Functional Programming in Haskell

The University of Glasgow

TP

- Your learning
- Wishlist
- Profile
- Settings
- Account
- Sign out

- To do
- Conversations
- Progress

Week 1

Haskell First Steps

Introduction

1.1

Welcome to the Course

6.11

6 more steps to go

Here be dragons

© Wim Vanderbauwhede

Example: the Maybe monad

1 comment

- We’ve already seen the Maybe type. Let’s look at the Maybe monad, which makes using the Maybe type a lot easier.

The *Maybe* type constructor

You already know the definition of the *Maybe* type:

```
data Maybe a = Just a | Nothing
```

Example use of Maybe: Safe *head* and *tail*

The *head* and *tail* functions from the Prelude are not safe in the sense that they fail when called on an empty list. We can define safe versions using *Maybe*:

```
myHead :: [a] -> Maybe a
myHead [] = Nothing
myHead (x:xs) = Just x

myTail :: [a] -> Maybe [a]
myTail [] = Nothing
myTail (x:xs) = Just xs
```

Monad instance of Maybe

Now we can make *Maybe* an instance of the *Monad* type class, simply by providing the appropriate definitions for *return*, *bind*, *then* and *fail*:

```
import Control.Monad

instance Monad Maybe where
    return         =   Just
    Nothing  >>= f = Nothing
    (Just x) >>= f = f x
    fail _         =   Nothing
```

There are a few additional functions defined in the *MonadPlus* type class:

```
instance MonadPlus Maybe where
    mzero               = Nothing
    Nothing `mplus` x = x
    x `mplus` _         = x
```

That’s it, we now have a *Maybe* monad!

Note: for users of `ghc 7.10` and higher, we need to do a little bit more work.

Explicit Maybe versus the Maybe Monad

Let’s see what this monad gives us:

A computation using explicit Maybe

```
foo :: [a] -> Maybe a
foo xs =
  case myTail xs of
    Nothing -> Nothing
    Just a -> case myTail a of
                Nothing -> Nothing
                Just b -> myHead b
```

To combine computations that use the *Maybe* type, we need explicit `case` expressions to pattern match against the type.

A computation using the Maybe monad

Let’s write this computation using the *Maybe* monad, first using the `>>=` operator:

```
bar :: [a] -> Maybe a
bar xs =
  myTail xs >>=
    (\a -> myTail a >>=
      (\b -> myHead b))
```

Now let’s change the line breaks and indentation a bit to make it look nicer:

```
bar2 :: [a] -> Maybe a
bar2 xs =
  myTail xs >>= (\a ->
  myTail a >>=  (\b ->
  myHead b))
```

Thanks to the associativity law, we can also remove unnecessary parentheses:

```
bar3 :: [a] -> Maybe a
bar3 xs =
  myTail xs >>= \a ->
  myTail a >>=  \b ->
  myHead b
```

This is already a lot cleaner, but finally we can use the `do`-notation:

```
bar3 :: [a] -> Maybe a
bar3 xs = do
  a <- myTail xs
  b <- myTail a
  myHead b
```

Clearly, the final monadic code is a lot more readable than the original non-monadic code.

Example: Reduction of bar [5,6]

```
    bar [5,6]

-- > substitute [5,6] for xs in definition of bar

    myTail [5,6] >>=
     (\a -> myTail a >>=
      (\b -> myHead b))

-- > def. myTail

    Just [6] >>=
     (\a -> myTail a >>=
      (\b -> myHead b))

-- >  def.2 of (>>=)
```

Video
2.4
Do it Yourself: Logical Thinking
Exercise
2.5
Nothing but the Truth
Quiz
Input and Output
2.6
Why I/O?
Video
2.7
Do it Yourself: Input/Output
Exercise
2.8
I/O and a First Encounter with Monads
Article
Installing GHC
2.9
Installing Haskell for Yourself
Article
2.10
How to Run GHCi
Video
2.11
Guessing Game
Article

```
    (\a -> myTail a >>=
     (\b -> myHead b))
       [6]
  -- > beta reduction, substitute [6] for a

    myTail [6] >>= (\b -> myHead b)

  -- > reduce myTail

    Just [] >>=  (\b -> myHead b)

  -- >  def.2 of (>>=)

   (\b -> myHead b) []

  -- > beta reduction, substitute [] for b

  myHead []

  -- > def.1 myHead
 Nothing
```

Example: Reduction of bar [5,6,7]

```
   bar [5,6,7]
  -- > substitute [5,6,7] for xs in definition of bar

   myTail [5,6,7] >>=
    (\a -> myTail a >>=
     (\b -> myHead b))

  -- > def. myTail

   Just [6,7]  >>=
    (\a -> myTail a >>=
     (\b -> myHead b))

  -- > def.2 of (>>=)

    (\a -> myTail a >>=
     (\b -> myHead b))
       [6,7]
  -- > beta reduction, substitute [6,7] for a

    myTail [6,7] >>= (\b -> myHead b)

  -- > reduce myTail

    Just [7] >>=  (\b -> myHead b)

  -- >  def.2 of (>>=)

   (\b -> myHead b) [7]

  -- > beta reduction, substitute [7] for b

   myHead [7]

  -- > def myHead

   Just 7
```

Write a Spelling Book Generator

Exercise

3.8

Summary

Article

3.9

Define Your Own Data Types

Video

3.10

3.11

Type Classes

Video

3.12

Interview with Simon Peyton Jones

Video

3.13

Brief History of Haskell

Article

3.14

Course Feedback

Article

3.15

End of Week 3

Share this article:

-
-
-
-

1 comment

Previous

Mark as complete

Next

Mark as complete

Comments

YP

Tomas P.

Add a comment... (plain text and markdown available)

0/1200

Post Learn more about markdown

Show: All comments

Sort by: Newest

Filter

- GY
  Gwee YC
  Follow
  05 NOV05 NOV
  Why MonadPlus is necessary for the a Monad is not clear? How is MonadPlus applied? Great if we can have some illustrative examples here.
  (edited)
  Like
  Reply
  Bookmark
  Report

Haskell in the Wild

Article

5.13

Course Feedback

Article

Week 6

Think like a Functional Programmer

Type Classes

6.1

Welcome to Week 6

Video

6.2

Types with Class

Video

6.3

Type Classes in more Detail

Article

6.4

Summary

Article

Geek Greek

6.5

Introduction to the Lambda calculus

Article

6.6

There are Only Functions! (Optional)

Video