Functors and Music

Heinrich Apfelmus
@ FARM 2019

HyperHaskell

cnoidal

HyperHaskell

— the strongly hyped
Haskell interpreter —

cnoidal

Reminder:
Functors

Functor

```
class Functor f where
    (<$>) :: (a → b) → f a → f b
```

Monad

Applicative Functor

```
zipWith
    :: (a  →  b  →  c)
    → [a] → [b] → [c]
```

Applicative Functor

```
zipWith
    :: (a  →  b  →  c)
    → [a] → [b] → [c]
```

```
zipWith (+)
    [1,2,3]
    [1,2,3]
=   [2,4,6]
```

Applicative Functor

```
zipWith3
    :: (a  →  b  →  c  →  d)
    → [a] → [b] → [c] → [d]
```

Applicative Functor

```
zipWith3
    :: (a  →  b  →  c  →  d)
    → [a] → [b] → [c] → [d]
```

```
zipWith4
zipWith5
 ...
```

Apply

```
(<*>) :: [a → b] → [a] → [b]
(<*>) = zipWith ($)
```

Apply

```
(<*>) :: [a → b] → [a] → [b]
(<*>) = zipWith ($)
```

```
f  :: a → b → c → d
xs :: [a], ys :: [b], zs :: [c]

f <$> xs          :: [b → (c → d)]
f <$> xs <*> ys   :: [    (c → d)]
f <$> xs <*> ys <*> zs ::     [d]
```

Applicative Functor

```
class Functor f ⇒ Applicative f where
    pure  :: a → f a
    (<*>) :: f (a → b) → f a → f b
```

Applicative Functor

```
class Functor f ⇒ Applicative f where
    pure  :: a → f a
    (<*>) :: f (a → b) → f a → f b
```

Laws

```
pure f <*> xs = fmap f xs
```

for lists with zip:

```
pure x = x : pure x
```

Lists: Two instances

```
instance Applicative []
```

```
instance Applicative ZipList
```

