CCLAW

Monad Transformers

01

May 20, 2021

02

Talking Points

- What/Why/How
- Monad Transfomers in baby-l4/lsp

What/Why/How

Monads (ugh)

monads are the interface (in haskell) for expressing useful effects [1]

Monad Transformers

A monad transformer is a context (defined for some monad) with well defined interactions with other monads.

[1] http://flint.cs.yale.edu/trifonov/cs629/WadlerMonadsForFP.pdf for the motivation behind monads in Haskell

What/Why/How

Monads don't play nice with each other

Unlike functions, Monads don't compose.

01 Functor Composition

(fmap f) . (fmap g) == fmap (f . g)

02 Monad Composition?

Does the composition of two monads give us something that behaves like a monad?

Compose?

Consider a data structure consisting of IO & Maybe,

```
data Composed = Compy IO Maybe a
```

The minimum complete definition of a monad consists of return & bind/join.
For Compy, it'll look something like:

```
join :: (Monad m1, Monad m2) => m1 (m2 (m1 (m2 a))) -> m1 (m2 a)
```

The join operation is possible if **m1 (m2 a) == m2 (m1 a)**, i.e.

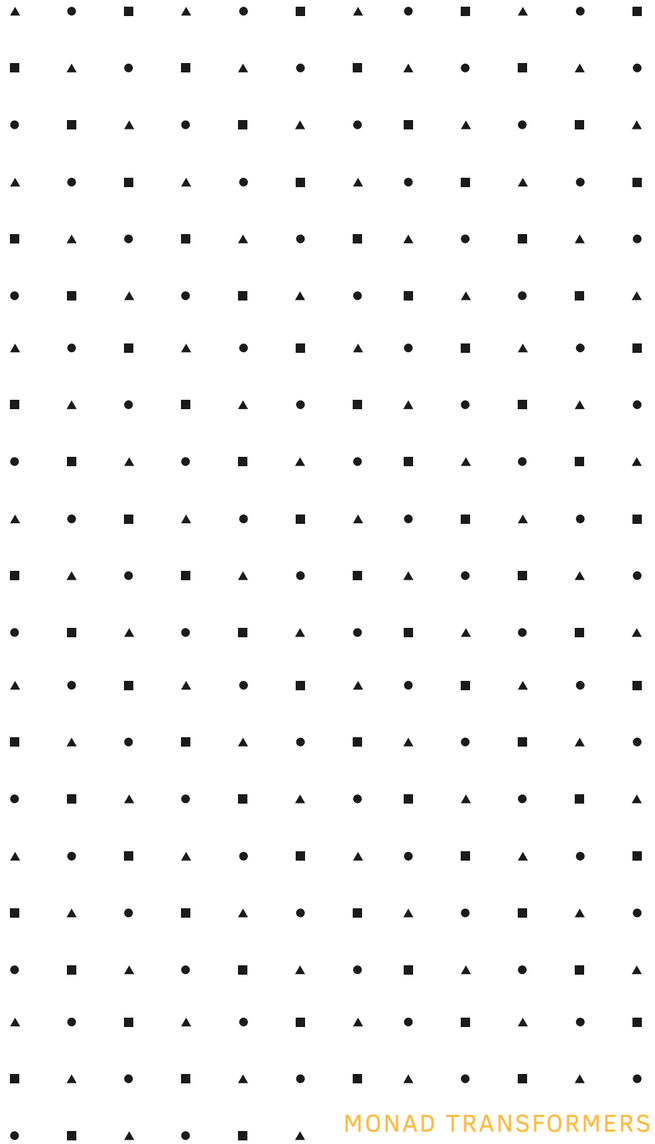
```
m1 (m2 (m1 (m2 a))) = m1 (m1 (m2 (m2 a)))
  = m1 (m2 a)
```

since m1 & m2 are necessarily monads

For Compy then, it means that we can have a well-defined monad instance if

```
IO (Maybe a) == Maybe (IO a)
```

A data structure consisting of 2 or more monads might not necessarily be itself a monad.



What/Why/How

Here's how to make monads play nice:

01 Take the precursor monad you'd like to use, and wrap it in a newtype

```
newtype MaybeT m a = MaybeT { runMaybeT :: m (Maybe a) }
```

02 Write a concrete instance of how you'd want it to interact with other monads

```
instance Monad m => Monad (MaybeT m) where
return = MaybeT . return . Just
(>>=) :: MaybeT m a -> (a -> MaybeT m b) -> MaybeT m b
x >>= f = MaybeT $ do maybe_val <- runMaybeT x
                      case maybe_val of
                          Nothing -> return Nothing
                          Just val -> runMaybeT $ f val
```

