Object Meets Function

Monad

Steven Lolong

steven.lolong(at)uni-tuebingen.de

Programming Language Research Group
Tuebingen University

Jan., 11 2023

Monad

1 Introduction

2 Monad

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

```
def f(i: Int) : String = i.toString()
def g(s: String) : Boolean = s == "7"
def h(b: Boolean) : Int = if b then 7 else sys.error("Other
    than 7")

// h after ! g after f(8)
def clientCode = h(!g(f(8)))
```

Listing: Composing function – PL1's lecture

Composing functions

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

Listing: Composing function with Option – PL1's Lecture

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

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

Listing: Composing function with Option – PL1's Lecture

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Composing functions

```
def clientCodeOp =
  fOp(8) match
    case Some(x) => gOp(x) match
        case Some(y) => hOp(!y)
        case None => None
    case None => None
```

Listing: Composing function with Option – PL1's Lecture

Composing functions

Add a new bindingFunction

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

Listing: Composing function with Option – PL1's Lecture

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

```
def clientCodeOp =
  fOp(8) match
    case Some(x) => gOp(x) match
        case Some(y) => hOp(!y)
        case None => None
    case None => None
```

Listing: Composing function with Option – PL1's Lecture

Composing functions

The new client code

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

Listing: 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

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

Listing: Monad interface

Client code

How about the client code?

Client code

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

Listing: Monad interface