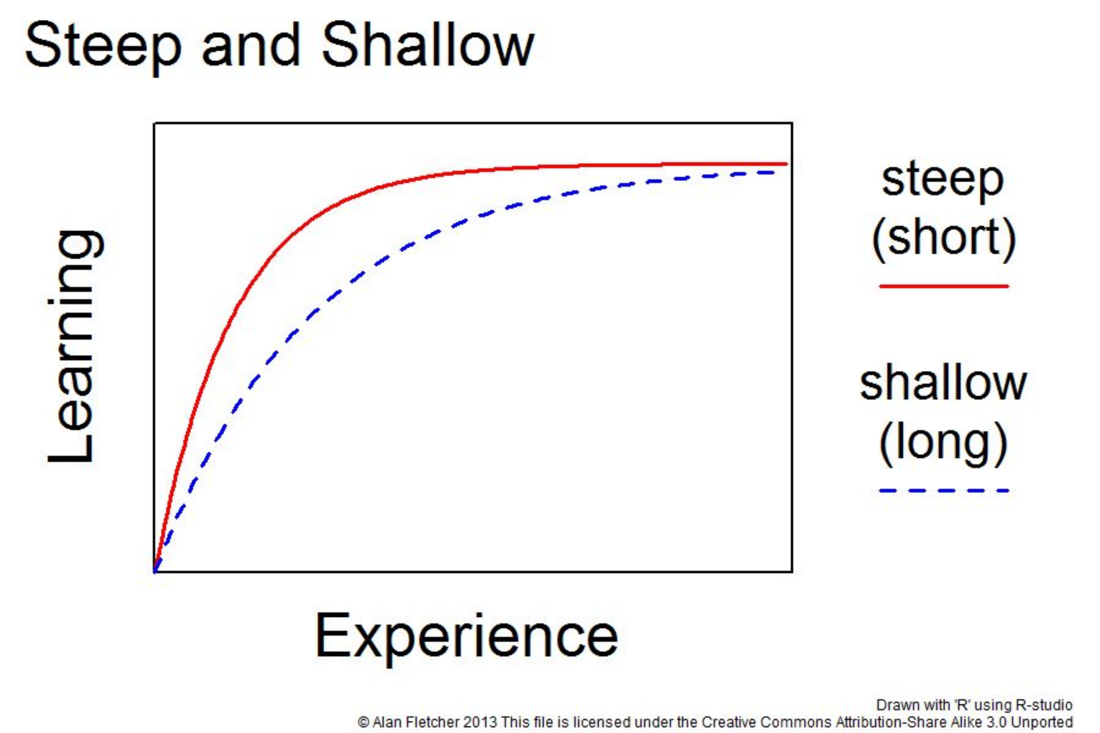
Mi Primer Validador Funcional (Effect agnostic)

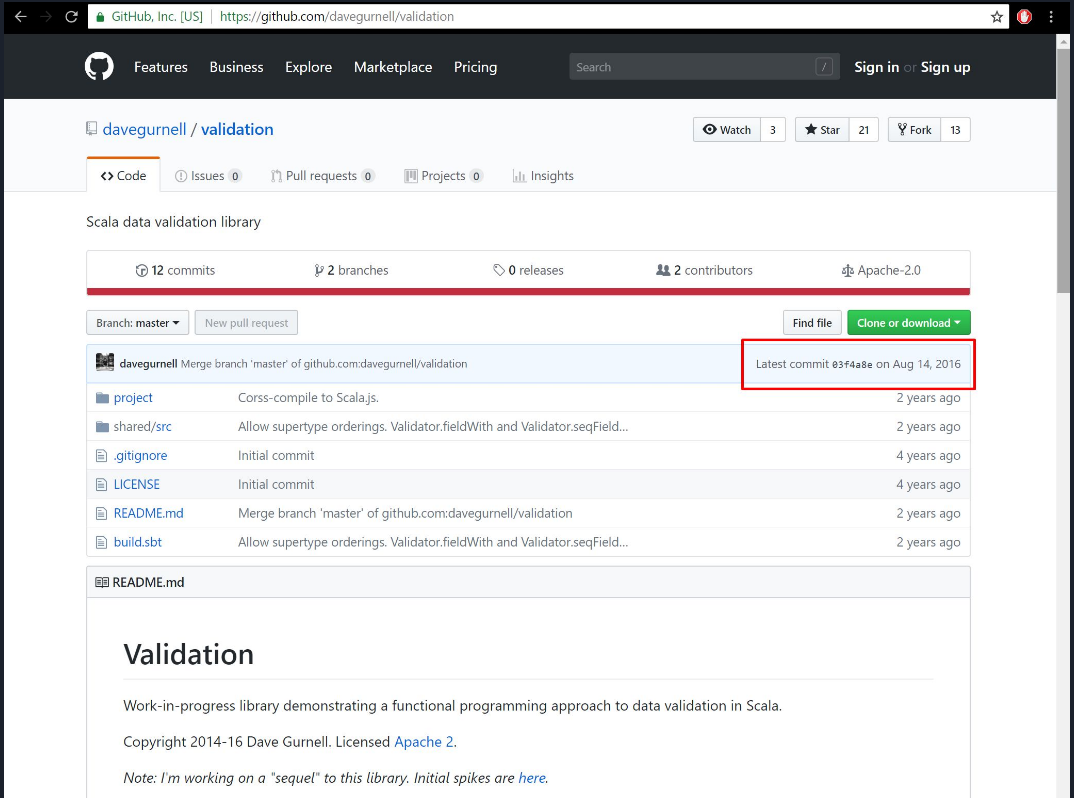
Raúl Piaggio - ScaLatam 2019 - 02/05/19

La larga (y no empinada) curva de aprendizaje de Scala



Drawn with 'R' using R-studio

Librería base: io.underscore.validation



davegurnell / validation

Scala data validation library

12 commits | 2 branches | 0 releases | 2 contributors | Apache-2.0

davegurnell Merge branch 'master' of github.com:davegurnell/validation — Latest commit 03f4a8e on Aug 14, 2016

project	Corss-compile to Scala.js.	2 years ago
shared/src	Allow supertype orderings. Validator.fieldWith and Validator.seqField...	2 years ago
.gitignore	Initial commit	4 years ago
LICENSE	Initial commit	4 years ago
README.md	Merge branch 'master' of github.com:davegurnell/validation	2 years ago
build.sbt	Allow supertype orderings. Validator.fieldWith and Validator.seqField...	2 years ago

README.md

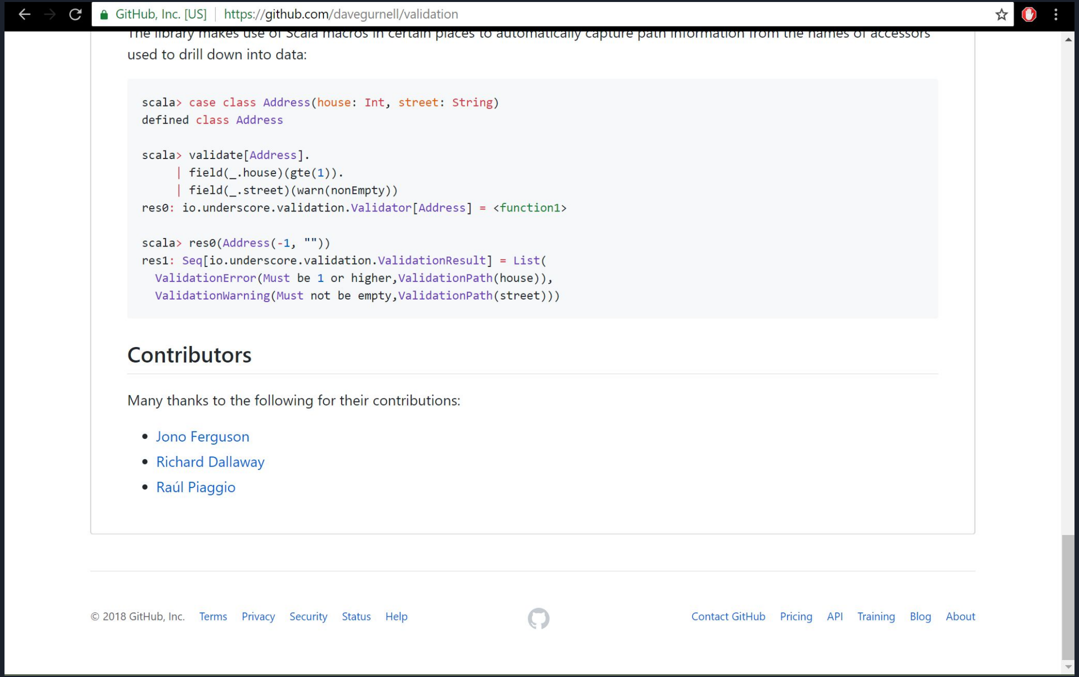
Validation

Work-in-progress library demonstrating a functional programming approach to data validation in Scala.