04

MONAD TRANSFORMERS

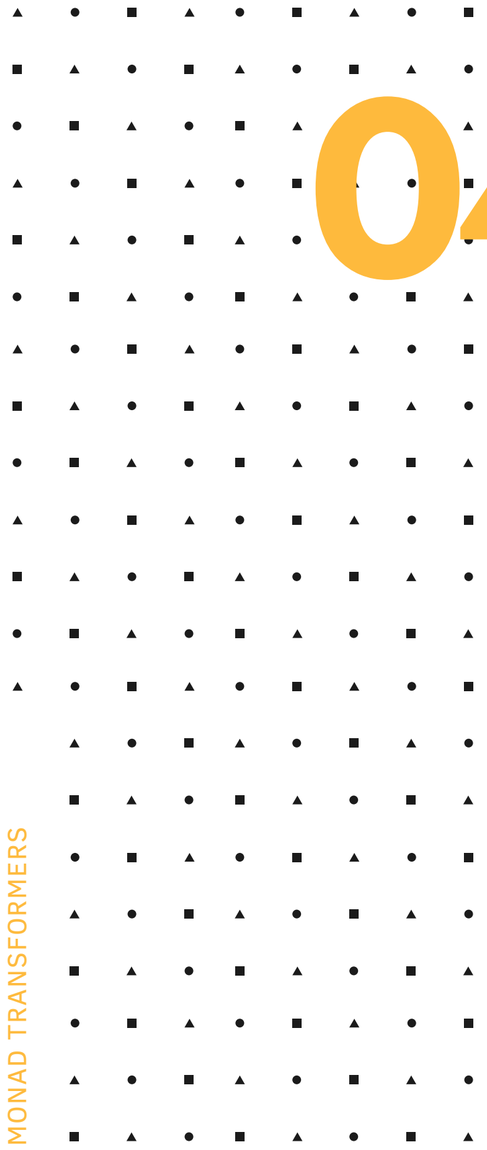
baby-l4/lsp

Language.LSP.Server

```
newtype LspT config m a =
        LspT { unLspT :: ReaderT (LanguageContextEnv config) m a }

type LspM config = LspT config IO
```

This module that contains the functions that deal with the running of the LSP server. LspT is but a record wrapper for the ReaderT Monad Transformer, so we will study that instead.



04

MONAD TRANSFORMERS

ReaderT (LanguageContextEnv config) IO a

ReaderT

newtype ReaderT r m a = ReaderT { runReaderT :: r -> m a }

This allows a computation which involves reading values from some environment of type ‘r’ and performing some monadic effect ‘m’ in order to get a value of type ’a’.

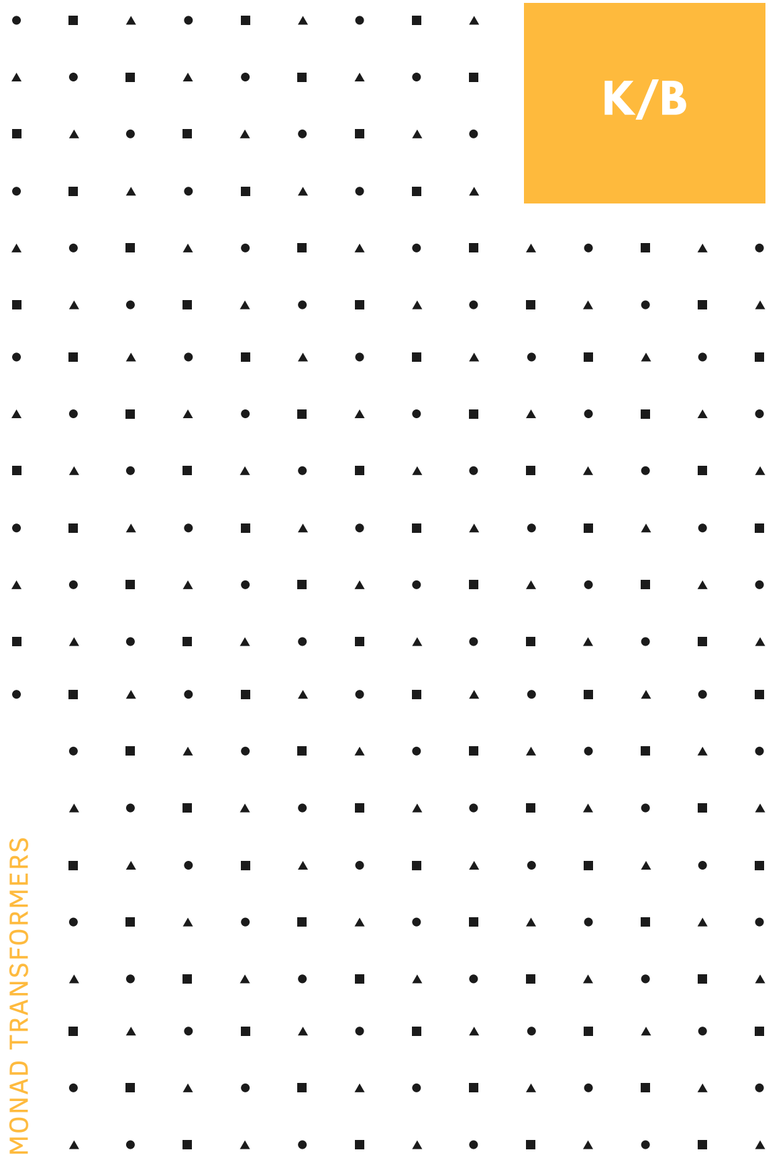
LanguageContextEnv

Contains the information about the lsp environment. This includes the event handlers, lsp client capabilities (conforming to the lsp specification),

IO

Handles the input/output of necessary data to and from the lsp server.


precursor
monad
base
monad
monad
transformer
POLESIE



Thanks for listening!

<< References >>

Haskellbook Chapter 25: Intro to Monad Transformers
Stack Overflow: Why are monads not closed under composition?
https://stackoverflow.com/a/5504669
Hoogle: lsp – Haskell Library for the Microsoft Language Server Protocol
https://hackage.haskell.org/package/lsp-1.2.0.0