Institut für Informatik
der Universität München
Dr. Andreas Abel
Dr. Hans-Wolfgang Loidl

SS 2009

Übungen zur Vorlesung
Funktionale Programmierung

Blatt 2

Aufgabe P-6: Holen Sie sich den Parser für arithmetische Ausdrücke von der Webseite. Modifizieren Sie ihn so, dass er einen Syntaxbaum zurückliefert, statt gleich das Ergebnis auszurechnen.

```
data Op             =   Plus | Times

data Expr           =   Const Int
                    |   Op Op  Expr Expr
```

Schreiben Sie einen Auswerter `eval :: Expr -> Int` für Syntaxbäume.

Aufgabe P-7: Der Auswerter der vorigen Aufgabe soll nun mitprotokollieren, welche Multiplikationen er durchführt. Dazu eignet sich eine Logging-Monade.

```
newtype Log a = Log { runLog :: (a, String) }

log     :: String -> Log ()
```

`log` gibt eine Log-Nachricht aus. Definieren Sie `return` und `>>=` für die Log-Monade. Die Sequenzoperation `>>=` muss die Log-Nachrichten in der richtigen Reihenfolge zusammenhängen. Schreiben Sie den Auswerter um zu

```
eval :: Expr -> Log Int
```

Aufgabe H-3: Erweitern Sie Syntaxbaum und Parser für arithmetische Ausdrücke um `let x = e in e'`.

Um Ausdrücke mit `let` auszuwerten, brauchen wir eine Umgebung `Env`, die den Wert der definierten Variablen speichert.

```
Env = String -> Int
```

```
empty :: Env
empty x = 0

extend :: String -> Int -> Env -> Env
extend x i env y = if x == y then i else env y
```

Implementieren Sie eine Monade

```
newtype WithEnv a = WithEnv { runWEnv :: Env -> a }

instance Monad WithEnv where ...

resolve :: String -> WithEnv Int
update  :: String -> Int -> WithEnv a -> WithEnv a
```

so dass z.B.

```
update x i $ resolve x = i
```

Schreiben Sie einen Auswerter für arithmetische Ausdrücke mit `let`:

```
eval :: Expr -> WithEnv Int

interp :: Expr -> Int
interp e = runWEnv (eval e) empty
```

Aufgabe H-4: Der Auswerter der vorigen Aufgabe liefert für undefinierte Variablen einfach `0`. Modifizieren Sie den Auswerter, dass er stattdessen eine passende Fehlermeldung generiert und die Auswertung abbricht. Dazu müssen Sie die Fehlermonade der Vorlesung in `WithEnv` integerieren. Die kombinierte Monade nennen wir schlicht `Eval`.

```
type Env = String -> Except Int
newtype Eval a = Eval { runEval :: Env -> Except a }
```

Modifizieren Sie den Code der vorigen Aufgabe entsprechend.