

Storage as a Monad Transformer

Julian Porter
julian.porter@lincoln.oxon.org

1 Introduction

1.1 Abstract

In [1] we proved that any persistent datastore is a monad, and is monoidal if its underlying data is a monoid. In this paper we extend this and show that if the underlying data is an (additive) monad, then so is the store. Hence the datastore is a monad transformer.

1.2 The simple storage monad

Recall that in [1] we made the following definition:

Definition 1.2.1. *A type s is a **store** if, for any type a in some class *Storable* there are operations:*

$$\begin{aligned} \text{get} &:: (\text{Storable } a) \Rightarrow s \ a \rightarrow a \\ \text{put} &:: (\text{Storable } a) \Rightarrow a \rightarrow s \ a \end{aligned}$$

subject to the constraints that $\text{get} \circ \text{put} = \text{id}$ and $\text{put} \circ \text{get} = \text{id}$.

and we proved:

Proposition 1.2.2. *If s is a store then it is a monad, with*

$$\begin{aligned} x \gg f &:= f (\text{get } x) \\ \text{return} &:= \text{put} \end{aligned}$$

and:

Proposition 1.2.3. *If s is a store and m is a monoid, then $s \ m$ is a monoid with:*

$$\begin{aligned} \square &:= \text{put } \odot \\ x \boxplus y &:= \text{put } \$ (\text{get } x) \oplus (\text{get } y) \end{aligned}$$

where we use the notations \odot and \square for the monoidal zero in m and $s \ m$ respectively, and similarly \oplus and \boxplus for monoidal addition.

2 Storage as monad transformer

2.1 Monad

Now we take an apparently different tack, and make the following definition:

Definition 2.1.1. *Let s be a store and m be a monad. Define the operations:*

$$\begin{aligned} (\gg) &:: s \ m \ a \rightarrow (a \rightarrow s \ m \ b) \rightarrow s \ m \ b \\ (\gg) \ x \ f &= \text{put } \$ (\text{get } x) \gg \text{get } \circ f \\ \underline{\text{return}} &:: a \rightarrow s \ m \ a \\ \underline{\text{return}} &= \text{put} \circ \text{return} \end{aligned}$$

Then we can prove:

Proposition 2.1.2. *These operations make $s\ m$ a monad.*

Proof. Considering the first monad law $\underline{\text{return}}\ x \gg\! = f \equiv f\ x$, we get

$$\begin{aligned} \underline{\text{return}}\ x \gg\! = f &\equiv (\text{put } \$ \text{ return } x) \gg\! = f \\ &\equiv \text{put } \$ (\text{get } \$ (\text{put } \$ \text{ return } x)) \gg\! = \text{get } \circ f \\ &\equiv \text{put } \$ \text{ return } x \gg\! = \text{get } \circ f \\ &\equiv \text{put } \$ \text{ get } \$ f\ x \\ &\equiv f\ x \end{aligned}$$

Considering the second law $x \gg\! = \underline{\text{return}} \equiv x$, we get

$$\begin{aligned} x \gg\! = \underline{\text{return}} &\equiv x \gg\! = \text{put } \circ \text{return} \\ &\equiv \text{put } \$ (\text{get } x) \gg\! = \text{get } \circ \text{put } \circ \text{return} \\ &\equiv \text{put } \$ \text{ get } x \gg\! = \text{return} \\ &\equiv \text{put } \$ \text{ get } x \\ &\equiv x \end{aligned}$$

and finally, the third law $s \gg\! = (\lambda x \rightarrow f\ x \gg\! = g) \equiv (s \gg\! = g) \gg\! = g$ gives

