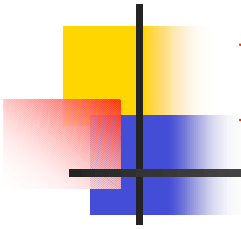
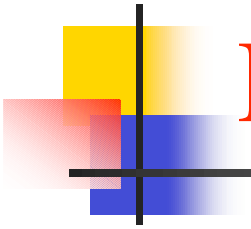
Generative Recursion Illustrated

Corky Cartwright
Department of Computer Science
Rice University



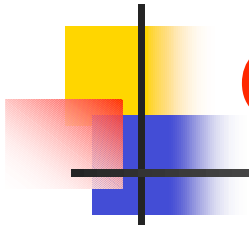
Big Picture

- Functional program design in Scheme
  - Data-directed (functional) program design 2-10
  - Algorithm design 11-15
  - Applied functional programming 16-18
- Object-oriented (OO) program design in Java 19-45
  - …



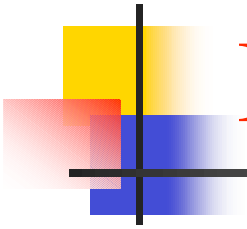
Plan for Today

- Template for Generative Recursion
- Looks at a simple example of generative recursion (algorithms) in detail: (very) simple parsing
- Book: focuses on more challenging numerical algorithms but the challenge is the underlying mathematics not the coding



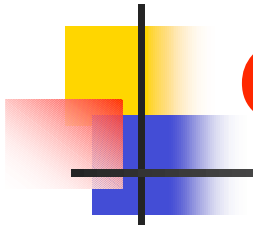
Generative Recursion

- Structural recursion
  - Template derived directly from data definition
  - Termination for all programs is the same
  - Technically includes *complete* structural recursion such as naive Fibbonacci
    f(n) = f(n-1) + f(n-2)
- Generative recursion
  - Data definition does not directly guide design of function
  - Must address termination in each such function



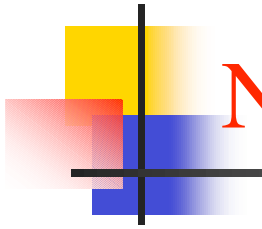
Impact on Design Recipe

- Only effects:
  - choice of template; and
  - inclusion of termination argument
- Impact on template:
  - "Divide and Conquer" decomposition of the problem requires some creativity
    - Determine solution for trivial problems
    - Determine how to break big problems into smaller ones
    - Determine how to combine solutions of smaller problems to solve the bigger problem



Generative Template

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



Numerical Algorithms; Stream Algorithms

Algorithms that process *real numbers* are not structural

Examples:

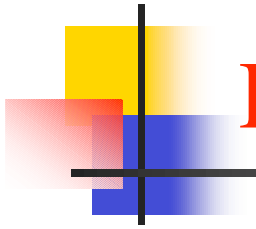
- Bi-section for finding roots
- Newton's algorithm for finding root of a function f (square root best known application)
- Formulas for constructing fractals
- Series approximations

Explanation: real numbers are not a structural type (Dedekind cuts, Cauchy sequences)

Algorithms that process *(infinite) streams* are not structural

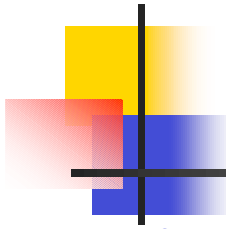
- Parsing
- Arithmetic operations on radix representations of real numbers

Explanation: (inifinte) streams are not a structural type



Example of stream-processing algorithm

Parsing



(Very) Simple Parsing

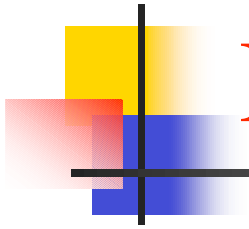
- Used by pretty every time a program reads a text file
- Basic idea: a file is a sequence of proper chars separated by newline (improper) chars. A read operation returns the sequence of chars starting at the cursor position ending with the next newline and advances the cursor. In a functional setting, a stream of chars is converted to a stream of lines

  ```
  parse ‘(a b newline c d e f newline g h i ...
  ```

  produces

  ```
  ‘((a b) (c d e f) (g h i) ... )
  ```

- Is there a divide and conquer problem decomposition for doing this?



Parsing cont.

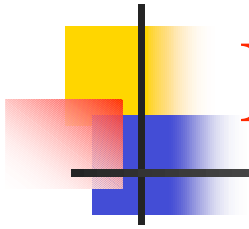
Consider writing the following function

```
; parse : (listOf symbol) -> (listOf (listOf symbol))
```

Note; `symbol` is a convenient subsitute for `char`

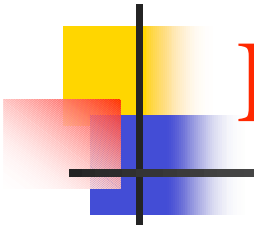
We will use helper functions:

- `first-line`
  which returns all symbols up to first `'newline`
- `rest-lines`
  which returns all symbols after first `'newline`



Parsing cont.

Collective in class exercise



For Next Class

- Homework due next Monday
- Reading:
  - Study chs. 25-28: many generative (non-structural) algorithms
- Lab
  - Practice with generative recursion