scalaz Typeclass Cheat Sheet

Adam Rosien (adam@rosien.net)

January 21, 2015

Installation

In your `build.sbt` file:

```
libraryDependencies += "org.scalaz" %% "scalaz-core" % "7.0.4"
```

Then in your `.scala` files:

```
import scalaz._
```

Defining Signatures

Each typeclass is defined by a particular function signature and a set of laws[1] (invariants) that the typeclass must obey.

[0] Typeclass laws are not listed here. See each typeclass' scaladoc link for more information.

Typeclass	Signature				
Functor	`F[A]`	`=>`	`(A => B)`	`=>`	`F[B]`
Contravariant	`F[A]`	`=>`	`(B => A)`	`=>`	`F[B]`
Apply[2]	`F[A]`	`=>`	`F[A => B]`	`=>`	`F[B]`
Bind	`F[A]`	`=>`	`(A => F[B])`	`=>`	`F[B]`
Traverse	`F[A]`	`=>`	`(A => G[B])`	`=>`	`G[F[B]]`
Foldable	`F[A]`	`=>`	`(A => B)`	`=>`	`B`
Plus	`F[A]`	`=>`	`F[A]`	`=>`	`F[A]`
Cobind	`F[A]`	`=>`	`(F[A] => B)`	`=>`	`F[B]`
Zip	`F[A]`	`=>`	`F[B]`	`=>`	`F[(A, B)]`
Unzip	`F[(A, B)]`	`=>`			`(F[A], F[B])`

[2] `Apply` has a (broader) subtype `Applicative`. See the expanded tables below.

Derived Functions

For each typeclass, its defining function is marked in **bold** and each derived function listed below it.

Typeclass		Signature				Function
Functor	F[A]	=>	(A => B)	=>	F[B]	**map**
		=>	B	=>	F[B]	as
		=>			F[(A, A)]	fpair
		=>	G[_]	=>	F[G[A]]	fpoint
		=>	(A => B)	=>	F[(A, B)]	fproduct
		=>	B	=>	F[(B, A)]	strengthL
		=>	B	=>	F[(A, B)]	strengthR
		=>			F[Unit]	void
Contravariant	F[A]	=>	(B => A)	=>	F[B]	**contramap**
Apply[3]	F[A]	=>	F[A => B]	=>	F[B]	**ap**
		=>	F[B]	=>	F[(A,B)]	tuple
		=>	F[B]	=>	F[B]	*>
		=>	F[B]	=>	F[A]	<*
Applicative	F[A]	=>	F[A => B]	=>	F[B]	**ap**
		=>	Boolean	=>	F[Unit]	unlessM
		=>	Boolean	=>	F[Unit]	whenM
		=>	Int	=>	F[List[A]]	replicateM
		=>	Int	=>	F[Unit]	replicateM_
Bind	F[A]	=>	(A => F[B])	=>	F[B]	**flatMap**
		=>	F[B]	=>	F[B]	>>
	F[F[A]]	=>		=>	F[A]	join
Traverse	F[A]	=>	(A => G[B])	=>	G[F[B]]	**traverse**
		=>	(A => G[F[B]])	=>	G[F[B]]	traverseM
		=>			F[A]	reverse
		=>	F[B]	=>	F[(A, Option[B])]	zipL
		=>	F[B]	=>	F[(Option[A], B)]	zipR
	F[G[A]]	=>			G[F[A]]	sequence

[3] Both the `Apply` and `Applicative` typeclasses implement the `ap` method; `Applicative` is a subtype of `Apply`, with an additional `point` method to lift a value into the `Applicative`.

Typeclass			Signature			Function
		=>	(A => B)	=>	B	**foldMap**
		=>	B => ((A, B) => B)	=>	B	foldRight
		=>	B => ((B, A) => B)	=>	B	foldLeft
		=>			A	fold
		=>			Int	length
		=>	Int	=>	Option[A]	index
		=>	(A, Int)	=>	A	indexOr
Foldable	F[A]	=>			A	suml
		=>			A	sumr
		=>			List[A]	toList
		=>			Set[A]	toSet
		=>			Stream[A]	toStream
		=>	(A => Boolean)	=>	Boolean	all
		=>	(A => Boolean)	=>	Boolean	any
		=>			Boolean	empty
Plus	F[A]	=>	F[A]	=>	F[A]	**plus**
Cobind	F[A]	=>	(F[A] => B)	=>	F[B]	**cobind**
		=>			F[F[A]]	cojoin
		=>	F[B]	=>	F[(A, B)]	**zip**
Zip	F[A]	=>	F[B] => ((A, B) => C)	=>	F[C]	zipWith
		=>	(F[A] => F[B])	=>	F[(A, B)]	apzip
		=>			(F[A], F[B])	**unzip**
Unzip	F[(A, B)]	=>			F[A]	firsts
		=>			F[B]	seconds