Filesystem - System plików

P. S.[*], **M. B.**[†]

23 grudnia 2020

Spis treści

[*]Politechnika Krakowska im. Tadeusza Kościuszki w Krakowie
[†]Politechnika Krakowska im. Tadeusza Kościuszki w Krakowie

1 System plików w Haskellu

Opis interakcji z systemem plików za pomocą haskella możemy opisać w poniższy sposób:

Ścieżka do pliku (FilePath) przekazuje informacje do wrapperów do obliczeń, które mogą być użyte przez System.IO.
Wrappery są zaprojektowane tak, aby działały jak najbardziej kompatybilnie w różnych wersjach GHC (Glasgow Haskell Compiler).

Cechuje je to, że nie wymagają, aby ścieżki plików POSIX były poprawnymi ciągami znaków, dlatego mogą otwierać ścieżki niezależnie od bieżącego kodowania regionalnego.

2 System plików - użycie w Haskellu

Haskell definiuje operacje odczytu i zapisu znaków z, i do plików, reprezentowanych przez wartości typu **Handle**.

Każda wartość typu **Handle** jest rekordem używanym przez system wykonawczy Haskell do zarządzania I/O (Input/Output) z obiektami systemu plików.

Wartość **Handle** posiada przynajmniej następujące właściwości:

- Czy zarządza wejściem czy wyjściem, czy obydwoma;
- Czy jest otwarta, zamknięta czy półzamknięta;
- Czy obiekt jest możliwy do wyszukania;
- Czy buforowanie jest wyłączone, czy włączone bazując na linii lub bloku;
- Bufor (którego długość może wynosić zero).

Większość wartości **Handle**, ma również bieżącą pozycję I/O wskazującą, gdzie nastąpi następna operacja wejścia lub wyjścia. **Handle** jest czytelna, jeśli zarządza tylko wejściem lub zarówno wejściem, jak i wyjściem; podobnie jest zapisywalna, jeśli zarządza tylko wyjściem lub zarówno wejściem, jak i wyjściem.

Wartość **Handle** jest otwarta kiedy zostaje po raz pierwszy asygnowana. Kiedy ją zamkniemy nie można jej już używać jako wejścia ani wyjścia, poprzez taką implementację nie może ponownie wykorzystać swojej pamięci, dopóki pozostają do niej odniesienia.

Wartości **Handle** znajdują się w klasach „Show" i „Eq". Ciąg utworzony przez pokazanie **Handle** jest zależny od systemu; powinien zawierać wystarczającą ilość informacji, aby zidentyfikować wartość **Handle** do debugowania.
Wartość **Handle** jest równa zgodnie ze znakiem == tylko sobie; nie jest wykonywana próba porównania stanu wewnętrznego różnych wartości **Handle** dla równości.

Tym podejściem programowania funkcyjnego Haskella, możemy wykonywać operacje na plikach jak i również katalogach.

Pliki:

- Binary files (pliki binarne);
- Text files (pliki tekstowe);

Katalogi:

- Tworzenie katalogów;
- Usuwanie katalogów;
- Bierzący katalog roboczy;
- Najczęściej używane ścieżki;

3 Cel projektu

Celem projektu jest wykorzystanie możliwości programowania funkcyjnego, za pomocą języka Haskell do operacji na plikach.

4 Opis programu

Program składa się z kilku funkcjonalności operacji na plikach, wykorzystujemy standardowe biblioteki Haskella, głównie Filesystem. Poniżej prezentujemy kilka funkcji z programu.

```haskell
copyFiles :: FilePath -> IO[FilePath]
copyFiles topdir = do
    names <- getDirectoryContents topdir
    let properNames = filter (`notElem` [".", ".."]) names
    paths <- forM properNames $ \name -> do
        let path = topdir </> name
        isDirectory <- doesDirectoryExist path
        let a = "/home/backup/" ++ name
        if isDirectory
            then putStr "Folder"
            else copyFile path a
        if isDirectory
            then copyFiles path
            else return [path]
    return (concat paths)
```

Rysunek 1: Kopiowanie plików

```haskell
moveFile :: FilePath -> IO()
moveFile path = do
    let a = "/home/move/" ++ takeFileName path
    copyFile path a
    copyPermissions path a
    removeFile path
```

Rysunek 2: Przenoszenie pliku

```haskell
copyOnly :: FilePath -> IO[FilePath]
copyOnly topdir = do
    names <- getDirectoryContents topdir
    let properNames = filter (`notElem` [".", ".."]) names
    paths <- forM properNames $ \name -> do
        let path = topdir </> name
        isDirectory <- doesDirectoryExist path
        let a = "/home/backup/" ++ name
        if isDirectory
            then putStr "Folder"
            else if takeExtension name == ".c"
                then copyFile path a
            else putStr "Diffrent extension"
        if isDirectory
            then copyFiles path
            else return [path]
    return (concat paths)
```

Rysunek 3: Kopiowanie plików tylko z danym rozszerzeniem

5 Podsumowanie

Programy wykonuje poprawnie kopiowanie jak i również przenoszenie plików. Widzimy że podejściem programowania funkcyjnego jesteśmy w stanie wykonać te operacje za pomocą Haskella.

6 Bibliografia

- `https://hackage.haskell.org/package/system-fileio-0.3.16.4/docs/Filesystem.html`(data dostępu: 16.12.2020)
- `http://hackage.haskell.org/package/filepath-1.4.2.1/docs/System-FilePath-Posi html`(data dostępu: 16.12.2020)
- Bryan O'Sullivan, John Goerzen, Don Stewart-Real World Haskell-O Reilly Media (2008)

  – I/O Case Study: A Library for Searchingthe Filesystem (Chapter 9)