Coding by Shape

□ △ ▽ ○

Andy Scott

August 13, 2019

who am I?

who am I?

- contributor to a few Scala libs; maintainer of even fewer
- fan of graphs, trees, recursive structures
- also dogs, hiking, coffee, books, music
- work on Scala & Bazel at Stripe

who am I?

- contributor to a few Scala libs; maintainer of even fewer
- fan of graphs, trees, recursive structures
- also dogs, hiking, coffee, books, music
- work on Scala & Bazel at Stripe

- github https://github.com/andyscott
- twitter https://twitter.com/andy·g·scott

Goals

- learn some diagramming basics

Goals

- learn some diagramming basics
- code $\rightarrow$ diagrams

Goals

- learn some diagramming basics
- code $\rightarrow$ diagrams
- diagrams $\rightarrow$ code

Goals

- learn some diagramming basics
- code $\rightarrow$ diagrams
- diagrams $\rightarrow$ code
- keep it simple but cover useful concepts

Goals

- learn some diagramming basics
- code $\rightarrow$ diagrams
- diagrams $\rightarrow$ code
- keep it simple but cover useful concepts

the real treasure was the friends we learned along the way

— Jon Pretty

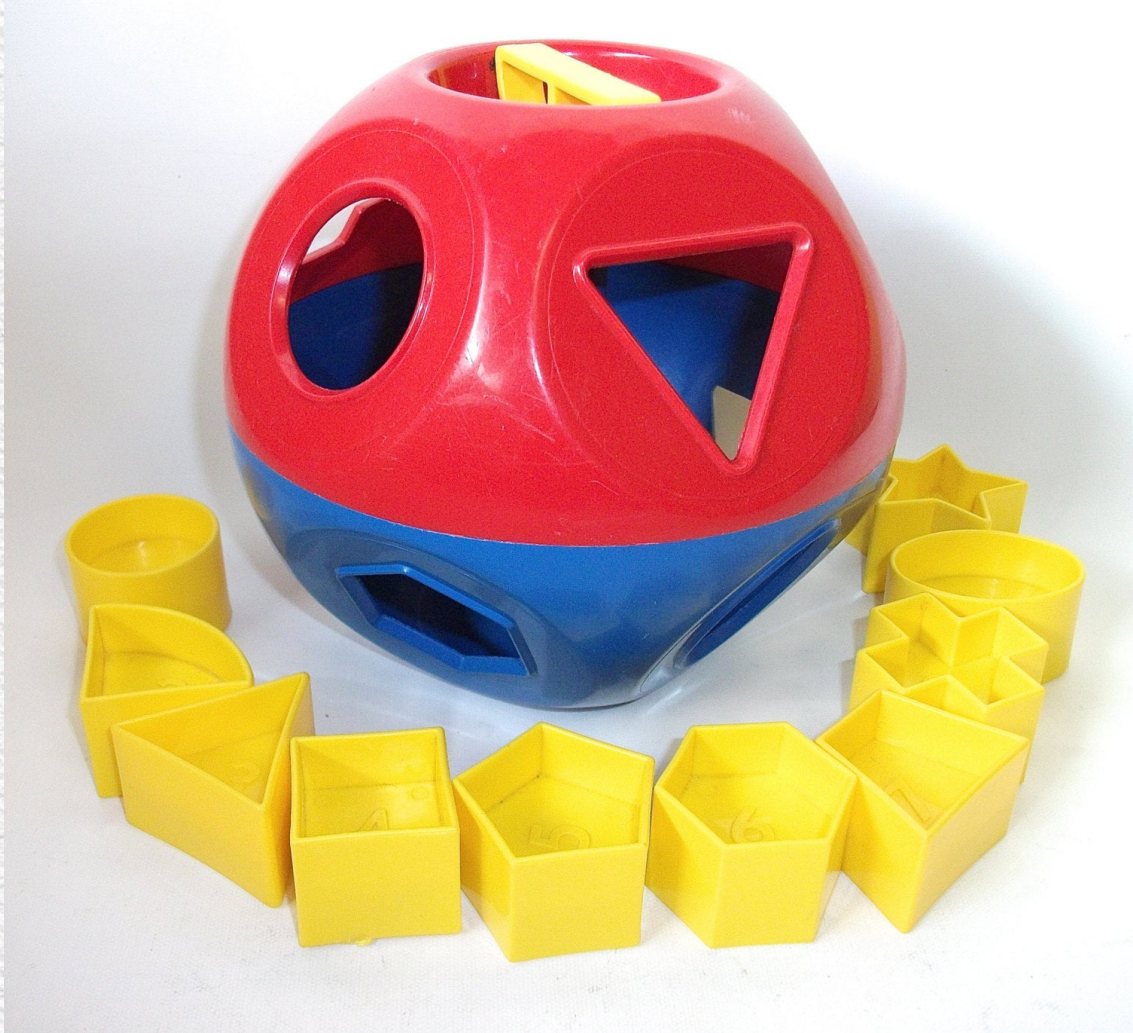
Goals

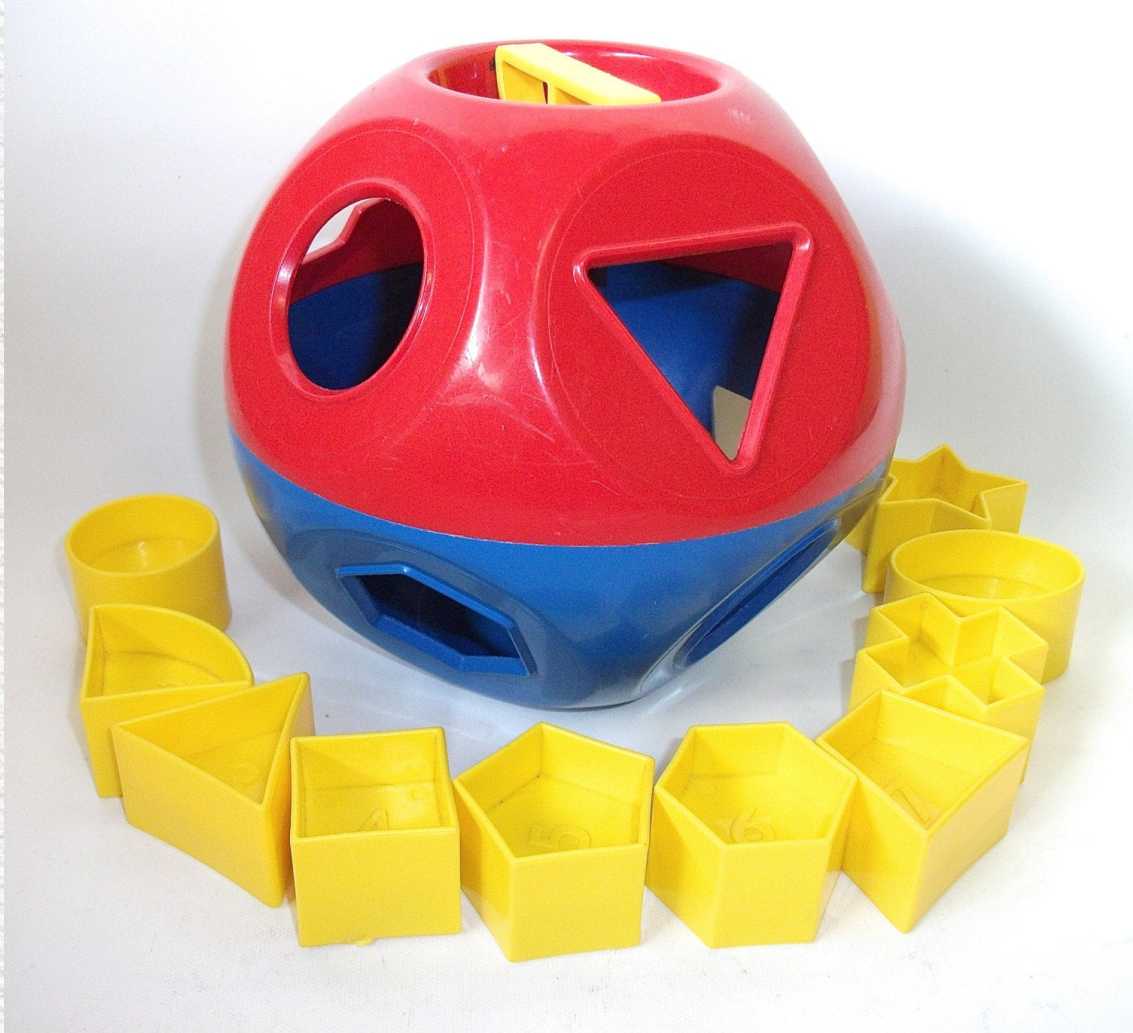
- learn some diagramming basics
- code $\rightarrow$ diagrams
- diagrams $\rightarrow$ code
- keep it simple but cover useful concepts