Copyright 2014-16 Dave Gurnell. Licensed Apache 2.

Note: I'm working on a "sequel" to this library. Initial spikes are here.

Librería base: io.underscore.validation



The library makes use of Scala macros in certain places to automatically capture path information from the names of accessors used to drill down into data:

```
scala> case class Address(house: Int, street: String)
defined class Address

scala> validate[Address].
     | field(_.house)(gte(1)).
     | field(_.street)(warn(nonEmpty))
res0: io.underscore.validation.Validator[Address] = <function1>

scala> res0(Address(-1, ""))
res1: Seq[io.underscore.validation.ValidationResult] = List(
  ValidationError(Must be 1 or higher,ValidationPath(house)),
  ValidationWarning(Must not be empty,ValidationPath(street)))
```

Contributors

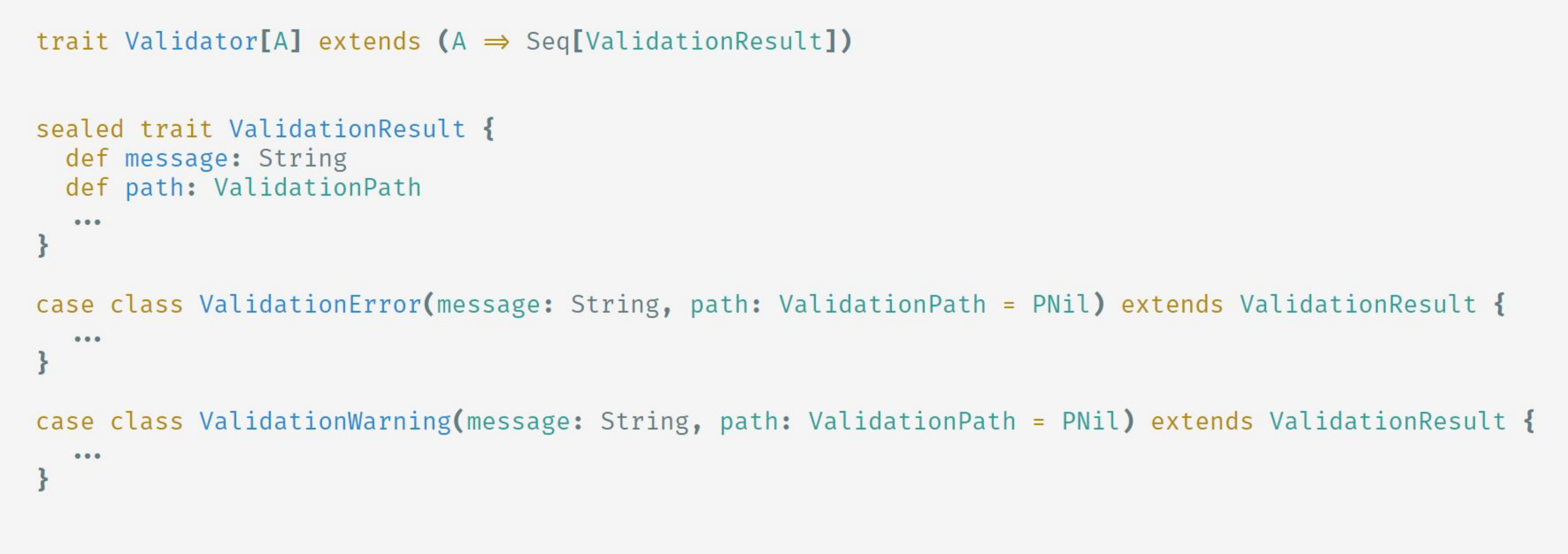
Many thanks to the following for their contributions:

- Jono Ferguson
- Richard Dallaway
- Raúl Piaggio

© 2018 GitHub, Inc. Terms Privacy Security Status Help

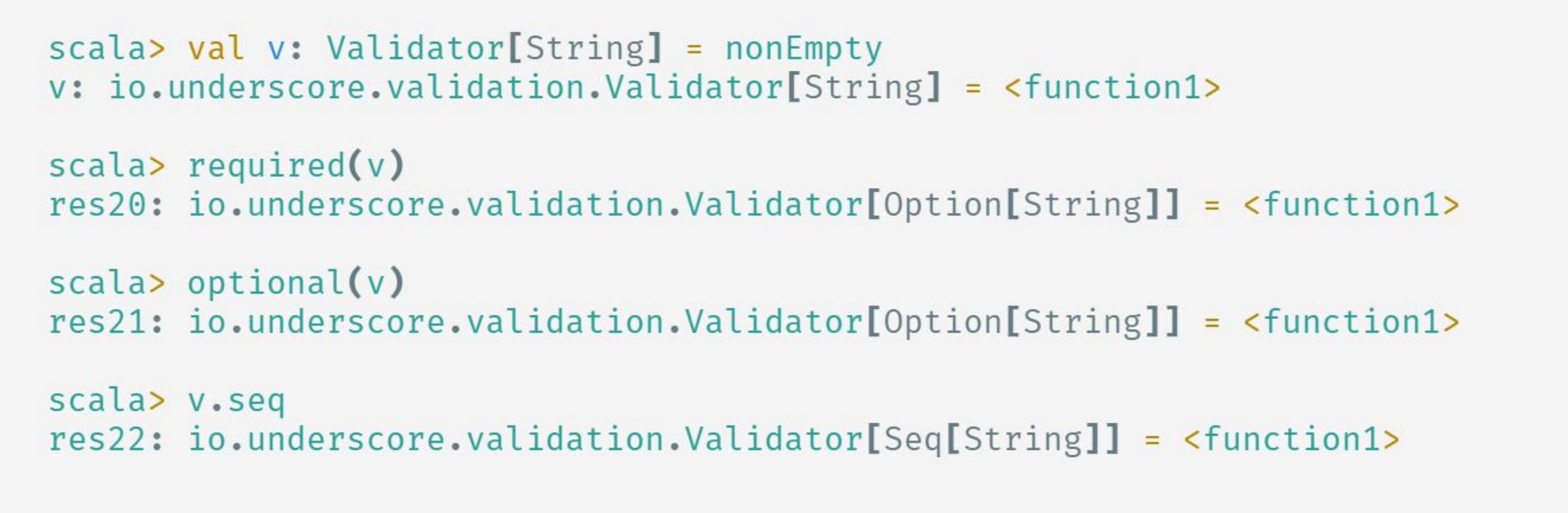
Contact GitHub Pricing API Training Blog About

