The missing diamond of Scala variance

Stephen Compall

Why does variance matter?

What works in Scala?

What doesn’t work?

Variance in a nutshell

`type Endo[A] = A => A`	invariant
`type Get[+A] = Foo => A`	covariant
`type Put[-A] = A => Foo`	contravariant
`X ~  Y → Endo[X] ~ Endo[Y]`	invariant
`X <: Y →  Get[X] <: Get[Y]`	covariant
`X <: Y →  Put[Y] <: Put[X]`	contravariant

Subtyping is incomplete without variance

	Type Equality
reflexivity	$A \sim A$
symmetry	$A \sim B \rightarrow B \sim A$
transitivity	$A \sim B \wedge B \sim C$ $\rightarrow A \sim C$
congruence	$A \sim B$ $\rightarrow F[A] \sim F[B]$

Subtyping is incomplete without variance

	Type Equality	**Type Conformance**
reflexivity	$A \sim A$	$A <: A$
symmetry	$A \sim B \rightarrow B \sim A$	$A <: B \wedge B <: A \rightarrow A = B$
transitivity	$A \sim B \wedge B \sim C$ $\rightarrow A \sim C$	$A <: B \wedge B <: C$ $\rightarrow A <: C$
congruence	$A \sim B$ $\rightarrow F[A] \sim F[B]$	???

Completing subtyping: variables

```
val aCat = Cat("Audrey")
val anAnimal: Animal = aCat
```

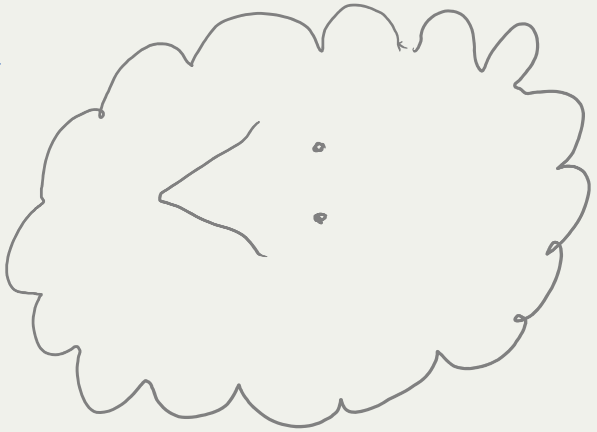
Completing subtyping: the harmony of a function call

```
def speak(a: Animal): IO[Unit]
speak(aCat)
```

covariant

aCat

: Cat



contravariant

speak

$: \text{Animal} \rightarrow \text{IO[Unit]}$

The rest

What else does Scala model here?

