Funktionale Programmierung

`http://proglang.informatik.uni-freiburg.de/teaching/functional-programming/2013/`

Übungsblatt 12 (Data Types à la Carte, Free Monads)

Fr, 2014-02-07

Hinweise

- Lösungen sollen in das persönliche Subversion (svn) Repository hochgeladen werden. Die Adresse des Repositories wird per Email mitgeteilt.
- **Alle** Aufgaben müssen bearbeitet und pünktlich abgegeben werden. Falls das sinnvolle Bearbeiten einer Aufgaben nicht möglich ist, kann eine stattdessen eine Begründung abgegeben werden.
- Wenn die Abgabe korrigiert ist, wird das Feedback in das Repository hochgeladen. Die Feedback-Dateinamen haben die Form `Feedback-<user>-ex<XX>.txt`.
- Allgemeinen Fragen zum Übungsblatt können im Forum (`http://proglang.informatik.uni-freiburg.de/forum/viewforum.php?f=38`) geklärt werden.

Abgabe: Sa, 2014-02-15

Hinweis: Für dieses Blatt benötigen Sie das Modul `DTC.hs` von der Vorlesungshomepage. Es enthält die relevanten Definitionen aus der Vorlesung.

Aufgabe 1 (QBF Ausdrücke)

Wenden Sie den Data Types à la Carte (DTC) Ansatz zur Definition von erweiterbaren Datentypen auf Boolesche Formeln mit Quantoren (QBF) an. Ein Beispiel einer solchen Formel ist:

$$\forall x.\ x \vee \neg x$$

Gehen Sie wie folgt vor:

1. Legen Sie zuerst ein Module `Ex12_QBF1.hs` an und definieren Sie dort boolesche Konstanten ($\top$ und $\bot$), Konjunktion ($\phi_1 \vee \phi_2$), Negation $\neg\phi$ und All-Quantifikation ($\forall x.\ \phi$). D.h. eine Formel wie

   $$\forall x.\ \neg(x \vee \top)$$

   sollte in ihrem Datentyp direkt ausdrückbar sein, eine wie

   $$\forall x.\ \neg(x \vee \top) \wedge \exists x.\ \bot \wedge x$$

   nicht.

2. Definieren Sie eine erweiterbare Auswertungsfunktion für den Datentyp. Hinweis: Sie werden während der Auswertung die aktuellen Belegungen der freien Variablen speichern müssen.

3. Legen Sie nun ein neues Modul `Ex12_QBF2.hs` an und erweitern Sie dort den Datentyp um Disjunktion ($\phi_1 \wedge \phi_2$) und Existenz-Quantifikation $\exists x.\ \phi$. Das ursprüngliche Modul sollte für die Erweiterung nicht verändert werden. Erweitern Sie auch die Auswertungsfunktion entsprechend.

4. Schreiben Sie in einem dritten Modul `Ex12_QBF3.hs` einen Pretty-Printer für ihren Datentyp. (Ein Pretty-Printer ist eine Funktion, die eine „gut-lesbare" String-Repräsentation einer Formel berechnet.)

Aufgabe 2 (FS)

Im Modul `FS.hs` (auf der Homepage) finden Sie die Monade `FS`, die ein (flaches) Dateisystem simuliert. Die Monade unterstützt die folgenden Operationen:

```
createFile :: String -> FS ()
deleteFile :: String -> FS ()
writeFile :: String -> String -> FS ()
readFile :: String -> FS String
```

und kann mit `runFS :: FS a -> FSRep -> (Either String a, FSRep)` ausgeführt werden. `FSRep` speichert die Dateien. Ein frisches, leeres Dateisystem erhält man mit `fsEmpty`.[1]

Im Modul `FS.hs` befinden sich auch zwei `FS`-Beispielprogramme:

```
-- This example finishes successfully
ex_fs1 = do
  createFile "Hello.txt"
  writeFile "Hello.txt" "Hello World!"
  createFile "diary.txt"
  writeFile "diary.txt" "Not much going on..."
  diary <- readFile "diary.txt"
  deleteFile "diary.txt"
  writeFile "Hello.txt" "Hello, again!"
  return diary
-- Output:
-- * > FS.runFS FS.ex_fs1  FS.fsEmpty
-- (Right "Not much going on...",["Hello.txt -> \"Hello, again!\""])
```

```
-- Aborts with a 'file-not-found' error
ex_fs2 = do
  createFile "Hello.txt"
  writeFile "Hello.txt" "Hello World!"
  createFile "diary.txt"
  writeFile "diary2.txt" "Not much going on..."
  diary <- readFile "diary.txt"
  deleteFile "diary.txt"
  writeFile "Hello.txt" "Hello, again!"
-- Output:
-- * > FS.runFS FS.ex_fs2  FS.fsEmpty
-- (Left "File 'diary2.txt' not found"
--  ,["Hello.txt -> \"Hello World!\"","diary.txt -> \"\""])
```

Operationen auf Dateisystemen werden häufig in Lese-, Schreib- und Verzeichnisoperationen (wie `createFile` und `deleteFile`) aufgeteilt und es ist möglich für diese drei Klassen die Rechte separat zu vergeben. Den `FS`-Programmen ist am Typ nicht anzusehen, zu welche Rechte sie benötigen. Benutzen Sie die in der Vorlesung am Rechner-Beispiel gezeigte Free-Monad Konstruktion, um eine Version von `FS` zu implementieren, in der die benötigten Rechte explizit als Summe von Funktoren im Typ eines Programms angegeben werden können.

1. Definieren Sie zuerst geeignete Funktoren für die drei Berechtigungsklassen.
2. Implementieren Sie dann Smart-Konstruktoren für Aktionen der freien Monade, die den `FS`-Operationen entsprechen.
3. Implementieren Sie einen Interpreter `runFSTerm`, der die Aktionen in `FS` Aktionen übersetzt. (Das heißt, `runTermFS tm :: FS a` für eine Term `tm` mit Ergebnis-Typ `a`.)

[1] Bei Interesse können Sie sich die weiteren Operationen für `FSRep` in `FS.hs` ansehen. Notwendig ist das zum Lösen der Aufgabe nicht.