

M-Structures *Continued*

Arvind
Laboratory for Computer Science
M.I.T.

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Mutable Lists

Any field in an algebraic type can be specified as an M-structure field by marking it with an “&”

```
data MList t = MNil
              | MCons {hd::t, tl::&(MList t)}
```

Allocate

```
x = MCons {hd = 5}
```

M-structure slot

Take

```
tl & x
```

Put

```
tl x := v
```

No side-effects while pattern matching

M-Cell: Dynamic Behavior

- Let allocated M-cells be represented by objects $o_1, o_2, \dots$
- Let the states of an M-cell be represented as:

$\text{empty}(o) \mid \text{full}(o,v) \mid \text{error}(o)$

- When a cell is allocated it is assigned a new object descriptor o and is empty, i.e., $\text{empty}(o)$
- Reading an M-cell
 $(x = \text{mFetch}(o) ; \text{full}(o,v)) \Rightarrow (x=v ; \text{empty}(o))$
- Storing into an M-cell
 $(\text{mStore}(o,v) ; \text{empty}(o)) \Rightarrow \text{full}(o,v)$
 $(\text{mStore}(o,v) ; \text{full}(o,v')) \Rightarrow \text{error}(o) ; \text{full}(o,v')$

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Barriers

- Barriers are needed to control to the execution of some operations
- A barrier discharges when all the bindings in its pre-region *terminate*, i.e., all expressions become *values*.

```
{ ( y = 1+7
  >>>
  z = 3 )  =>
in
  z }
```

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Insert: Functional and Non Functional

Functional solution:

```
insertf [] x = [x]
insertf (y:ys) x = if (x==y) then y:ys
                  else y:(insertf ys x)
```

M-structure solution:

```
insertm ys x =
  case ys of
    MNil      -> MCons x MNil
    MCons y ys' ->
      if x == y then ys
      else let tl ys := insertm (tl&ys) x
           in  ys
```

*In pattern matching
m-fields have the
"examine semantics"*

Can we replace **tl&ys** by **ys'**?

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>



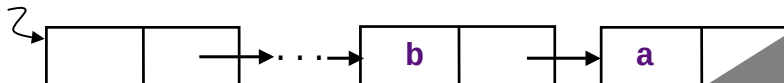
Out-of-order Insertion

Compare **ys2's** assuming **a** and **b** are not in **ys**.

```
ys1 = insertf ys a
ys2 = insertf ys1 b
```

```
ys1 = insertm ys a
ys2 = insertm ys1 b
```

ys2 Can the following list be produced?



ys1 can be returned before the insertion of **a** is complete.

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>



Avoiding out-of-order insertion

```

insertm ys x =
  case ys of
    MNil      -> MCons x MNil
    MCons y ys' ->
      if x == y then ys
      else let
          tl ys := insertm (tl&ys) x
        in  ys

```

Notice **(tl&ys)** can't be read again before **(tl ys)** is set



Membership and Insertion

insertm' is the same as **insertm** except that it also returns a flag that indicates if a match was found

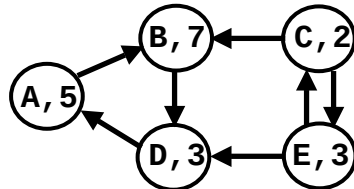
```

insertm' ys x =
  case ys of
    MNil      -> (False, (MCons x MNil))
    MCons y ys' ->
      if x == y then (True, ys)
      else let
          (flag, ys'') = (insertm' (tl&ys) x)
          tl ys := ys''
        in
          (flag, ys)

```



Graph Traversal



```

data GNode =
  GNode {id :: Nodeid,
         val :: Int,
         nbrs:: [GNode] }
a = GNode "A" 5 [b]
b = GNode "B" 7 [d]
c = GNode "C" 2 [b]
d = GNode "D" 3 [a]
e = GNode "E" 3 [c,d]
  
```

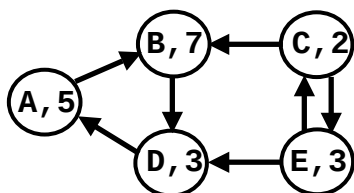
Write function **rsum** to sum the nodes reachable from a given node.

rsum a ==> ?

