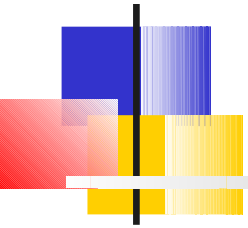
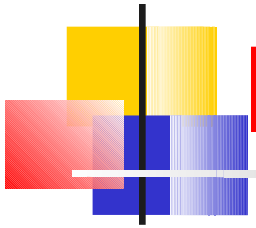


Trees

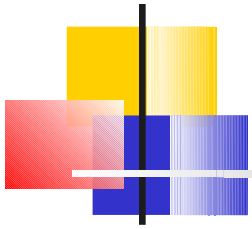
Corky Cartwright
Stephen Wong
Department of Computer Science
Rice University



Recap of Previous Lecture

Data-directed design

- Start with *data definition*
  - Specifies structure of data
- Derive *function template* from data definition
  - Model for any function that can be performed on data
  - Use generic name (e.g., `f`) for function
- Create *template instantiation* for a specific function and primary argument
  - Use specific name (e.g., `sort`) for function
  - Define separate auxiliary functions for other arguments if needed
- Develop *code* based on template instantiation



Today's Goals

- Loose Ends
  - Catching mistakes and raising errors
  - `and` & `or` operations
- Trees
  - Significantly more expressive type
  - "Lists with many tails"
- Examples:
  - Family tree
  - Binary search tree

Using **and** & **or**

- Scheme and abbreviates a conditional and takes an arbitrary number of arguments (our student dialects require at least 2)

```
(and arg1 arg2 ... argn)    abbreviates
(cond [(not arg1) false]
      [(not arg2) false]
      ...
      [else argn])
```

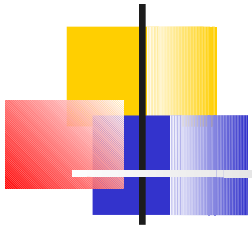
- Hence,

```
(and true true false (zero? (/ 2 0)) …) => false
```

  Called "short-circuit" or "non-strict" semantics for **and**

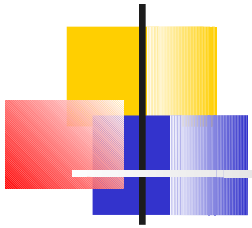
- What does **or** do? It is the *dual* of **and**.

```
(or arg1 arg2 ... argn)    abbreviates
(cond [arg1 true]
      [arg2 true]
      ...
      [else argn])
```



`and` & `or` cont.

- What are the reduction rules (laws) for **`and`**?
  - **`(and false ... argn) => false`**
  - **`(and true arg2 ... argn) => (and arg2 ... argn)`**
  - **`(and v) => v`**
- What are the reduction rules (laws) for **`or`**?
  - **`(or true ... argn) => true`**
  - **`(or false arg2 ... argn) => (or arg2 ... argn)`**
  - **`(or v) => v`**



Error Reporting

To report an error in (a student dialect of) Scheme invoke:

```
(error msg)
```

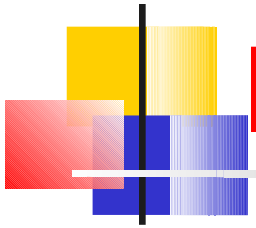
where **msg** is a string enclosed in quotation marks. In full Scheme, **error** takes additional arguments. See Help Desk.

HTDP (our online book) describes an *obsolete* version of **error**.

Semantics: the entire computation is aborted and an error message **msg** is printed.

Example:

```
(define (len aloa
  (cond
    [(empty? Aloa) 0]
    [(cons? aloa) (add1 (len (rest aloa)))]
    [else (error "len: expects argument of type <list>")]))
```

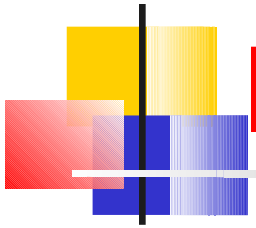


Error Reporting (cont.)

Questions:

- Is error reporting a good idea.
- Should error behavior be documented?

- Answers to questions are surprising subtle and lacking in consensus. In the case above, it is probably a good idea but it is often not done because it clutters the code and adds overhead. Moreover, the error is caught anyway with a slightly less informative diagnostic. On the other hand, DrScheme libraries do perform such checks, partly because Scheme does not perform static type checking.
- Error checking should not be included in a contract (purpose) unless the client code can depend on it and use it (by "catching" the error). We will cover error catching in Java.



