1} = x_n + \Delta x \]

(read \( \Delta x \) as "velocity")

» But if we have acceleration, we need to model how it changes velocity:

\[ \Delta x_{n+1} = \Delta x_n + \Delta(\Delta x) \]

(read \( \Delta(\Delta x) \) as "acceleration")
DEMO #1: LOOSE CANNONS

(ka-boom!)
THE SIMULATION

(define-struct obj (x y vx vy))
; sim : obj -> true
; repeatedly apply velocity to position, accel to velocity
; simulation stops when o hits the “ground” (y=0)
; assume fixed gravity in the y direction of -10 m/s
(define (sim o)
  (cond
   [(< (obj-y o) 0) true] ; stop when we hit the ground
   [else (and
      (draw o)
      (sim (make-obj
        (+ (obj-x o) (obj-vx o)) ; x_{n+1} = x_n + \Delta x
        (+ (obj-y o) (obj-vy o)) ; y_{n+1} = y_n + \Delta y
        (obj-vx o)
        (+ (obj-vy o) -10)))]))) ; \Delta y_{n+1} = \Delta y_n + \Delta(\Delta y)
WHERE WAS THE TEMPLATE?

- We didn’t know how to write one
  - There’s no data definition for “physics simulation”
  - …so it can’t be structural recursion

- But it IS recursive.
  - Evidence: *sim* called *sim* again.
  - So what do we call it?

- The book calls it
  "GENERATIVE RECURSION"
TEMPLATE FOR THE CANNON SIMULATION

Instead of a cond based on structure:

A cond based on an idea for simulating things hitting the “ground”

1. Keep moving an object
2. If the object’s y-value goes below 0, stop

We can write a template for all simulations following this idea

(define (f o)
  (cond
    [(< (obj-y o) 0) ...]
    [else ... (f ...) ... ])))
THE STUDY OF

ALGORITHMS

CLEVER IDEAS FOR SOLVING PROBLEMS

A FUNDAMENTAL TOPIC IN COMPUTER SCIENCE

SYSTEMS, NETWORKS, LANGUAGES, CRYPTO, ROBOTICS, &C.

(OUTSIDE OF COMPUTER SCIENCE, TOO!)
DIVIDE AND CONQUER

➢ A general class of algorithms

➢ If your problem is easy to solve, solve it and stop
  ➢ Otherwise, break it into strictly easier problems
    ➢ And recursively examine those problems
      ➢ If those problems are easy to solve ...
        ➢ (I think you get the idea)

➢ Conveniently expressed as recursive functions

➢ Here’s a D&C algorithm for …
PROBLEM #2: SORTING

THE PARTY HAT ALGORITHM

For a line of people to be sorted by birthday,

- Pick someone to put on a party hat and shout out her birthday.
- Everyone whose birthday comes before hers: move to her right.
- Everyone whose birthday comes after: move to her left.
- Start the game over with the people on her left.
- Also start over with the people on her right.
- At any point, if the line of people is empty, for goodness’ sake, stop!
DEMO #2: THE PARTY HAT ALGORITHM AT WORK

(hopefully Dan remembered the party hats)
THAT’S A NEAT ALGORITHM

This is actually an “old” algorithm (1960)

(oh, and, it’s not called “the party hat algorithm”)

It’s called QUICKSORT*

... because it’s quick! It’s a lot better than insertion sort, which we saw earlier.


ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.
procedure quicksort (A,M,N); value M,N;
array A; integer M,N;
comment Quick sort is a very fast and convenient method of sorting an array in the random-access store of a computer. The entire contents of the store may be sorted, since no extra space is
Our algorithm said that we’d stop on an empty list, and perform a generative recursion otherwise

```scheme
;; qsort-esque-func : [X] -> [X]
(define (qsort-esque-func L)
  (cond
    [(empty? L) ...]
    [else ... (qsort-esque-func ...) ... ]))
```
IMPLEMENTATION OF PARTY HAT QUICKSORT

;; qsort : [num] -> [num]
(define (qsort L)
  (local
    ((define (elements-before i L) (filter (lambda (x) (<= x i)) L))
     (define (elements-after i L) (filter (lambda (x) (> x i)) L)))
  (cond
   [(empty? L) empty]
   [else
    (append
     (qsort (elements-before (first L) (rest L)))
     (list (first L))
     (qsort (elements-after (first L) (rest L))))])))
SOME FINAL THOUGHTS

▶ Is "generative recursion" really something fundamentally new?

▶ Algorithms are inventions, by people
  ▶ They haven’t all been found yet!