CS314. Функциональное программирование

Лекция 16. Примеры использования преобразователей монад

В. Н. Брагилевский

Направление «Фундаментальная информатика и информационные технологии»
Институт математики, механики и компьютерных наук имени И. И. Воровича
Южный федеральный университет

19 ноября 2016 г.

Содержание

Содержание

Подсчёт количества файлов в дереве каталогов

Задача

Даны имя каталога и глубина поиска. Посчитать количество файлов в каждом подкаталоге данного каталога, опускаясь не более чем на заданную глубину в дереве каталогов.

Компоненты решения

- Конфигурация: максимальная глубина.
- Состояние: текущая глубина, имя каталога.
- Результат: журнал со списком посещённых каталогов и количеством файлов в них.
- Используемые монады: IO, Writer, Reader, State.

Решение: используемые расширения и модули

```
{-# LANGUAGE GeneralizedNewtypeDeriving #-}
import System.Environment
import System.Directory
import System.FilePath
import Control.Monad.Writer
import Control.Monad.Reader
import Control.Monad.State
import Control.Applicative
```

Решение: конфигурация, состояние и журнал

```haskell
data AppConfig = AppConfig {
      cfgMaxDepth :: Int
    } deriving (Show)

data AppState = AppState {
      stCurDepth :: Int,
      stCurPath :: FilePath
    } deriving (Show)

type AppLog = [(FilePath, Int)]
```

Решение: стек монад

```
newtype MyApp a = MyA {
      runA :: ReaderT AppConfig (StateT AppState
                                 (WriterT AppLog IO)) a
    } deriving (Functor, Applicative, Monad,
                MonadIO,
                MonadReader AppConfig,
                MonadWriter [(FilePath, Int)],
                MonadState AppState)

runMyApp :: MyApp a -> Int -> FilePath -> IO (a, AppLog)
runMyApp app maxDepth path =
    let config = AppConfig maxDepth
        state = AppState 0 path
    in runWriterT (
           evalStateT (
              runReaderT (runA app) config) state)
```

Решение: вспомогательная функция

```
listDirectory :: FilePath -> IO [String]
listDirectory = liftM (filter notDots) . getDirectoryContents
    where notDots p = p /= "." && p /= ".."
```

Решение: вычисление в монаде MyApp

```
constrainedCount :: MyApp ()
constrainedCount = do
  maxDepth <- liftM cfgMaxDepth ask
  st <- get
  let curDepth = stCurDepth st
  when (curDepth < maxDepth) $ do
    let path = stCurPath st
    contents <- liftIO $ listDirectory path
    tell [(path, length contents)]
    forM_ contents $ \name -> do
      let newPath = path </> name
      isDir <- liftIO $ doesDirectoryExist newPath
      when isDir $ do
        put $ st {stCurDepth = curDepth + 1,
                  stCurPath = newPath}
        constrainedCount
```

Решение: запуск вычисления в функции main

```
main = do
  [md, p] <- getArgs
  (_, xs) <- runMyApp constrainedCount (read md) p
  print xs
```

Пример запуска

```
$ ./count 2 "."
[(".",4),("./A",1),("./B",0)]
$ ./count 3 "."
[(".",4),("./A",2),("./A/B2",0),("./A/A2",2),("./B",0)]
```

Результаты

Что в итоге?

Разработана схема приложения со следующими возможностями:

- чтение конфигурационной информации;
- запись журнала;
- использование изменяемого состояния;
- работа с вводом-выводом.

Гибкость полученного решения

- Возможно расширение конфигурационной информации.
- Возможно расширение состояния.
- Возможно изменение формата журнала.

Содержание

Постановка задачи (Constraints Satisfaction Problem)

Имеется логическая задача с переменными и ограничениями в форме предикатов. Необходимо установить возможные значения переменных, удовлятворяющие всем ограничениям.

Пример задачи

В племени калотанцев есть давний обычай: их мужчины всегда говорят только правду, тогда как женщины никогда не говорят правду или неправду два раза подряд. Антрополог Джонс начал изучать племя калотанцев, хотя с их языком пока не знаком. Однажды он встречает разнополую пару калотанцев с ребёнком Киби. Джонс спрашивает Киби: «А ты мальчик?», на что ребёнок отвечает на калотанском, которого Джонс не понимает. Затем Джонс обращается к родителям, знающим английский, за разъяснениями. Один из них говорит: «Киби сказал, что он мальчик». Другой же сообщает: «Киби девочка, она соврала». Каков пол Киби и каждого из родителей?

Идеи решения

- Переменные: родители и ребёнок, их значения — пол (мужской/женский).
- Формулировка необходимых предикатов (как общих, так и для конкретной задачи) => множество ограничений.
- Перебор множества решений (подстановок значений переменных) и проверка каждого на предмет удовлетворения множества ограничений.
- Списковая монада для реализации недетерминированных вычислений и фильтрации.
- Монада State для работы с переменными.