Librería base: io.underscore.validation



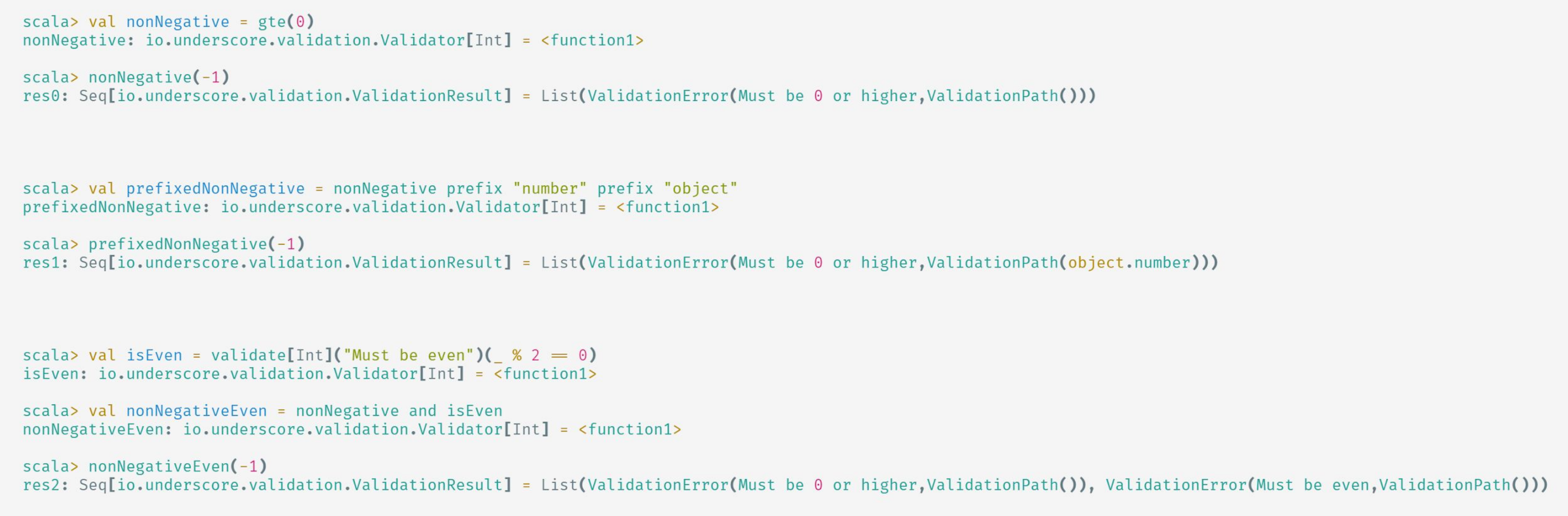
```scala
trait Validator[A] extends (A ⇒ Seq[ValidationResult])


sealed trait ValidationResult {
  def message: String
  def path: ValidationPath
   ...
}

case class ValidationError(message: String, path: ValidationPath = PNil) extends ValidationResult {
   ...
}

case class ValidationWarning(message: String, path: ValidationPath = PNil) extends ValidationResult {
   ...
}
```

Librería base: io.underscore.validation DSL



```
scala> val v: Validator[String] = nonEmpty
v: io.underscore.validation.Validator[String] = <function1>

scala> required(v)
res20: io.underscore.validation.Validator[Option[String]] = <function1>

scala> optional(v)
res21: io.underscore.validation.Validator[Option[String]] = <function1>

scala> v.seq
res22: io.underscore.validation.Validator[Seq[String]] = <function1>
```

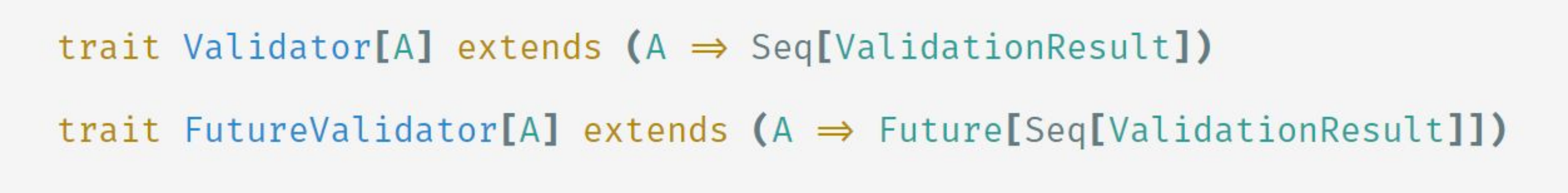
Librería base: io.underscore.validation DSL



```
scala> val nonNegative = gte(0)
nonNegative: io.underscore.validation.Validator[Int] = <function1>

scala> nonNegative(-1)
res0: Seq[io.underscore.validation.ValidationResult] = List(ValidationError(Must be 0 or higher,ValidationPath()))



scala> val prefixedNonNegative = nonNegative prefix "number" prefix "object"
prefixedNonNegative: io.underscore.validation.Validator[Int] = <function1>

scala> prefixedNonNegative(-1)
res1: Seq[io.underscore.validation.ValidationResult] = List(ValidationError(Must be 0 or higher,ValidationPath(object.number)))



scala> val isEven = validate[Int]("Must be even")(_ % 2 == 0)
isEven: io.underscore.validation.Validator[Int] = <function1>

scala> val nonNegativeEven = nonNegative and isEven
nonNegativeEven: io.underscore.validation.Validator[Int] = <function1>

scala> nonNegativeEven(-1)
res2: Seq[io.underscore.validation.ValidationResult] = List(ValidationError(Must be 0 or higher,ValidationPath()), ValidationError(Must be even,ValidationPath()))
```

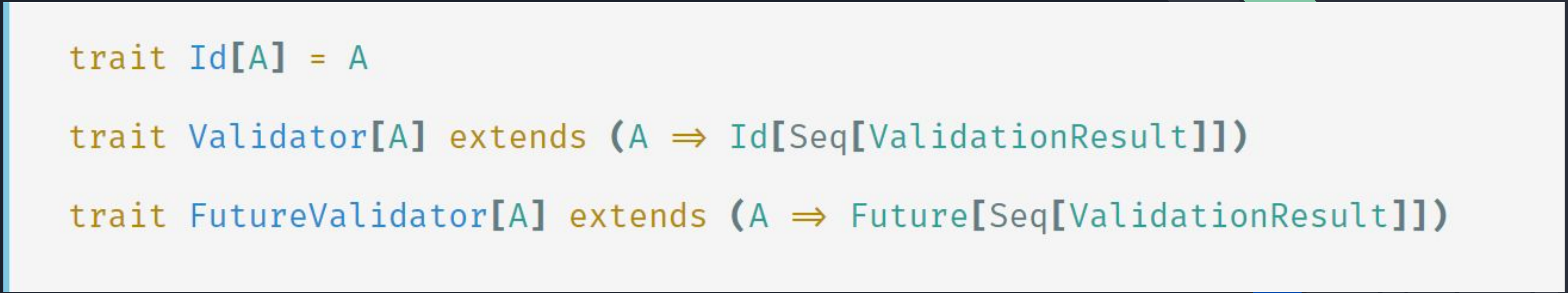
Preguntas hasta acá?

Dilema



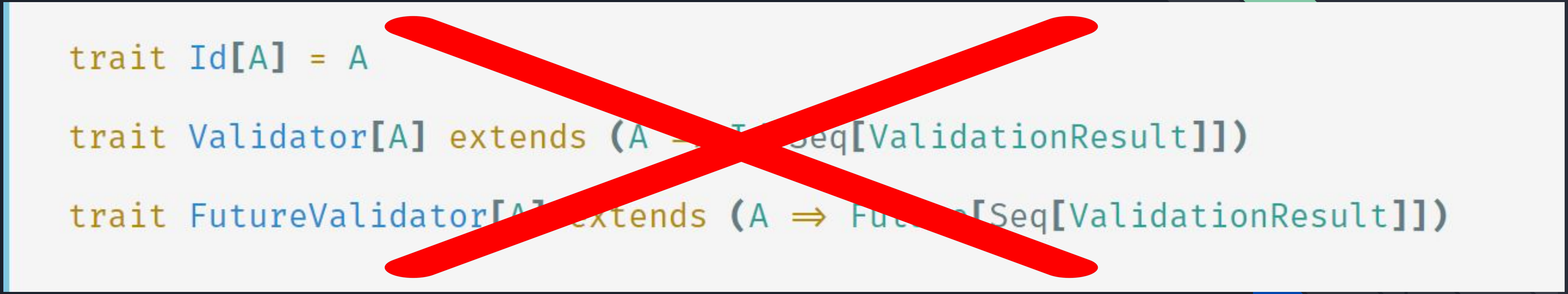
```scala
trait Validator[A] extends (A ⇒ Seq[ValidationResult])

trait FutureValidator[A] extends (A ⇒ Future[Seq[ValidationResult]])
```

