Christoph Lüth <cxl>
Felix Beckwermert <foetus>
Michael Skibbe <mskibbe>
Dennis Walter <dw>
Rafael Trautmann <pirate>
Thomas Meyer <mclee>

5. Übungsblatt

Ausgabe: 16.12.2002

Bearbeitungszeit: Zwei Wochen

Leider war das Officepaket HASKELL XY nicht der erhoffte Kassenschlager. Zwei verkaufte Exemplare reichten nicht aus, die fünfwöchige Werbekampagne zu finanzieren, und so ist das *venture capital* schnell aufgebraucht.

Etwas desillusioniert wechseln Sie die Seiten, und schließen sich der *open source*-Bewegung[1] an. Deshalb werden wir in diesem Aufgabenblatt ein bekanntes Unix-Kommando reimplementieren, natürlich in Haskell.

9 *Synchronisation* — *20 Punkte*

In dieser Aufgabe soll ein Programm implementiert werden, welches es ermöglicht, Änderungen zwischen zwei Verzeichnissen abzugleichen (im Sinne des Unix-Kommandos `rsync(1)`, allerdings nur zwischen lokalen Verzeichnissen).

Das Programm soll wie folgt aus der Kommandozeile aufgerufen werden können:

```
sync [-i] src trg
```

wobei `src` das Quellverzeichnis und `trg` das Zielverzeichnis ist. Das `sync`-Utility soll alle Dateien in `src`, die in `trg` nicht existieren, oder die in `src` später als in `trg` modifiziert wurden, nach `trg` kopieren. Unterverzeichnisse werden rekursiv behandelt. Verzeichnisse in `trg`, einschließlich `trg` selbst, sollen wenn nötig angelegt werden.

Die Option `-i` bezeichnet den *interaktiven* Modus. Im interaktiven Modus soll der Benutzer vorher um Bestätigung gefragt werden, bevor eine alte Datei überschrieben wird.

Beachten Sie die Fehlerbehandlung, das Programm sollte bei nicht lesbaren oder schreibbaren Verzeichnissen/Dateien nicht undefiniert abbrechen.

Hinweise: Folgende Funktionen sind für diese Aufgabe sehr hilfreich:

- Die meisten Funktionen im Modul `Directory`, insbesondere die folgenden:

[1] `http://www.fsf.org/`

```
getDirectoryContents :: FilePath -> IO [FilePath]
doesFileExist        :: FilePath -> IO Bool
doesDirectoryExist   :: FilePath -> IO Bool
createDirectory      :: FilePath -> IO ()
getModificationTime  :: FilePath -> IO ClockTime
```

`getDirectoryContents` liefert als Inhalt des Verzeichnisses die Liste der Dateinamen, allerdings nicht den vollen Pfad. `doesDirectoryExist` und `doesFileExist` prüfen, ob das angegebene Verzeichnis bzw. die angegebene Datei existiert; mit diesen Funktionen läßt sich auch herausfinden, ob ein Dateiname, der von `getDirectoryContents` zurückgegeben wurde, ein Verzeichnis oder eine Datei ist.

`getModificationTime` liefert die Zeit der letzten Modifikation. Der Rückgabetype ist eine Instanz der Klasse `Ord` und kann daher mit `<` etc. verglichen werden.

- Das Modul `System` bietet den Zugriff auf Kommandozeilenargumente (`getArgs`) und Shell-Kommandos (`system`):

```
data ExitCode = ExitSuccess | ExitFailure Int
                deriving (Eq, Ord, Read, Show)

getArgs  :: IO [String]
system   :: String -> IO ExitCode
```

- Es gibt keine explizite Funktion, um Dateien zu kopieren; man kann entweder einen geeigneten Shell-Befehl zusammenstellen, und mit der Funktion `System.system` ausführen (was aber nicht portabel ist), oder die Datei mit `readFile` lesen und `writeFile` schreiben (was aber nicht so effizient ist).

Dieses ist Version 1.0 von 16. Dezember 2002.