$$\begin{aligned} s \gg\! = (\lambda x \rightarrow f\ x \gg\! = g) &\equiv \text{put } \$ (\text{get } s) \gg\! = \text{get } \circ (\lambda x \rightarrow f\ x \gg\! = g) \\ &\equiv \text{put } \$ (\text{get } s) \gg\! = \text{get } \circ (\lambda x \rightarrow \text{put } \$ \text{ get } (f\ x) \gg\! = \text{get } \circ g) \\ &\equiv \text{put } \$ (\text{get } s) \gg\! = (\lambda x \rightarrow (\text{get } \circ f)\ x) \gg\! = \text{get } \circ g \\ &\equiv \text{put } \$ ((\text{get } s) \gg\! = \text{get } \circ f) \gg\! = \text{get } \circ g \\ &\equiv \text{put } \$ (\text{get } (\text{put } \$ (\text{get } s) \gg\! = \text{get } \circ f)) \gg\! = \text{get } \circ g \\ &\equiv \text{put } \$ \text{ get } (s \gg\! = f) \gg\! = \text{get } \circ g \\ &\equiv (s \gg\! = f) \gg\! = g \end{aligned}$$

and so $s\ m$ is a monad. □

Now we appear to have two different monadic structures on $s\ m$: the monad transformer and the monad defined in 1.2.1. They are in fact equivalent.

Proposition 2.1.3. *The natural monadic structure on s defined in 1.2.1 is the monad transformer associated to the trivial monad on the underlying data.*

Proof. Let m be the trivial monad defined by letting values in $m\ a$ be just values in a , and taking

$$\begin{aligned} \text{return } x &= x \\ x \gg\! = f &= f\ x \end{aligned}$$

Then in $s\ m$ we have $x \gg\! = f \equiv \text{put } \$ (\text{get } x) \gg\! = \text{get } \circ f \equiv \text{put } \circ \text{get } \circ f (\text{get } x) \equiv f (\text{get } x)$ and $\underline{\text{return}}\ x \equiv \text{put } (\text{return } x) \equiv \text{put } x$. □

2.2 Transformer

Proposition 2.2.1. *If s is a store and m a monad, and we define*

$$\begin{aligned} \text{lift} &:: m\ a \rightarrow s\ m\ a \\ \text{lift} &= \text{put} \end{aligned}$$

then s is a monad transformer.

Proof. We have already proved that $s\ m$ is a monad, so we need only prove that the additional laws involving $lift$ hold. Start with the first law $lift \circ return \equiv return$. Trivially:

$$lift \circ return \equiv put \circ return \equiv return$$

The second law is $lift (x \gg f) = (lift\ x) \gg (lift \circ f)$. Expanding, we get:

$$\begin{aligned} lift (x \gg f) &\equiv put \$ x \gg f \\ &\equiv put \$ get (put\ x) \gg get (put \circ f) \\ &\equiv (put\ x) \gg put \circ f \\ &\equiv (lift\ x) \gg (lift \circ f) \end{aligned}$$

□

2.3 Additive

Proposition 2.3.1. *If s is a store and m is an additive monad, then with definitions as in 1.2.3, $s\ m$ is an additive monad.*

Proof. We have proved that $s\ m$ is a monad, and in [1] we proved that the three monoid laws apply, as any additive monad is a monoid. Therefore we need only prove the fourth and fifth laws for an additive monad. The fourth law is $\square \gg f \equiv \square$. Expanding, we get

$$\begin{aligned} \square \gg f &\equiv put \$ (get\ \square) \gg get\ f \\ &\equiv put \$ \odot \gg get\ f \\ &\equiv put \$ get\ f \\ &\equiv f \end{aligned}$$

And the fifth law $x \gg \square \equiv \square$ gives

$$\begin{aligned} x \gg \square &\equiv put \$ (get\ x) \gg get\ \square \\ &\equiv put \$ (get\ x) \gg \odot \\ &\equiv put\ \odot \\ &\equiv \square \end{aligned}$$

□

References

- [1] Julian Porter. “Distributed Storage with CloudHaskell”. In: (2011). URL: <http://jpembeddedsolutions.files.wordpress.com/2011/10/storage.pdf>.