Aha! Id Type

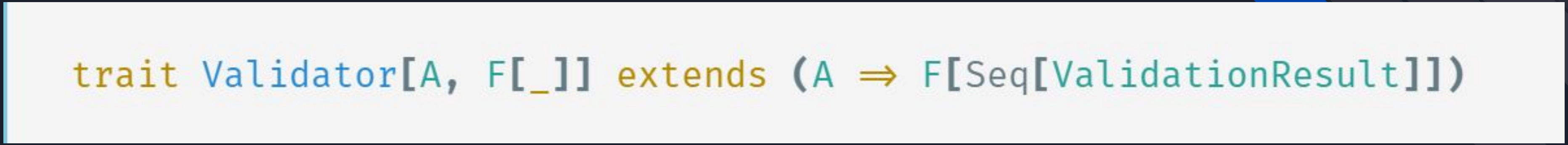


```
trait Id[A] = A

trait Validator[A] extends (A ⇒ Id[Seq[ValidationResult]])

trait FutureValidator[A] extends (A ⇒ Future[Seq[ValidationResult]])
```

Aha! Id Type



```
trait Id[A] = A

trait Validator[A] extends (A ⇒ Try[Seq[ValidationResult]])

trait FutureValidator[A] extends (A ⇒ Future[Seq[ValidationResult]])
```



```
trait Validator[A, F[_]] extends (A ⇒ F[Seq[ValidationResult]])
```

Objetivos

01 Seguir usando código existente sin modificaciones (o modificaciones mínimas).

02 Poder combinar contextos fácilmente.

03 Inferir contexto resultante.

Inspiración

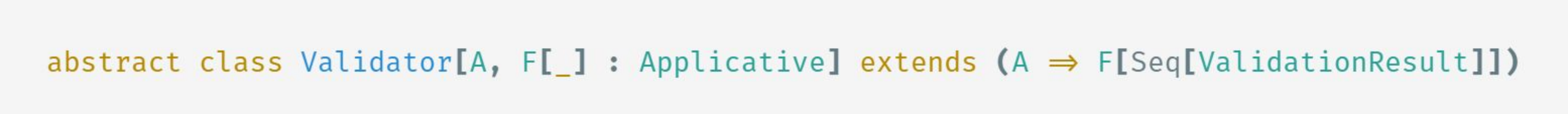
- Abstraer contexto:
    - Tagless Final
    - Free Monads
    - Doobie

- Scala Meetup Montevideo
    - Natural Transformations: F[_] ~> G[_]

Cómo generalizamos?

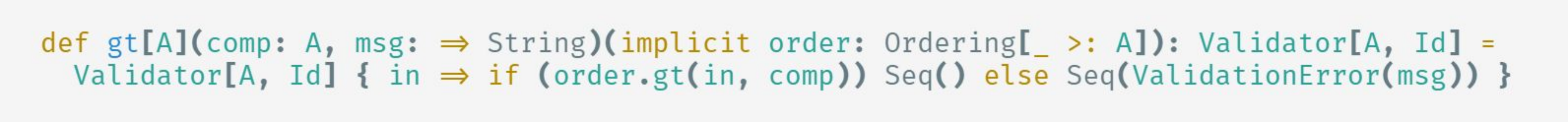
- Functor? Monad? Applicative!
- Múltiples evaluaciones **independientes.**
- Paralelismo!
- Qué obtenemos del contexto?
    - map
    - pure
    - traverse/compose

Primeros pasos



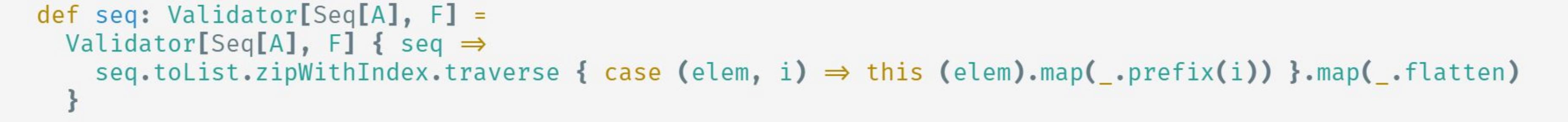
```
abstract class Validator[A, F[_] : Applicative] extends (A ⇒ F[Seq[ValidationResult]])
```

- Todos los “generadores base” existentes ahora validan en contexto Id[_]. Ejemplo:



```
def gt[A](comp: A, msg: ⇒ String)(implicit order: Ordering[_ >: A]): Validator[A, Id] =
  Validator[A, Id] { in ⇒ if (order.gt(in, comp)) Seq() else Seq(ValidationError(msg)) }
```

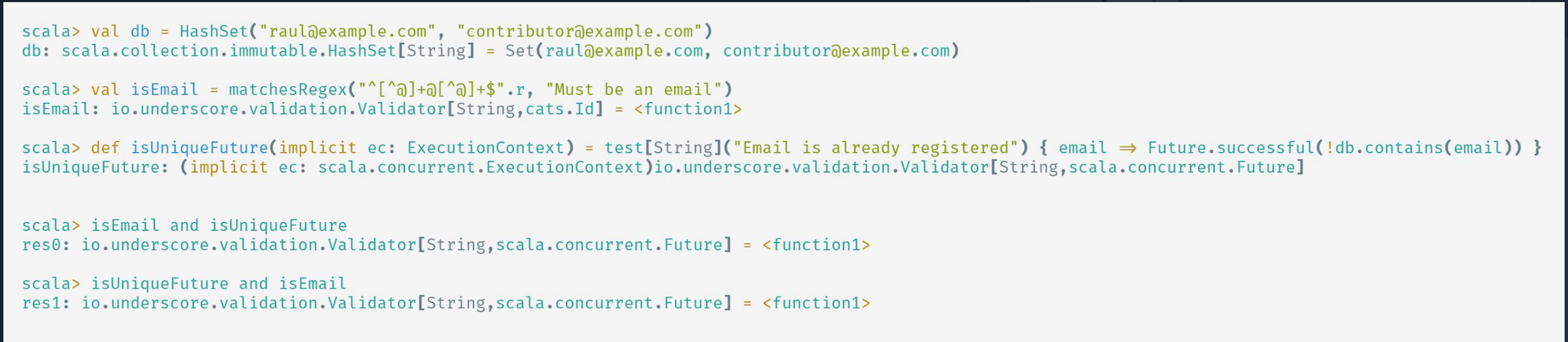
- Todos los “modificadores” existentes respetan el contexto. Ejemplo:



```
def seq: Validator[Seq[A], F] =
  Validator[Seq[A], F] { seq ⇒
    seq.toList.zipWithIndex.traverse { case (elem, i) ⇒ this (elem).map(_.prefix(i)) }.map(_.flatten)
  }
```

Y qué hacemos con “and” ???