Reductions for Errors

- First, note how errors work for functions you already know. In any context, erroneous primitive function applications like `(/ 1 0)` abort the computation and return an error *at the top level:*

  ```
  (/ 1 0) => /: division by zero
  ```

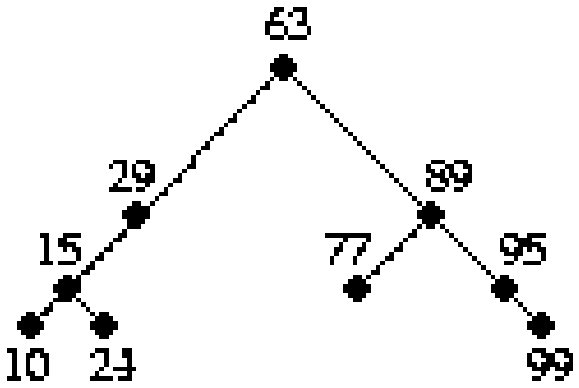
  This aborting behavior is unique among our rules.
- The error construct gives program text access to this mechanism. In any context

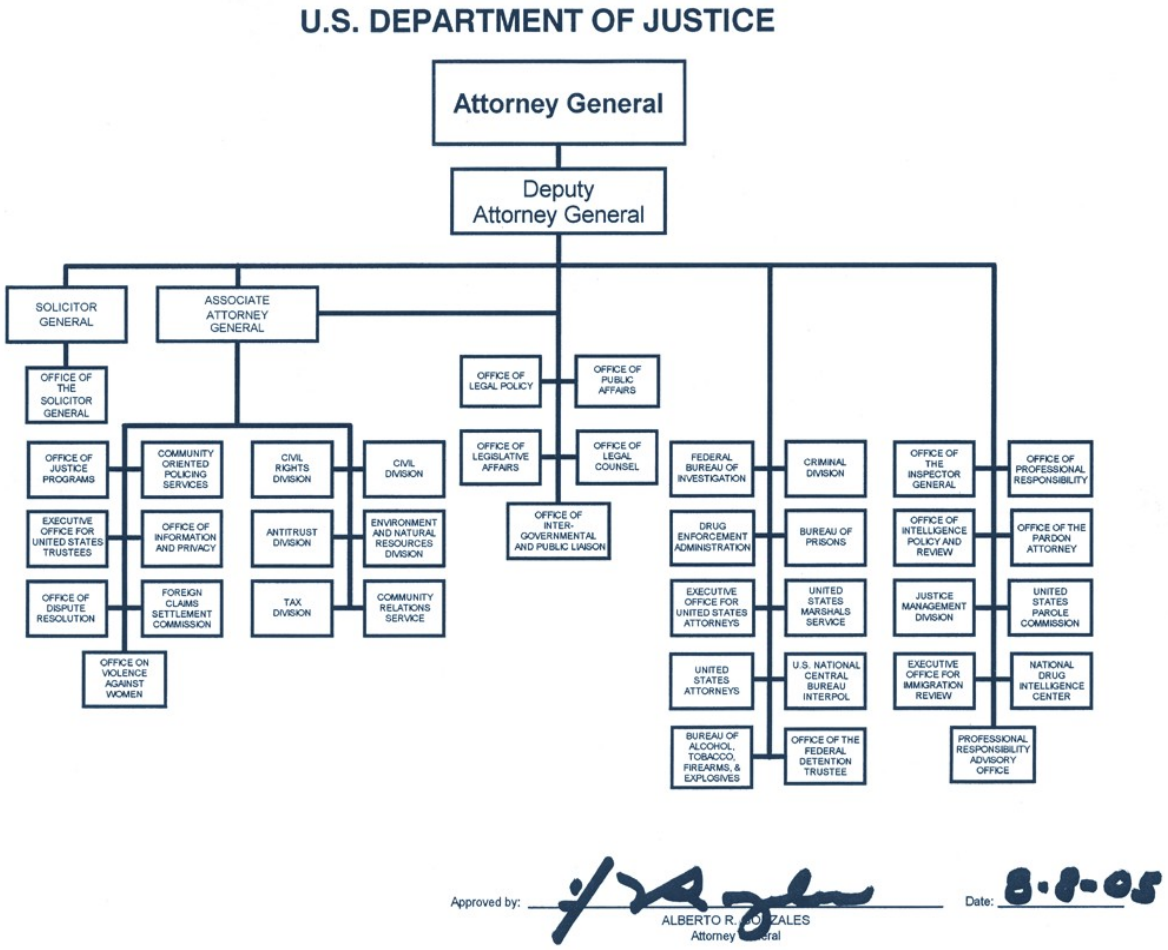
  ```
  (error "append: expects <list> as first argument") =>
  append: expects <list> as first argument
  ```
  [at top level]
- Use errors *only* as required by the problem or recipe.