Temporal Media

Temporal Media

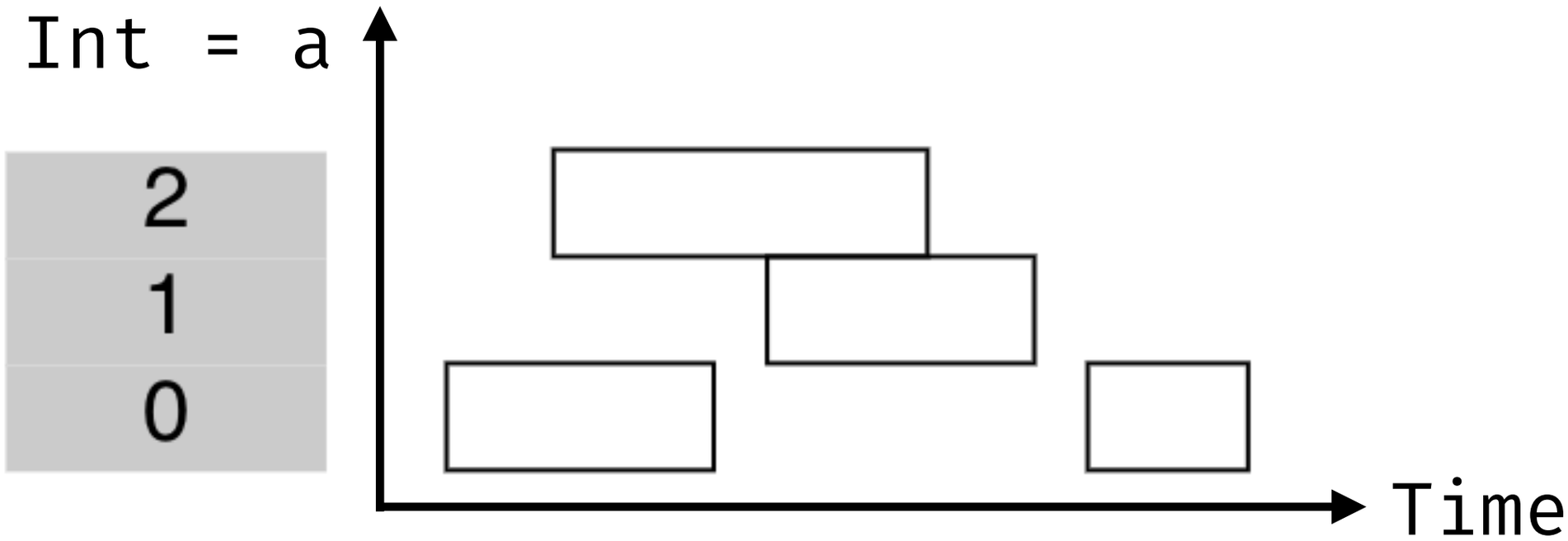
```
data Media a
```

Temporal Media

```
data Media a
```

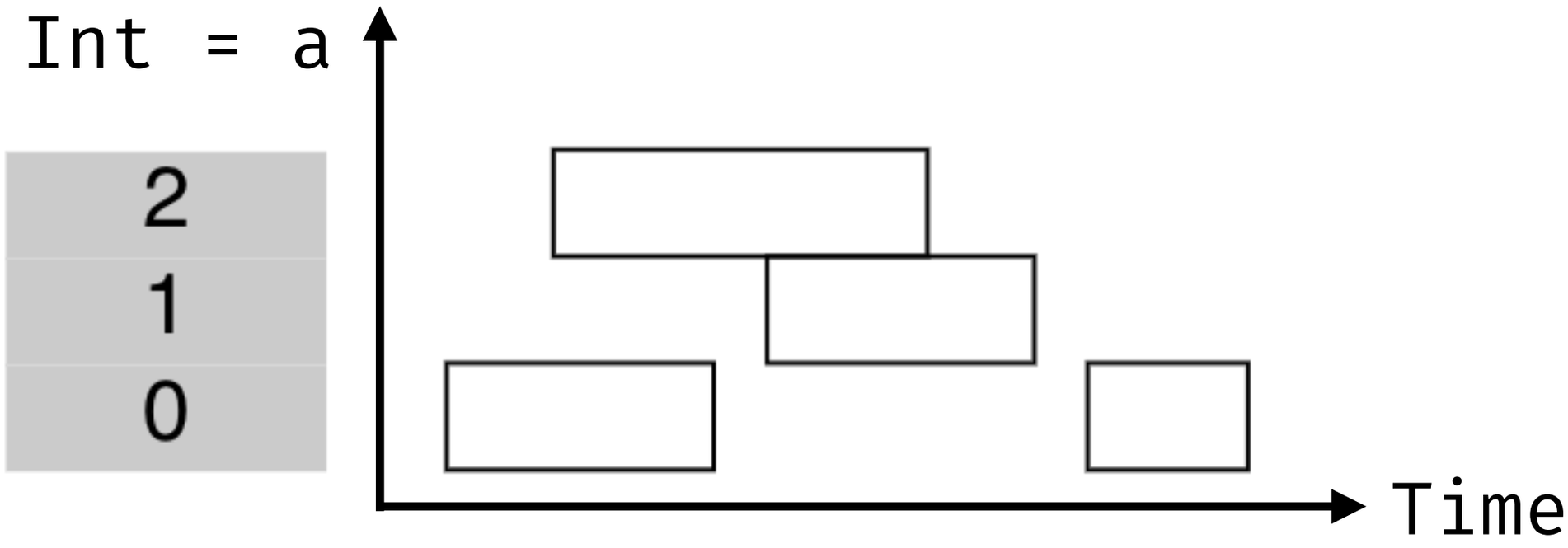
Temporal Media

```
data Media a
```



Temporal Media

```
data Media a
```



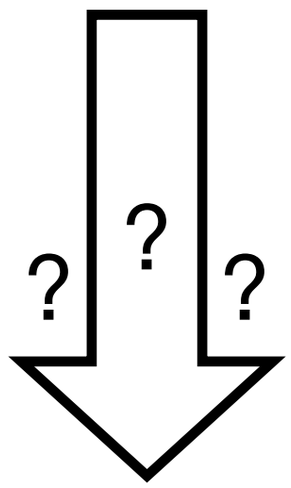

```
toIntervals ::
    Media a → Set (Interval, a)
```

Equality?

```
toIntervals x = toIntervals y
```

Equality?

```
toIntervals x = toIntervals y
