A Brief Intro to Tagless Final

Walter Chang 張瑋修 @weihsiu

Agenda

- What is tagless final
- Expression problem
- What are effects
- Pishen’s problem
- More examples
- Tagless final vs. Free
- Libraries that build on tagless final

What is tagless final

- Originated from a paper “Finally Tagless, Partially Evaluated”
- “The characteristic feature of the tagless-final approach is extensibility: At any time one may add more interpreters, more optimization passes, and even more expression forms to the DSL while re-using the previous interpreters as they were. “ -- http://okmij.org/ftp/tagless-final/index.html
- “The best way to structure a purely functional program in Scala.” -- me

Expression problem

- Coined by Philip Wadler
- Object Style
- Function Style
- Tagless Final Style

https://scastie.scala-lang.org/558q8LNGRqivJpIX1u7qcg

Expression problem - Object style

```
sealed trait Expr {
  def eval: Int
}
case class Lit(v: Int) extends Expr {
  def eval = v
}
```

Expression problem - Object style

```
case class Add(e1: Expr, e2: Expr) extends Expr {
  def eval = e1.eval + e2.eval
}
case class Mul(e1: Expr, e2: Expr) extends Expr {
  def eval = e1.eval * e2.eval
}
```

Expression problem - Object style

```
val expr = Add(Lit(1), Mul(Lit(2), Lit(3)))

println(expr.eval) // 7

// expr.view ?
```

Expression problem - Function style

```
sealed trait Expr

case class Lit(v: Int) extends Expr

case class Add(e1: Expr, e2: Expr) extends Expr
```

Expression problem - Function style

```
def eval(expr: Expr): Int = expr match {
  case Lit(v) => v
  case Add(e1, e2) => eval(e1) + eval(e2)
}
```

Expression problem - Function style

```scala
def view(expr: Expr): String = expr match {
  case Lit(v) => v.toString
  case Add(e1, e2) =>
    s"(${view(e1)} + ${view(e2)})"
}
```

Expression problem - Function style

```
val expr = Add(Lit(1), Add(Lit(2), Lit(3)))

println(eval(expr)) // 6

println(view(expr)) // (1 + (2 + 3))

// Add(Lit(1), Mul(Lit(2), Lit(3))) ?
```

Expression problem - Tagless final style

```
trait ExprAlg[A] {

  def lit(v: Int): A

  def add(e1: A, e2: A): A

}
```

Expression problem - Tagless final style

```
val evalExprAlg = new ExprAlg[Int] {
  def lit(v: Int) = v
  def add(e1: Int, e2: Int) = e1 + e2
}
```

Expression problem - Tagless final style

```scala
val viewExprAlg = new ExprAlg[String] {
  def lit(v: Int) = v.toString
  def add(e1: String, e2: String) =
    s"($e1 + $e2)"
}
```

Expression problem - Tagless final style

```
def program1[A](exprAlg: ExprAlg[A]): A = {
  import exprAlg._
  add(lit(1), add(lit(2), lit(3)))
}
println(program1(evalExprAlg)) // 6
println(program1(viewExprAlg)) // (1 + (2 + 3))
```

Expression problem - Tagless final style

```
trait MulAlg[A] {

  def mul(e1: A, e2: A): A

}
```

Expression problem - Tagless final style

```
val evalMulAlg = new MulAlg[Int] {

  def mul(e1: Int, e2: Int) = e1 * e2

}

val viewMulAlg = new MulAlg[String] {

  def mul(e1: String, e2: String) = s"($e1 *
$e2)"

}
```

Expression problem - Tagless final style

```
def program2[A](exprAlg: ExprAlg[A], mulAlg: MulAlg[A]): A = {
  import exprAlg._, mulAlg._
  add(lit(1), mul(lit(2), lit(3)))
}
println(program2(evalExprAlg, evalMulAlg)) // 7
println(program2(viewExprAlg, viewMulAlg)) // (1 + (2 * 3))
```

What are effects

```
type Eff[A] = Option[A]

type Eff[A] = Either[Throwable, A]

type Eff[A] = Future[A]

type Eff[A] = EitherT[Future, Throwable, A]

type Eff[A] = IO[A]

type Eff[A] = Task[A]
```

Pishen’s problem

- Functions returning different effects
  - Either[E, A]
  - Future[A]
  - Future[Either[E, A]]
- Monad transformer to the rescue
- Effect is fixed
  - Difficult to change

https://scastie.scala-lang.org/YyqwIfrPREKulUohdV7CBg

More examples

- Video example
  - Sub program
  - Handling exception
  - Parallel program
  - Log program

https://scastie.scala-lang.org/0jxB9p66SNKocbQqEWstVA

Even more concrete examples

- Protochain
  - A simple blockchain implementation
  - The allure of cats ecosystem -- this project uses:
    - cats
    - cats-effect
    - cats-mtl
    - monix
    - scodec
    - fs2
    - circe
    - http4s
    - doobie

Tagless final vs. Free

- Tagless final is simpler
- Tagless final is easier to compose
- Tagless final is more memory efficient
- Tagless final is easier to compose both sequential and parallel programs
- Tagless final is not automatically stack safe

Libraries that build on tagless final

- freestyle
- mainecoon
- sphynx

The end