Typelevel's path to Scala 3

Lars Hupel
Scala Love in the City
2021-02-13

INNOQ

Typelevel

Typelevel.scala

Let the Scala compiler work for you. We provide type classes, instances, conversions, testing, supplements to the standard library, and much more.

Projects

Our projects cover a wide range of domains, from general functional programming to tooling.

Cats More

Functional Programming

Shapeless

Generic Programming

spire

Numeric abstractions

typelevel.scala

Let the Scala compiler work for you.

http://typelevel.org info@typelevel.org Verified

Repositories 62 Packages People 25

Pinned repositories

cats

Lightweight, modular, and extensible library for functional programming.

Scala 4.2k 1k

fs2

Compositional, streaming I/O library for Scala

Scala 1.8k 471

scalacheck

Property-based testing for Scala

Scala 1.7k 363

spire

Powerful new number types and numeric abstractions for Scala.

Scala 1.6k 237

cats-effect

The purely functional runtime system for Scala

Scala 1.1k 298

discipline

Flexible law checking for Scala

Scala 282 43

Find a repository… Type: All Language: All

Typelevel

Founded in 2013 at
Northeast Scala Symposium

Typelevel

Founded in 2013 at
Northeast Scala Symposium

Today: 70+ projects,
vibrant ecosystem

Typelevel projects

Central theme: Scala-idiomatic Functional Programming

Typelevel projects

Central theme: Scala-idiomatic Functional Programming

- ... with as little hassle as possible

Typelevel projects

Central theme: Scala-idiomatic Functional Programming

- ... with as little hassle as possible
- ... with as little runtime overhead as possible

Typelevel projects

Central theme: Scala-idiomatic Functional Programming

- ... with as little hassle as possible
- ... with as little runtime overhead as possible
- ... as safe as possible

Type classes

Type classes

Supremely useful tool, pioneered in Haskell

Type classes

Supremely useful tool, pioneered in Haskell

```
class Semigroup a => Monoid a where
  mempty :: a
  mconcat :: [a] -> a
  mconcat = foldr mappend mempty
```

Type classes

Supremely useful tool, pioneered in Haskell

```
class Semigroup a => Monoid a where
  mempty :: a
  mconcat :: [a] -> a
  mconcat = foldr mappend mempty
```

It Just Works™!


HULK SMASH THE STATE

Type classes in Scala

... now we just need to encode them in Scala

Type classes in Scala

... now we just need to encode them in Scala

- multiple inheritance?

Type classes in Scala

... now we just need to encode them in Scala

- multiple inheritance?
- syntax??

Type classes in Scala

... now we just need to encode them in Scala

- multiple inheritance?
- syntax??
- global confluence???

Type classes in Scala

... now we just need to encode them in Scala

- multiple inheritance?
- syntax??
- global confluence???

The Limitations of Type Classes as Subtyped Implicits (Short Paper)

Adelbert Chang
adelbertc@gmail.com

Abstract

Type classes enable a powerful form of ad-hoc polymorphism which provide solutions to many programming design problems. Inspired by this, Scala programmers have striven to emulate them in the design of libraries like Scalaz and Cats.

The natural encoding of type classes combines subtyping and implicits, both central features of Scala. However, this encoding has limitations. If the type class hierarchy branches, seemingly valid programs can hit implicit resolution failures. These failures must then be solved by explicitly passing the implicit arguments which is cumbersome and negates the advantages of type classes.

In this paper we describe instances of this problem and show that they are not merely theoretical but often arise in practice. We also discuss and compare the space of solutions to this problem in Scala today and in the future.

CCS Concepts • *Software and its engineering* → *Language features*

the type class resolver automatically searches through the dictionary of instances to ensure the appropriate instances are defined.

Scala programmers have sought to emulate type classes to leverage this kind of ad-hoc polymorphism. The natural encoding of type classes uses implicits for instance definition and resolution and subtyping for specifying type class relationships.

As a running example consider the (stubbed) encoding of the `Functor` and `Monad` type classes. Each type class becomes a trait, and relationships between type classes become subtype relationships. For example, every `Monad` gives rise to a `Functor`, so `Monad[F] extends Functor[F]`.

```
trait Functor[F[_]] { }
trait Monad[F[_]] extends Functor[F] { }
```

It is also possible to write functions abstracting over these type classes.

The rise of the macros

Type classes, encoded

In 2015, Michael Pilquist started *simulacrum*.