```




```
x = y
```

Two Variants

```
toIntervals
```

$$\text{Media}_1 \cong \text{Set} \circ (\text{Interval} \times)$$

Two Variants

```
toIntervals
```

$$\text{Media}_1 \cong \text{Set} \circ (\text{Interval} \times)$$

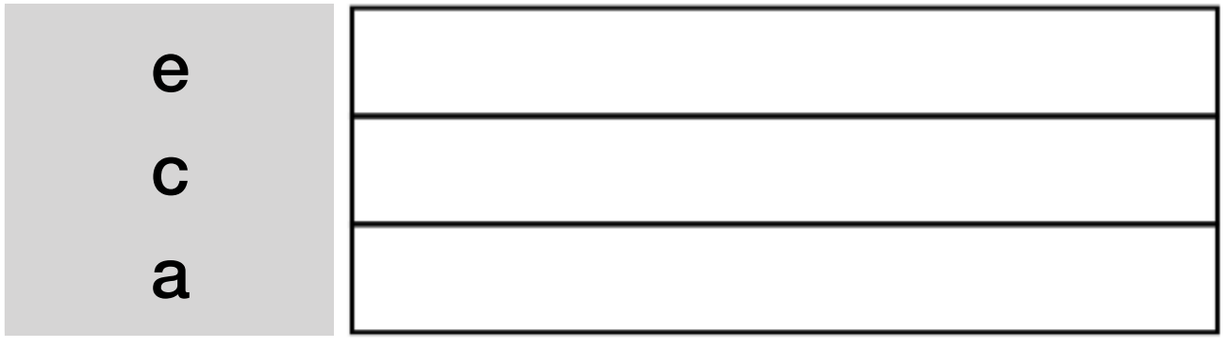
```
toIntervals, duration
duration :: Media a → Time
```

$$\text{Media}_2 \cong (\text{Time} \times) \circ \text{Set} \circ (\text{Interval} \times)$$