the real treasure was the ~~*friends*~~ *folds* *we learned along the way*

— Jon Pretty

...

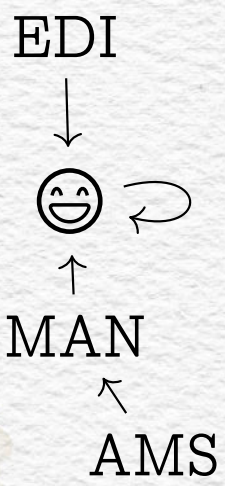


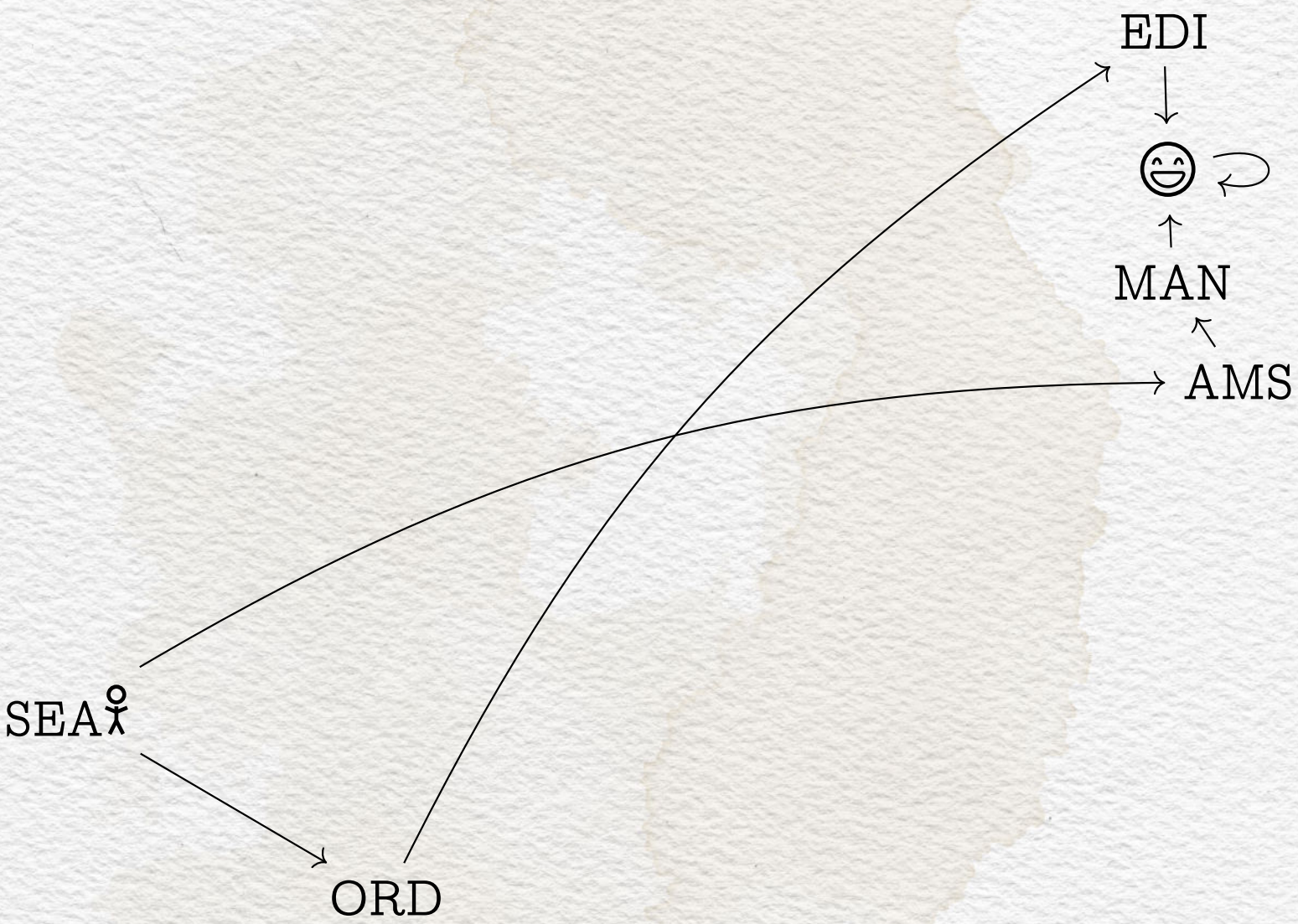


Tupperware Shape-O

Commutative Diagram 101

- it's a picture of composition
- nodes are objects
- edges are morphisms


EDI
MAN
AMS


EDI
MAN
AMS
SEA
ORD

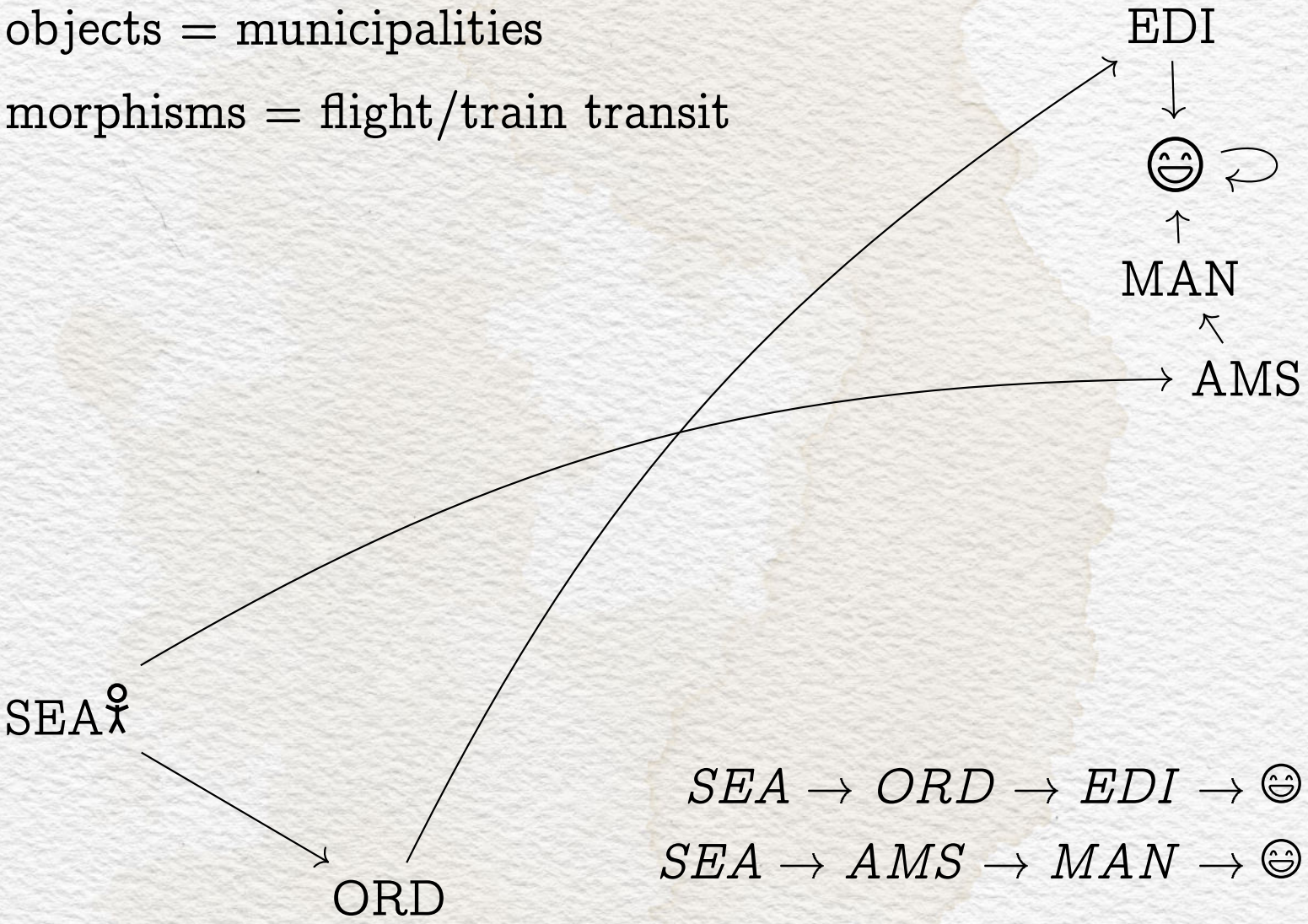
objects = municipalities

morphisms = flight/train transit

$$
\begin{tikzcd}
 & & \mathrm{EDI} \arrow[d] \\
 & & \text{😄} \arrow[loop right] \\
 & & \mathrm{MAN} \arrow[u] \\
 & & \mathrm{AMS} \arrow[ul] \\
\mathrm{SEA} \arrow[urr, bend left] \arrow[dr] & & \\
 & \mathrm{ORD} \arrow[uur] &
\end{tikzcd}
$$

objects = municipalities

morphisms = flight/train transit



