

MOVE OVER FREE MONADS:
MAKE WAY FOR FREE APPLICATIVES!

JOHN A. DE GOES — @JDEGOES

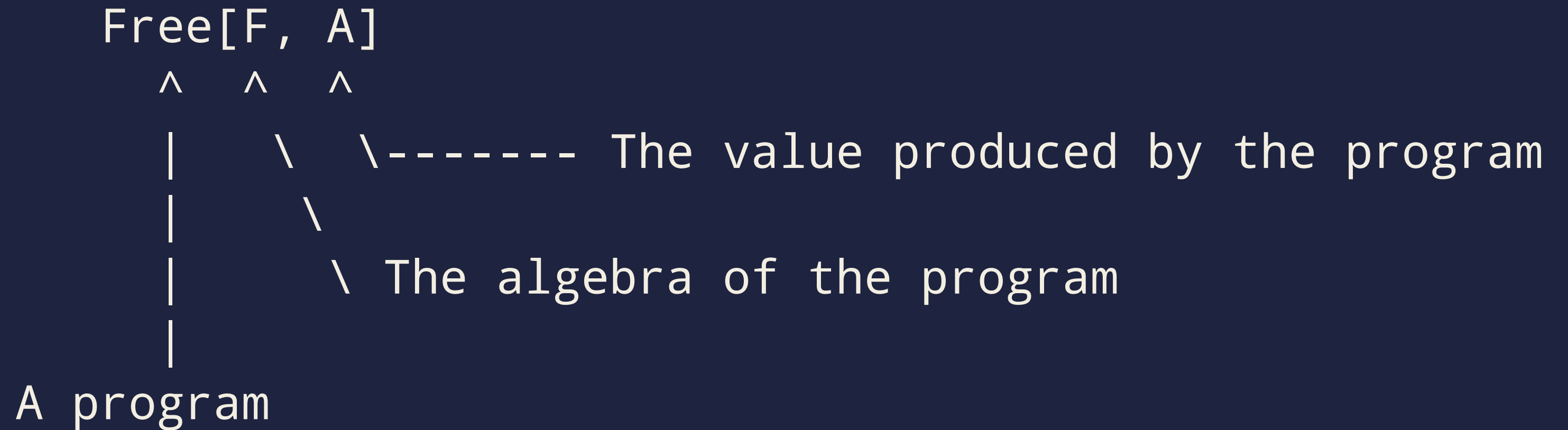
<http://github.com/jdegoes/scalaworld-2015>

FREE MONADS

Free Monads: Don't build a program. Build a description of a program.

- ▶ Easier to transform, compose, reason about
 - ▶ Extensible effects
 - ▶ Reflection without remorse
 - ▶ Aspect-oriented programming

FREE MONADS: THE PROGRAM PERSPECTIVE



THE TROUBLE WITH MONADS

```
trait Monad[F[_]] {  
  ...  
  def bind[A, B](fa: F[A])(f: A => F[B]): F[B]  
}
```

The diagram illustrates the components of the `bind` function signature. Vertical dashed lines connect the following elements:

- `fa: F[A]` to **First Program**
- `A` to **Runtime Value**
- `A => F[B]` to **Second Program**
- `F[B]` to **Result Program**

Horizontal dashed lines are placed below `F[A]`, `A`, and `F[B]`. The `Second Program` label is positioned between the `A` and `F[B]` lines.

Monads embody the essence of sequential computation: a program can depend on a value produced by a previous program.

THE TROUBLE WITH FREE MONADS

The structure of monadic programs is dynamic. Free monad programs can only be interpreted, not introspected and transformed prior to interpretation.

```
def bind[A, B](fa: F[A])(f: A => F[B]): F[B]
```

^

|
|
|
|

!@!#%!@!!

BUT....

Applicatives are, obviously, strictly less powerful than Monads.

```
(doX(a) |@| doY(b) |@| doZ(c))(doResults(_, _, _))
```

Static structure has a cost.

FREE APPLICATIVES

- ▶ Record the structure of applicative composition.
 - ▶ Interpret it later in any way.

```
sealed trait FreeAp[F[_], A]
```

^

|

Any * -> * (not even functor!)

CONFREE: CONFIGURATION LIBRARY

ALGEBRA¹

```
object algebra {  
  sealed trait ConfigF[A]  
  
  case class ConfigInt    [A](field: String, value: Int    => A) extends ConfigF[A]  
  case class ConfigFlag   [A](field: String, value: Boolean => A) extends ConfigF[A]  
  case class ConfigPort   [A](field: String, value: Int     => A) extends ConfigF[A]  
  case class ConfigServer [A](field: String, value: String  => A) extends ConfigF[A]  
  case class ConfigFile   [A](field: String, value: String  => A) extends ConfigF[A]  
  case class ConfigSub    [A](field: String, value: FreeAp[ConfigF, A]) extends ConfigF[A]  
}
```

¹ See Appendix A.