Temporal Media: Functors

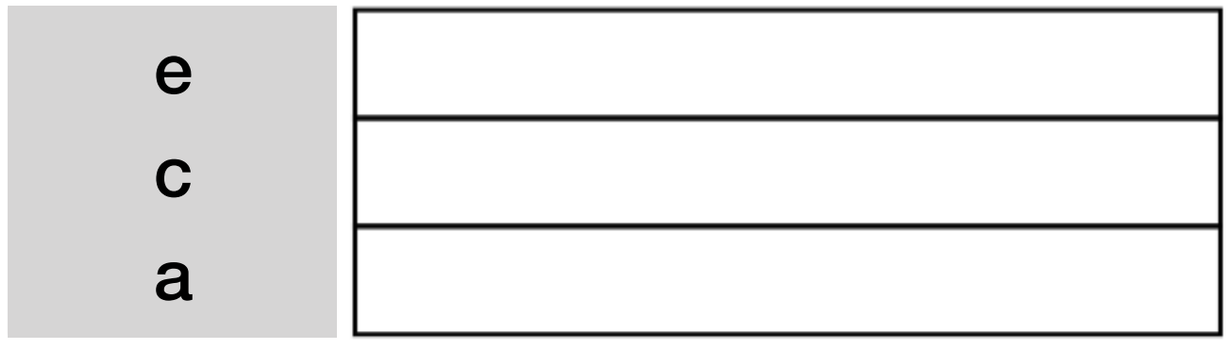
Applicative Functor

```
harmony =
```



Applicative Functor

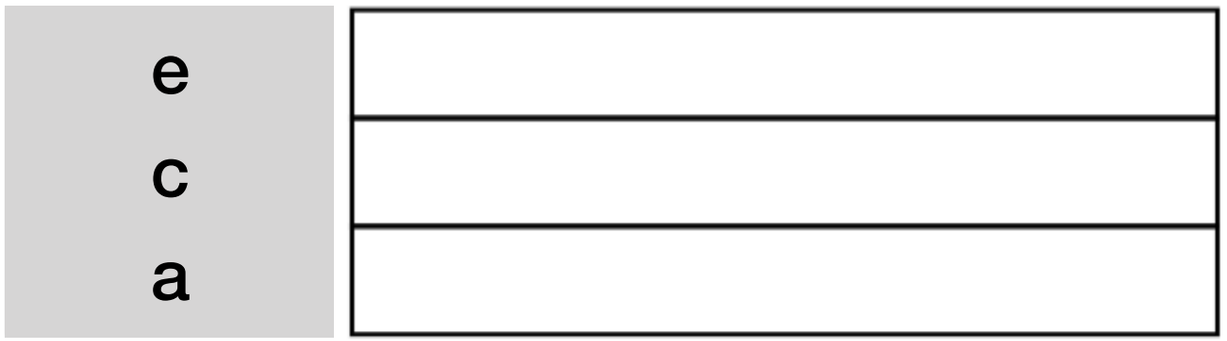
```
harmony =
```




```
rhythm =
```

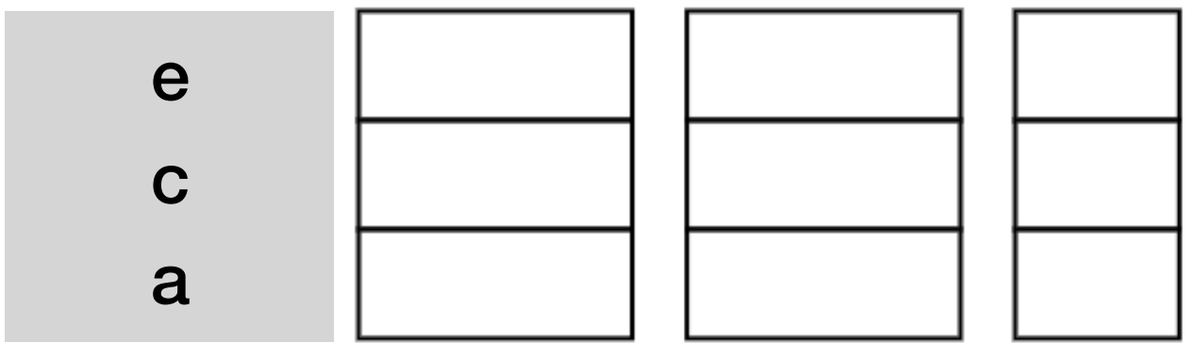
Applicative Functor

```
harmony      =
```




```
rhythm       =
```


```
const
<$> harmony
<*> rhythm   =
```



Applicative Functor

$$\text{Media}_1 \cong \text{Set} \circ (\text{Interval} \times)$$

composition of Applicative Functors

Monoid: *Intersection* of Intervals

Applicative Functor

$$\text{Media}_1 \cong \text{Set} \circ (\text{Interval} \times)$$

composition of Applicative Functors

Monoid: *Intersection* of Intervals

Intervals start at ≥ 0, may go to $+\infty$

```
type Interval = (Time, Maybe Time)
```

