CapsConflicts — Detecting Directory Entries that Differ only in Capitalisation

Wolfram Kahl

2004-06-03

The default filesystem on MacOS X is case insensitive — before using tools like Unison between a Mac and a real Unix box makes sense, all these conflicts have to be resolved. The tool here just finds them.

```
module Main (main) where

import System.Posix.Files
import Directory
import IO
import Char
import System
import Data.FiniteMap
```

We start from either all command-line arguments, or the current directory:

```
main = do
  args ← getArgs
  paths ← case args of
    [] → fmap (:[]) getCurrentDirectory
    _ → return args
  mapM_ processDirectory paths
```

A useful utility in this context is concatenation of file paths:

```
fpCat :: FilePath → FilePath → FilePath
fpCat dir file = dir ++ '/' : file
```

The following data structure for a case-insensitive finite map could be encapsulated in a separate module exporting only *conflictGroups*. The conflict detection uses finite maps that map lower-case strings to non-empty lists of directory entries.

```
newtype DirMap = DM{ unDM :: (FiniteMap String [FilePath]) }
emptyDirMap = DM emptyFM

addToDirMap :: DirMap → FilePath → DirMap
addToDirMap (DM m) s = let key = map toLower s in
  DM ∘ addToFM m key ∘ (s:) ∘ maybe [] id
    $ lookupFM m key

listToDirMap :: [FilePath] → DirMap
listToDirMap = foldl addToDirMap emptyDirMap
```

Only groups with more than one element represent conflicts:

```
conflictGroups :: [FilePath] → [[FilePath]]
conflictGroups = foldFM f [] ∘ unDM ∘ listToDirMap
where
  f key [] r = error "conflictGroups: impossible case"
  f key [s] r = r
  f key ss r = ss : r
```

The groups are displayed without particular efforts:

```
groupLines :: FilePath → [FilePath] → [String]
groupLines path = ("":) ∘ map (fpCat path)

printGroups :: FilePath → [[FilePath]] → IO ()
printGroups path [] = return ()
printGroups path gs = putStrLn ∘ unlines ∘ concatMap (groupLines path) $ gs
```

We define a variant of *FilePath* concatenation that checks whether the resulting path is a sub-directory, not counting or following symbolic links:

```
catSubDir :: FilePath → FilePath → IO (Maybe FilePath)
catSubDir dir "." = return Nothing
catSubDir dir ".." = return Nothing
catSubDir dir file = let path = fpCat dir file in
catch (do
  st ← getSymbolicLinkStatus path
  if isSymbolicLink st
    then return Nothing
    else do
      st ← getFileStatus path
      return $ if isDirectory st
        then Just path
        else Nothing
  ) (λerr → do
    hPutStrLn stderr ("catSubDir error: " ++ path ++
      " - " ++ ioeGetErrorString err)
    return Nothing)
```

Processing a directory:

```
processDirectory path = do
  contents ← catch (getDirectoryContents path)
    (λerr → do
      hPutStrLn stderr ("getDirectoryContents " ++ path ++
        " - " ++ ioeGetErrorString err)
      return [])
  printGroups path ∘ reverse ∘ conflictGroups $ contents
  mapM_ (catSubDir path =>>? = processDirectory) contents
```

Here we used a variant of monadic composition, specialised for branches that are applied only to *Just* results:

```
(=>>? =) :: Monad m ⇒ (a → m (Maybe b)) → (b → m ()) → (a → m ())
f =>>? = g = λx → f x >>= maybe (return ()) g
```