$SEA \to ORD \to EDI \to$ 😄

$SEA \to AMS \to MAN \to$ 😄

objects = municipalities

morphisms = flight/train transit

$$\begin{tikzcd}
 & & \text{EDI} \arrow[d] \\
 & & \text{😄} \arrow[loop right] \\
 & & \text{MAN} \arrow[u] \\
 & & \text{AMS} \arrow[ul] \arrow[uull, bend left] \\
\text{SEA} \arrow[urrr, bend left] \arrow[dr] & & \\
 & \text{ORD} \arrow[uur] &
\end{tikzcd}$$

$$SEA \to ORD \to EDI \to \text{😄}$$

$$SEA \to AMS \to MAN \to \text{😄}$$

$$SEA \to \text{😄}$$

imagine...

imagine...

- a list of strings

imagine...

- a list of strings
- computing the total length of all of the strings

$List[String]$

$List[String]$ Int

$List[String]$ Int

```
def f(x: List[String]): List[Int] = x.map(_.length)
```

$$List[String] \xrightarrow{\quad f \quad} List[Int] \qquad\qquad Int$$

```
def f(x: List[String]): List[Int] = x.map(_.length)
```

$$List[String] \xrightarrow{f} List[Int] \xrightarrow{g} Int$$

```
def f(x: List[String]): List[Int] = x.map(_.length)
def g(x: List[Int]): Int = x.foldLeft(0)(_ + _)
```

$$\begin{tikzcd} List[String] \arrow[r, "f"] \arrow[rr, bend right, "h"'] & List[Int] \arrow[r, "g"] & Int \end{tikzcd}$$

```
def f(x: List[String]): List[Int] = x.map(_.length)
def g(x: List[Int]): Int = x.foldLeft(0)(_ + _)
```

$$\begin{tikzcd} List[String] \arrow[r, "f"] \arrow[rr, bend right, "h"'] & List[Int] \arrow[r, "g"] & Int \end{tikzcd}$$

```
def f(x: List[String]): List[Int] = x.map(_.length)
def g(x: List[Int]): Int = x.foldLeft(0)(_ + _)

def h(x: List[String]): Int = g(f(x))
```