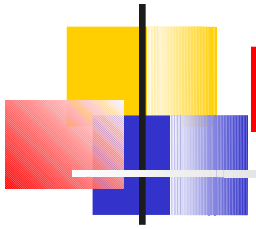
Another Inductive Type: Trees (Structures in Structures)

- Labeled trees
- Organizational charts
- Decision trees
- Search trees

and many more!





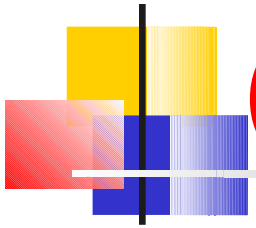


From Lists to Trees

Example of a List Data Definition

```
; A list-of-symbols is
;   empty, or
;   (cons s los)
; where s is a symbol and los is a list-of-symbols
```

A list has one embedded structure (**`rest`**)



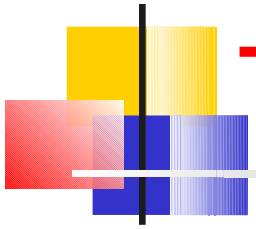
(Ancestor) Family Trees

Example of a Family Tree Data Definition

```
; A child is
;    empty                          // Represents "unknown"
;    (make-child n m f)             // Two self-references
; where n is a symbol, m is a child and f is a child

(define-struct child (name mother father))
```

A **`child`** has two embedded structures (**`mother`**, **`father`**) which belong to type **`child`**. Perhaps **`child`** is misnamed.



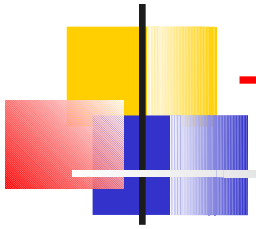
Tree Depth (in class ex.)

- Consider the following problem
  - Given an ancestry tree, compute the maximum number of generations for which we know something about this person.
- Contract (or "type") is
  - **`child -> natural`**
- Examples (next slide)

Tree Depth Examples

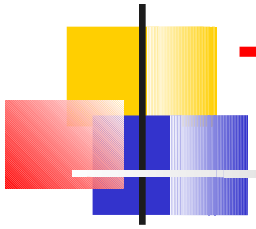
```
(define cat (make-child 'Cat empty empty))
(define tom (make-child 'Tom cat empty))
(define jane (make-child empty tom))
(define johnny (make-child 'Johnny empty empty))
(define ray (make-child 'Ray empty johnny))
(define sue (make-child 'Sue empty ray))
(define rob (make-child 'Rob empty sue))
(define bob (make-child 'Bob jane rob))

(check-expect (max-depth cat) 1)
(check-expect (max-depth tom) 2)
(check-expect (max-depth jane) 3)
(check-expect (max-depth johnny) 1)
(check-expect (max-depth ray) 2)
(check-expect (max-depth sue) 3)
(check-expect (max-depth rob) 4)
(check-expect (max-depth bob) 5)
```



Tree Depth Template Instantiation

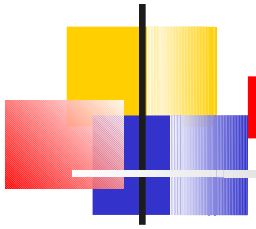
```
; Template Instantiation (trivial)
(define (max-depth c)
 (cond
   [(empty? c) ...]
   [else ...
     ... (max-depth (child-mother c)) ...
     ... (max-depth (child-father c)) ...)))
```



Tree Depth Code

```
; Code
(define (max-depth c)
 (cond
   [(empty? c) 0]
   [else (add1
    (max (max-depth (child-mother c))
         (max-depth (child-father c))))]))
```

Examples can help in writing code. Work through simple examples by hand.



