"Real World" Application Programming with Haskell:

the RIO library & useful patterns

Alessandro Marrella @ Earnest Research

Dublin Haskell Meetup - 2019-11-12

Today we are talking about...

- ReaderT
- The ReaderT design pattern
- RIO

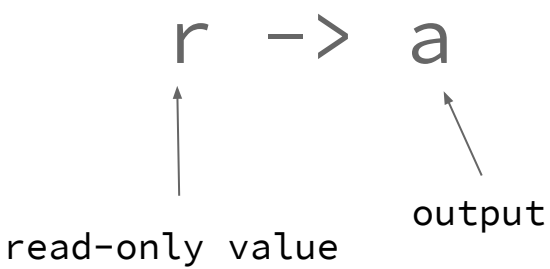
Reader

From hackage:

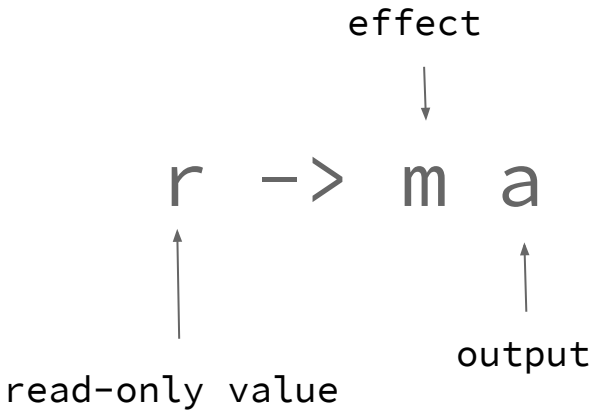
The Reader monad (also called the Environment monad) represents a computation, which can:

- read values from a shared environment
- pass values from function to function
- execute sub-computations in a modified environment

Reader is a function



ReaderT is an effectful function



In practice...

```
import Control.Monad.Trans.Reader
```

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

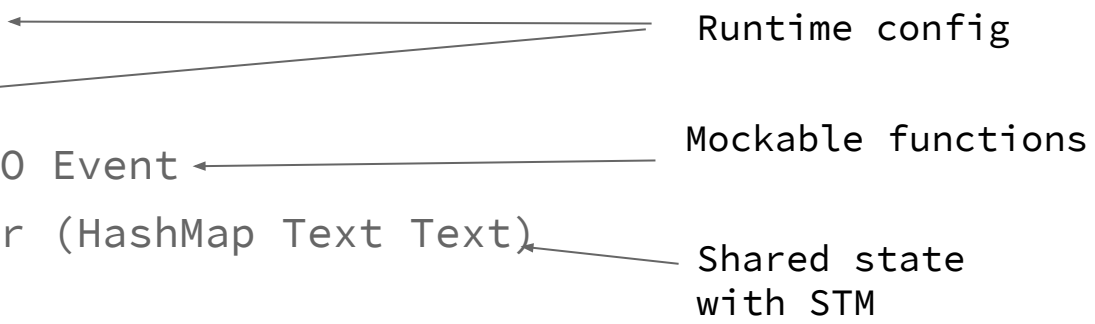
```
type Reader r a = ReaderT r Identity a
```

The ReaderT design pattern: define an env type

```
type App a = ReaderT Env IO a
```

What do we put in *Env*?

```
data Env = {
  awsEnv :: !AWS.Env
, dryRun :: !Bool
, readNextEvent :: !IO Event
, honeyObject :: !TVar (HashMap Text Text)
}
```



The ReaderT design pattern: initialize your app

```
main :: IO ()
main = do
  aws <- AWS.newEnv AWS.Discover
  dryRun <- readDryRunOpt
  initObject <- emptyObject
  let
   env =
    Env aws dryRun getNextEvent' initObject
  runReaderT program env

program :: ReaderT Env IO ()
```

The ReaderT design pattern: use the environment

Alternative 1: use Env directly

```
program :: ReaderT Env IO ()
program = do
  env <- ask
  event <- liftIO (readNextEvent env)
  liftIO $ print event
  liftIO $ atomically $ modifyTVar' (honeyObject env) (body event)
```

The ReaderT design pattern: use the environment

Alternative 2: capabilities

```
data Env = {
  awsEnv :: !AWS.Env
, dryRun :: !Bool
, readNextEvent :: IO Event
, honeyObject :: TVar (HashMap Text Text)
}

class HasAWS a where
  awsEnv :: AWS.Env
instance HasAWS Env where
  awsEnv Env = awsEnv env
```

The ReaderT design pattern: use the environment

Alternative 2: capabilities

```
program :: (HasHoney env, HasReadEvent env) => ReaderT env IO ()
program = do
  env <- ask
  event <- liftIO (readNextEvent env)
  liftIO $ print event
  liftIO $ atomically $ modifyTVar' (honeyObject env) (body event)
```