$$\begin{tikzcd} List[String] \arrow[r, "f"] \arrow[rr, bend right, "g \circ f"'] & List[Int] \arrow[r, "g"] & Int \end{tikzcd}$$

```
def f(x: List[String]): List[Int] = x.map(_.length)
def g(x: List[Int]): Int = x.foldLeft(0)(_ + _)

def h(x: List[String]): Int = g(f(x))
                        /* or */ g compose f
```

options

options

```scala
sealed trait Option[+A]

case class  Some[+A](value: A) extends Option[A]
case object None extends Option[Nothing]
```

options

```scala
sealed trait Option[+A]

case class  Some[+A](value: A) extends Option[A]
case object None extends Option[Nothing]
```

- put a value in an Option

options

```
sealed trait Option[+A]

case class  Some[+A](value: A) extends Option[A]
case object None extends Option[Nothing]
```

- put a value in an Option

$$A \xrightarrow{\quad some \quad} Option[A]$$

options

```
sealed trait Option[+A]

case class  Some[+A](value: A) extends Option[A]
case object None extends Option[Nothing]
```

- put a value in an Option

$$A \xrightarrow{\quad some \quad} Option[A]$$

```
def some[A](a: A): Option[A] = Some(a)
```

```scala
sealed trait Option[+A]

case class  Some[+A](value: A) extends Option[A]
case object None extends Option[Nothing]
```

```scala
sealed trait Option[+A]

case class  Some[+A](value: A) extends Option[A]
case object None extends Option[Nothing]
```

- create an empty Option

```scala
sealed trait Option[+A]

case class  Some[+A](value: A) extends Option[A]
case object None extends Option[Nothing]
```

- create an empty Option

```scala
def none[A]: Option[A] = None
```

```scala
sealed trait Option[+A]

case class  Some[+A](value: A) extends Option[A]
case object None extends Option[Nothing]
```

- create an empty Option

```scala
def none[A]: Option[A] = None
```

$$1 \xrightarrow{\quad none \quad} Option[A]$$

$$A \xrightarrow{\quad some \quad} Option[A]$$

$$1 \xrightarrow{\quad none \quad} Option[A]$$

$$A \xrightarrow{\quad some \quad} Option[A]$$

$$1 \xrightarrow{\quad none \quad} Option[A]$$

$$1 + A \xrightarrow{\quad [empty, some] \quad} Option[A]$$

$$A \xrightarrow{\quad some \quad} Option[A]$$

$$1 \xrightarrow{\quad none \quad} Option[A]$$

$$1 + A \xrightarrow{\quad [empty, some] \quad} Option[A]$$

$$1 + A \xleftarrow{\quad ??? \quad} Option[A]$$

```scala
def fold[A, B](x: Option[A])
              (ifEmpty: B)(f: A => B): B = ???
```

```scala
def fold[A, B](x: Option[A])
              (ifEmpty: B)(f: A => B): B = ???


val x0   : Option[Int] = Some(2)
val res0: Int          = x0.fold(0)(_ * 10)
// 20

val x1   : Option[Int] = None
val res1: Int          = x1.fold(0)(_ * 10)
// 0
```

```scala
val x0   : Option[Int] = Some(2)
val res0: Int          = x0.fold(0)(_ * 10)
// 20

val x1   : Option[Int] = None
val res1: Int          = x1.fold(0)(_ * 10)
// 0

def fold[A, B](x: Option[A])
              (ifEmpty: B)(f: A => B): B =
  x match {
    case None     => ifEmpty
    case Some(a) => f(a)
  }
```

```scala
def fold[A, B](x: Option[A])
              (ifEmpty: B)(f: A => B): B =
  x match {
    case None     => ifEmpty
    case Some(a) => f(a)
  }
```

$$\mathit{Option}[A] \xrightarrow{\mathit{fold}} B$$

$$1 \xrightarrow{\mathit{ifEmpty}} B$$

$$A \xrightarrow{f} B$$

```scala
def fold[A, B](x: Option[A])
              (ifEmpty: B)(f: A => B): B =
  x match {
    case None     => ifEmpty
    case Some(a) => f(a)
  }
```