CONFREE: CONFIGURATION LIBRARY

DSL

```
object dsl {  
  import algebra._  
  
  type Dsl[A] = FreeAp[ConfigF, A]  
  
  private def lift[A](value: ConfigF[A]): Dsl[A] = FreeAp.lift[ConfigF, A](value)  
  
  def int    (field: String): Dsl[Int]      = lift(ConfigInt    (field, identity))  
  def flag   (field: String): Dsl[Boolean]  = lift(ConfigFlag   (field, identity))  
  def port   (field: String): Dsl[Int]      = lift(ConfigPort   (field, identity))  
  def server (field: String): Dsl[String]   = lift(ConfigServer (field, identity))  
  def file   (field: String): Dsl[String]   = lift(ConfigFile   (field, identity))  
  def sub[A](field: String)  
    (value: Dsl[A])              = lift(ConfigSub    (field, value))  
}
```

CONFREE: CONFIGURATION LIBRARY

USAGE

```
object example {  
  ...  
  import dsl._  
  
  case class AuthConfig(port: Int, host: String)  
  case class ServerConfig(logging: Boolean, auth: AuthConfig)  
  
  val authConfig    = (int("port") |@| server("host"))(AuthConfig)  
  val serverConfig = (flag("logging") |@| sub("auth")(authConfig))(ServerConfig)  
  ...  
}
```

CONFREE: CONFIGURATION LIBRARY

INSTANT HELP

```
def genHelp[A](config: Dsl[A]): String = ...
```

```
scala> genHelp(serverConfig)
res10: String =
"auth      - a sub-configuration
  host     - a server address
  port     - an integer
logging    - a boolean flag
"
```

CONFREE: CONFIGURATION LIBRARY

INSTANT DESERIALIZATION

```
def genDecode[A](config: Dsl[A]): DecodeJson[A] = ...

val serverConfigJson =
  Json("logging" -> jTrue,
    "auth" -> Json("port" -> jNumberOrNull(2020),
      "host" -> jString("localhost")))

scala> genDecode(serverConfig).decodeJson(serverConfigJson)
res11: argonaut.DecodeResult[confree.example.ServerConfig] =
  DecodeResult(\\-(ServerConfig(true,AuthConfig(2020,localhost))))
```

CONFREE: CONFIGURATION LIBRARY

HOW THE SAUSAGE IS MADE

```
def genDecode[A](config: Dsl[A]): DecodeJson[A] =
  config.foldMap(new NaturalTransformation[ConfigF, DecodeJson] {
    def apply[A](value: ConfigF[A]): DecodeJson[A] = DecodeJson(json => value match {
      case ConfigInt    (n, v) => (json --\ n).as[Int]    .map(v)
      case ConfigFlag   (n, v) => (json --\ n).as[Boolean].map(v)
      case ConfigPort   (n, v) => (json --\ n).as[Int]    .map(v)
      case ConfigServer(n, v) => (json --\ n).as[String] .map(v)
      case ConfigFile   (n, v) => (json --\ n).as[String] .map(v)
      case ConfigSub    (n, v) => for {
        sub <- (json --\ n).as[Json]
        a   <- genDecode(v).decodeJson(sub)
      } yield a
    })
  })
```

CONFREE: CONFIGURATION LIBRARY

HOW THE SAUSAGE IS MADE

```
case class HelpState(help: String = "", indent: Int = 0) {  
  def --> (h: String): HelpState =  
    copy(help = help + (0 until indent * 2).foldLeft[String]("")(a, _) => a + " ") + h + "\n")  
  
  def indented: HelpState = copy(indent = indent + 1)  
  def dedented: HelpState = copy(indent = indent - 1)  
}
```

CONFREE: CONFIGURATION LIBRARY

HOW THE SAUSAGE IS MADE

```
def genHelp[A](config: Dsl[A]): String = {
  type G[A] = State[HelpState, A]

  def genHelp0[A](config: Dsl[A]): G[A] = {
    config.foldMap(new NaturalTransformation[ConfigF, G] {
      def apply[A](value: ConfigF[A]): G[A] = value match {
        case ConfigInt    (n, v) => State.modify[HelpState](_ --> (n + "\t - an integer"      )) *> v(0).point[G]
        case ConfigFlag   (n, v) => State.modify[HelpState](_ --> (n + "\t - a boolean flag"   )) *> v(false).point[G]
        case ConfigPort   (n, v) => State.modify[HelpState](_ --> (n + "\t - a port number"    )) *> v(0).point[G]
        case ConfigServer (n, v) => State.modify[HelpState](_ --> (n + "\t - a server address")) *> v("").point[G]
        case ConfigFile   (n, v) => State.modify[HelpState](_ --> (n + "\t - a file path"      )) *> v("").point[G]
        case ConfigSub    (n, v) => for {
          _ <- State.modify[HelpState](_ --> (n + "\t - a sub-configuration"))
          _ <- State.modify[HelpState](_.indented)
          a <- genHelp0(v)
          _ <- State.modify[HelpState](_.dedented)
        } yield a
      }
    })
  }

  genHelp0(config).exec(HelpState()).help
}
```


YUCK!!!!!!

