FREESTYLE

A COHESIVE & PRAGMATIC FRAMEWORK OF FP CENTRIC SCALA LIBRARIES

Getting into typed FP is hard because...

- No previous CT knowledge or math foundations
- Leaving styles one is used to (ex. OOP)
- Lack of docs on how to properly use MTL/Tagless and other techniques required for concise FP.
- Rapid changing ecosystem
- Scala's encoding
- Lack of good examples where FP is applied

Freestyle's Goals

- Approachable to newcomers
- Stack-safe
- Dead simple integrations with Scala's library ecosystem
- Help you build pure FP apps, libs & micro-services
- Pragmatism

In this talk

- Freestyle programming style
- **@free**, **@tagless**, **@module**, **@service**
- Effects
- Integrations
- Optimizations (iota Coproduct, stack-safe @tagless)
- RPC based Microservices

Interface + Impl driven design

```
@free trait Interact {
  def ask(prompt: String): FS[String]
  def tell(msg: String): FS[Unit]
}

implicit val handler: Interact.Handler[Future] = new Interact.Handler[Future] {
  def ask(prompt: String): Future[String] = ???
  def tell(msg: String): Future[Unit] = ???
}
```

Boilerplate Reduction > Declaration

```
+ @free trait Interact {
+  def ask(prompt: String): FS[String]
+  def tell(msg: String): FS[Unit]
+ }
- sealed trait Interact[A]
- case class Ask(prompt: String) extends Interact[String]
- case class Tell(msg: String) extends Interact[Unit]
-
- class Interacts[F[_]](implicit I: InjectK[Interact, F]) {
-  def tell(msg: String): Free[F, Unit] = Free.inject[Interact, F](Tell(msg))
-  def ask(prompt: String): Free[F, String] = Free.inject[Interact, F](Ask(prompt))
- }
-
- object Interacts {
-  implicit def interacts[F[_]](implicit I: InjectK[Interact, F]): Interacts[F] = new Interacts[F]
-  def apply[F[_]](implicit ev: Interacts[F]): Interacts[F] = ev
- }
```

Boilerplate Reduction > Composition

```
+ @module trait App {
+  val exerciseOp: ExerciseOp
+  val userOp: UserOp
+  val userProgressOp: UserProgressOp
+  val githubOp: GithubOp
+ }
- type C01[A] = Coproduct[ExerciseOp, UserOp, A]
- type C02[A] = Coproduct[UserProgressOp, C01, A]
- type ExercisesApp[A] = Coproduct[GithubOp, C02, A]
- val exerciseAndUserInterpreter: C01 ~> M = exerciseOpsInterpreter or userOpsInterpreter
- val userAndUserProgressInterpreter: C02 ~> M = userProgressOpsInterpreter or exerciseAndUserInterpreter
- val allInterpreters: ExercisesApp ~> M = githubOpsInterpreter or userAndUserProgressInterpreter
```

Freestyle's Workflow

- Declare your algebras
- Group them into modules
- Compose your programs
- Provide implicit implementations of each algebra Handler
- Run your programs at the edge of the world

Freestyle's Workflow

Declare your algebras

```
import freestyle._
import freestyle.tagless._

object algebras {
  /* Handles user interaction */
  @free trait Interact {
    def ask(prompt: String): FS[String]
    def tell(msg: String): FS[Unit]
  }

  /* Validates user input */
  @tagless trait Validation {
    def minSize(s: String, n: Int): FS[Boolean]
    def hasNumber(s: String): FS[Boolean]
  }
}
```

Freestyle's Workflow

Group them into modules

```
import algebras._
import freestyle.effects.error._
import freestyle.effects.error.implicits._
import freestyle.effects.state
val st = state[List[String]]
import st.implicits._

object modules {

  @module trait App {
    val validation: Validation.StackSafe
    val interact: Interact
    val errorM : ErrorM
    val persistence: st.StateM
  }

}
```

