Object Meets Function

Monad

Steven Lolong

steven.lolong(at)uni-tuebingen.de

Programming Language Research Group
Tuebingen University

Jan., 11 2023

Monad

Eberhard Karls
Universität
Tübingen

Resource

Good resources for this topic:

Intro. to Monad `https://ps-tuebingen-courses.github.io/pl1-lecture-notes/20-monads-intro/monads-intro.html`

Monad in Picture `https://www.adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html#monads`

Monad (SPOOKY? No.) Haskell Programming from First Principles (book).

Composing functions

```
1 def f(i: Int) : String = i.toString()
2 def g(s: String) : Boolean = s == "7"
3 def h(b: Boolean) : Int = if b then 7 else sys.error("Other
      than 7")
4
5 // h after ! g after f(8)
6 def clientCode = h(!g(f(8)))
```

Listing 1: Composing function – PL1's lecture

Composing functions

```
11 def fOp(i: Int): Option[String] = if (i < 100) Some(i.
       toString()) else None
12 def gOp(s: String): Option[Boolean] = Some(s == "7")
13 def hOp(b: Boolean): Option[Int] = if (b) Some(7) else None
```

Listing 2: Composing function with Option – PL1's Lecture

How about the client code, do we need to change it?

```
6 def clientCode = h(!g(f(8)))
```

Listing 3: Composing function with Option – PL1's Lecture

Composing functions

```
1 def clientCodeOp =
2   fOp(8) match
3     case Some(x) => gOp(x) match
4         case Some(y) => hOp(!y)
5         case None => None
6     case None => None
```

Listing 4: Composing function with Option – PL1's Lecture

Composing functions

Add a new bindingFunction

```
21 def bindOption[A, B](a: Option[A], f: A => Option[B]):
       Option[B] = a match {
22   case Some(x) => f(x)
23   case None => None
```

Listing 5: Composing function with Option – PL1's Lecture

How about the client code, do we need to change it?

```
13 def clientCodeOp =
14   fOp(8) match
15     case Some(x) => gOp(x) match
16         case Some(y) => hOp(!y)
17         case None => None
18     case None => None
```

Listing 6: Composing function with Option – PL1's Lecture

Composing functions

The new client code

```
26 def clientCodeOpBind =
27   bindOption(fOp(27), (x: String) =>
28     bindOption(gOp(x + "z"), (y: Boolean) =>
29       hOp(!y)))
```

Listing 7: Composing function with Option – PL1's Lecture

Monad

Monad $\cong$ Compose functions

Monad

Monad laws:

- "unit" acts as a kind of neutral element of "bind", ex.: $bind(unit(x), f) == f(x)$ and $bind(x, y => unit(y)) == x$
- Bind enjoys an associative property
  bind(bind(x, f), g) == bind(x, y => bind(f(y), g))

Monad Interface

```
1 trait Monad[M[_]]:
2   def unit[A](a: A): M[A]
3   def bind[A, B](m: M[A], f: A => M[B]): M[B]
4 end Monad
```

Listing 8: Monad interface

Client code

How about the client code?

Client code

```
1 def clientCode2Op(m: Monad[Option]) = m.bind(fOp(27), (x:
      String) =>
2   m.bind(gOp(x + "z"), (y: Boolean) => m.unit(!y)))
```

Listing 9: Client code

The Option Monad

Option Monad is a monad to compose functions with Option type for the function's parameter and return's type.

```
1 object OptionMonad extends Monad[Option]:
2   override def bind[A, B](a: Option[A], f: A => Option[B]):
      Option[B] =
3     a match {
4       case Some(x) => f(x)
5       case None => None
6     }
7   override def unit[A](a: A) = Some(a)
8 end OptionMonad
```

Listing 10: Option Monad

```
1 def v: Option[Boolean] = clientCode2Op(OptionMonad)
```

Listing 11: Application of Clientcode over option monad

The Identity Monad

Application of identity over a function. When it parameterized the monadic code with the identity monad, it will return the behavior of the original no-monadic code.

```
1 type Id[X] = X
2
3 object IdMonad extends Monad[Id]:
4     override def unit[A](a: A): Id[A] = a
5     override def bind[A, B](m: Id[A], f: A => Id[B]): Id[B]
      = f(m)
6 end IdMonad
```

Listing 12: The Identity Monad

```
1 def fId(i: Int) = i
2 def idMonadTest(id: Monad[Id]) = id.bind(3, fId)
```

Listing 13: Applicaton of Identity Monad

The Reader Monad

The Reader Monad a.k.a environment monad. It captures the essence of environment passing style.

```
1 trait ReaderMonad[R] extends Monad[[A] =>> R => A]:
2   override def unit[A](a: A): R => A = (_) => a
3   override def bind[A, B](m: R => A, f: A => R => B): R => B
      = r => f(m(r))(r)
4 end ReaderMonad
```

Listing 14: The Reader Monad

Eberhard Karls
Universität
Tübingen

The Reader Monad

Functions with environment passing style.
Case: Every function will use the same data from environment.

```
1  def fRead(n: Int) : Int => String = (x: Int) =>
2    print("I will use this env: " + x.toString() + " for some
       purposes \n")
3    n.toString()
4
5  def gRead(s: String) : Int => Boolean = (x: Int) =>
6    print("You can use this env: " + x.toString() + " for some
       purposes \n")
7    s == "7"
8
9  def hRead(b: Boolean) : Int => Int = (x: Int) =>
10   print("X: " + x.toString() + " is env data passing over
       functions \n")
11   if b then 7 + x else sys.error("Other than 14")
```

Listing 15: Environt passing stype

The Reader Monad

Flashback to function composition.

```
1 def f(i: Int) : String = i.toString()
2 def g(s: String) : Boolean = s == "7"
3 def h(b: Boolean) : Int = if b then 7 else sys.error("Other
      than 7")
4
5 // h after ! g after f(8)
6 def clientCode = h(!g(f(8)))
```

Listing 16: Function composition

Can we do it like this?

```
1 // def clientCode = h(!g(f(8)))
2 // def clientCodeNewFunc(env: Int) = hRead(!gRead((fRead(8))
      ))
```

Listing 17: Function composition with environment

The Reader Monad

The answer is NO. We must do it like this:

```
1 def clientCodeNewFunc(env: Int) = hRead(!gRead((fRead(8)(env
      )))(env))(env)
2
3 clientCodeNewFunc(3)
```

Listing 18: Function composition with environment

How to do it using ReaderMonad?

The Reader Monad

How to do it using ReaderMonad?

```
1 object ReaderMonad extends ReaderMonad[Int]
2
3 def clientCodeRM(using rm: ReaderMonad[Int]) = (env: Int) =>
4     rm.bind(fRead(8),
5       (s: String) => rm.bind(gRead(s), (b: Boolean) => hRead
      (!b))) (env)
6
7 clientCodeRM(using ReaderMonad)(3)
```

Listing 19: Function composition with environment using ReaderMonad

Eberhard Karls
Universität
Tübingen