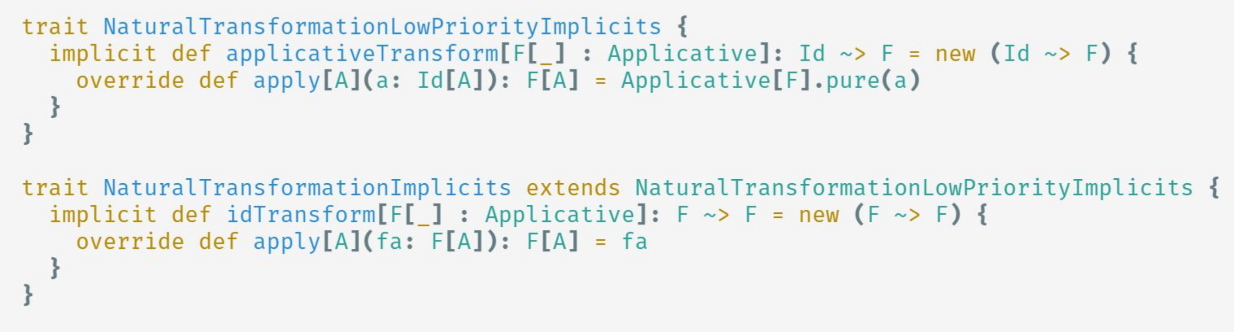
Comportamiento deseado



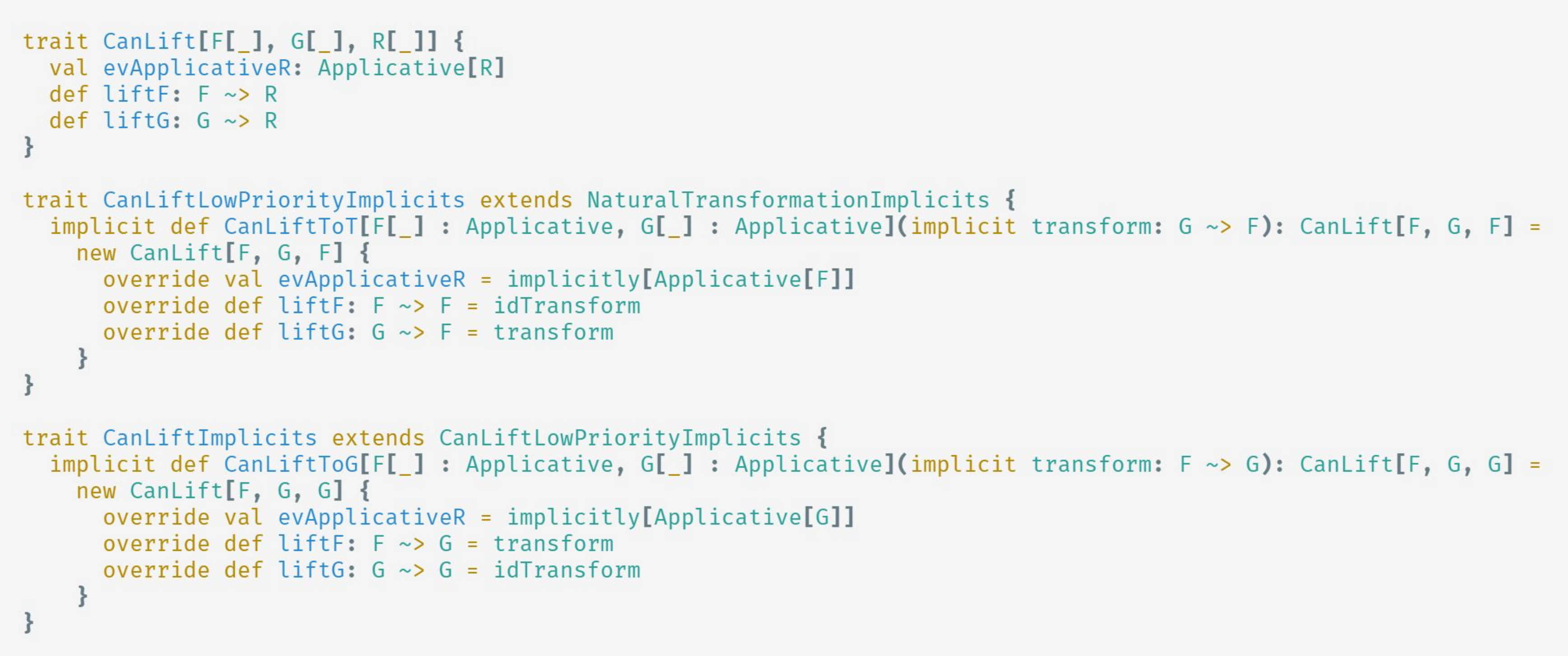
```
scala> val db = HashSet("raul@example.com", "contributor@example.com")
db: scala.collection.immutable.HashSet[String] = Set(raul@example.com, contributor@example.com)

scala> val isEmail = matchesRegex("^[^@]+@[^@]+$".r, "Must be an email")
isEmail: io.underscore.validation.Validator[String,cats.Id] = <function1>

scala> def isUniqueFuture(implicit ec: ExecutionContext) = test[String]("Email is already registered") { email => Future.successful(!db.contains(email)) }
isUniqueFuture: (implicit ec: scala.concurrent.ExecutionContext)io.underscore.validation.Validator[String,scala.concurrent.Future]


scala> isEmail and isUniqueFuture
res0: io.underscore.validation.Validator[String,scala.concurrent.Future] = <function1>

scala> isUniqueFuture and isEmail
res1: io.underscore.validation.Validator[String,scala.concurrent.Future] = <function1>
```

Más o menos se entiende hasta acá?

Solución

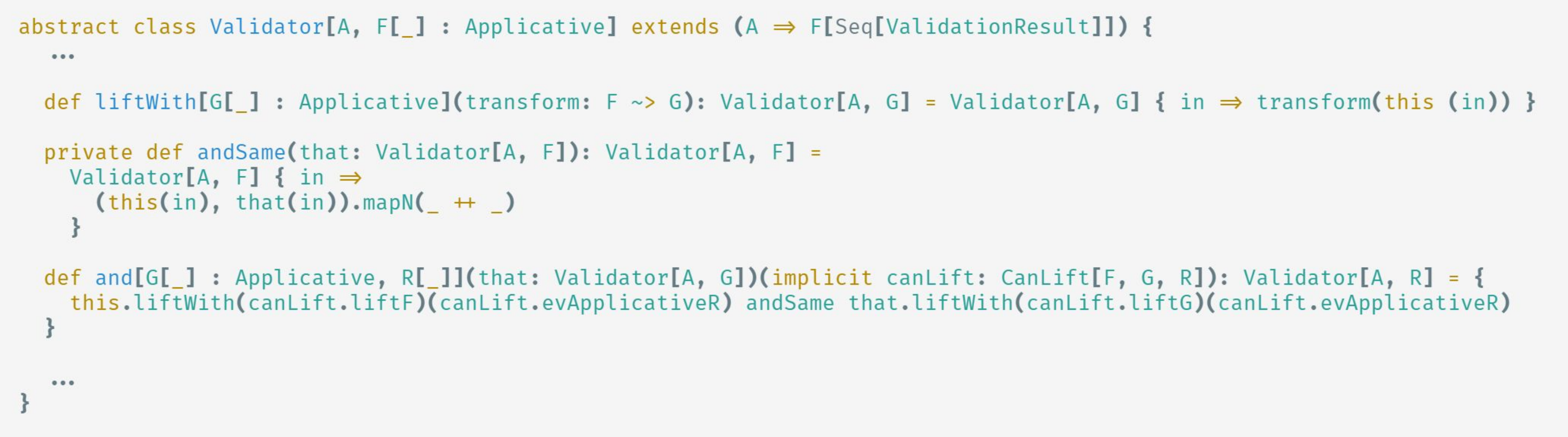


```scala
trait NaturalTransformationLowPriorityImplicits {
  implicit def applicativeTransform[F[_] : Applicative]: Id ~> F = new (Id ~> F) {
    override def apply[A](a: Id[A]): F[A] = Applicative[F].pure(a)
  }
}

trait NaturalTransformationImplicits extends NaturalTransformationLowPriorityImplicits {
  implicit def idTransform[F[_] : Applicative]: F ~> F = new (F ~> F) {
    override def apply[A](fa: F[A]): F[A] = fa
  }
}
```