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>


Graph Traversal: First Attempt



```

data GNode =
  GNode {id :: Nodeid,
         val :: Int,
         nbrs:: [GNode] }
  
```

```

rsum (GNode x i nbs) =
  i + sum (map rsum nbs)
  
```

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>


Mutable Markings

Keep an updateable boolean flag to record if a node has been visited. Initially the flag is set to false in all nodes.

```
data GNode = GNode {id::Nodeid, val::Int,
                    nbrs::[GNode], flag::&Bool}
```

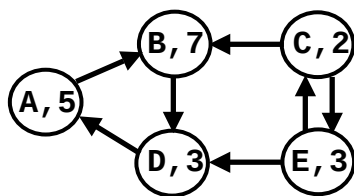
A procedure to return the current flag value of a node and to simultaneously set it to true

```
marked node = let m = flag & node >>>
              flag node := True
              in
              m
```

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Graph Traversal: Mutable Markings



```
data GNode =
  GNode {id :: Nodeid,
         val :: Int,
         nbrs:: [GNode]
         flag::&Bool }
```

```
rsum node =
  if marked node then 0
  else
    (val node)
    + sum (map rsum (nbrs node))
```

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Book-Keeping Information

```
data GNode = GNode {id::Nodeid, val::Int,
                    nbrs::[GNode], flag::&Bool}
```

The graph should not be mutated!

Keep the visited flags in a separate data structure -
a *notebook* with the following functions

```
mkNotebook :: () -> Notebook
member     :: Notebook -> Nodeid -> Bool
```

Immutable (functional) notebook

```
insert :: Notebook -> Nodeid -> Notebook
```

Mutable notebook: insertion causes a side-effect

```
insert :: Notebook -> Nodeid -> ()
```

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Graph Traversal: *Immutable Notebook*

Thread the notebook and the current sum through
the reachable nodes of the graph in any order

```
data GNode =
  GNode {id::Nodeid, val::Int, nbrs::[GNode]}

rsum node =
  let nb = mkNotebook ()      -- a new notebook
      (s,_) = thread (0, nb) node
      thread (s,nb) (GNode x i nbs) =
        if member nb x then (s,nb)
        else let nb' = insert nb x
              s' = s + i
              in
  in s
```

?

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Graph Traversal: *Mutable Notebook*

```

rsum node =
  let nb = mkNotebook ()      -- a new notebook

  rsum' (GNode x i nbs) =
    if (member nb x) then 0
    else let
      insert nb x >>>
      s = i + sum (map rsum' nbs)
    in s
  in rsum' node

- No threading
- No copying

```

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Mutable Notebooks: *revisited*

The test for membership and subsequent insertion has to be done atomically to avoid races.

```

isMemberInsertm :: Notebook -> Nodeid -> Bool
rsum node =
  let nb = mkNotebook ()      -- a new notebook
      rsum' (GNode x i nbs) =
        if (isMemberInsert nb x)
        then 0
        else i + sum (map rsum' nbs)
  in
    rsum' node

```

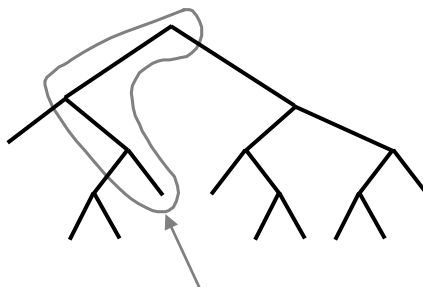
October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Notebook Representation: Tree

We can maintain the notebook as a (balanced) binary tree

```
data Tree = TEmpty | TNode Int Tree Tree
```

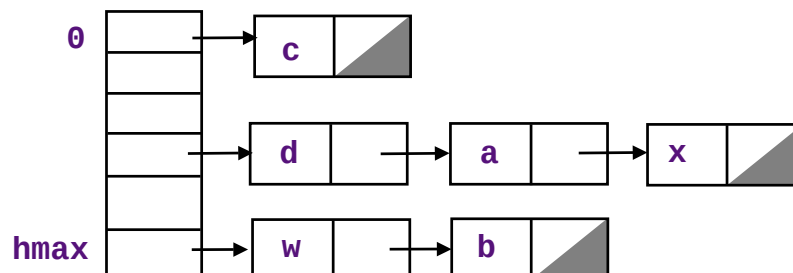


Nodes above the point of insertion have to be copied in a functional solution

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>


Notebook Representation: Hash Table



```
data MList t = MNil
              | MCons {hd::t, tl::&(MList t)}
```

```
mkNotebook () =
  mArray (0,hmax) [(j,MNil) | j <- [0..hmax]]
```

