2013年3月7日
15:51

```
newtype State s a = State {runState :: s -> (a, s)}

get :: State s s
put :: s -> State s ()
```

```
data Tree a = Tip a | Bin (Tree a) (Tree a)

label :: Tree a -> State [b] (Tree (a, b))
label (Tip a) = do {(b:y) <- get; put y; return (Tip (a, b))}
label (Bin u v) = do { u' <- label u
                     ; v' <- label v
                     ; return (Bin u' v') }
```

```
labels :: Tree (a, b) -> [b]
labels (Tip (a, b)) = [b]
labels (Bin u v) = labels u ++ labels v

runState (label t) xs = (u, ys)
    => labels u ++ ys = xs
```

```
class Functor m => Idiom m where
    pure :: a -> m a
    (⊛)  :: m (a -> b) -> m a -> m b
```

and four laws...

```
instance Monad m => Idiom m where
    pure a    = return a
    mf ⊛ ma = do { f <- mf; a <- ma; return (f a) }
```

Reader monad $\rho \to a$

```
instance Idiom (ρ →) where
    pure a = λρ → a
    mf ⊛ ma = λρ → (mf ρ) (ma ρ)
```

"Napierian idiom" $a^?$

Stream $a \cong Nat \to a$

Pair $a \cong Bool \to a$

```
newtype K b a = K { unK :: b }
instance Monoid b => Idiom (K b) where
    pure a    = K mempty
    K b ⊛ K b' = K (b `mappend` b')
```

---

pure id ⊛ ma = ma

((pure (.) ⊛ mf) ⊛ mg) ⊛ ma = mf ⊛ (mg ⊛ ma)

pure f ⊛ pure a = pure (f a)

mf ⊛ pure a = pure (λf → f a) ⊛ mf

```
instance Functor (K b) where
  fmap f (K b) = K b

(>>=) :: m a -> (a -> m b) -> m b
```

```
class Functor t => Traversable t where
  traverse :: Idiom m => (a -> m b) -> t a -> m (t b)

instance Traversable Tree where
  traverse f (Tip a) = pure Tip <*> f a
  traverse f (Bin t u)
    = pure Bin <*> traverse f t <*> traverse f u
```

```
distr :: (Traversable t, Idiom m) =>
         t (m a) -> m (t a)
distr = traverse id

contents :: Traversable t => t a -> [a]
contents = unk . traverse (\a -> K [a])
```

```
adorn :: a -> State [b] (a,b)
adorn a = do { b:y <- get; put y; return (a,b) }

label = traverse adorn
```

$m \cdot n \qquad m\,(n\,a)$

$(\odot) :: (\text{Idiom } m, \text{Idiom } n) \Rightarrow$
$(b \to n\ c) \to (a \to m\ b) \to (a \to (m \cdot n)\ c)$

$\text{traverse}\ (g \odot f) = \text{traverse}\ g \odot \text{traverse}\ f$

$\varphi :: m\ a \to n\ a \qquad$ idioms m, n

$\varphi\ (\text{pure}_m\ a) = \text{pure}_n\ a$

$\varphi\ (mf \circledast_m ma) = \varphi\ mf \circledast_n \varphi\ ma$

is an idiom morphism

$\varphi \cdot \text{traverse}_m\ f = \text{traverse}\ (\varphi \cdot f)$

$\Rightarrow$

$\text{traverse}\ \text{pure} = \text{pure}$

$\text{traverse}\ g \bullet \text{traverse}\ f = \text{traverse}\ (g \bullet f)$ (Kleisli)

commutative monad

$\text{unlabel} :: \text{Tree}\ (a,b) \to \text{State}\ [b]\ (\text{Tree}\ a)$

$\text{unlabel} = \text{traverse}\ \text{strip}$

$\text{strip}\ (a,b) = \text{do}\ \{ y \leftarrow \text{get}; \text{put}\ (b:y); \text{return}\ a \}$

---

unlabel ⊙ label ? —

unlabel • label?)

no: strip ⊙ adom ≠ pure!

(strip • adom = return

no, not commutative monad!

---

newtype B m a = B {unB :: m a}

instance Idiom m ⇒ Idiom (B m) where

pure a = B (pure a)

B mf ⊛ B ma = B (pure (flip ($)) ⊛ ma ⊛ mf)

traverse f = unB . traverse (B . f)

traverse strip • traverse adom = return

---

f • g = return ⇒ traverse f • traverse g = return