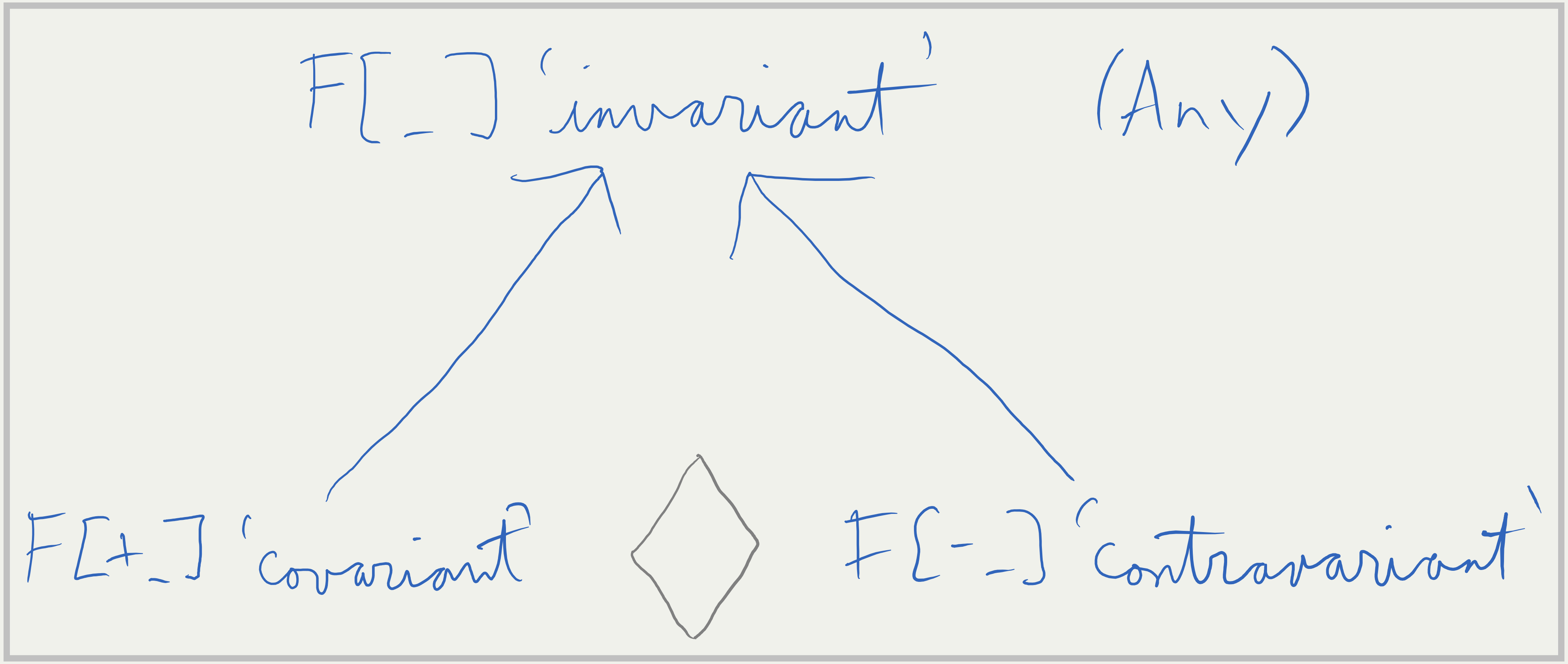
What is it missing?

You *need* this

```
def same[A,                F[_]]
  (fa: F[A]): F[A] = fa

def widen[A, B >: A, F[+_]]
  (fa: F[A]): F[B] = fa
```

Variance exhibits a subkinding relation



Flipping variances

```
mutable.Seq[A] extends Seq[+A]

CovCoy[F[+_]] extends Coy[F[_]]
```

Flipping variances

```
mutable.Seq[A]          extends     Seq[+A]
CovCoy[F[+_]]           extends     Coy[F[_]]
 InvMT[T[_[_]]]         extends   CovMT[T[_[+_]]]
CovMTT[W[_[_[+_]]]]     extends  InvMTT[W[_[_[_]]]]
```