$$Option[A] \xrightarrow{fold} B$$

$$1 + A \xrightarrow{[ifEmpty, f]} B$$

```scala
def fold[A, B](x: Option[A])
              (ifEmpty: B)(f: A => B): B =
  x match {
    case None     => ifEmpty
    case Some(a) => f(a)
  }
```

$$\begin{tikzcd}
Option[A] \arrow[rr, "fold"] & & B \\
& 1 + A \arrow[ur, "{[ifEmpty, f]}"'] &
\end{tikzcd}$$

```
def fold[A, B](x: Option[A])
              (ifEmpty: B)(f: A => B): B =
  x match {
    case None     => ifEmpty
    case Some(a) => f(a)
  }
```

$$\begin{tikzcd}
Option[A] \arrow[rr, "fold"] \arrow[dr, "match"'] & & B \\
& 1 + A \arrow[ur, "{[ifEmpty, f]}"'] &
\end{tikzcd}$$

lists

lists

```scala
sealed trait List[A]
case object Nil extends List[Nothing]
case class ::[A]( // or 'Cons'
  head: A,
  tail: List[A]) extends List[A]
```

lists

```
sealed trait List[A]
case object Nil extends List[Nothing]
case class ::[A]( // or 'Cons'
  head: A,
  tail: List[A]) extends List[A]
```

$$1 \xrightarrow{\quad empty \quad} List[A]$$

lists

```
sealed trait List[A]
case object Nil extends List[Nothing]
case class ::[A]( // or 'Cons'
  head: A,
  tail: List[A]) extends List[A]
```

$$1 \xrightarrow{\quad empty \quad} List[A]$$

$$A \times List[A] \xrightarrow{\quad cons \quad} List[A]$$

lists

```scala
sealed trait List[A]
case object Nil extends List[Nothing]
case class ::[A]( // or 'Cons'
  head: A,
  tail: List[A]) extends List[A]
```

$$1 \xrightarrow{\quad empty \quad} List[A]$$

$$A \times List[A] \xrightarrow{\quad cons \quad} List[A]$$

```scala
def empty[A]: List[A] = Nil
def cons[A](head: A, tail: List[A]): List[A]
  = head :: tail
```

$$1 \xrightarrow{\quad empty \quad} List[A]$$

$$A \times List[A] \xrightarrow{\quad cons \quad} List[A]$$

$$1 \xrightarrow{\quad empty \quad} List[A]$$

$$A \times List[A] \xrightarrow{\quad cons \quad} List[A]$$

$$1 + A \times List[A] \xrightarrow{\quad [empty, cons] \quad} List[A]$$

$$1 \xrightarrow{\quad empty \quad} List[A]$$

$$A \times List[A] \xrightarrow{\quad cons \quad} List[A]$$

$$1 + A \times List[A] \xrightarrow{\quad [empty, cons] \quad} List[A]$$

$$1 + A \times List[A] \xleftarrow{\quad match \quad} List[A]$$

```
def foldRight[A, B](x: List[A])
                   (z: B)(f: (A, B) => B): B = ???
```

```
def foldRight[A, B](x: List[A])
                   (z: B)(f: (A, B) => B): B = ???
```

$$List[A] \xrightarrow{foldRight} B$$

$$1 \xrightarrow{z} B$$

$$A \times B \xrightarrow{f} B$$

```
def foldRight[A, B](x: List[A])
                    (z: B)(f: (A, B) => B): B = ???
```

$$List[A] \xrightarrow{foldRight} B$$

$$1 + A \times B \xrightarrow{[z,f]} B$$

```
def foldRight[A, B](x: List[A])
                    (z: B)(f: (A, B) => B): B = ???
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] & B \\
& 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

