Funktionale Programmierung

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

Übungsblatt 8 (Interpreter, Monaden)

Mi, 2013-12-11

Hinweise

- Lösungen sollen als Haskell Quellcode 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: Mi, 2013-12-18

Achtung: Dies ist eine korrigierte Version (2013-12-12)

Aufgabe 1 (Matcher)
Die (zum Zweck der Übung erfundene) Monade Matcher a b ist eine Kombination aus Reader und Error Monade. Sie erlaubt das Lesen aus einer Umgebung mit der Aktion

```
matchTarget :: Matcher a b a
```

und das Abbrechen der Berechnung mit

```
matchSuccess :: b -> Matcher a b c
```

Definieren Sie die erforderlichen Datentypen, Funktionen und Instanzen für die Matcher Monade

- direkt
- mit der Verwendung der in der Vorlesung definierten Monaden-Transformer (Modul `Transformers.lhs` auf der Homepage. **Achtung:** Der Code wurde 2013-12-12 korrigiert. Es wurde ein ErrorT Transformer hinzugefügt, der ähnlich wie die Error-Monade `M2` aus der Vorlesung funktioniert.

Die folgende Funktion benutzt die Matcher-Monade (siehe auch `MatcherExample.hs` auf der Homepage):

```
data FileType = Haskell | Java | Tex | Binary | Other String
  deriving (Eq, Show)

fileType :: String -> FileType
fileType name = runMatcher name $ do
  matchExt "hs" Haskell
  matchExt "java" Java
  matchExt "tex" Tex
  match "a.out" Binary
  ext <- getExt
  return $ Other ext

tests_matcher =
```

```
testGroup "matcher"
[ testProperty "haskell" $ fileType "Hello.hs" == Haskell
, testProperty "java" $ fileType "Hello.java" == Java
, testProperty "tex" $ fileType "Hello.tex" == Tex
, testProperty "bin" $ fileType "a.out" == Binary
, testProperty "fail" $ fileType "bla.blub" == Other "blub"
]
```

Ergänzen Sie den Code so dass er kompiliert und die Testfälle durchlaufen.

Aufgabe 2 (While)
Implementieren Sie einen Interpreter für eine While-Sprache ähnlich der von Blatt Ex07. Die einzige Erweiterung ist ein neues Statement `print e` das den Ausdruck `e` ausgibt. Benutzen Sie den StateT Transformer um die Werte der Variablen zu verwalten und den WriterT Transformer für die Ausgabe. Das Resultat der Interpretation soll die Variablenbelegung sein, die das While-Programm produziert, zusammen mit der Ausgabe.

- Nehmen Sie zunächst an, dass Variablen immer **Integer** Werte speichern. Zu Beginn haben alle Variablen den Wert 0. (Da die While-Sprache auch boolesche Werte unterstützt, müssen Sie diese, à la C, als Zahlen darstellen)
- (Optional) Erweitern Sie den Interpreter, so dass boolesche Werte und **Integer** Werte zur Laufzeit unterschieden werden. Bei unzulässigen Statements und Operationen, wie `while 5 do ... done` und `(1 == 1) + (2 == 2)`, soll die Ausführung mit einer Fehlermeldung abbrechen.

Den Datentyp für die While-Sprache sowie einen Parser und Pretty-Printer finden Sie auf der Hompage als Modul `While2.hs` (benötigt `Parser2.hs`).

Einige Hinweise:

- Ein möglicher Zustandtyp wäre zum Beispiel eine Liste aus Name-Wert Paaren, die die aktuelle Variablenbelegung repräsentiert:

```
type WhileState = [(Id, Value)]
type Id = String
type Value = ...
```

- Die nötigen Operationen zur Zustandsmodifikation lassen sich z.B. mit Hilfe der Funktionen **filter**, **insertBy** und **lookup** implementieren
- Beim Zugriff auf nicht initialisierte Variablen können Sie einen Default-Wert zurückgeben.
- Achten Sie darauf, keine mehrfachen Einträge für eine Variable zu erzeugen, damit auch bei längeren Ausführungen nicht zu viel Speicher verbraucht wird.
- Das Modul Control.**Monad** enthält viele nützliche Funktionen für das Programmieren mit Monaden.