Freestyle's Workflow

Declare and compose programs inside `@free`, `@tagless` or `@module`

```
import cats.implicits._

object modules {

  @module trait App {
    val validation: Validation.StackSafe
    val interact: Interact
    val errorM : ErrorM
    val persistence: st.StateM

    def program: FS.Seq[Unit] =
      for {
        cat <- interact.ask("What's the kitty's name?")
        isValid <- (validation.minSize(cat, 5), validation.hasNumber(cat)).mapN(_ && _) //may run ops in parallel
        _ <- if (isValid) persistence.modify(cat :: _) else errorM.error(new RuntimeException("invalid name!"))
        cats <- persistence.get
        _ <- interact.tell(cats.toString)
      } yield ()
  }

}
```

Freestyle's Workflow

Declare and compose programs anywhere else

```
def program[F[_]]
  (implicit I: Interact[F], R: st.StateM[F], E: ErrorM[F], V: Validation.StackSafe[F]): FreeS[F, Unit] = {
  for {
    cat <- I.ask("What's the kitty's name?")
    isValid <- (V.minSize(cat, 5), V.hasNumber(cat)).mapN(_ && _) //may run ops in parallel
    _ <- if (isValid) R.modify(cat :: _) else E.error(new RuntimeException("invalid name!"))
    cats <- R.get
    _ <- I.tell(cats.toString)
  } yield ()
}
```

Freestyle's Workflow

Provide implicit evidence of your handlers to any desired target M[_]

```
import cats.effect.IO
import cats.effect.implicits._
import cats.data.StateT

type Target[A] = StateT[IO, List[String], A]

implicit val interactHandler: Interact.Handler[Target] = new Interact.Handler[Target] {
  def ask(prompt: String): Target[String] = tell(prompt) >> StateT.lift("Isidoro1".pure[IO])
  def tell(msg: String): Target[Unit] = StateT.lift(IO { println(msg) })
}

implicit val validationHandler: Validation.Handler[Target] = new Validation.Handler[Target] {
  def minSize(s: String, n: Int): Target[Boolean] = StateT.lift((s.length >= n).pure[IO])
  def hasNumber(s: String): Target[Boolean] = StateT.lift(s.exists(c => "0123456789".contains(c)).pure[IO])
}
```

Freestyle's Workflow

Run your program to your desired target `M[_]`

```
import modules._
// import modules._

import freestyle.implicits._
// import freestyle.implicits._

import cats.mtl.implicits._
// import cats.mtl.implicits._

val concreteProgram = program[App.Op]
// concreteProgram: freestyle.FreeS[modules.App.Op,Unit] = Free(...)

concreteProgram.interpret[Target].runEmpty.unsafeRunSync
// What's the kitty's name?
// List(Isidoro1)
// res6: (List[String], Unit) = (List(Isidoro1),())
```

Effects

An alternative to monad transformers

- **error**: Signal errors
- **either**: Flattens if `Right` / short-circuit `Left`
- **option**: Flatten `Some` / short-circuit on `None`
- **reader**: Deffer dependency injection until program interpretation
- **writer**: Log / Accumulate values
- **state**: Pure functional state threaded over the program monadic sequence
- **traverse**: Generators over `Foldable`
- **validation**: Accumulate and inspect errors throughout the monadic sequence
- **async**: Integrate with callback based API's

Effects

Error

```
import freestyle.effects.error._
// import freestyle.effects.error._

import freestyle.effects.implicits._
// import freestyle.effects.implicits._

type EitherTarget[A] = Either[Throwable, A]
// defined type alias EitherTarget

def shortCircuit[F[_]: ErrorM] =
  for {
    a <- FreeS.pure(1)
    b <- ErrorM[F].error[Int](new RuntimeException("BOOM"))
    c <- FreeS.pure(1)
  } yield a + b + c
// warning: there was one feature warning; for details, enable `:setting -feature' or `:replay -feature'
// shortCircuit: [F[_]](implicit evidence$1: freestyle.effects.error.ErrorM[F])cats.free.Free[[β$0$]cats.free.FreeApplicative[F,β$0$],Int]

shortCircuit[ErrorM.Op].interpret[EitherTarget]
// res7: EitherTarget[Int] = Left(java.lang.RuntimeException: BOOM)

shortCircuit[ErrorM.Op].interpret[IO].attempt.unsafeRunSync
// res8: Either[Throwable,Int] = Left(java.lang.RuntimeException: BOOM)
```

