```
class Idiom i where
  ii :: x -> i x
  <%> :: i (s -> t) -> i s -> i t

class IFunctor f where
  imap :: Idiom i => (s -> i t) -> f s -> i (f t)

class Monoid z where
  zero :: z
  <+> :: z -> z -> z
```

Idiom notation

$$[\![f\ a_1 \ldots a_n]\!]$$

$$\hookrightarrow ii\ f <\%> a_1 <\%> \ldots <\%> a_n$$

$$\mathrm{i}[\![\ \ldots\ ]\!]\mathrm{i}$$

```
instance Idiom [] where
  ii = repeat
  (<%>) = zipWith ($)

type Box = [[Char]]

juxV = (++)                (#-#)
~~juxH~~                      (#|#)
juxH xss yss
  = ⟦(++) xss yss⟧

transpose :: [[x]] -> [[x]]
transpose [] = ⟦[]⟧
transpose (xs : xss) =
  ⟦(:) xs (transpose xss)⟧
instance IFunctor [] where
  imap f [] = ⟦[]⟧
  imap f (x:xs) = ⟦(:) (fx) (imap f xs)⟧
```

```
instance Idiom (r ->) where
  ii x ~~const~~ y = x
(rst <*> rs) r = rst r (rs r)
```

```
instance Monoid z => Monoid (r -> z) where
  zero = ⟦zero⟧
  x <+> y = ⟦(<+>) x y⟧
```

Exercise: given
  instance Functor (r ->)
  solve the Halting Problem

```
newtype Acc z x = Acc {accumulated :: z}

instance Monoid z => Idiom (Acc z) where
  ii _ = Acc zero
  Acc fz <$> Acc sz = Acc (fz <+> sz)


icrush :: (IFunctor f, Monoid z) =>
          (x -> z) -> f x -> z
icrush c = accumulated . (imap (Acc . c))


itrail :: (IFunctor f, Idiom i, Monoid (i x)) =>
          f x -> i x
itrail = icrush ii
```

Exercise: count the Haskell library
functions which are just
itrail up to newtype
isos.

```haskell
class Functor i => Idiom i where
  unit :: i ()
  mult :: i s -> i t -> i (s,t)

{-  ii x = fmap (const x) unit
    fi <%> si = fmap ($) (mult fi si)  -}
```

```haskell
mult unit ti = fmap ((),) ti
mult si unit = fmap (,()) si
mult si (mult ti ui) = fmap assoc
                         (mult (mult si ti) ui)
```

```haskell
mult (ii s) ti = fmap swap (mult ti (ii s))
mult si (ii t) = fmap swap (mult (ii t) si)
```

```haskell
mult (ii s) (ii t) = ii (s, t)
```

$[\![p\ i_1 \ldots i_n]\!]$