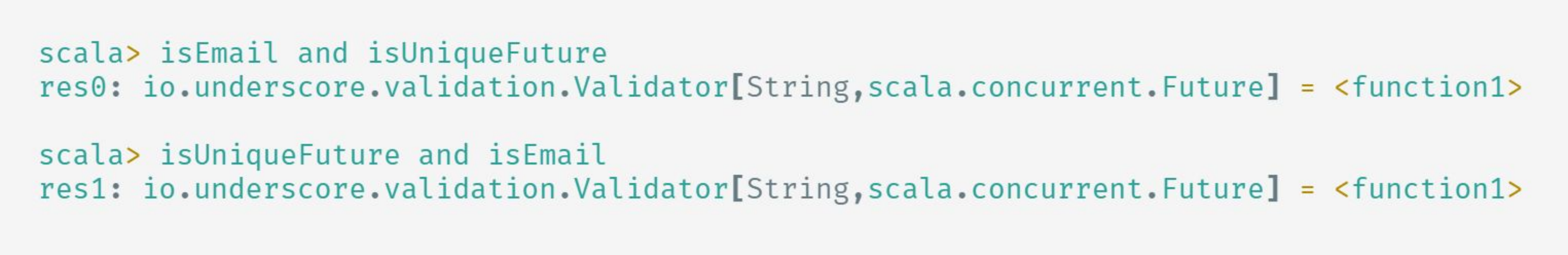
```scala
trait CanLift[F[_], G[_], R[_]] {
  val evApplicativeR: Applicative[R]
  def liftF: F ~> R
  def liftG: G ~> R
}

trait CanLiftLowPriorityImplicits extends NaturalTransformationImplicits {
  implicit def CanLiftToT[F[_] : Applicative, G[_] : Applicative](implicit transform: G ~> F): CanLift[F, G, F] =
    new CanLift[F, G, F] {
      override val evApplicativeR = implicitly[Applicative[F]]
      override def liftF: F ~> F = idTransform
      override def liftG: G ~> F = transform
    }
}

trait CanLiftImplicits extends CanLiftLowPriorityImplicits {
  implicit def CanLiftToG[F[_] : Applicative, G[_] : Applicative](implicit transform: F ~> G): CanLift[F, G, G] =
    new CanLift[F, G, G] {
      override val evApplicativeR = implicitly[Applicative[G]]
      override def liftF: F ~> G = transform
      override def liftG: G ~> G = idTransform
    }
}
```

Solución



```
abstract class Validator[A, F[_] : Applicative] extends (A ⇒ F[Seq[ValidationResult]]) {
  ...

  def liftWith[G[_] : Applicative](transform: F ~> G): Validator[A, G] = Validator[A, G] { in ⇒ transform(this (in)) }

  private def andSame(that: Validator[A, F]): Validator[A, F] =
    Validator[A, F] { in ⇒
      (this(in), that(in)).mapN(_ ++ _)
    }

  def and[G[_] : Applicative, R[_]](that: Validator[A, G])(implicit canLift: CanLift[F, G, R]): Validator[A, R] = {
    this.liftWith(canLift.liftF)(canLift.evApplicativeR) andSame that.liftWith(canLift.liftG)(canLift.evApplicativeR)
  }


  ...
}
```

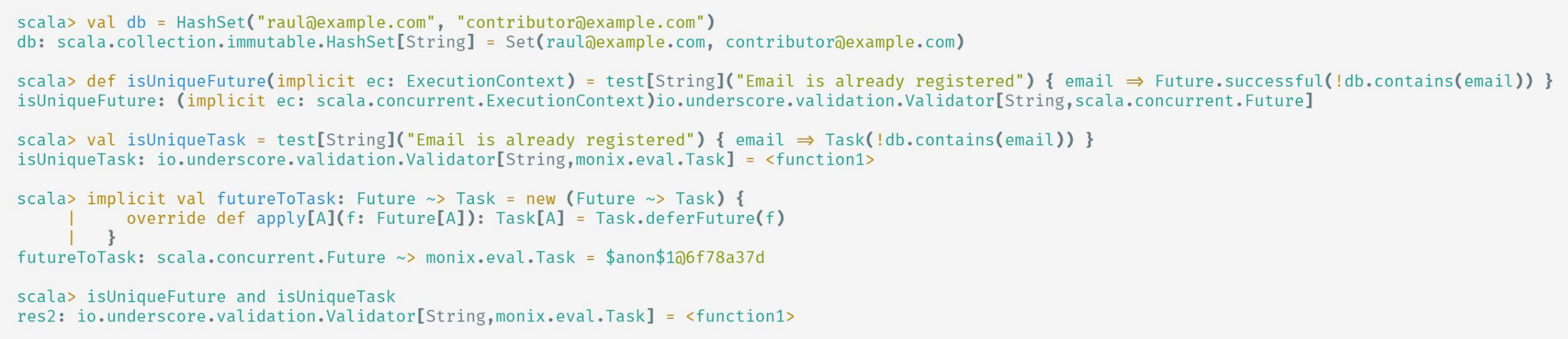
Listo!



```
scala> isEmail and isUniqueFuture
res0: io.underscore.validation.Validator[String,scala.concurrent.Future] = <function1>

scala> isUniqueFuture and isEmail
res1: io.underscore.validation.Validator[String,scala.concurrent.Future] = <function1>
```

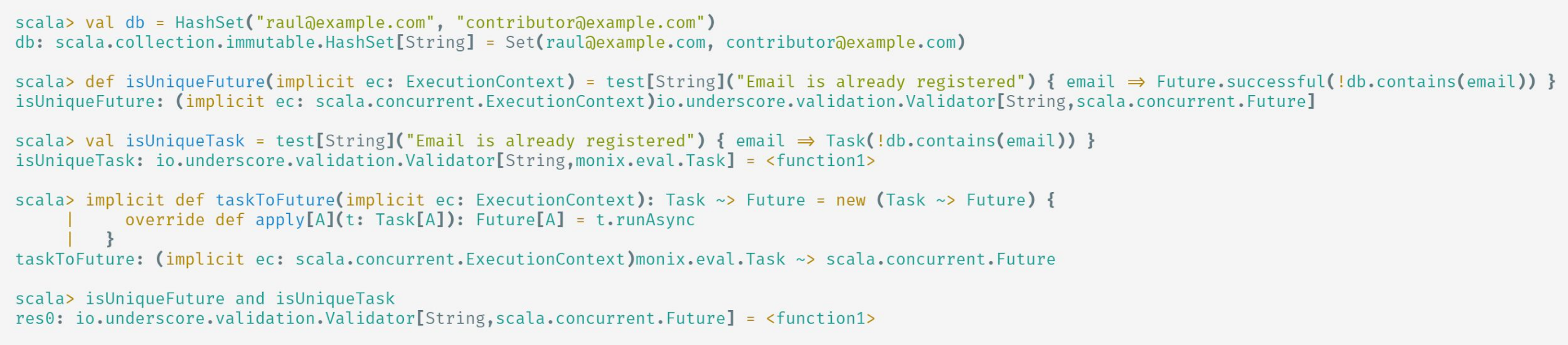
Código Real

Combinar contextos



```
scala> val db = HashSet("raul@example.com", "contributor@example.com")
db: scala.collection.immutable.HashSet[String] = Set(raul@example.com, contributor@example.com)

scala> def isUniqueFuture(implicit ec: ExecutionContext) = test[String]("Email is already registered") { email ⇒ Future.successful(!db.contains(email)) }
isUniqueFuture: (implicit ec: scala.concurrent.ExecutionContext)io.underscore.validation.Validator[String,scala.concurrent.Future]

scala> val isUniqueTask = test[String]("Email is already registered") { email ⇒ Task(!db.contains(email)) }
isUniqueTask: io.underscore.validation.Validator[String,monix.eval.Task] = <function1>

scala> implicit val futureToTask: Future ~> Task = new (Future ~> Task) {
     |     override def apply[A](f: Future[A]): Task[A] = Task.deferFuture(f)
     |   }
futureToTask: scala.concurrent.Future ~> monix.eval.Task = $anon$1@6f78a37d

scala> isUniqueFuture and isUniqueTask
res2: io.underscore.validation.Validator[String,monix.eval.Task] = <function1>
```

