CS314. Функциональное программирование

Лекция 15. Монада ST. Преобразователи монад

В. Н. Брагилевский

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

18 ноября 2016 г.

Содержание

1. Монада ST
2. Комбинирование монад: простой пример
3. Преобразователи стандартных монад
4. Анатомия преобразователей монад

Содержание

1. Монада ST
2. Комбинирование монад: простой пример
3. Преобразователи стандартных монад
4. Анатомия преобразователей монад

Вычисление чисел Фибоначчи

```haskell
fib' :: Int -> Integer
fib' 0 = 0
fib' 1 = 1
fib' n = fib' (n-1) + fib' (n-2)
```

- Долгие вычисления.
- Большой расход памяти.

```haskell
fib'' :: Int -> Integer
fib'' n = fst $ fib2 n
  where
    fib2 0 = (0, 1)
    fib2 n =
       let (a, b) = fib2 (n-1)
       in (b, a+b)
```

- Большой расход памяти.

Монада ST (state-transformer, state-thread)

```
data ST s a
```

- s не обозначает конкретный тип состояния, это просто метка, позволяющая различать вычисления с разными состояниями (вычислительные потоки с состоянием) и запрещающая их смешивать;
- a — это тип результата вычислений.

```
runST :: (forall s. ST s a) -> a
```

Изменяемые переменные (модуль Data.STRef)

```
data STRef s a
```

```
newSTRef :: a -> ST s (STRef s a)
readSTRef :: STRef s a -> ST s a
writeSTRef :: STRef s a -> a -> ST s ()
modifySTRef :: STRef s a -> (a -> a) -> ST s ()
modifySTRef' :: STRef s a -> (a -> a) -> ST s ()
```

Эффективное вычисление чисел Фибоначчи

```haskell
fibST :: Int -> ST s Integer
fibST n = do
  a <- newSTRef 0
  b <- newSTRef 1
  replicateM_ n $ do
      x <- readSTRef a
      y <- readSTRef b
      writeSTRef a y
      writeSTRef b $! x + y
  readSTRef a
```

- Две переменные (ссылки) a и b.
- Повторяемое n раз действие (тело цикла).
- Строгое вычисление x+y (операция $!).
- Извлечение результата вычисления из переменной a.

```haskell
fib :: Int -> Integer
fib n = runST (fibST n)
```

- Чистая функция!

Примеры

```
ghci> fib' 25
75025
(0.29 secs, 85928376 bytes)
ghci> fib'' 55
139583862445
(0.01 secs, 2062704 bytes)
ghci> fib 55
139583862445
(0.00 secs, 0 bytes)
ghci> fib 155
1105603071560908172376327542123 45
(0.00 secs, 1058472 bytes)
```

Изменяемые массивы в монаде ST

```
newListArray ::
        (MArray a e m, Ix i) => (i, i) -> [e] -> m (a i e)
readArray ::
        (MArray a e m, Ix i) => a i e -> i -> m e
writeArray ::
        (MArray a e m, Ix i) => a i e -> i -> e -> m ()
```

```
runSTArray ::
     Ix i => (forall s . ST s (STArray s i e)) -> Array i e
```

Пример вычисления (обмен значениями элементов)

```
import Data.STRef
import Control.Monad.ST

import Data.Array
import Data.Array.ST
import Data.Array.MArray

swapElems :: Ix i => i -> i -> STArray s i e -> ST s ()
swapElems i j arr = do
    vi <- readArray arr i
    vj <- readArray arr j

    writeArray arr i vj
    writeArray arr j vi
```

Пример использования

```
test :: Int -> Int -> [a] -> [a]
test i j xs = elems $ runSTArray $ do
    arr <- newListArray (0, length xs - 1) xs
    swapElems i j arr
    return arr
```

```
ghci> test 1 3 [0,1,2,3,4]
[0,3,2,1,4]
ghci> test 4 2 [0,1,2,3,4]
[0,1,4,3,2]
```

Более сложные примеры

- Сортировки методами пузырька, вставки, поиска.
- Быстрая сортировка Хоара (настоящая, inplace).

Содержание

1 Монада ST

2 Комбинирование монад: простой пример

3 Преобразователи стандартных монад

4 Анатомия преобразователей монад

Запрос пароля

Постановка задачи

При регистрации в некоторой системе пользователь должен указать пароль, который он будет использовать в дальнейшем для аутентификации.
Необходимо организовать чтение предлагаемого пользователем пароля с клавиатуры, совместив его с проверкой, является ли он достаточно стойким.