Goal: consistent encoding across different projects, 0 boilerplate

Simulacrum

Input

```
import simulacrum._

@typeclass trait Semigroup[A] {
  @op("|+|") def append(x: A, y: A): A
}
```

Simulacrum

Output

```
object Semigroup {
  def apply[A](implicit instance: Semigroup[A]): Semigroup[A] = instance

  // ...
}
```

Simulacrum

More output

```
object Semigroup {
  trait Ops[A] {
    def typeClassInstance: Semigroup[A]
    def self: A
    def |+|(y: A): A = typeClassInstance.append(self, y)
  }
}
```

Simulacrum

Even more output

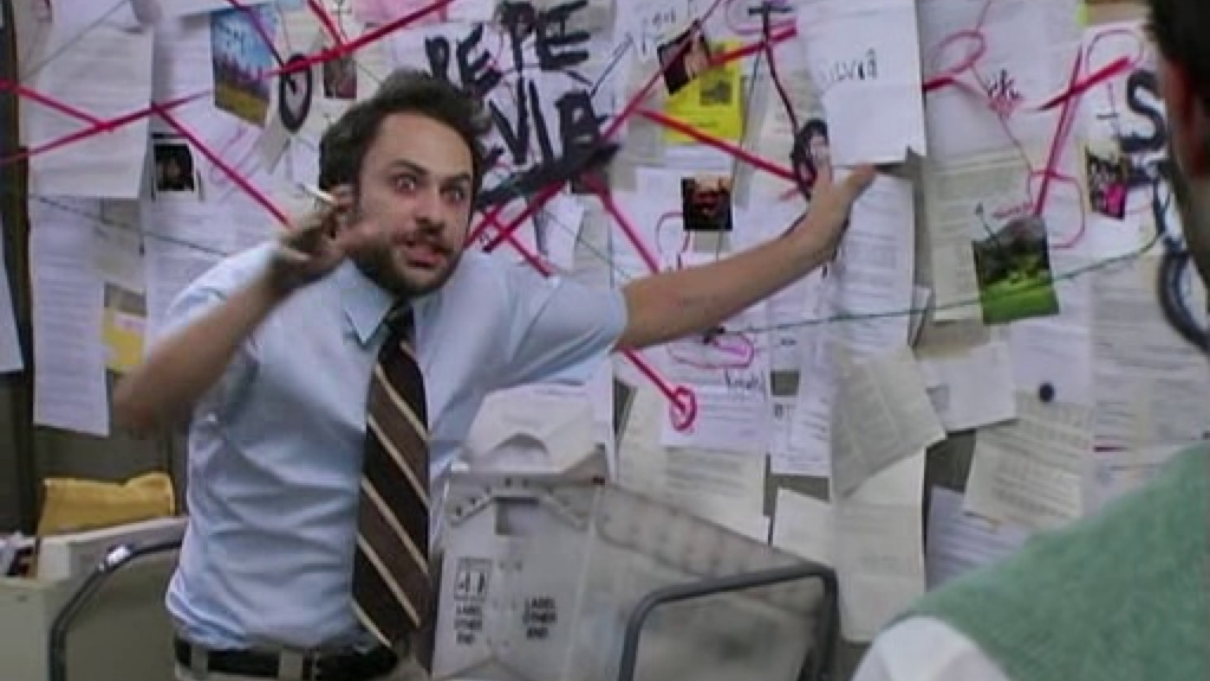
```
object Semigroup {
  trait ToSemigroupOps {
    implicit def toSemigroupOps[A](target: A)(implicit tc: Semigroup[A]): Ops[A]
      val self = target
      val typeClassInstance = tc
    }
  }

  object nonInheritedOps extends ToSemigroupOps
}
```

Simulacrum

Yet more output

```
object Semigroup {
  trait AllOps[A] extends Ops[A] {
    def typeClassInstance: Semigroup[A]
  }
  object ops {
    implicit def toAllSemigroupOps[A](target: A)(implicit tc: Semigroup[A]): AllO
      val self = target
      val typeClassInstance = tc
    }
  }
}
```



But it works!

Simulacrum solved a ton of issues

We can write `x |+| y`!

But it works!

Simulacrum solved a ton of issues

We can write `x |+| y`!

Used by Cats and tons of third-party libraries

We're not done yet

Simulacrum didn't solve the performance issue.

We're not done yet

Simulacrum didn't solve the performance issue.

Input

```
x |+| y
```

We're not done yet

Simulacrum didn't solve the performance issue.

Output

```
Semigroup.ops.toAllSemigroupOps(x).|+|(y)
```

Enter Machinist

Split out of Spire by Erik Osheim in 2014

Enter Machinist

Split out of Spire by Erik Osheim in 2014

Now (2020) archived and re-incorporated into Spire

Cats 3 Roadmap #3757

Open 10 comments

larsrh 4 days ago Member

Continued from #2296.