Effects

Option

```
import freestyle.effects.option._
// import freestyle.effects.option._

def programNone[F[_]: OptionM] =
  for {
    a <- FreeS.pure(1)
    b <- OptionM[F].option[Int](None)
    c <- FreeS.pure(1)
  } yield a + b + c
// warning: there was one feature warning; for details, enable `:setting -feature' or `:replay -feature'
// programNone: [F[_]](implicit evidence$1: freestyle.effects.option.OptionM[F])cats.free.Free[[β$0$]cats.free.FreeApplicative[F,β$0$],Int]

programNone[OptionM.Op].interpret[Option]
// res9: Option[Int] = None

programNone[OptionM.Op].interpret[List]
// res10: List[Int] = List()
```

Optimizations

Freestyle provides optimizations for Free + Inject + Coproduct compositions as in DataTypes a la Carte

Optimizations

A fast Coproduct type based on Iota with constant evaluation time based on `@scala.annotation.switch` on the Coproduct's internal indexed values.

```
import iota._
// import iota._

import iota.debug.options.ShowTrees
// import iota.debug.options.ShowTrees

val interpreter: FSHandler[App.Op, Target] = CopK.FunctionK.summon
// <console>:67: {
//   class CopKFunctionK$macro$2 extends _root_.iota.internal.FastFunctionK[Op, Target] {
//     private[this] val arr0 = scala.Predef.implicitly[cats.arrow.FunctionK[algebras.Validation.StackSafe.Op, Target]](algebras.this.Validation.
//     private[this] val arr1 = scala.Predef.implicitly[cats.arrow.FunctionK[algebras.Interact.Op, Target]](interactHandler).asInstanceOf[_root_.
//     private[this] val arr2 = scala.Predef.implicitly[cats.arrow.FunctionK[freestyle.effects.error.ErrorM.Op, Target]](freestyle.effects.implic
// interpreter: freestyle.FSHandler[modules.App.Op,Target] = FastFunctionK[modules.App.Op, Target]<<generated>>
```

Optimizations

Freestyle does not suffer from degrading performance as the number of Algebras increases in contrast with `cats.data.EitherK`

Optimizations

Optimizations over the pattern matching of `FunctionK` for user defined algebras to translate them into a JVM switch with `@scala.annotation.switch`.

Optimizations

Brings ADT-less stack safety to `@tagless` Algebras without rewriting interpreters to `Free[M, ?]` where `M[_]` is stack unsafe.

```
program[Option] // Stack-unsafe
program[StackSafe[Option]#F] // lift handlers automatically to Free[Option, ?] without the `@free` ADTs overhead
```

Integrations

- **Monix**: Target runtime and async effect integration.
- **Fetch**: Algebra to run fetch instances + Auto syntax Fetch ~> FS.
- **FS2**: Embed FS2 Stream in Freestyle programs.
- **Doobie**: Embed ConnectionIO programs into Freestyle.
- **Slick**: Embed DBIO programs into Freestyle.
- **Akka Http**: EntityMarshallers to return Freestyle programs in Akka-Http endpoints.
- **Play**: Implicit conversions to return Freestyle programs in Play Actions.
- **Twitter Util**: Capture instances for Twitter's Future & Try.
- **Finch**: Mapper instances to return Freestyle programs in Finch endpoints.
- **Http4s**: EntityEncoder instance to return Freestyle programs in Http4S endpoints.

Standalone libraries (WIP)

- **frees-kafka**: Consumer, Producer and Streaming algebras for Kafka
- **frees-cassandra**: Algebras for Cassandra API's, object mapper and type safe query compile time validation.
- **frees-rpc**: Purely functional RPC Services.
- **frees-microservices**: Purely functional monitored microservices.