Проверка стойкости

```
isValid :: String -> Bool
isValid s =
    length s >= 8
    && any isAlpha s
    && any isNumber s
    && any isPunctuation s
```

Используемые монады

- `IO` — ввод-вывод;
- `Maybe` — `Just <пароль>` либо `Nothing` (возможная неудача — пароль не прошёл проверку на стойкость).

Запрос пароля: вариант решения

Чтение пароля

```
getPassword = do
    putStrLn "Введите новый пароль:"
    s <- getLine
    return (if isValid s then Just s else Nothing)

askPassword = do
    mpasswd <- getPassword
    if isJust mpasswd
        then do
            putStrLn "Сохранение..."
            -- необходимые действия
        else
            askPassword
```

- В какой монаде исполняется этот код?
- Используются ли здесь возможности монады Maybe?

Ответы: IO, нет.

Комбинирование монад

- Хочется иметь средство для комбинирования монад: то есть уметь создавать монаду, объединяющую возможности двух или более имеющихся монад.
- Оказывается, не любые две монады можно скомбинировать.
- Не существует общего способа скомбинировать две произвольные монады.
- Можно написать код, добавляющий возможности некоторой конкретной монады к любой другой. Например: монада IO с возможной неудачей в вычислениях.
- Основные понятия: преобразователи монад (monad transformers), стек монад, подъём по стеку.

Решение с построением стека монад

```
getPassword :: MaybeT IO String
getPassword = do
  lift $ putStrLn
       "Введите новый пароль:"
  s <- lift getLine
  guard (isValid s)
  return s

askPassword :: MaybeT IO ()
askPassword = do
  value <- msum $
           repeat getPassword
  lift $ putStrLn "Сохранение..."

main = runMaybeT askPassword
```

Компоненты решения

- MaybeT — преобразователь монад.
- MaybeT IO String — стек монад (тоже монада).
- Maybe — базовая монада, IO — внутренняя монада.
- lift — «поднятие» функции до внутренней монады.
- guard, msum — возможности экземпляра MonadPlus для Maybe.
- runMaybeT — запуск новой монады.

Импортируемые модули и типы

```
import Control.Monad
import Control.Monad.Trans
import Control.Monad.Trans.Maybe
```

```
getPassword :: MaybeT IO String
askPassword :: MaybeT IO ()
```

```
lift :: (Monad m, MonadTrans t) => m a -> t m a
msum :: (Foldable t, MonadPlus m) => t (m a) -> m a
runMaybeT :: MaybeT m a -> m (Maybe a)
```

Содержание

1. Монада ST
2. Комбинирование монад: простой пример
3. Преобразователи стандартных монад
4. Анатомия преобразователей монад

Преобразователи стандартных монад

- MaybeT, ReaderT, WriterT, StateT, ...

Пример: версия 1

```
m :: MaybeT (ReaderT Integer IO) String
m = do
  n <- lift ask
  guard (n > 0)
  lift $ lift $ print n
  return "OK"

main = runReaderT (runMaybeT m) 2
```

- Базовая монада — Maybe.
- Внутренняя монада — IO.
- Стек содержит три монады.
- Поднятие (lift).

Автоматическое поднятие

Пример: версия 2 с автоматическим поднятием

```
m :: MaybeT (ReaderT Integer IO) String
m = do
  n <- ask
  guard (n > 0)
  liftIO $ print n
  return "OK"

main = runReaderT (runMaybeT m) 2
```

- Функция `lift` вызывается автоматически (в зависимости от структуры стека монад).
- Функция `liftIO` всегда поднимает до внутренней монады `IO`.

Когда необходимо явное поднятие?

```
m :: StateT Integer (State String) ()
m = do
  n <- get
  s <- lift $ get
  return ()
```

```
m2 :: ReaderT Bool (StateT Integer (State String)) ()
m2 = do
  b <- ask
  n <- lift $ get
  s <- lift $ lift $ get
  return ()
```

- Необходимы явные обёртки.
- Слишком жёсткий расчёт на структуру стека монад.

