COSC 3015: Lecture 16

Lectured and scribed by Sunil Kothari

21 October 2008

1 Review

We looked at a couple of trees and their corresponding datatypes. For instance, the binary trees are given as:

```
data Btree a = Leaf a | Fork (Btree a) (Btree a)
```

and the binary search trees are given as:

```
data (Ord a) => Stree a = Null | Fork (Stree a) a (Stree a)
```

The induction principle for the Stree is:
$(P(Null) \wedge \forall x : a, \forall t1, t2 : (Stree\ a), P(t1) \wedge P(t2) \Rightarrow P(Fork\ t1\ x\ t2))$
$\Rightarrow \forall t : Stree\ a, P(t)$.

Note that we haven't taken into account the constraint on the types. But if we do consider the constraint, the induction principle is:
$\forall\ a : Type, (Ord\ a) \Rightarrow$
$[(P(Null) \wedge \forall x : a, \forall t1, t2 : (Stree\ a), P(t1) \wedge P(t2) \Rightarrow P(Fork\ t1\ x\ t2))]$
$\Rightarrow \forall t : Stree\ a, P(t)$.

Today, we will look at two different types of trees:

- Binary heap tree
- Rose tree

1.1 Binary Heap Tree

The binary search tree is given by:

```
data (Ord a) => Htree a = Null | Fork a (Htree a)  (Htree a)
```

This looks familiar. In fact, it is very similar to the way a Stree is defined. The only difference is the position of the label in the Stree. In Htree, the label on a non-null node is given before the two subtrees.

The book says that the Htree has to satisfy the condition that the label on a given node cannot be greater than label on any subtree of the node. Actually, depending upon the order there are two kinds of binary heap trees: min and max. In the min binary heap trees, the root node has the smallest label. In this lecture, we consider only min binary heap trees.
When we try to flatten a heap tree, the order becomes important.

```
flatten::(Ord a) => Htree a -> [a]
flatten Null = []
flatten (Fork x yt zt) = x:merge (flatten yt) (flatten zt)
```

and the function *merge* is defined as

```
merge::(Ord a) => [a] -> [a] -> [a]
merge []  ys = ys
merge xs [] = xs
merge (x:xs) (y:ys) = if x <= y
                      then x: merge xs (y:ys)
                      else y: merge (x:xs) ys
```

The thing to note is that merge will return a list that has labels in some order (ascending order for min heap trees).

```
Main> merge ['a'..'c'] ['b'..'d']
"abbccd"
Main>
```

There are two ways of creating a heap tree.

1. Insert one node at a time and then push-up the inserted element so that the tree satisfies the heap property - slightly more than linear. In fact n (log n), where n is the size of the tree.

2. Make a minimum height tree and then impose heap property - Linear in the size of the tree

The book defines a function *mkHeap* which does the latter as :

```
mkHeap = heapify. mkHtree
```

Both *heapify* and *mkHtree* are linear in the size of the tree. If

Now we can come back to our question of the difference between a Stree and a Htree. Htree will satisfy the following predicate:

```
heapOrdered :: Ord a => Htree a -> Bool
heapOrdered = ordered . flatten
```

1.2 Rose Trees

With Rose trees, described by Lambert Meertens, we can represent trees of arbitrary arity.

They are also known as General trees, and we will see why it's so.

In Haskell, the trees are defined as:

```
data Rosa a = Node a [Rose a]
```

Note that compared to other trees, we have only one data constructor but still we can represent any k-ary tree as a Rose tree.

This is in contrast with the binary search tree and heap trees, which are 2-ary trees.

The induction principle for Rose trees is:
$(\forall xts : [Rose\ a].(\forall xt \in xts.P(xt) \Rightarrow P(Node\ x\ xts))) \Rightarrow \forall xt : Rose\ a.P(xt)$

Let's look at some examples:

1. $Node\ 0\ []$ - is a single tree with no sub trees.
2. $Node\ 0\ [Node 1[]]$ - is a tree with one child

In general, we can define a rose with infinite children as:
$Node\ 0\ [Node\ n\ [] \mid n \leftarrow [1..n]]$

Finally, note that any rose tree can expressed as a binary tree and any binary tree can be expressed as a rose tree. The former is harder than the latter.