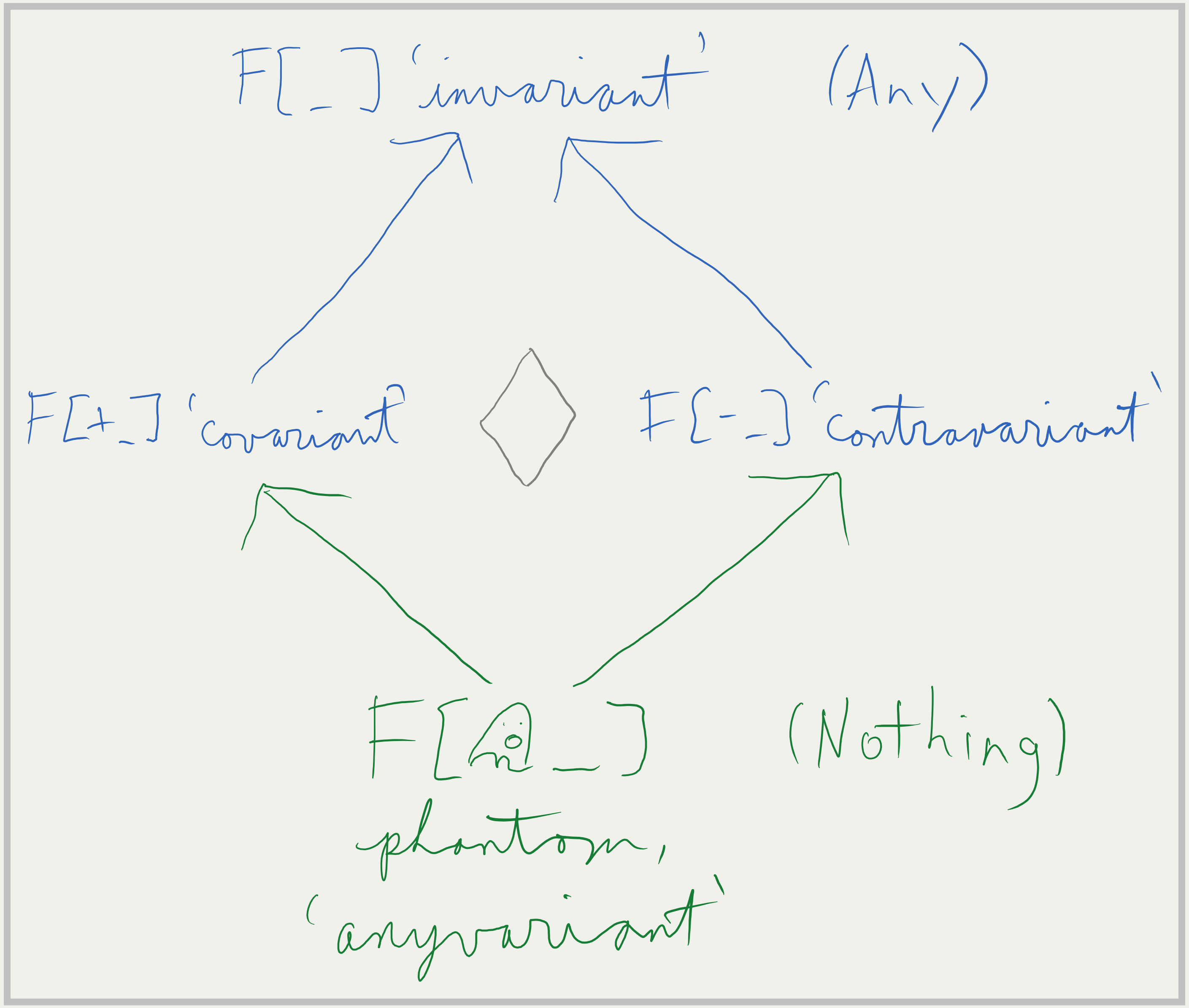
Type parameter positions are variance-contravariant

(variance variance)

Complete diamond: the bottom variance

```
type ConstI[👻A] = Int
ConstI[A] ~ ConstI[B]   phantom, or 👻
```

The diamond


F[_] 'invariant'
(Any)
F[+_] 'covariant'
F[-_] 'contravariant'
F[_]
phantom,
'anyvariant'
(Nothing)

Variance of monad transformers

```
case class NewOptionT[F[_],   A]
case class OldOptionT[F[+_], +A]

  (run: F[Option[A]]) // in both cases
```

Variance variables

```
case class OptionT[😟V, F[V😟_], V😟A]
```

GHC type roles

`a ~ b → f a ~ f b`	nominal
$a \sim_w b \to f\ a \sim_w f\ b$	representational
$f\ a \sim_w f\ b$	phantom

Breakout roles

```
newtype MaybeT m a =
  MaybeT (m (Maybe a))
```

Harder cases

```
case class Compose[F[_], G[_], A]
  (run: F[G[A]])
```

There are
[only]
four variances.

References

- “Of variance and functors”, Adelbert Chang, `https://is.gd/yumowo`
- “SI-2066 Unsoundness in overriding methods with higher-order type parameters”, Scala JIRA, `https://is.gd/jifixe`
- “Roles”, GHC Wiki, `https://is.gd/pugupu`

Reach me at @S11001001