Содержание

1. Монада ST
2. Комбинирование монад: простой пример
3. Преобразователи стандартных монад
4. Анатомия преобразователей монад

Монада Identity

```haskell
newtype Identity a = Identity { runIdentity :: a }

instance Functor Identity where
    fmap :: (a->b) -> Identity a -> Identity b
    fmap     = coerce

instance Applicative Identity where
    pure     = Identity
    (<*>) :: Identity (a->b) -> Identity a -> Identity b
    (<*>)    = coerce

instance Monad Identity where
    return   = Identity
    m >>= k  = k (runIdentity m)
```

```haskell
coerce :: Coercible a b => a -> b
```

Базовые классы-преобразователи

Класс MonadTrans

```
class MonadTrans t where
    lift :: (Monad m) => m a -> t m a
```

Класс MonadIO

```
class (Monad m) => MonadIO m where
    liftIO :: IO a -> m a

instance MonadIO IO where
    liftIO = id
```

Преобразователь MaybeT

```haskell
newtype MaybeT m a = MaybeT { runMaybeT :: m (Maybe a) }

instance MonadTrans MaybeT where
    lift :: (Monad m) => m a -> MaybeT m a
    lift = MaybeT . liftM Just

instance (MonadIO m) => MonadIO (MaybeT m) where
    liftIO = lift . liftIO
```

Преобразователь MaybeT создаёт монаду

```haskell
instance (Monad m) => Monad (MaybeT m) where
    return = lift . return
    x >>= f = MaybeT $ do
        v <- runMaybeT x
        case v of
            Nothing -> return Nothing
            Just y  -> runMaybeT (f y)
```

- В какой монаде работает код в блоке do?

Монада Writer и преобразователь WriterT

```
type Writer w = WriterT w Identity
```

```
newtype WriterT w m a = WriterT { runWriterT :: m (a, w) }
```

```
instance (Monoid w, Monad m) => Monad (WriterT w m) where
    return a = writer (a, mempty)
    m >>= k  = WriterT $ do
        (a, w)  <- runWriterT m
        (b, w') <- runWriterT (k a)
        return (b, w `mappend` w')
```

Монада Writer и преобразователь WriterT (2)

```
instance (Monoid w) => MonadTrans (WriterT w) where
    lift m = WriterT $ do
        a <- m
        return (a, mempty)

instance (Monoid w, MonadIO m) => MonadIO (WriterT w m) where
    liftIO = lift . liftIO
```

- Как сделать так, чтобы другие монады могли воспользоваться возможностями монады Writer?

Класс MonadWriter и его использование

```haskell
class (Monoid w, Monad m) => MonadWriter w m | m -> w where
    writer :: (a,w) -> m a
    tell   :: w -> m ()
    listen :: m a -> m (a, w)
    pass   :: m (a, w -> w) -> m a
```

```haskell
instance MonadWriter w m => MonadWriter w (MaybeT m) where
    writer = lift . writer
    tell   = ...
    listen = ...
    pass   = ...

instance MonadWriter w m => MonadWriter w (ReaderT r m) where
    ...
instance MonadWriter w m => MonadWriter w (StateT s m) where
    ...
```

Вернёмся к примеру

Пример: версия 2 с автоматическим поднятием

```
m :: MaybeT (ReaderT Integer IO) String
m = do
  n <- ask
  guard (n > 0)
  liftIO $ print n
  return "OK"
```

- Есть класс `MonadReader` с функцией `ask`.
- Есть экземпляр:

```
instance MonadReader r m => MonadReader r (MaybeT m) where
    ask    = lift ask
```

- Есть `MonadIO` для `MaybeT` и `ReaderT`, поэтому `liftIO` превращается в `lift.liftIO` и далее в lift.(lift.liftIO) = lift.lift.id.

Итоги

- Монада Identity.
- Монада есть преобразователь над Identity.
- Преобразователь создаёт обёртку над произвольной монадой, которая сама оказывается монадой, при этом bind базовой (новой) монады вызывает bind внутренней.
- Возможности монад из строящегося стека (классы MonadReader, MonadWriter и пр.) используются либо вручную с помощью `lift` из `MonadTrans`, либо автоматически с помощью экземпляров `MonadIO` и попарных реализаций вида

```
instance MonadWriter w m => MonadWriter w (ReaderT r m) where

instance (Monoid w, MonadReader r m) =>
                              MonadReader r (WriterT w m) where
```

Источник примера

1. Запрос пароля: Haskell/Monad transformers на wikibooks (`http://en.wikibooks.org/wiki/Haskell/Monad_transformers`)