Freestyle RPC

Define your proto messages

```
import freestyle._
import freestyle.rpc.protocol._

trait ProtoMessages {

  @message
  case class Point(latitude: Int, longitude: Int)

  @message
  case class Rectangle(lo: Point, hi: Point)

  @message
  case class Feature(name: String, location: Point)

  @message
  case class FeatureDatabase(feature: List[Feature])

  @message
  case class RouteNote(location: Point, message: String)

  @message
  case class RouteSummary(point_count: Int, feature_count: Int, distance: Int, elapsed_time: Int)

}
```

Freestyle RPC

Expose Algebras as RPC services

```
import monix.reactive.Observable

@option(name = "java_package", value = "routeguide", quote = true)
@option(name = "java_multiple_files", value = "true", quote = false)
@option(name = "java_outer_classname", value = "RouteGuide", quote = true)
object protocols extends ProtoMessages {

  @free
  @service
  trait RouteGuideService {

    @rpc
    def getFeature(point: Point): FS[Feature]

    @rpc
    @stream[ResponseStreaming.type]
    def listFeatures(rectangle: Rectangle): FS[Observable[Feature]]

    @rpc
    @stream[RequestStreaming.type]
    def recordRoute(points: Observable[Point]): FS[RouteSummary]

    @rpc
    @stream[BidirectionalStreaming.type]
    def routeChat(routeNotes: Observable[RouteNote]): FS[Observable[RouteNote]]
  }

}
```

Freestyle RPC gives you for free:

- **gRPC Server**: gRPC based server.
- **client** gRPC client.
- **.proto** files to interoperate with other langs.

Scala First, FP first approach to .proto generation

```
sbt protoGen
```

Scala FP First approach to .proto generation

Find a complete example at https://github.com/frees-io/freestyle-rpc-examples

```
syntax = "proto3";

option java_package = "routeguide";
option java_multiple_files = true;
option java_outer_classname = "RouteGuide";

message Point {
   int32 latitude = 1;
   int32 longitude = 2;
}
...

service RouteGuideService {
   rpc getFeature (Point) returns (Feature) {}
   rpc listFeatures (Rectangle) returns (stream Feature) {}
   rpc recordRoute (stream Point) returns (RouteSummary) {}
   rpc routeChat (stream RouteNote) returns (stream RouteNote) {}
}
```