Combinar contextos



```
scala> val db = HashSet("raul@example.com", "contributor@example.com")
db: scala.collection.immutable.HashSet[String] = Set(raul@example.com, contributor@example.com)

scala> def isUniqueFuture(implicit ec: ExecutionContext) = test[String]("Email is already registered") { email ⇒ Future.successful(!db.contains(email)) }
isUniqueFuture: (implicit ec: scala.concurrent.ExecutionContext)io.underscore.validation.Validator[String,scala.concurrent.Future]

scala> val isUniqueTask = test[String]("Email is already registered") { email ⇒ Task(!db.contains(email)) }
isUniqueTask: io.underscore.validation.Validator[String,monix.eval.Task] = <function1>

scala> implicit def taskToFuture(implicit ec: ExecutionContext): Task ~> Future = new (Task ~> Future) {
     |     override def apply[A](t: Task[A]): Future[A] = t.runAsync
     |   }
taskToFuture: (implicit ec: scala.concurrent.ExecutionContext)monix.eval.Task ~> scala.concurrent.Future

scala> isUniqueFuture and isUniqueTask
res0: io.underscore.validation.Validator[String,scala.concurrent.Future] = <function1>
```

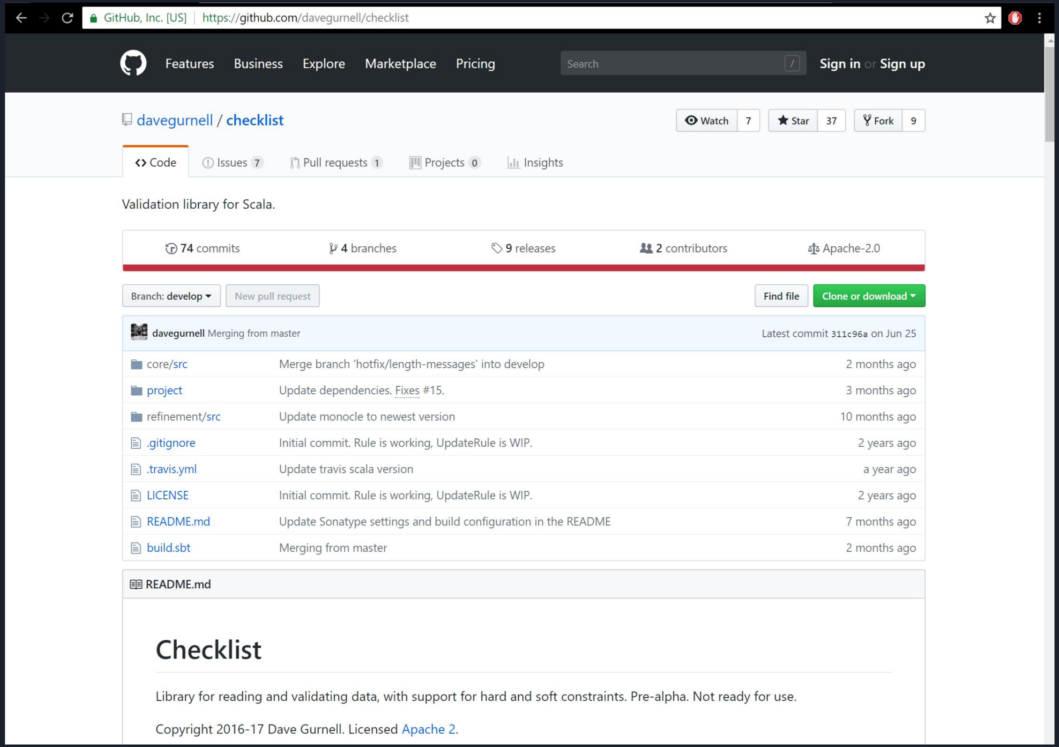
Gracias!

rpiaggio@gmail.com

@RaulPiaggio

Y las Monads??

Nueva encarnación de librería base: checklist



davegurnell / checklist

Validation library for Scala.

74 commits | 4 branches | 9 releases | 2 contributors | Apache-2.0

davegurnell Merging from master — Latest commit 311c96a on Jun 25

core/src	Merge branch 'hotfix/length-messages' into develop	2 months ago
project	Update dependencies. Fixes #15.	3 months ago
refinement/src	Update monocle to newest version	10 months ago
.gitignore	Initial commit. Rule is working, UpdateRule is WIP.	2 years ago
.travis.yml	Update travis scala version	a year ago
LICENSE	Initial commit. Rule is working, UpdateRule is WIP.	2 years ago
README.md	Update Sonatype settings and build configuration in the README	7 months ago
build.sbt	Merging from master	2 months ago

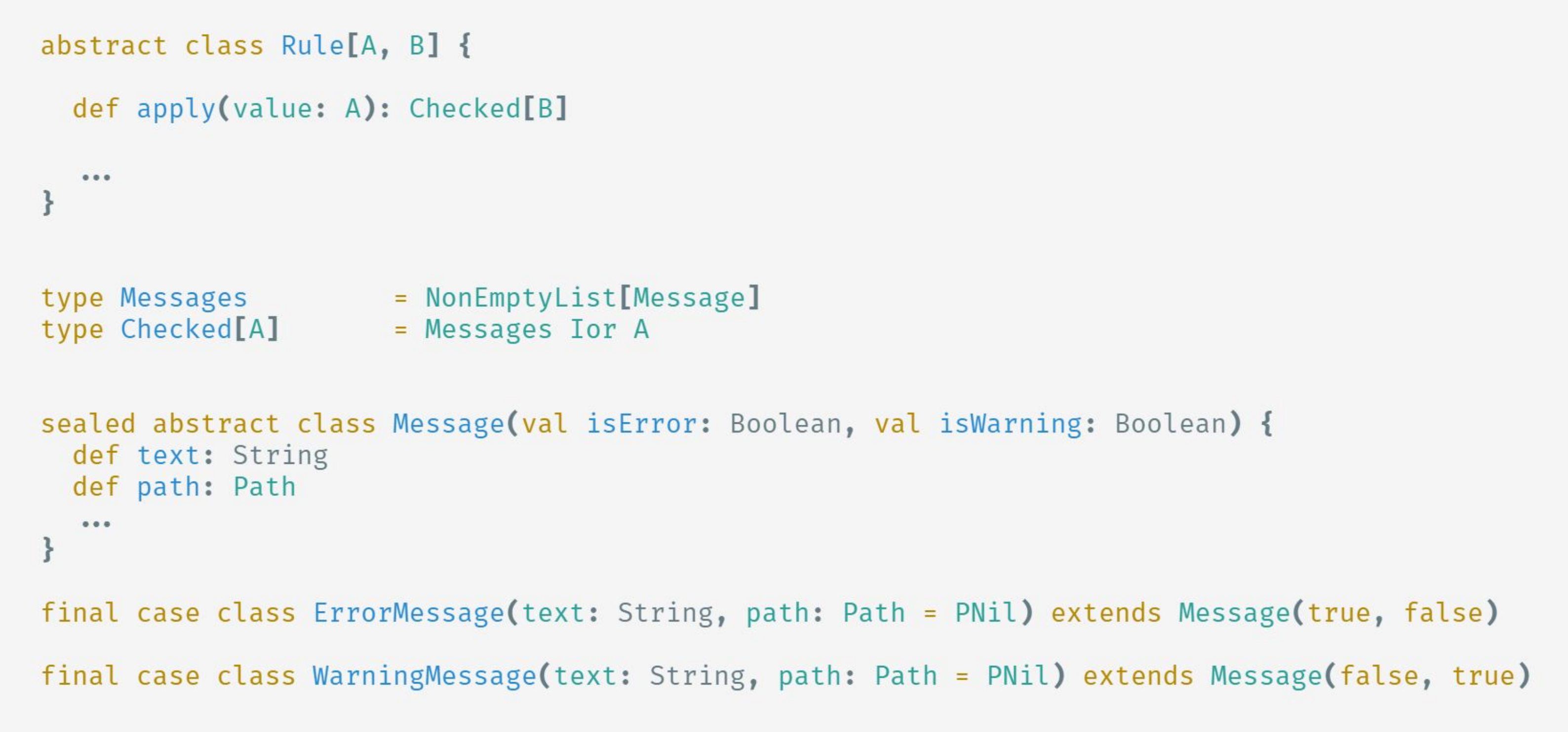
README.md

Checklist

Library for reading and validating data, with support for hard and soft constraints. Pre-alpha. Not ready for use.

