Give me freedom!

Or let me forget

Joseph Tel Abrahamson

May 20, 2015

Table of Contents

`Free` is a noun

The Free monad

```
data Free f a
  = Return a
  | Free (f (Free f a))
```

Construction and destruction

- When `f` is a `Functor`, `Free f` is a `Monad`

Construction and destruction

- For any value `x :: f a` we have `lift x :: Free f a`

Construction and destruction

- For any
  1. `Monad m` and
  2. *interpretation* of `f` into `m`, `phi :: forall x. f x -> m x`,

  we have `fold phi :: forall a . Free f a -> m a`.

e.g.

```
lift :: Functor f ⇒ f a → Free f a
fold :: Monad m ⇒ (∀ x . f x → m x) → (∀ x . Free f x → m x)
```

Very nice embedded DSLs… for *less*!

```
data TeletypeF a
  = PutStrLn String a
  | GetLine (String → a)
    deriving ( Functor )

type Teletype = Free TeletypeF

putStrLnTT :: String → Teletype ()
putStrLnTT line = lift (PutStrLn line ())

getLineTT :: Teletype String
getLineTT = lift (GetLine id)
```

Very nice embedded DSLs… for *less*!

```
echoTT :: Teletype ()
echoTT = forever $ do
  line ← getLineTT
  putStrLineTT line
```

Very nice embedded DSLs... for *less*!

```
interp :: TeletypeF a → IO a
interp x = case x of
  PutStrLn line a → putStrLn line  »  return a
  GetLine next → do
    line ← getLine
    return (next line)

echoIO :: IO ()
echoIO = fold interp echoTT
```

Free is an adjective

Free

Other Freedoms we have

Other Freedoms we have

1. Free `Monad`s

Other Freedoms we have

1. Free `Monad`s
2. Free `MonadPlus`es

Other Freedoms we have

1. Free `Monad`s
2. Free `MonadPlus`es
3. Free `Applicative`s

Other Freedoms we have

1. Free `Monad`s
2. Free `MonadPlus`es
3. Free `Applicative`s
4. Free `Alternative`s

Other Freedoms we have

1. Free `Monad`s
2. Free `MonadPlus`es
3. Free `Applicative`s
4. Free `Alternative`s
5. Free `Monoid`s (“lists”)

Other Freedoms we have

1. Free `Monad`s
2. Free `MonadPlus`es
3. Free `Applicative`s
4. Free `Alternative`s
5. Free `Monoid`s (“lists”)
6. Free `Category`s

Other Freedoms we have

1. Free `Monad`s
2. Free `MonadPlus`es
3. Free `Applicative`s
4. Free `Alternative`s
5. Free `Monoid`s (“lists”)
6. Free `Category`s
7. …

What does it mean to be free?

What does it mean to be free?

This is the wrong question!

Free is a verb

What is `Free`, really?

```
> : kind Free_Monad
```

What is `Free`, really?

```
> : kind Free_Monad
Free_Monad :: (* -> *) -> (* -> *)
```

What is `Free`, really?

But really more like...

$$> : \texttt{kind}\ \texttt{Free}_{\texttt{Monad}}$$

$$\texttt{Free}_{\texttt{Monad}}\ \texttt{::}\ (\star \to \star)\,\texttt{Functor} \to (\star \to \star)\,\texttt{Monad}$$

What is Free, really?

But really more like...

$$> : \texttt{kind Free}_{\texttt{Monad}}$$

$$\texttt{Free}_{\texttt{Monad}} \texttt{ :: } (\star \to \star)\,\texttt{Functor} \to (\star \to \star)\,\texttt{Monad}$$

```
-- remember...
instance Functor f ⇒ Monad (Free f)
```

Freedom is a process

```
data IsAFunctor f
  = IsAFunctor
    { fmap :: ∀ a b . (a → b) → (f a → f b)
    }

data IsAMonad f
  = IsAMonad
    { return :: ∀ a   . a → f a
    , bind   :: ∀ a b . (a → f b) → (f a → f b)
    }

free :: IsAFunctor f → IsAMonad (Free f)
```

Other “Freedoms” we have

1. Free `Monad`s
2. Free `MonadPlus`es
3. Free `Applicative`s
4. Free `Alternative`s
5. Free `Monoid`s
6. Free `Category`s

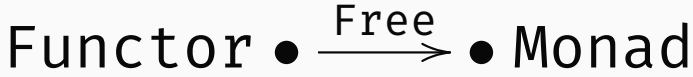
Other “Freedoms” we have

1. Free `Monad`s
2. Free `MonadPlus`es
3. Free `Applicative`s
4. Free `Alternative`s
5. Free `Monoid`s
6. Free `Category`s

All *underdefined*. Freedom goes from a source to a target!

A picture of “Free monads”

$\texttt{Free}_{\texttt{Monad}}$



A picture of “Free monads”

$\texttt{Free}_{\texttt{Monad}}$

$$\texttt{Functor} \bullet \xrightarrow{\texttt{Free}} \bullet \texttt{Monad}$$

$\texttt{List}$

$$\texttt{Hask} \bullet \xrightarrow{\texttt{Free}} \bullet \texttt{Monoid}$$

A picture of “Free monads”

$\texttt{Free}_{\texttt{Monad}}$

$$\texttt{Functor} \bullet \xrightarrow{\texttt{Free}} \bullet \texttt{Monad}$$

$\texttt{List}$

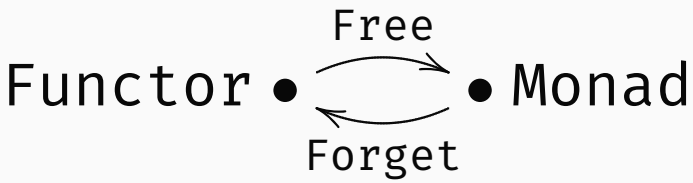
$$\texttt{Hask} \bullet \xrightarrow{\texttt{Free}} \bullet \texttt{Monoid}$$

$\texttt{Coyoneda}$

$$\texttt{Hask}_{(\star \to \star)} \bullet \xrightarrow{\texttt{Free}} \bullet \texttt{Functor}$$

Gaining Freedom and Forgetting it all


Free
Functor
Monad


Free
Functor
Monad
Forget

It’s easy to forget

$$\texttt{Forget}_{\texttt{Monad}} \;\texttt{::}\; (\star \to \star)_{\texttt{Monad}} \to (\star \to \star)_{\texttt{Functor}}$$

```
type Forget f a = f a

forget :: IsAMonad f → IsAFunctor (Forget f)
forget (IsAMonad { bind, return }) = IsAFunctor fmap where
  fmap f = bind (return . f)
```

Too much to ask for

$$(\texttt{Free} \circ \texttt{Forget})(M) \neq M$$

$$(\texttt{Forget} \circ \texttt{Free})(F) \neq F$$

Just right

If

$$M = \mathtt{Free}(F)$$

for some $\mathtt{Functor}$ F, then

$$(\mathtt{Free} \circ \mathtt{Forget})(M) = M$$

Just right

If

$$F = \mathtt{Forget}(M)$$

for some **Monad** M, then

$$(\mathtt{Forget} \circ \mathtt{Free})(F) = F$$

Adjunctions

$$F \circ G \circ F = Id$$

$$G \circ F \circ G = Id$$