Applicative Functor

$$\text{Media}_1 \cong \text{Set} \circ (\text{Interval} \times)$$

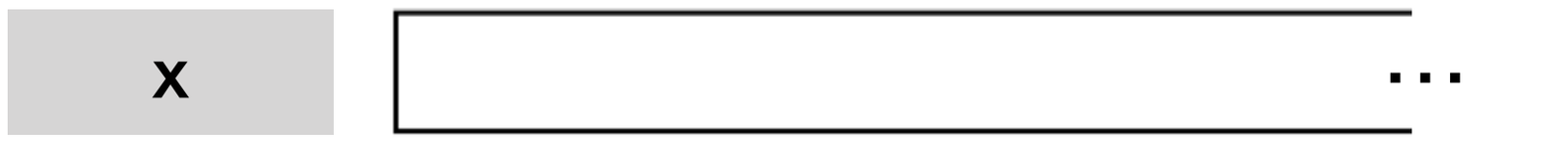
composition of Applicative Functors

Monoid: *Intersection* of Intervals

Intervals start at ≥ 0, may go to $+\infty$

```
type Interval = (Time, Maybe Time)
```

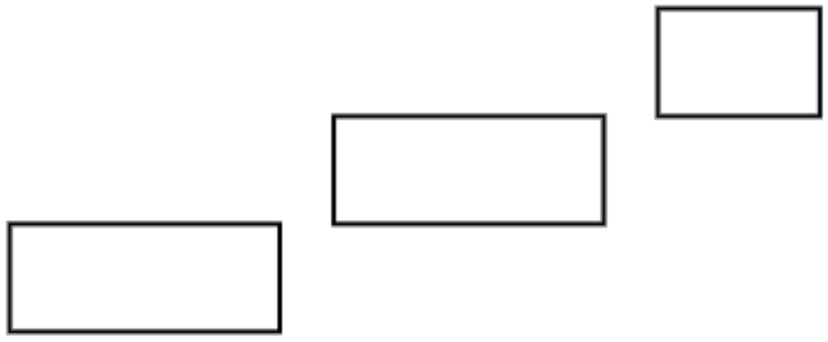
```
pure x =
```



Monad

```
music =
```

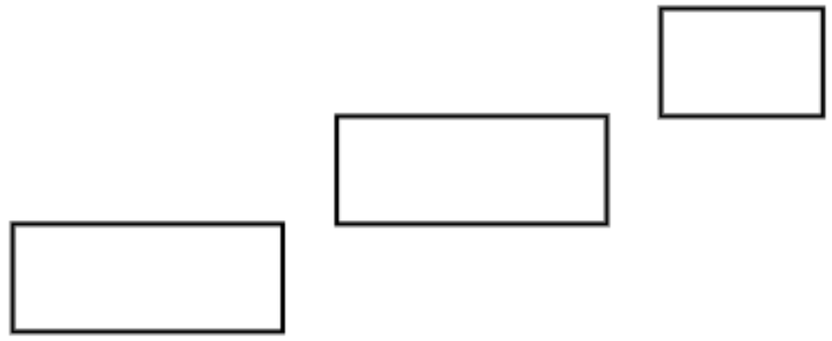
```
e
d
c
```



Monad

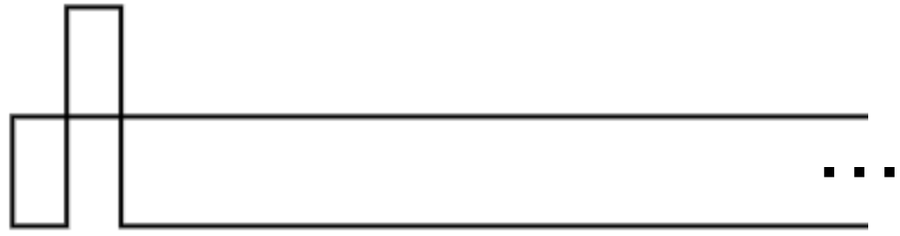
```
music          =
```

```
e
d
c
```



```
mordent x    =
```

