Relationship between Functor, Applicative Functor, and Monad

Asked 12 years, 11 months ago Modified 12 years, 11 months ago Viewed 4k times

37

When reading up on type classes I have seen that the relationship between Functors, Applicative Functors, and Monads is that of strictly increasing power. Functors are types that can be mapped over. Applicative Functors can do the same things with *certain* effects. Monads the same with *possibly unrestrictive* effects. Moreover:

```
Every Monad is an Applicative Functor
Every Applicative Functor is a Functor
```

The definition of the Applicative Functor shows this clearly with:

```
class Functor f => Applicative f where
  pure  :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b
```

But the definition of Monad is:

```
class Monad m where
  return :: a -> m a
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)   :: m a -> m b -> m b
  m >> n = m >>= \_ -> n
  fail   :: String -> m a
```

According to Brent Yorgey's great typeclassopedia that an alternative definition of monad could be:

```
class Applicative m => Monad' m where
  (>>=) :: m a -> (a -> m b) -> m b
```

which is obviously simpler *and* would cement that Functor < Applicative Functor < Monad. So why isn't this the definition? I know applicative functors are new, but according to the 2010 Haskell Report page 80, this hasn't changed. Why is this?

haskell

Share edited Dec 28, 2011 at 19:23 asked Dec 28, 2011 at 16:17

Improve this question

Follow

Magnus Kronqvist
1,589 ● 12 ● 22

1 "Monads the same with unrestrictive effects." Where did you get that idea? – Conal Dec 28, 2011 at 18:23

1 I am of course making the mistake of putting an equal sign between Monads and IO. What would be more correct to say? – Magnus Kronqvist Dec 28, 2011 at 19:22

7 @MagnusKronqvist: Perhaps "dependent" would be a better word. The main difference between Applicative and Monad is the types of `(<*>) :: f (a -> b) -> f a -> f b` and `(=<<) :: (a -> m b) -> m a -> m b`, i.e. the `m` part of the result can depend on the `a` from the input, while for an applicative the `f` part of the result must be the same independent of the value of the `a` input. – hammar Dec 28, 2011 at 19:40

1 `class Applicative m => Monad'' m where join :: m (m a) -> m a` is another possible minimal complete definition. (also noted in typeclassopedia) – Dan Burton Dec 28, 2011 at 20:14

7 @MagnusKronqvist: Here's an explanation of what Monad gives you over Applicative that I like: Applicative forces the structure of the computation to be *static*, whereas Monad allows it to be *dynamic*, depending on the results of other computations. – ehird Dec 28, 2011 at 23:04

2 Answers

Sorted by: Highest score (default)

28

Everyone wants to see Applicative become a superclass of Monad, but it would break so much code (if `return` is eliminated, every current Monad instance becomes invalid) that everyone wants to hold off until we can extend the language in such a way that avoids breaking the code (see here for one prominent proposal).

Haskell 2010 was a conservative, incremental improvement in general, standardising only a few uncontroversial extensions and breaking compatibility in one area to bring the standard in line with every existing implementation. Indeed, Haskell 2010's libraries don't even include Applicative — less of what people have come to expect from the standard library is standardised than you might expect.

Hopefully we'll see the situation improve soon, but thankfully it's usually only a mild inconvenience (having to write `liftM` instead of `fmap` in generic code, etc.).

Share Improve this answer Follow

answered Dec 28, 2011 at 17:20

ehird
40.8k ● 3 ● 184 ● 184

1 Not *everyone* wants `Applicative` to become a superclass of `Monad` – Bertie Wheen May 23, 2015 at 18:33

8

Changing the definition of Monad at this point, would have broken a lot of existing code (any piece of code that defines a Monad instance) to be worthwhile.

Breaking backwards-compatibility like that is only worthwhile if there is a large practical benefit to the change. In this case the benefit is not that big (and mostly theoretical anyway) and wouldn't justify that amount of breakage.

Share Improve this answer Follow

answered Dec 28, 2011 at 16:36
sepp2k
370k ● 56 ● 679 ● 681

1 In what way do you mean that it would break compatibility? 'Applicative m => Monad m' probably breaks certain existing code. But the fix should be easy and Haskell 2010 already is not entirely backwards compatible. – Magnus Kronqvist Dec 28, 2011 at 16:44

Haskell 2010 is entirely backwards compatible with real implementations of Haskell 98, I believe; the breaking change in syntax was something that every compiler already implemented the way Haskell 2010 mandates. – ehird Dec 28, 2011 at 16:49

3 Everyone wants to see Applicative become a superclass of Monad, but it would break so much code that everyone wants to hold off until we can extend the language in such a way that avoids breaking the code. See here for one prominent proposal. – ehird Dec 28, 2011 at 16:50

@ehird Thanks, it should almost be an answer though! – Magnus Kronqvist Dec 28, 2011 at 16:56

@MagnusKronqvist: I couldn't resist expanding it into an answer :) – ehird Dec 28, 2011 at 17:20