The ReaderT design pattern: use the environment

Alternative 3: optics

```
data Env = {
  awsEnv :: !AWS.Env
, dryRun :: !Bool
, readNextEvent :: IO Event
, honeyObject :: TVar (HashMap Text Text)
}

class HasAWS a where
  awsEnvL :: Lens' a AWS.Env
```

```
instance HasAWS Env where
  awsEnvL Env = lens awsEnv
   (\x y -> x { awsEnv = y })
```

The ReaderT design pattern: use the environment

Alternative 3: optics

```
program :: (HasHoney env, HasReadEvent env) => ReaderT env IO ()
program = do
  env <- ask
  readNextEvent <- view readNextEventL env
  honeyObject <- view honeyObjectL env
  event <- liftIO readNextEvent
  liftIO $ print event
  liftIO $ atomically $ modifyTVar' honeyObject (body event)
```

The ReaderT design pattern: use the environment

Alternative 3: optics+

```
program :: (HasHoney env, HasReadEvent env) => ReaderT env IO ()
program = do
  readNextEvent <- view readNextEventL
  honeyObject <- view honeyObjectL
  event <- liftIO readNextEvent
  liftIO $ print event
  liftIO $ atomically $ modifyTVar' honeyObject (body event)
```

Can we simplify it further?

Meet RIO:

newtype RIO env a = RIO (ReaderT env IO a)

- Like ReaderT, but with IO hardcoded (sensible default for applications)
- Less polymorphic -> better error messages :)
- Shorter to type :P

RIO aims to be a "standard library" for Haskell

- Custom prelude with partial/unsafe functions hidden

```
{-# LANGUAGE NoImplicitPrelude #-}
import RIO
```

- Opinionated, simple logging

```
sayHelloRIO :: HasLogFunc env => RIO env ()
sayHelloRIO = logInfo "Hello World!"
```

RIO: Define an env type

```
data Env = {
  awsEnv :: !AWS.Env
, dryRun :: !Bool
, readNextEvent :: !IO Event
, honeyObject :: !TVar (HashMap Text Text)
, appLogFunc :: !LogFunc
}

instance HasLogFunc Env where
  logFuncL = lens appLogFunc (\x y -> x { appLogFunc = y })
```

RIO: Initialize your app

```
main :: RIO App a
main = do
  logOptions' <- logOptionsHandle stderr False
  let logOptions = setLogUseTime True $ setLogUseLoc True logOptions'
  withLogFunc logOptions $ \logFunc -> do
    let env = Env aws dryRun getNextEvent’ initObject logFunc
    runRIO env program
```

RIO: use the environment

Alternative 1: use Env directly

```
program :: RIO Env ()
program = do
  env <- ask
  event <- liftIO (readNextEvent env)
  liftIO $ print event
  liftIO $ atomically $ modifyTVar' (honeyObject env) (body event)
```

RIO: use the environment

Alternative 2: capabilities

```
program :: (HasHoney env, HasReadEvent env) => RIO env ()
program = do
  env <- ask
  event <- liftIO (readNextEvent env)
  liftIO $ print event
  liftIO $ atomically $ modifyTVar' (honeyObject env) (body event)
```

RIO: use the environment

Alternative 3: optics

```
program :: (HasHoney env, HasReadEvent env) => RIO env ()
program = do
  env <- ask
  readNextEvent <- view readNextEventL env
  honeyObject <- view honeyObject env
  event <- liftIO readNextEvent
  liftIO $ print event
  liftIO $ atomically $ modifyTVar' honeyObject (body event)
```

RIO: use the environment

Alternative 3: optics+

```
program :: (HasHoney env, HasReadEvent env) => RIO env()
program = do
  readNextEvent <- view readNextEventL
  honeyObject <- view honeyObjectL
  event <- liftIO readNextEvent
  liftIO $ print event
  liftIO $ atomically $ modifyTVar' honeyObject (body event)
```

RIO QuickStart

with stack: **stack new myapp rio**

This command will create a new RIO app skeleton for you to use.

- app/ contains the main app logic
- src/ the library logic
- test/ the tests

Example repository using RIO

https://github.com/EarnestResearch/k8s-volume-discovery/

References:

The ReaderT design pattern:
https://www.fpcomplete.com/blog/2017/06/readert-design-pattern

The RIO monad:
https://www.fpcomplete.com/blog/2017/07/the-rio-monad

The RIO standard library:
https://tech.fpcomplete.com/haskell/library/rio

Stackage docs:

https://www.stackage.org/lts-14.14/package/rio-0.1.12.0

Thank you!

Any questions?