- **@joroKr21** suggests that Cats 3 is Scala-3-only.
- **@LukaJCB** and **@kailuowang** advocated that we only bring break binary compatibility changes to new Scala major versions.
- **@kailuowang** drafted a plan in 2019.

larsrh mentioned this issue 4 days ago

[Meta Thread] Post 1.2 Roadmap #2296 Closed

Type constructors

Type constructor polymorphism

```scala
case class EitherT[F[_], A, B](value: F[Either[A, B]]) {
  // ...
}
```

Type constructor polymorphism

```scala
case class EitherT[F[_], A, B](value: F[Either[A, B]]) {
  // ...
}
```

Scala is the *only* language that can do that!*

Type constructor polymorphism

Landed in Scala 2.5 by Adriaan Moors (2007)

Type constructor polymorphism

Landed in Scala 2.5 by Adriaan Moors (2007)

Complete game-changer

> *As soon as you give Scala programmers a new toy, they will start abusing it in ways you can't imagine.*
>
> – ancient Scala proverb

Scala isn't Haskell

Haskell

```haskell
instance Monad (Either a) where
  {- ... -}
```

Scala isn't Haskell

Haskell

```haskell
instance Monad (Either a) where
  {- ... -}
```

Scala

```scala
implicit val eitherMonad[A]: Monad[({ type λ[β] = Either[A, β] })#λ] =
  // ...
```

kind-projector

kind-projector

Even macros can't fix missing syntax.*

We need a compiler plugin!

kind-projector

Even macros can't fix missing syntax.*

We need a compiler plugin!

```
implicit val eitherMonad[A]: Monad[Either[A, ?]] = // ...
```

implement higher-order unification for type constructor inference

Log In

Details

Type:	Improvement	Status:	Open
Priority:	Minor	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	None
Component/s:	Misc Compiler		
Labels:	None		
Environment:	tcpoly_infer		

Description

implement what's described in "Partial polymorphic type inference and higher-order unification"

```
object Test {
  def meh[M[_], A](x: M[A]): M[A] = x
  meh{(x: Int) => x} // should solve ?M = [X] X => X and ?A = Int ...
}
```

Issue Links

blocks	SI-5993 Unexpected compiler error involving type lambda and implicit conversion	
is duplicated by	SI-6744 It is impossible to pattern match on a case class containing partially applied type constructors	
relates to	SI-5075 Anonymous type function is accepted as higher-kinded parameter type, but does not unify with it	

Activity

All Comments History Activity Commits

SI-2712

```
def foo[F[_], A](fa: F[A]) = // ...

// doesn't compile!
// Either[_, _] is not an F[_]
foo(x: Either[String, Int])
```

Unapply

Dependent method types + tons of boilerplate = SI-2712 hack

Unapply

```
trait Unapply[TC[_[_]], MA] {
  type M[_]
  type A

  def TC: TC[M]

  def subst: MA => M[A]
}

// okay ...
```

Unapply

```
implicit def unapply3MTLeft[TC[_[_]], F[_[_],_,_], AA[_], B, C]
      (implicit tc: TC[F[AA,?,C]]): Aux3MTLeft[TC,F[AA, B, C], F, AA, B, C] =
  new Unapply[TC, F[AA,B,C]] {
    type M[X] = F[AA, X, C]
    type A = B
    def TC: TC[F[AA, ?, C]] = tc
    def subst: F[AA, B, C] => M[A] = identity
  }

// the what now?!?!
```

Seven years later ...

Miles fixes it for Scala 2.12!

Seven years later ...

Miles fixes it for Scala 2.12!

(... and for 2.10 and 2.11 with a compiler plugin)

Seven years later ...

Miles fixes it for Scala 2.12!

(... and for 2.10 and 2.11 with a compiler plugin)

Type constructors in Dotty

The Dotty team has taken great care to consolidate the current “hacks”

Type lambdas now built in!


