∼ Lecture VII ∼

Keywords:

recursive `datatype`s: lists, trees, λ calculus; tree manipulation; tree listings: preorder, inorder, postorder; tree exploration: breadth-first and depth-first search; polymorphic exceptions; isomorphisms.

References:

- [MLWP, Chapter 4]

Recursive datatypes

Datatype definitions, including polymorphic ones, can be recursive.

The built-in type operator of *lists* might be defined as follows:

```
infixr 5 :: ;
datatype 'a list
  = nil | :: of 'a * 'a list ;
```

In the same vein, the polymorphic `datatype` of (planar) *binary trees* with nodes where data items are stored is given by:

```
datatype 'a tree
  = empty | node of 'a * 'a tree * 'a tree ;
```


Semantics

The set $\mathrm{Val}(\tau\ \mathtt{list})$ of *values* of the type $\tau\ \mathtt{list}$ is inductively given by the following rules:

$$\mathtt{nil} \in \mathrm{Val}(\tau\ \mathtt{list})$$

$$\frac{\nu \in \mathrm{Val}(\tau) \qquad \ell \in \mathrm{Val}(\tau\ \mathtt{list})}{\nu :: \ell \in \mathrm{Val}(\tau\ \mathtt{list})}$$

That is, $\mathrm{Val}(\tau\ \mathtt{list})$ is the smallest set containing `nil` and closed under performing the operation $\nu :: \_$ for values ν of type τ.

? What is the set of values of $\tau\ \mathtt{tree}$?

Further recursive datatypes

Examples:

1. Non-empty planar finitely-branching trees and forests.
   (a) Recursive version.

```
datatype
  'a FBtree = node of 'a * 'a FBtree list ;
type
  'a FBforest = 'a FBtree list ;
```

   (b) Mutual-recursive version.

```
datatype
  'a FBtree = node of 'a * 'a FBforest
and
  'a FBforest = forest of 'a FBtree list ;
```

? What are the set of values of $\tau\ \mathtt{FBtree}$ and $\tau\ \mathtt{FBforest}$?

2. λ calculus.

```
datatype
  D = f of D -> D ;
```

NB: It is non-trivial to give semantics to D. This was done by Dana Scott in the early 70's, and gave rise to *Domain Theory*.

References:

- D. Scott. Continuous lattices. In *Toposes, Algebraic Geometry and Logic*, pages 97–136, Lecture Notes in Mathematics 274, 1972.
- D. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. Unpublished notes, Oxford University, 1969. (Published in *Theoretical Computer Science*, 121(1-2):411–440, 1993.)

Tree manipulation

Examples:

1.
```
fun count empty = 0
  | count( node(_,l,r) ) = 1 + count l + count r ;
```

2.
```
fun depth empty = 0
  | depth( node(_,l,r) )
      = 1 + Int.max( depth l , depth r ) ;
```

3.
```
fun treemap f empty = empty
  | treemap f ( node(n,l,r) )
      = node( f n , treemap f l , treemap f r ) ;
```


4.
```
datatype
  dir = L | R ;

exception E ;

fun subtree [] t = t
  | subtree ( L::D ) ( node(_,l,_) )
      = subtree D l
  | subtree ( R::D ) ( node(_,_,r) )
      = subtree D r
  | subtree _ _
      = raise E ;
```


Tree listings

1. Preorder.
```
fun preorder empty = []
  | preorder( node(n,l,r) )
      = n :: (preorder l) @ (preorder r) ;
```

2. Inorder.
```
fun inorder empty = []
  | inorder( node(n,l,r) )
      = (inorder l) @ n :: (inorder r) ;
```

3. Postorder.
```
fun postorder empty = []
  | postorder( node(n,l,r) )
      = (postorder l) @ (postorder r) @ [n] ;
```

Inorder without append

```
fun inorder t
  = let
      fun accinorder acc empty = acc
        | accinorder acc ( node(n,l,r) )
            = accinorder (n :: accinorder acc r) l
    in
      accinorder [] t
    end ;
```

```
- inorder( node(3,node(2,node(1,empty,empty),
                       empty),
                 node(4,empty,
                       node(5,empty,empty))) );
val it = [1,2,3,4,5] : int list
```


Tree exploration

Breadth-first search[a]

```
datatype
  'a FBtree = node of 'a * 'a FBtree list ;

fun bfs P t
  = let fun auxbfs [] = NONE
          | auxbfs( node(n,F)::T )
              = if P n then SOME n
                else auxbfs( T @ F ) ;
    in  auxbfs [t]  end ;

val bfs = fn : ('a -> bool) -> 'a FBtree -> 'a option
```

[a]See: Chris Okasaki. Breadth-first numbering: Lessons from a small exercise in algorithm design. ICFP 2000. (Available on-line from ⟨http://www.eecs.usma.edu/Personnel/okasaki/pubs.html⟩.)

Tree exploration

Depth-first search

1. fun dfs P t

```
fun dfs P t
  = let fun auxdfs [] = NONE
          | auxdfs( node(n,F)::T )
              = if P n then SOME n
                else auxdfs( F @ T ) ;
    in
      auxdfs [t]
    end ;

val dfs = fn : ('a -> bool) -> 'a FBtree -> 'a option
```


2. DFS without append

```
fun dfs' P t
  = let
      fun auxdfs( node(n,F) )
        = if P n then SOME n
          else
            foldl
              ( fn(t,r) => case r of
                              NONE => auxdfs t | _ => r )
              NONE
              F ;
    in  auxdfs t  end ;

val dfs' = fn : ('a -> bool) -> 'a FBtree -> 'a option
```

3. DFS without append; raising an exception when successful.

```
fun dfs0 P (t: 'a FBtree)
  = let
      exception Ok of 'a;
      fun auxdfs( node(n,F) )
        = if P n then raise Ok n
          else foldl (fn(t,_) => auxdfs t) NONE F ;
    in
      auxdfs t handle Ok n => SOME n
    end ;
val dfs0 = fn :
      ('a -> bool) -> 'a FBtree -> 'a option
```


Warning: When a *polymorphic exception* is declared, ML ensures that it is used with only one type. The type of a top level exception must be monomorphic and the type variables of a local exception are frozen.

Consider the following nonsense:

```
exception Poly of 'a ;   (*** ILLEGAL!!! ***)
(raise Poly true) handle Poly x => x+1 ;
```


Further topics

Quite surprisingly, there are very sophisticated *non-recursive* programs between recursive datatypes.

References:

- M. Fiore. Isomorphisms of generic recursive polynomial types. In 31^{st} Symposium on Principles of Programming Languages (POPL 2004), pages 77-88. ACM Press, 2004.
- M. Fiore and T. Leinster. An objective representation of the Gaussian integers. Journal of Symbolic Computation 37(6):707-716, 2004.