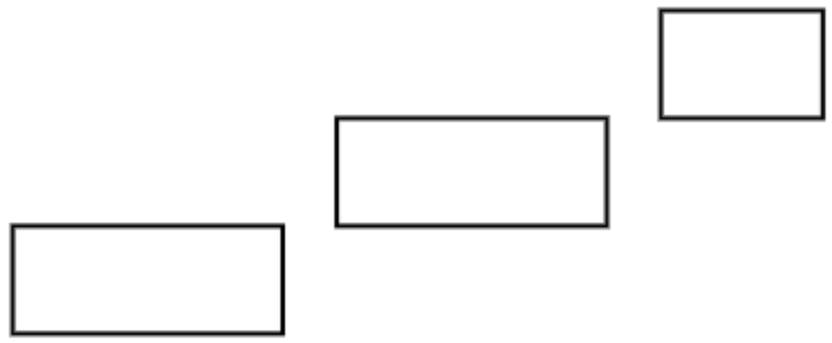
```
x+1
x
```



Monad

```
music                =
```

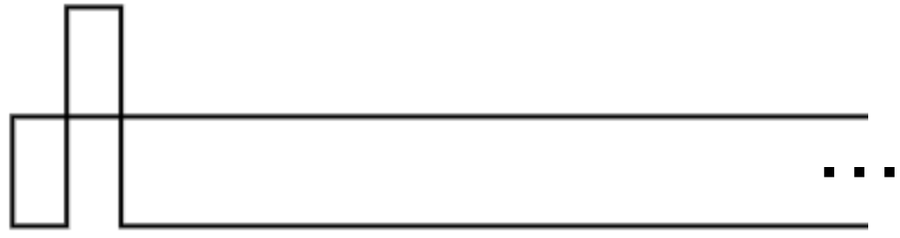
```
e
d
c
```



```
mordent x    =
```

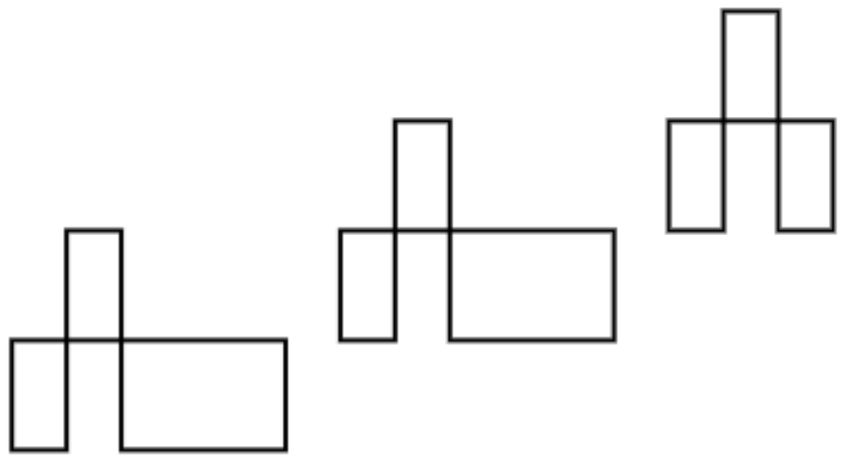
```
x+1

x
```



```
music ≫=
  mordent    =
```

```
f
e
d
c
```



Monad

$$\text{Media}_1 \cong \text{WriterT Interval Set}$$

application of Monad Transformer

Monad

$$\text{Media}_1 \cong \text{WriterT Interval Set}$$

application of Monad Transformer

Monoid: *Intersection* and *shift* of Intervals

```
i <> j = intersect i (shift (start i) j)
```

Not a Monad

$$\text{Media}_2 \cong (\text{Interval} \times) \circ \text{WriterT Interval Set}$$

still useful!

```
adorn ::
  (a → Media b) → Media a → Media b
```

Music