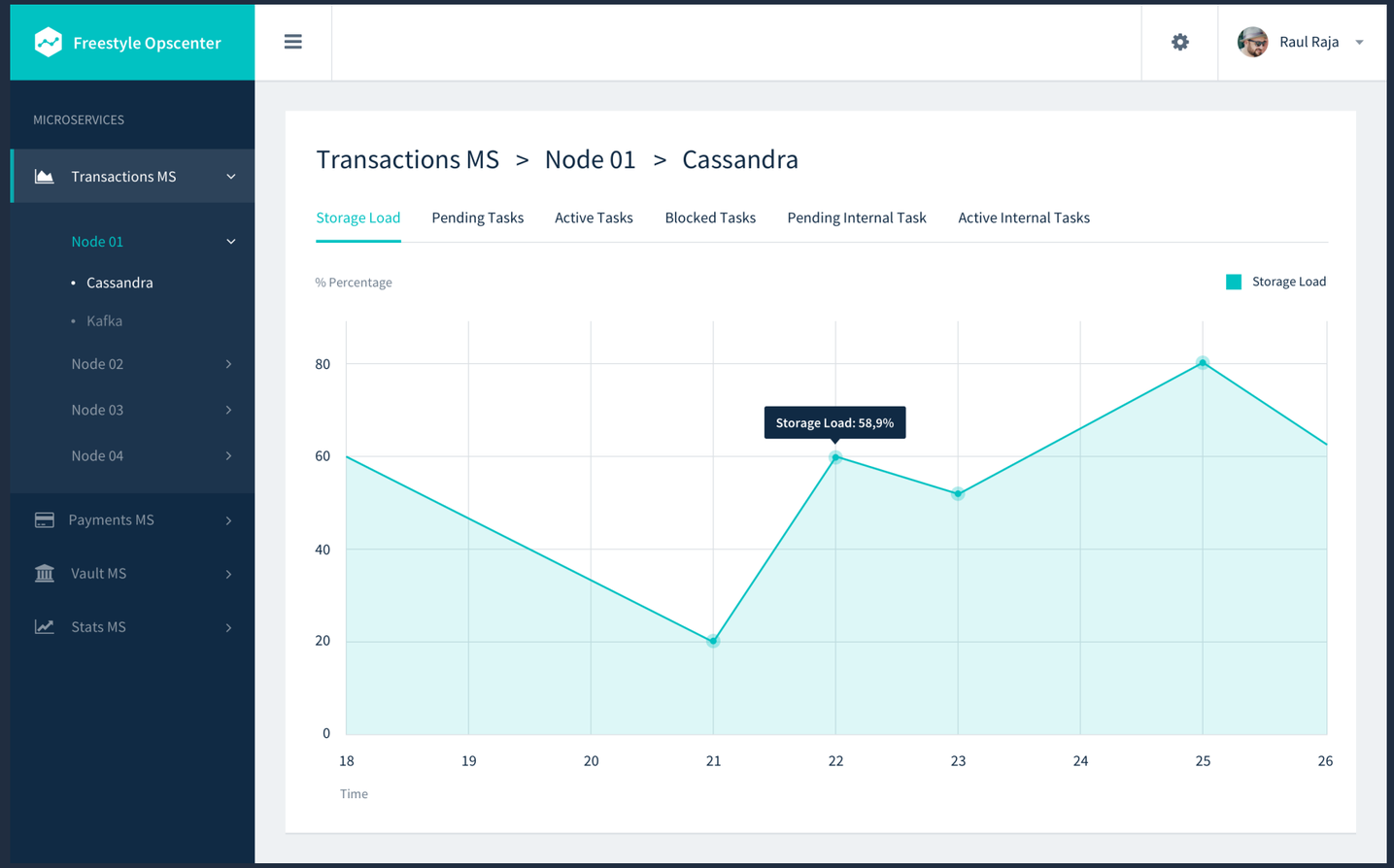
Freestyle Microservices

Provides a reference impl over RPC optionally including Kafka & Cassandra Algebras and Handlers (WIP unfinished design)

```
@free
@service
trait MyService {

  @subscribe[Topic.type]
  def listen(r: ConsumerRecord[Topic#Key, Topic#Value]): FS[Ack]

}

implicit def myServiceHandler
  (implicit
     producer: Producer[Topic#Key, Topic#Value],
     persistence: Persistence[MyModel]): RouteGuideService.Handler[IO] =
  new RouteGuideService.Handler[IO] {

    def listen(r: ConsumerRecord[Topic#Key, Topic#Value]): IO[Ack] = ???

  }
```

Freestyle Microservices OpsCenter

Lightweight monitoring of micro-services through automatic routes



Inspired by

- Cats
- Scalaz
- KΛTEGORY
- Eff
- Fetch
- Simulacrum

Brought to you by...

[colin-passiv](https://github.com/colin-passiv)
Adrián Ramírez Fornell <[AdrianRaFo](https://github.com/AdrianRaFo)>
Alejandro Gómez <[dialelo](https://github.com/dialelo)>
Ana Mª Marquez <[anamariamv](https://github.com/anamariamv)>
Andy Scott <[andyscott](https://github.com/andyscott)>
Diego Esteban Alonso Blas <[diesalbla](https://github.com/diesalbla)>
Domingo Valera <[dominv](https://github.com/dominv)>
Fede Fernández <[fedefernandez](https://github.com/fedefernandez)>
Francisco Diaz <[franciscodr](https://github.com/franciscodr)>
Giovanni Ruggiero <[gruggiero](https://github.com/gruggiero)>
Javi Pacheco <[javipacheco](https://github.com/javipacheco)>
Javier de Silóniz Sandino <[jdesiloniz](https://github.com/jdesiloniz)>
Jisoo Park <[guersam](https://github.com/guersam)>
Jorge Galindo <[jorgegalindocruces](https://github.com/jorgegalindocruces)>
Juan Pedro Moreno <[juanpedromoreno](https://github.com/juanpedromoreno)>
Juan Ramón González <[jrgonzalezg](https://github.com/jrgonzalezg)>
Maureen Elsberry <[MaureenElsberry](https://github.com/MaureenElsberry)>
Peter Neyens <[peterneyens](https://github.com/peterneyens)>
Raúl Raja Martínez <[raulraja](https://github.com/raulraja)>
Sam Halliday <[fommil](https://github.com/fommil)>
Suhas Gaddam <[suhasgaddam](https://github.com/suhasgaddam)>
... and many more contributors

Thanks!

- http://frees.io
- https://github.com/frees-io/freestyle-rpc-examples