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>


isMemberInsert

```
isMemberInsert nb x =  
  let i = hash x  
  ys = nb!&i  
  (flag, ys') = insertm' ys x  
  nb!i := ys'  
  in flag
```

`insertm'` is the same as `insertm` except that it also returns a flag to indicate if a match was found

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Summary

- M-structures have been used heavily to program
 - Monsoon run-time system, including I/O
 - Id compiler in Id
 - Non-deterministic numerical algorithms
- Programming with M-structures is full of perils!
 - Encapsulate M-structures in functional data structures, if possible

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

The λ_S Calculus

- An extension of λ_{let} with side-effects and barriers

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

λ_S Syntax

$E ::= x \mid \lambda x.E \mid E E \mid \{ S \text{ in } E \}$

$\mid \text{Cond}(E, E, E)$

$\mid \text{PF}_k(E_1, \dots, E_k)$

$\mid \text{CN}_0 \mid \text{CN}_k(E_1, \dots, E_k) \mid \underline{\text{CN}}_k(x_1, \dots, x_k)$

$\mid \text{allocate}()$

$\mid o_i$ object descriptors

Not in initial expressions

$\text{PF}_1 ::= \text{negate} \mid \text{not} \mid \dots \mid \text{Prj}_1 \mid \text{Prj}_2 \mid \dots \mid \text{ifetch} \mid \text{mfetch}$

$\dots$

$\text{CN}_0 ::= \text{Number} \mid \text{Boolean} \mid ()$

$S ::= \varepsilon \mid x = E \mid S; S$

$\mid S \gg S$

$\mid \text{sstore}(E, E)$

$\mid \text{allocator} \mid \text{empty}(o_i) \mid \text{full}(o_i, E) \mid \text{error}(o_i)$

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Values and Heap Terms

Values

$$V ::= \lambda x.E \mid \text{CN}_0 \mid \underline{\text{CN}}_k(x_1, \dots, x_k) \mid o_i$$

Simple expressions

$$\text{SE} ::= x \mid V$$

Heap Terms

$$H ::= x = V \mid H; H \mid \text{allocator} \\ \mid \text{empty}(o_i) \mid \text{full}(o_i, V)$$

Terminal Expressions

$$E^T ::= V \mid \text{let } H \text{ in } \text{SE}$$

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Side-effect Rules

- Allocation rule

$$(\text{allocator}; x = \text{allocate}()) \Rightarrow (\text{allocator}; x = o; \text{empty}(o))$$
 where o is a new object descriptor
- Fetch and Take rules

$$\begin{aligned} (x = \text{iFetch}(o); \text{full}(o, v)) &\Rightarrow (x = v; \text{full}(o, v)) \\ (x = \text{mFetch}(o); \text{full}(o, v)) &\Rightarrow (x = v; \text{empty}(o)) \end{aligned}$$
- Store rules

$$\begin{aligned} (\text{mStore}(o, v); \text{empty}(o)) &\Rightarrow \text{full}(o, v) \\ (\text{mStore}(o, v); \text{full}(o, v')) &\Rightarrow \text{error}(o); \text{full}(o, v') \end{aligned}$$
- Lifting rules

$$\begin{aligned} \text{sstore}(\{ S \text{ in } e \}, e_2) &\Rightarrow (S; \text{sstore}(e, e_2)) \\ \text{sstore}(e_1, \{ S \text{ in } e \}) &\Rightarrow \text{error}(S); \text{sstore}(e_1, e) \end{aligned}$$

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Barrier Rules

- Barrier discharge

$$(\epsilon \ggg S) \Rightarrow S$$

- Barrier equivalence

$$((H ; S_1) \ggg S_2) \equiv (H ; (S_1 \ggg S_2))$$

$$(H \ggg S) \Rightarrow (H ; S) \text{ (derivable)}$$

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>

Multiple-Store Error

A program with “exposed” store error is suppose to blow up!

Program $\rightarrow$ T

The Top represents a contradiction

Exposed error: A **error(o)** cell that is not below a barrier, inside an arm of a conditional or inside a lambda abstraction

October 28, 2002

<http://www.csg.lcs.mit.edu/6.827>