DOTTY AND TYPES:
THE STORY SO FAR
Guillaume Martres - EPFL
TEKNOLOGIHUSET
House of Communities

Implementing Higher-Kinded Types in Dotty

Martin Odersky, Guillaume Martres, Dmitry Petrashko

EPFL, Switerland: {first.last}@epfl.ch

Abstract

dotty is a new, experimental Scala compiler based on DOT, the calculus of Dependent Object Types. Higher-kinded types are a natural extension of first-order lambda calculus, and have been a core construct of Haskell and Scala. As long as such types are just partial applications of generic classes, they can be given a meaning in DOT relatively straightforwardly. But general lambdas on the type level require extensions of the DOT calculus to be expressible. This paper is an experience report where we describe and discuss four implementation strategies that we have tried out in the last three years. Each strategy was fully implemented in the *dotty* compiler. We discuss the usability and expressive power of each scheme, and give some indications about the amount of implementation difficulties encountered.

proved to be challenging, so much so that we evaluated four different strategies before settling on the current direct representation encoding. The strategies are summarized as follows:

- A *simple encoding* in the DOT-inspired [9] core type structures that can express partial applications and not much more
- A *direct representation* that adds support for full type lambdas and higher-kinded applications, without reusing much of the existing concepts of the calculus and the compiler.
- A *projection encoding*, that encodes higher-kinded types as first-order generic types using type projections `T#A`.

Implementing Higher-Kinded Types in Dotty

Martin Odersky, Guillaume Martres, Dmitry Petrashko

EPFL, Switerland: {first.last}@epfl.ch

Ab

dot

the

types are a natural extension of first-order lambda calculus, and have been a core construct of Haskell and Scala. As long as such types are just partial applications of generic classes, they can be given a meaning in DOT relatively straightforwardly. But general lambdas on the type level require extensions of the DOT calculus to be expressible. This paper is an experience report where we describe and discuss four implementation strategies that we have tried out in the last three years. Each strategy was fully implemented in the *dotty* compiler. We discuss the usability and expressive power of each scheme, and give some indications about the amount of implementation difficulties encountered.

- A *direct representation* that adds support for full type lambdas and higher-kinded applications, without reusing much of the existing concepts of the calculus and the compiler.

evaluated four
ent direct rep-
narized as fol-
lows:

- A *simple encoding* in the DOT-inspired [9] core type structures that can express partial applications and not much more
- A *direct representation* that adds support for full type lambdas and higher-kinded applications, without reusing much of the existing concepts of the calculus and the compiler.
- A *projection encoding*, that encodes higher-kinded types as first-order generic types using type projections `T#A`.

Macros

Scala 2 macros

Landed in Scala 2.10 (2012)

Enabled lots of innovation across the board

Scala 2 macros

Landed in Scala 2.10 (2012)

Enabled lots of innovation across the board

Sadly, not available any more in Dotty

main scodec-bits / core / shared / src / main /

Add file ··· Download

mpilquist Upgrade to Scala 3.0.0-M3 ✓ 51e9638 on Dec 17, 2020 History

..

scala-2.11/scodec/bits	Convert build to sbt-spiewak	5 months ago
scala-2.12/scodec/bits	Convert build to sbt-spiewak	5 months ago
scala-2.13/scodec/bits	Convert build to sbt-spiewak	5 months ago
scala-2/scodec/bits	avoid using type parameters on Left or Right	2 months ago
scala-3.0.0-M1/scodec/bits	Prepare for M3	2 months ago
scala-3.0.0-M2/scodec/bits	Upgraded to Scala 3.0.0-M2 (#252)	2 months ago
scala-3.0.0-M3/scodec/bits	Upgrade to Scala 3.0.0-M3	2 months ago
scala/scodec/bits	Scalafmt, build updates	3 months ago

Migration of scodec

First supported version: Dotty 0.22.0*

Migration of scodec

First supported version: Dotty 0.22.0*

Requires complete reimplementation between Scala 2 and Dotty

What's next?

Scala 3.0.0 is near!

✓ language features
✓ simulacrum
✓ many major Typelevel projects
✓ release train

Scala 3.0.0 is near!

✓ language features
✓ simulacrum
✓ many major Typelevel projects
✓ release train

Still lots to do (Monix, http4s, ...)

Q & A

INNOQ
www.innoq.com

Lars Hupel


lars.hupel@innoq.com


@larsr_h

LARS HUPEL

Senior Consultant
innoQ Deutschland GmbH

Lars is known as one of the founders of the Type-level initiative which is dedicated to providing principled, type-driven Scala libraries in a friendly, welcoming environment. A frequent conference speaker, they are active in the open source community, particularly in Scala.

Sources

- https://twitter.com/ohbadiah/status/299937487713878016
- https://twitter.com/travisbrown/status/300015411125174273
- https://secure.trifork.com/dl/techmesh-london-2012/slides/JohnHughes_and_PhilipWadler_and_SimonPeytonJones_KeynoteHaskellPracticalAsWellAsCool.pdf
- https://www.reddit.com/r/shiba/comments/e6ec1e/angry_shiba/
- https://pixabay.com/vectors/people-jump-silhouette-group-male-4894818/