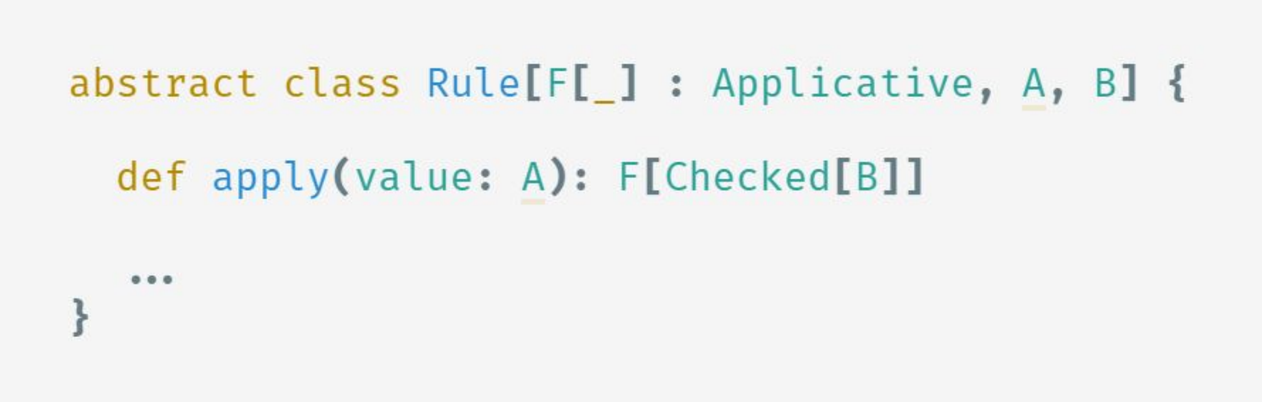
Copyright 2016-17 Dave Gurnell. Licensed Apache 2.

checklist puede modificar el valor



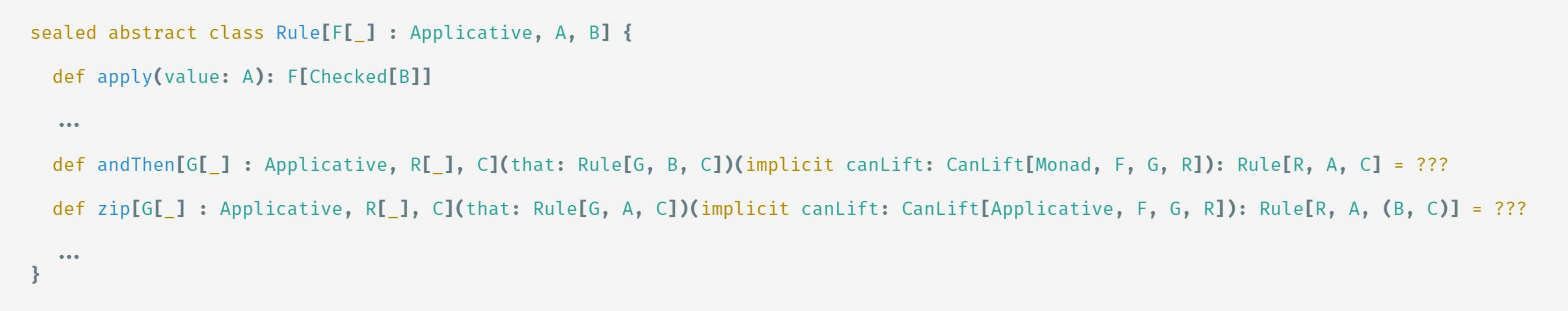
```scala
abstract class Rule[A, B] {

  def apply(value: A): Checked[B]

  ...
}


type Messages         = NonEmptyList[Message]
type Checked[A]       = Messages Ior A


sealed abstract class Message(val isError: Boolean, val isWarning: Boolean) {
  def text: String
  def path: Path
  ...
}

final case class ErrorMessage(text: String, path: Path = PNil) extends Message(true, false)

final case class WarningMessage(text: String, path: Path = PNil) extends Message(false, true)
```

Hacemos lo mismo...



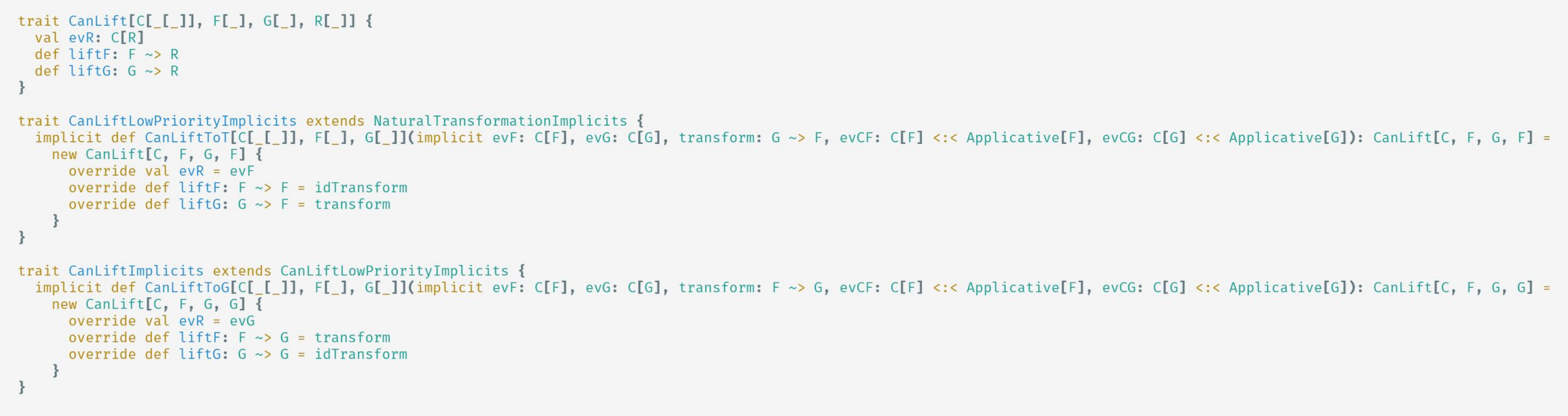
```
abstract class Rule[F[_] : Applicative, A, B] {
  def apply(value: A): F[Checked[B]]
  ...
}
```

Pero si el valor se modifica no podemos paralelizar! Hay dependencia...



```
sealed abstract class Rule[F[_] : Applicative, A, B] {

  def apply(value: A): F[Checked[B]]

   ...

  def andThen[G[_] : Applicative, R[_], C](that: Rule[G, B, C])(implicit canLift: CanLift[Monad, F, G, R]): Rule[R, A, C] = ???

  def zip[G[_] : Applicative, R[_], C](that: Rule[G, A, C])(implicit canLift: CanLift[Applicative, F, G, R]): Rule[R, A, (B, C)] = ???

   ...
}
```

Generalizamos CanLift (aún más)



```scala
trait CanLift[C[_[_]], F[_], G[_], R[_]] {
  val evR: C[R]
  def liftF: F ~> R
  def liftG: G ~> R
}

trait CanLiftLowPriorityImplicits extends NaturalTransformationImplicits {
  implicit def CanLiftToT[C[_[_]], F[_], G[_]](implicit evF: C[F], evG: C[G], transform: G ~> F, evCF: C[F] <:< Applicative[F], evCG: C[G] <:< Applicative[G]): CanLift[C, F, G, F] =
    new CanLift[C, F, G, F] {
      override val evR = evF
      override def liftF: F ~> F = idTransform
      override def liftG: G ~> F = transform
    }
}

trait CanLiftImplicits extends CanLiftLowPriorityImplicits {
  implicit def CanLiftToG[C[_[_]], F[_], G[_]](implicit evF: C[F], evG: C[G], transform: F ~> G, evCF: C[F] <:< Applicative[F], evCG: C[G] <:< Applicative[G]): CanLift[C, F, G, G] =
    new CanLift[C, F, G, G] {
      override val evR = evG
      override def liftF: F ~> G = transform
      override def liftG: G ~> G = idTransform
    }
}
```