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:

$$\begin{aligned}(\texttt{List a}, \{\texttt{++}, \texttt{Nil}\},\\ \{&\texttt{xs ++ [] = xs},\\ &\texttt{[] ++ xs = 0},\\ &\texttt{xs ++ (ys ++ zs) = (xs ++ ys) ++ zs}\})\end{aligned}$$

- `||` is a monoid:

$$\begin{aligned}(\texttt{Bool}, \{\texttt{||}, \texttt{False}\},\\ \{&\texttt{p || False = p},\\ &\texttt{False || p = p},\\ &\texttt{p || (q || r) = (p || q) || r}\})\end{aligned}$$

Can you think of others?

Verified Monoids

- Time for the demo.

References

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