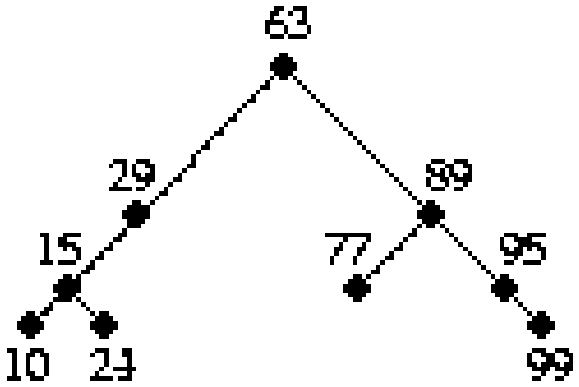
Binary Trees and Binary Search Trees

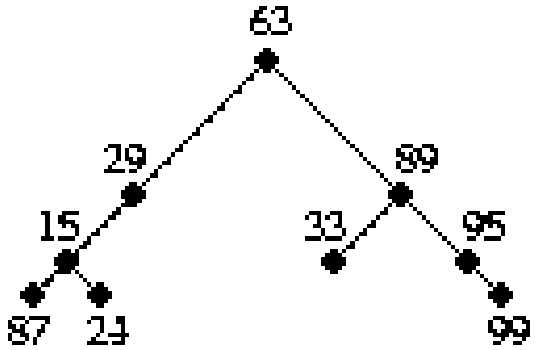
```
;  A binary tree (BT) is either
;     empty, or
;     (make-node n l r)
;  where n is a number, l and r are BTs.
(define-struct node (num left right))

;  A binary search tree (BST) is a binary tree where
;  the following invariants hold:
;  1. Numbers in l are less than or equal to n
;  2. Numbers in r are greater than n
```

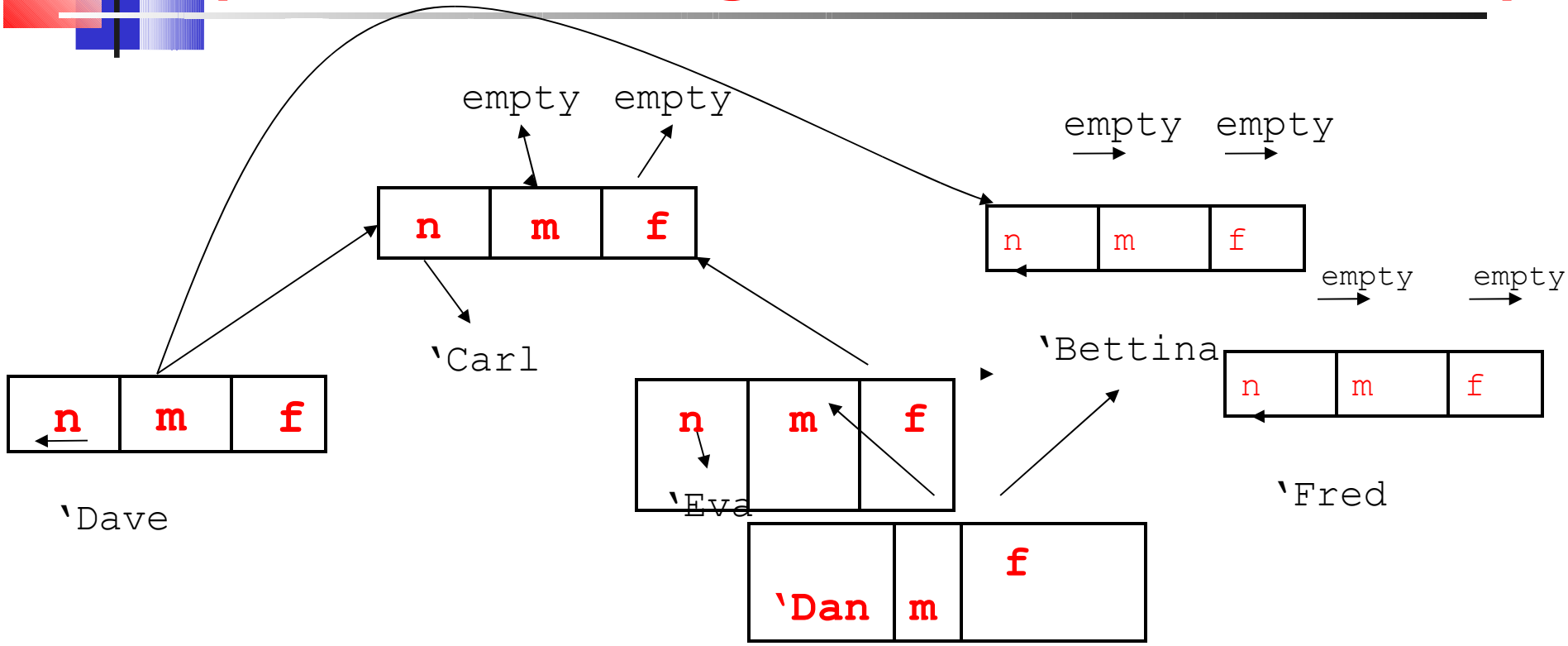
Binary Search Trees

Which binary tree satisfies the invariants of a binary search tree?





Example Family Tree (variant of Figure 35 in textbook)



Challenge Problem

- Can you think of a Scheme program that can create a cycle among structures?