99 questions/Solutions/69

From HaskellWiki

< 99 questions | Solutions

Dotstring representation of binary trees.

We consider again binary trees with nodes that are identified by single lower-case letters, as in the example of problem P67. Such a tree can be represented by the preorder sequence of its nodes in which dots (.) are inserted where an empty subtree (nil) is encountered during the tree traversal. For example, the tree shown in problem P67 is represented as 'abd..e..c.fg...'. First, try to establish a syntax (BNF or syntax diagrams) and then write a predicate tree_dotstring/2 which does the conversion in both directions. Use difference lists.

```
data Tree a = Empty | Branch a (Tree a) (Tree a) deriving (Show, Eq)

ds2tree []      = (Empty,"")
ds2tree ('.':xs) = (Empty,xs)
ds2tree (x:xs)  = (Branch x left right, rest2)
  where (left,rest) = ds2tree xs
        (right,rest2) = ds2tree rest

tree2ds Empty = "."
tree2ds (Branch x l r) = x:(tree2ds l ++ tree2ds r)
```

The tree2ds function is straightforward: Empty trees become a dot, and non-empty trees become their label, with left and right trees appended.

ds2tree is a bit trickier because you have to know which part of the string belongs to which subtree. Ds2tree returns the part of the string that hasn't been consumed yet.

The following solution uses Parsec for 'dsToTree':

```
import Text.Parsec.String
import Text.Parsec hiding (Empty)


pTree :: Parser (Tree Char)
pTree = do
    pBranch <|> pEmpty

pBranch = do
    a <- letter
    t0 <- pTree
    t1 <- pTree
    return $ Branch a t0 t1
```

```haskell
pEmpty = do
    char '.'
    return Empty

dsToTree str =
    case parse pTree "" str of
        Right t -> t
        Left e  -> error (show e)
```