FINALLY TAGLESS: PAIN-FREE(AP)

```
object algebra {  
  sealed trait AsmF[F[_]] extends Applicative[F] {  
    def push(v: Int): F[Unit]  
  
    def pop: F[Int]  
  
    def sum: F[Int]  
  
    def mul: F[Int]  
  }  
}
```

FINALLY TAGLESS: PAIN-FREE(AP)

```
object dsl {
  import algebra._

  sealed trait Dsl[A] {
    def apply[F[_]: AsmF]: F[A]
  }

  def push(v: Int): Dsl[Unit] = new Dsl[Unit] { def apply[F[_]: AsmF] = implicitly[AsmF[F]].push(v) }

  def pop: Dsl[Int] = new Dsl[Int] { def apply[F[_]: AsmF] = implicitly[AsmF[F]].pop }

  def sum: Dsl[Int] = new Dsl[Int] { def apply[F[_]: AsmF] = implicitly[AsmF[F]].sum }

  def mul: Dsl[Int] = new Dsl[Int] { def apply[F[_]: AsmF] = implicitly[AsmF[F]].mul }

  implicit val ApplicativeDsl: Applicative[Dsl] = new Applicative[Dsl] {
    def point[A](a: => A): Dsl[A] = new Dsl[A] { def apply[F[_]: AsmF] = a.point[F] }
    def ap[A, B](fa: => Dsl[A])(f: => Dsl[A => B]): Dsl[B] = new Dsl[B] {
      def apply[F[_]: AsmF] = fa.apply[F] <*> f.apply[F]
    }
  }
}
```

FINALLY TAGLESS: PAIN-FREE(AP)

```
def program =  
  push(1) *>  
  push(2) *>  
  sum    *>  
  push(9) *>  
  mul
```

FINALLY TAGLESS: PAIN-FREE(AP)

```
def print[A](p: Dsl[A]): String = {  
  type Printer[A] = String  
  
  p.apply(new AsmF[Printer] {  
    def point[A](a: => A): Printer[A] = ""  
  
    def ap[A, B](fa: => Printer[A])(f: => Printer[A => B]): Printer[B] = f + fa  
  
    def push(v: Int) = "push " + v + "\n"  
  
    def pop = "pop\n"  
  
    def sum = "sum\n"  
  
    def mul = "mul\n"  
  })  
}
```

FINALLY TAGLESS: PAIN-FREE(AP)

```
def evaluate[A](p: Dsl[A]): A = {
  type EvaluatorState = List[Int]

  type Evaluator[A] = State[List[Int], A]

  p.apply(new AsmF[Evaluator] {
    def point[A](a: => A): Evaluator[A] = a.point[Evaluator]

    def ap[A, B](fa: => Evaluator[A])(f: => Evaluator[A => B]): Evaluator[B] = fa <*> f

    def push(v: Int) = State.modify[EvaluatorState](v :: _)

    def pop = State.get[EvaluatorState].map(_.head)

    def sum = for {
      l <- State.get[EvaluatorState]
      (x1 :: x2 :: Nil, ys) = l.splitAt(2)
      v = x1 + x2
      _ <- State.put[EvaluatorState](v :: ys)
    } yield v

    def mul = for {
      l <- State.get[EvaluatorState]
      (x1 :: x2 :: Nil, ys) = l.splitAt(2)
      v = x1 * x2
      _ <- State.put[EvaluatorState](v :: ys)
    } yield v
  }).eval(Nil)
}
```

INTUITION FOR FREE FUNCTOR HIERARCHY

- ▶ **Free Functors: Programs that Change Values**
- ▶ **Free Applicatives: Programs that Build Data**
- ▶ **Free Monads: Programs that Build Programs**

EXPRESSIVE POWER OF FREE APPLICATIVES

- ▶ **Parsers - Auto-complete, optimization, & even lookahead!**
 - ▶ **Codecs**
 - ▶ **Simple programs**
- ▶ **Static "Runtime Branching" - Alternative, fixed equivalence**
 - ▶ **???**

THANK YOU!

JOHN A. DE GOES — @JDEGOES

<http://github.com/jdegoes/scalaworld-2015>

APPENDIX A: GADTS IN SCALA

```
sealed trait ConfigF[A]

case class ConfigInt    (field: String) extends ConfigF[Int]
case class ConfigFlag   (field: String) extends ConfigF[Boolean]
case class ConfigPort   (field: String) extends ConfigF[Int]
case class ConfigServer(field: String) extends ConfigF[String]
case class ConfigFile   (field: String) extends ConfigF[String]
case class ConfigSub    [A](field: String, value: FreeAp[ConfigF, A]) extends ConfigF[A]

def fail[A](v: ConfigF[A]) = {
  case ConfigInt(f) => // A must be Int!
  ...
}
```