99 questions/Solutions/67A

From HaskellWiki

< 99 questions | Solutions

A string representation of binary trees

Somebody represents binary trees as strings of the following type:

a(b(d,e),c(,f(g,)))

a) Write a Prolog predicate which generates this string representation, if the tree is given as usual (as nil or t(X,L,R) term). Then write a predicate which does this inverse; i.e. given the string representation, construct the tree in the usual form. Finally, combine the two predicates in a single predicate tree_string/2 which can be used in both directions.

```
treeToString :: Tree Char -> String
treeToString Empty = ""
treeToString (Branch x Empty Empty) = [x]
treeToString (Branch x l r) =
        x : '(' : treeToString l ++ "," ++ treeToString r ++ ")"

stringToTree :: (Monad m) => String -> m (Tree Char)
stringToTree "" = return Empty
stringToTree [x] = return $ Branch x Empty Empty
stringToTree str = tfs str >>= \ ("", t) -> return t
    where tfs a@(x:xs) | x == ',' || x == ')' = return (a, Empty)
          tfs (x:y:xs)
                | y == ',' || y == ')' = return (y:xs, Branch x Empty Empty)
                | y == '(' = do (',':xs', l) <- tfs xs
                                (')':xs'', r) <- tfs xs'
                                return $ (xs'', Branch x l r)
          tfs _ = fail "bad parse"
```

Note that the function `stringToTree` works in any Monad.

The following solution for 'stringToTree' uses Parsec:

```
import Text.Parsec.String
import Text.Parsec hiding (Empty)
-- these modules require parsec-3
-- to install parsec-3: cabal install parsec

pTree :: Parser (Tree Char)
pTree = do
    pBranch <|> pEmpty
pBranch = do
    a <- letter
    char '('
    t0 <- pTree
    char ','
    t1 <- pTree
    char ')'
    return $ Branch a t0 t1

pEmpty =
    return Empty

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

The above solution cannot parse such inputs as x(y,a(,b)) but demands a more rigid format x(y(,),a(,b(,))). To parse a less rigid input:

```
pBranch = do
  a <- letter
  do char '('
     t0 <- pTree
     char ','
     t1 <- pTree
     char ')'
     return $ Branch a t0 t1
   <|> return (Branch a Empty Empty)
```

This solution should be attributed to Daniel Fischer @StackOverflow[1] (http://stackoverflow.com/questions/9058914/cant-find-parsec-modules-in-ghci/9059321#9059321)

Retrieved from "https://wiki.haskell.org/index.php?title=99_questions/Solutions/67A&oldid=44285"

- This page was last modified on 30 January 2012, at 08:51.
- Recent content is available under a simple permissive license.