Monoids and Monads

Donovan Crichton

August 2022

Preliminaries

- Slides and Examples available at:
  https://github.com/donovancrichton/ANU-FP
- This talk: /MonoidsAndMonads

Last Week

- Type Classes in Haskell are almost an Algebra, Haskell has no way to express the laws.
- Idris lets us express the laws via proofs of equality.
- We saw Functor and Applicative type classes.
- We left some future work on the proof of Applicative laws for this week.

Functors

- The Functor interface lets us map over a structure. Letting us transform the underlying elements into new elements.
- Applicative lets us apply pure functions to ‘funny types’ [McBride and Paterson, 2008].

For example:

```
-- change a list of number to a list of functions.
(\x => MkPair x) <$> [1, 2, 3] : Num a => [b -> (a, b)]

-- lift a pure function up to apply to maybe types
(pure (+)) <*> (Just 2) <*> (Just 3) = Just 5
```

Functors

- The Functor interface lets us map over a structure. Letting us transform the underlying elements into new elements.
- Applicative lets us apply pure functions to 'funny types' [McBride and Paterson, 2008].

For example:

```
-- change a list of number to a list of functions.
(\x => MkPair x) <$> [1, 2, 3] : Num a => [b -> (a, b)]

-- lift a pure function up to apply to maybe types
(pure (+)) <*> (Just 2) <*> (Just 3) = Just 5
```

Proofs of Applicative Laws

- Thanks to the hard work of Dr Hideyuki Kawabata!
- Let's see the code.

Monoids as an Algebra

- An algebra $(A,\ F,\ L)$.
- A carrier set A.
- Two operations:

$$F = \{\langle + \rangle : A \to A \to A, id : A\}$$

- Three laws:

$$\begin{aligned} L = \{ & rightid : a\langle + \rangle id = a, \\ & leftid : id\langle + \rangle a = a, \\ & assoc : a\langle + \rangle (b\langle + \rangle c) = (a\langle + \rangle b)\langle + \rangle c\} \end{aligned}$$

Lots of familiar things are Monoids

- Addition is a monoid:

$$
\begin{aligned}
(\mathbb{Z}, \{&+, 0\},\\
&\{x + 0 = x,\\
&\ 0 + x = x,\\
&\ x + (y + z) = (x + y) + z\})
\end{aligned}
$$

- Multiplication is a monoid:

$$
\begin{aligned}
(\mathbb{Z}, \{&\times, 1\},\\
&\{x \times 1 = x,\\
&\ 1 \times x = x,\\
&\ x \times (y \times z) = (x \times y) \times z\})
\end{aligned}
$$

Still more Monoids

- `++` is a monoid:

```
(List a, {++, Nil},
        {xs ++ [] = xs,
         [] ++ xs = 0,
         xs ++ (ys ++ zs) = (xs ++ ys) ++ zs})
```

- `||` is a monoid:

```
(Bool, {||, False},
        {p || False = p,
         False || p = p,
         p || (q || r) = (p || q) || r})
```

Can you think of others?

Verified Monoids

- Time for the demo.

Finally the M-Word!

- *In addition to it's being good and useful...it's also cursed.*[1]
- "*Just a Monoid in the category of Endofunctors.*"[2]
- Burritos!? [3] [4]

[1] Thanks to Douglas Crawford

[2] Thanks to James Iry.

[3] Thanks to Brent Yorgy.

[4] Not to be confused with Ed Morehouse's excellent paper for Hungry readers.

Ok...So what's a Monad?

Formal definition [edit]

Throughout this article C denotes a category. A *monad* on C consists of an endofunctor $T\colon C \to C$ together with two natural transformations: $\eta\colon 1_C \to T$ (where 1_C denotes the identity functor on C) and $\mu\colon T^2 \to T$ (where T^2 is the functor $T \circ T$ from C to C). These are required to fulfill the following conditions (sometimes called coherence conditions):

- $\mu \circ T\mu = \mu \circ \mu T$ (as natural transformations $T^3 \to T$); here $T\mu$ and μT are formed by "horizontal composition"
- $\mu \circ T\eta = \mu \circ \eta T = 1_T$ (as natural transformations $T \to T$; here 1_T denotes the identity transformation from T to T).

We can rewrite these conditions using the following commutative diagrams:

$$
\begin{tikzcd}
T^3 \arrow[r, "T\mu"] \arrow[d, "\mu T"'] & T^2 \arrow[d, "\mu"] \\
T^2 \arrow[r, "\mu"'] & T
\end{tikzcd}
\qquad
\begin{tikzcd}
T \arrow[r, "\eta T"] \arrow[d, "T\eta"'] \arrow[dr, equal] & T^2 \arrow[d, "\mu"] \\
T^2 \arrow[r, "\mu"'] & T
\end{tikzcd}
$$

See the article on natural transformations for the explanation of the notations $T\mu$ and μT, or see below the commutative diagrams not using these notions:

$$
\begin{tikzcd}
T(T(T(X))) \arrow[r, "T(\mu_X)"] \arrow[d, "\mu_{T(X)}"'] & T(T(X)) \arrow[d, "\mu_X"] \\
T(T(X)) & T(X)
\end{tikzcd}
\qquad
\begin{tikzcd}
T(X) \arrow[r, "\eta_{T(X)}"] \arrow[d, "T(\eta_X)"'] \arrow[dr, equal] & T(T(X)) \arrow[d, "\mu_X"] \\
T(T(X)) & T(X)
\end{tikzcd}
$$

Lets try again...

- A Monad is an algebra[5].
- A Monad is a way to sequence effectful computations.
- A Monad is a way to enforce referential transparency and purity.
- A Monad is a way to reason about the universe.

[5] Technically a Kleisli Triple

Let's start with the Algebra

Let $C =$ M must also be an Applicative!

Let $M = (C, A, F, L)$

Let $F = \{$

$$\mu : M(M(A)) \to M(A),$$

$$>>= : M(A) \to (A \to M(B)) \to M(B)\}$$

Let $L = \{$

$$pureIdLeft : pure(x) >>= f = f(x),$$

$$pureIdRight : m >>= pure = m,$$

$$assoc : m >>= (\lambda x.f(x) >>= g) = (m >>= f) >>= g\}$$

Verified Monads

- Time for the demo.

what about the rest?

- ~~A Monad is an algebra.~~
- A Monad is a way to sequence effectful computations.
- A Monad is a way to enforce referential transparency and purity.
- A Monad is a way to reason about the universe.

References

C. McBride and R. Paterson. Applicative programming with effects. *Journal of functional programming*, 18(1):1–13, 2008.