Переменные и предикаты

```
type Var = String
type Value = String
data Predicate =
             Is Var Value
           | Equal Var Var
           | And Predicate Predicate
           | Or Predicate Predicate
           | Not Predicate
  deriving (Eq, Show)

type Variables = [(Var, Value)]
```

Составные предикаты

```haskell
isNot :: Var -> Value -> Predicate
isNot var value = Not (Is var value)

implies :: Predicate -> Predicate -> Predicate
implies a b = Not a `Or` b

orElse :: Predicate -> Predicate -> Predicate
orElse a b = (a `And` (Not b)) `Or` ((Not a) `And` b)
```

Проверка значения предиката

```
check :: Variables -> Predicate -> Maybe Bool
check vars (Is var val) = liftM (==val) (lookup var vars)
check vars (Equal v1 v2) = liftM2 (==) (lookup v1 vars)
                                       (lookup v2 vars)
check vars (And p1 p2) = liftM2 (&&) (check vars p1)
                                     (check vars p2)
check vars (Or  p1 p2) = liftM2 (||) (check vars p1)
                                     (check vars p2)
check vars (Not p) = liftM not (check vars p)
```

Состояние задачи

```
data ProblemState = PS {vars :: Variables,
                        constraints :: [Predicate]}

findVar :: Var -> ProblemState -> Maybe Value
findVar v (PS vs _) = lookup v vs

updateVar :: Var -> Value -> ProblemState -> ProblemState
updateVar v x (PS vs cs)
            = PS ((v, x) : filter ((v/=).fst) vs) cs
```

Состояние + список = недетерминированное состояние

```haskell
type NDS a = StateT ProblemState [] a

getVar :: Var -> NDS (Maybe Value)
getVar v = findVar v <$> get

setVar :: Var -> Value -> NDS ()
setVar v x = modify (updateVar v x)
```

```haskell
get :: m s
gets :: MonadState s m => (s -> a) -> m a
modify :: MonadState s m => (s -> s) -> m ()
```

Проверка удовлетворения ограничений

```
isConsistent :: Bool -> NDS Bool
isConsistent partial = do
  cs <- gets constraints
  vs <- gets vars
  return $ all (maybe partial id . check vs) cs

getFinalVars :: NDS Variables
getFinalVars = do
  c <- isConsistent False
  guard c
  gets vars
```

- Решение частично согласовано (partial), если не все переменные определены.

```
maybe :: b -> (a -> b) -> Maybe a -> b
```

Поиск решения (перебор значений и результаты)

```
tryAllValues :: Var -> [Value] -> NDS ()
tryAllValues var values = do
  msum $ map (setVar var) values
  c <- isConsistent True
  guard c
```

```
getAllSolutions :: ProblemState -> NDS a -> [a]
getAllSolutions ps c = evalStateT c ps

getSolution :: ProblemState -> NDS a -> Maybe a
getSolution ps c = listToMaybe (evalStateT c ps)
```

Формулировка задачи на естественном языке

В племени калотанцев есть давний обычай: их мужчины всегда говорят только правду, тогда как женщины никогда не говорят правду или неправду два раза подряд. Антрополог Джонс начал изучать племя калотанцев, хотя с их языком пока не знаком. Однажды он встречает разнополую пару калотанцев с ребёнком Киби. Джонс спрашивает Киби: «А ты мальчик?», на что ребёнок отвечает на калотанском, которого Джонс не понимает. Затем Джонс обращается к родителям, знающим английский, за разъяснениями. Один из них говорит: «Киби сказал, что он мальчик». Другой же сообщает: «Киби девочка, она соврала». Каков пол Киби и каждого из родителей?

Специфичные для задачи предикаты

```
said :: Var -> Predicate -> Predicate
said v p = (v `Is` "male") `implies` p

saidBoth :: Var -> Predicate -> Predicate -> Predicate
saidBoth v p1 p2 =
     And ((v `Is` "male") `implies` (p1 `And` p2))
         ((v `Is` "female") `implies` (p1 `orElse` p2))

lied :: Var -> Predicate -> Predicate
lied v p = ((v `said` p) `And` (Not p))
                        `orElse` ((v `said` (Not p)) `And` p)
```

Формулировка задачи на Haskell

```
main = do
  let variables = []
      values = ["male", "female"]
      constraints = [
        Not (Equal "parent1" "parent2"),
        "parent1" `said` ("child" `said`
                              ("child" `Is` "male")),
        saidBoth "parent2" ("child" `Is` "female")
                           ("child" `lied`
                              ("child" `Is` "male"))]
       problem = PS variables constraints
  print $ getSolution problem $ do
     tryAllValues "parent1" values
     tryAllValues "parent2" values
     tryAllValues "child" values
     getFinalVars
```

Источники примеров

1. Приложение с конфигурацией, состоянием и журналом: глава 18 из Real World Haskell (`http://book.realworldhaskell.org/read/monad-transformers.html`)
2. Логическая задача: пункт 22.3 StateT with List из All About Monads (`https://www.haskell.org/haskellwiki/All_About_Monads#StateT_with_List`)

Решение задачи о калотанцах

```
ghci> :main
Just [("child","female"),
      ("parent2","male"),
      ("parent1","female")]
```

Что дальше?

Лекции

- Расширения компилятора GHC: семейства типов, Template Haskell.
- Использование внешних библиотек, сборка проектов, тестирование и профилирование.
- Веб-программирование (Yesod).
- Эффективный ввод-вывод, обработка исключений.
- Параллельное, конкурентное и распределённое программирование.
- Обзор внутреннего устройства компилятора GHC (24 декабря).

Практика

- Вторая контрольная работа (9 и 10 декабря).
- Проектное задание (25 ноября — 24 декабря).