What a Monad is not

From HaskellWiki

Contents

1 Warning

Don't be surprised if you leave this page more confused than before. That just means that it has successfully destroyed your false assumptions, or that you've fallen for some horrible inside joke. Beware of Zygohistomorphic prepromorphisms. Go for warm and fuzzy (http://en.wikibooks.org/wiki/Haskell/Understanding_monads) , instead.

2 Monads are not a good choice as topic for your first Haskell blog entry

...just accept that they're burritos (http://blog.plover.com/prog/burritos.html) , and wait until later.

3 Monads are not a language feature

Really. They are defined in terms of Haskell, not Haskell in terms of them. Conversely,

4 Haskell doesn't need Monads

...well, apart from the current Haskell standard defining the way IO is done in terms of Monads: It could be done differently and still work (http://donsbot.wordpress.com/2009/01/31/reviving-the-gofer-standard-prelude-circa-1994/) .

5 Monads are not impure

...In no way whatsoever. You don't even need flexible morals to claim it. To be more specific, it's IO that's impure. That makes the IO monad impure. But that's not a general property of monads - just IO. And even then, we can pretend that Haskell is a purely functional description language for imperative programs. But we didn't want to employ flexible morals, now did we?

6 Monads are not about state

While it is certainly possible to abstract away explicit state passing by using a Monad, that's not what a monad is. Some examples for monads that are not about state: Identity monad, Reader monad, List monad, Continuation monad, Exception monad.

7 Monads are not about strictness

Monad operations (bind and return) have to be non-strict in fact, always! However other operations can be specific to each monad. For instance some are strict (like IO), and some are non-strict (like []). Then there are some that come in multiple flavours, like State.

Try the following:

```
runState (sequence . repeat $ state (\x -> (x,x+1))) 0
```

Having a look at the implementation of fixIO might be helpful, too.

8 Monads are not values

This point might be driven home best by pointing out that instance Monad Foo where ... is not a data type, but a declaration of a typeclass instance. However, to elaborate:

Monads are not values in the same sense that addition and multiplication are not numbers: They capture a -- very specific -- relationship between values of a

specific domain into a common abstraction. We're going to call these values monads manage *mobits*, somewhat like this:

```
type Mobit m a = Monad m => m a
```

The IO monad manages mobits representing side-effects ("IO actions").

The List monad manages mobits representing multiple values ("[a]")

The Reader monads manages mobits that are pure computations that use asks to propagate information instead of explicit arguments

...and while addition and multiplication are both monoids over the positive natural numbers, a monad is a monoid object in a category of endofunctors: return is the unit, and join is the binary operation. It couldn't be more simple. If that confuses you, it might be helpful to see a Monad as a lax functor from a terminal bicategory.

9 Monads are not a replacement for applicative functors

Instead, every monad *is* an applicative functor (as well as a functor). It is considered good practice not to use >>= if all you need is <*>, or even fmap.

Not confusing which features of monads are specific to monads only and which stem from applicative functors is vitally important for a deeper understanding of monads. As an example, the applicative functor interface of parser libraries can parse context-free languages (modulo hacks abusing open recursion), while the monadic interface can parse context-sensitive grammars: Monads allow you to influence further processing by inspecting the result of your parse. To understand why, have a look at the type of >>=. To understand why applicative functors by themselves are sufficient to track the current parsing position and express sequencing, have a look at the uu-parsinglib tutorial (pdf) (http://www.cs.uu.nl/research/techreps/repo/CS-2008/2008-044.pdf) .

The exact differences are elaborated in even greater detail in Brent Yorgey's excellent Typeclassopedia.

10 Monads are not about ordering/sequencing

Monads are commonly used to order sequences of computations. But this is misleading. Just as you can use monads for state, or strictness, you can use them to order computations. But there are also commutative monads, like Reader, that don't order anything. So ordering is not in any way essential to what a monad is.

Let's have a look at what's meant by ordering. Consider an expression like

```
let x = a
    y = b
in  f x y
```

That gives the same result as

```
let y = b
    x = a
in  f x y
```

It doesn't matter what order we write the two bindings. But for doing I/O we'd like ordering. Monads allow us to express

```
do
    x <- getChar
    y <- getChar
    return (x,y)
```

and have it be different from

```
do
    y <- getChar
    x <- getChar
    return (x,y)
```

Unlike the first, the second example returns a pair of characters in the opposite order to which they were entered.

It might help to meditate about the difference between 'assignment' and 'binding', right now.

However, just to spoil the enlightenment you just attained, there are monads for which swapping the order of lines like this makes no difference: For example, the Reader monad.

So while it is correct to say that monads can be used to order operations, it would be wrong to say that monads are a mechanism for ordering operations.

This notion of commutativity looks superficially very different from the familiar one in vanilla algebra where a+b=b+a. It doesn't mean that

```
m >> n === n >> m
```

which doesn't hold in general for *any* non-trivial monad, as for the most part

```
return 1 >> return 2 === return 2 =/= return 1 === return 2 >> return 1
```

This shouldn't be too surprising, though, as >> isn't the binary operation of a monoid. The category-theoretic definition (http://ncatlab.org/nlab/show/commutative+algebraic+theory) of a commutative monad is rather more

abstract.

11 See also

- Do notation considered harmful

Categories:

- FAQ
- Monad
