Monadic Design Patterns for the Web

Episode IV - Code Talks!

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Outline

- Inheritance and reuse
- The Monadic API as a view
- Enough structure to support multiple views
- Conway Games
- Poking a hole in Conway
- Reuse again?

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Acknowledgement

Scala is a BIG tent. While everybody else is scaling the web and DSLing, lgm is abstracting Conway games over arbitrary monads.

James Iry, recent tweet

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Inheritance and reuse

- What does reuse really mean?
- Proposal -- we can reuse code just when we know we can substitute that code for other code that "does the same thing"
- Types give us a first approximation of "does the same thing" -- but in many cases are not enough -- we usually need some additional context information

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Inheritance and reuse

- For example, we could say that Sets *are* functions from the Universe from which they are drawn to Boolean
- In code: Set[A] ≈ A => Boolean
- But this actually presumes a great deal about what we mean by function, Set and Universe

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Inheritance and reuse

- In a world of higher-order functions it is easy to reify our assumptions:

```
trait SetsCanBeFunction[S,T] {
    def asFunction( s : Set[S] ) : S => T
}
trait SetsCanBeTotalFunctions[S]
    extends SetsCanBeFunctions[S,Boolean]
```

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Inheritance and reuse

- But also

trait SetsCanBeFuzzy[S]

extends SetsCanBeFunctions[S,Double]

trait SetsCanBePartialFunctions[S]

extends SetsCanBeFunctions[S,Option[Boolean]]

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Inheritance and reuse

- Is-a relationships are more often than not meaningful or applicable *in a context*
- With the higher-order approach, we can easily reify the context and have a representative of our assumptions
- Inheritance sublimates the context of an is-a relationship, eliminating the representative of our assumptions

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

The Monadic API as a View

- The view from here is pretty nice!
- Structures like List, Set, Tree, ... are *not* monads -- but they can be viewed as having monadic structure

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

The Monadic API as a View

shape

wrap

```
trait Monad[M[_]] {

  def unit [A] ( a : A ) : M[A]

  def bind [A,B] ( ma : M[A], fn : A => M[B] ) : M[B]

}
```

jelly roll

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

The Monadic API as a View

```
class ListM[A] extends Monad[List] {

  override def unit [S] ( s : S ) : List[S] = { List[S]( e ) }

  override def bind [S,T] ( ls : List[S], fn : S => List[T] ) = {

    ( ( Nil : List[T] ) /: ls )( { ( acc, e ) => { acc ++ f( e ) } } )

  }

}
```

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

The Monadic API as a View

- With these containers -- together with the constraints like functoriality -- it's harder to see how there might be different interpretations of the monadic API -- it's all so canonical!

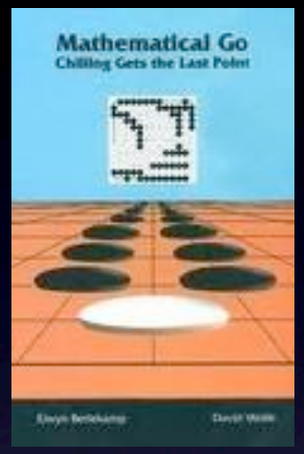
Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Enough structure to support multiple views

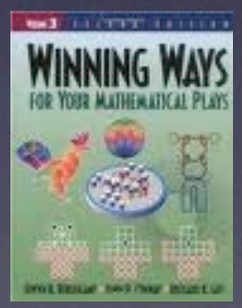
- A container with components that are containers is just the right sort of device to illustrate this point

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Conway Games



- The 0 game -- i offer you the first move...



Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Conway Games

- A game captures the options of Player and Opponent
- An option is a move to a new position in which Player and Opponent have new options

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Conway Games

Actually, these are the droids you're looking for!

```
trait ConwayGame {
    def left : Set[ConwayGame]
    def right : Set[ConwayGame]
}
```

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Conway Games

```
case object EmptyGame
      extends ConwayGame {
   override def left : Set[ConwayGame] =
   Set.empty

   override def right : Set[ConwayGame] =
   Set.empty

}
```

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Conway Games

```
case class Game(
    override val left : Set[ConwayGame],
    override val right : Set[ConwayGame]
) extends ConwayGame
```

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Conway Games

```
val cgZero = EmptyGame

val cgOne = Game( Set( cgZero ), Set.empty )

val cgTwo = Game( Set( cgOne ), Set.empty )

...

val cgMinusOne = Game( Set.empty, Set( cgZero ) )

...

 val cgOneHalf = Game( Set( cgOne ), Set( cgZero ) )
```

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Conway Games

```
trait Calculator {

    def minus( g : ConwayGame ) : ConwayGame = {

        g match {

            case EmptyGame => EmptyGame

            case Game( gL, gR ) =>

            Game( gR.map( minus ), gL.map( minus ) )

        }

    }
```

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Conway Games

def add(g1 : ConwayGame, g2 : ConwayGame) : ConwayGame

def multiply(g1 : ConwayGame, g2 : ConwayGame) : ConwayGame

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Conway's notation

$0 = \{ \mid \}$

$1 = \{ 0 \mid \} = \{ \{ \mid \} \mid \}$

$2 = \{ 1 \mid \} = \{ \{ \{ \mid \} \mid \} \mid \}$

...

$G = \{ G^L, \ldots \mid G^R, \ldots \}$

$-G := \{ -G^R \mid -G^L \}$

$G1 + G2 := \{ G1^L + G2, G2^L + G1 \mid G1^R + G2, G2^R + G1 \}$

Can you say 'DSL'?
i knew you could!

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Poking (a hole in) Conway

- Conway Games are as pure as Sets
- Ordinary Set Theory offers no place to put data! Sets are either empty or contain other Sets.
- The same is true for ConwayGames

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Poking (a hole in) Conway

- Just as we understand intuitively what we mean by Set[A] ...
- We can have an intuitive understanding of ConwayGame[A]
  - Yet, when it comes to arithmetic, we have to proceed with caution!

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Poking (a hole in) Conway

```
trait GenConwayGame[A] {
    left : Set[Either[A,GenConwayGame[A]]]
    right : Set[Either[A,GenConwayGame[A]]]
}
```

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Poking (a hole in) Conway

- When we negate, add or multiply these widgets we get into situations where we are asked to combine A's, and GenConwayGame[A]'s.
- We defer and reify! This is the basic power of monads!

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Poking (a hole in) Conway

- From a the point of view of the Monadic API we have two containers -- left and right -- that would support monadic structure
- We have two views!
    - unit wraps A's into left component
    - unit wraps A's into right component
    - bind is forced to do the same thing in either view

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Poking (a hole in) Conway

- This kind of generalization and the ease with which we make it is one of the real contributions that Computer Science makes back to society
- It invites even better questions, for look:

```
trait GenConwayGame[M[_],A] {
    left : M[Either[A,GenConwayGame[A]]]
    right : M[Either[A,GenConwayGame[A]]]
}
```

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Poking (a hole in) Conway

- It turns out our calculator code works as is with this structure!
- This means we can build notions of quantity as rich as the Field of the Reals over arbitrary monads!

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Reuse again?

So, i took it apart and put it back together, but i'm not sure what to do with these?

- In Scala Set[A] inherits from A => Boolean
- This means that the variance constraints of List[A] and Set[A] differ!
- This is not consistent!
- It meant i had to write my own Set for this talk

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

What does this have to do with Monadic Design Patterns for the Web?

- Scaling the web isn't just about volume or load or sheer numbers
- Scaling the web is as much about complexity
- You can have the most scalable architecture ever invented, but if nobody can program to it, it's not useful

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC

Location, location, location!

- Conway's construction actually gives us a generic notion of location
- Once we see it from the Monadic vantage point we see that it can be compared with Huet's proposal for location

Lucius Gregory Meredith, Managing Partner, Biosimiliarity LLC