```
def foldRight[A, B](x: List[A])
                    (z: B)(f: (A, B) => B): B = ???
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

```
def foldRight[A, B](x: List[A])
                    (z: B)(f: (A, B) => B): B = ???
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "???"] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

$$1 + A \times List[A] \xrightarrow{???} 1 + A \times B$$

$$1 + A \times List[A] \xrightarrow{???} 1 + A \times B$$

$$1 + A \times List[A] \xrightarrow{[id, ???]} 1 + A \times B$$

$$1 + A \times List[A] \xrightarrow{???} 1 + A \times B$$

$$1 + A \times List[A] \xrightarrow{[id, ???]} 1 + A \times B$$

$$1 + A \times List[A] \xrightarrow{[id, \langle id, ??? \rangle]} 1 + A \times B$$

$$1 + A \times List[A] \xrightarrow{???} 1 + A \times B$$

$$1 + A \times List[A] \xrightarrow{[id,???]} 1 + A \times B$$

$$1 + A \times List[A] \xrightarrow{[id,\langle id,???\rangle]} 1 + A \times B$$

$$1 + A \times List[A] \xrightarrow{id+id\times???} 1 + A \times B$$

```
def foldRight[A, B](x: List[A])
                    (z: B)(f: (A, B) => B): B = ???
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "???"] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

```
def foldRight[A, B](x: List[A])
                    (z: B)(f: (A, B) => B): B = ???
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "id + id \times ???"] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

```
def foldRight[A, B](x: List[A])
                    (z: B)(f: (A, B) => B): B = ???
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "id + id \times foldRight"] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "id + id \times foldRight"] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

```
def foldRight[A, B](x: List[A])
                   (z: B)(f: (A, B) => B): B =
  ???
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "id + id \times foldRight"] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

```
def foldRight[A, B](x: List[A])
                   (z: B)(f: (A, B) => B): B =
  x match {
    case Nil            => ???
    case head :: tail => ???
  }
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "id + id \times foldRight"] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

```
def foldRight[A, B](x: List[A])
                   (z: B)(f: (A, B) => B): B =
  x match {
    case Nil           => z
    case head :: tail => ???
  }
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "id + id \times foldRight"] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

```
def foldRight[A, B](x: List[A])
                   (z: B)(f: (A, B) => B): B =
  x match {
    case Nil            => z
    case head :: tail =>
      foldRight(tail)(z)(f)
      ???
  }
```

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "id + id \times foldRight"] & 1 + A \times B \arrow[u, "{[z,f]}"']
\end{tikzcd}
$$

```
def foldRight[A, B](x: List[A])
                   (z: B)(f: (A, B) => B): B =
  x match {
    case Nil            => z
    case head :: tail =>
      f(head, foldRight(tail)(z)(f))
  }
```

things we learned

things we learned

- categories
- commutative diagrams
- folds in Scala

things we learned

- categories
- commutative diagrams
- folds in Scala

but also...

things we learned

- categories
- commutative diagrams
- folds in Scala

but also...

- f-algebras

things we learned

- categories
- commutative diagrams
- folds in Scala

but also...

- f-algebras
- initial algebras

things we learned

- categories
- commutative diagrams
- folds in Scala

but also...

- f-algebras
- initial algebras
- recursion schemes

$$
\begin{tikzcd}
List[A] \arrow[r, "foldRight"] \arrow[d, "match"'] & B \\
1 + A \times List[A] \arrow[r, "id + id \times foldRight"] & 1 + A \times B \arrow[u, "{[z, f]}"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
X \arrow[r, "foldRight"] \arrow[d] & B \\
1 + A \times X \arrow[r, "id + id \times foldRight"] & 1 + A \times B \arrow[u, "{[z, f]}"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
X \arrow[r, "foldRight"] \arrow[d] & B \\
F[X] \arrow[r] & F[B] \arrow[u]
\end{tikzcd}
$$

$$\begin{tikzcd}
X \arrow[r] \arrow[d] & B \\
F[X] \arrow[r] & F[B] \arrow[u]
\end{tikzcd}$$

$$
\begin{tikzcd}
X \arrow[r, "hylo"] \arrow[d, "coalg"'] & B \\
F[X] \arrow[r, "map\ \ hylo"] & F[B] \arrow[u, "alg"']
\end{tikzcd}
$$

questions?