

Programming in Vinyl

Jon Sterling
FOBO

May 11, 2014

Records in GHC 7.8

Records in GHC 7.8

- ▶ Haskell records are nominally typed

Records in GHC 7.8

- ▶ Haskell records are nominally typed
- ▶ They may not share field names

Records in GHC 7.8

- ▶ Haskell records are nominally typed
- ▶ They may not share field names

data R = R { x :: X }

Records in GHC 7.8

- ▶ Haskell records are nominally typed
- ▶ They may not share field names

data R = R { x :: X }

data R' = R' { x :: X } -- ^ *Error*

Records in GHC 7.8

Records are...

anticompositional

Structural Typing

Structural Typing

- ▶ Sharing field names and accessors

Structural Typing

- ▶ Sharing field names and accessors
- ▶ Record types may be characterized *extensionally*

Row Polymorphism

Row Polymorphism

How do we express the type of a function which adds a field to a record?

Row Polymorphism

How do we express the type of a function which adds a field to a record?

$$\frac{x : \{foo : A\}}{f(x) : \{foo : A, bar : B\}}$$

Row Polymorphism

How do we express the type of a function which adds a field to a record?

$$\frac{x : \{foo : A; \vec{r}\vec{s}\}}{f(x) : \{foo : A, bar : B; \vec{r}\vec{s}\}}$$

Row Polymorphism

Row Polymorphism

- ▶ supports type inference

Row Polymorphism

- ▶ supports type inference
- ▶ is compositional

Row Polymorphism

- ▶ supports type inference
- ▶ is compositional

Roll Your Own in Haskell

Roll Your Own in Haskell

```
data (s :: Symbol) :: (t :: *) = Field
```

Roll Your Own in Haskell

```
data (s :: Symbol) :: (t :: *) = Field
```

```
data Rec :: [*] → * where
```

Roll Your Own in Haskell

```
data (s :: Symbol) :: (t :: *) = Field
```

```
data Rec :: [*] → * where  
  RNil :: Rec '[]
```

Roll Your Own in Haskell

```
data (s :: Symbol) ::: (t :: *) = Field
```

```
data Rec :: [*] → * where
```

```
  RNil :: Rec '[]
```

```
  (:&) :: !t → !(Rec rs) → Rec ((s ::: t) ': rs)
```


Roll Your Own in Haskell

```
data (s :: Symbol) ::: (t :: *) = Field
```

```
data Rec :: [*] → * where
```

```
  RNil :: Rec '[]
```

```
  (:&) :: !t → !(Rec rs) → Rec ((s ::: t) ': rs)
```

```
class s ∈ (rs :: [*])
```

Roll Your Own in Haskell

```
data (s :: Symbol) :: (t :: *) = Field
```

```
data Rec :: [*] → * where
```

```
  RNil :: Rec '[]
```

```
  (:&) :: !t → !(Rec rs) → Rec ((s :: t) ': rs)
```

```
class s ∈ (rs :: [*])
```

```
class ss ⊆ (rs :: [*]) where
```

```
  cast :: Rec rs → Rec ss
```

Roll Your Own in Haskell

```
data (s :: Symbol) ::: (t :: *) = Field
```

```
data Rec :: [*] → * where
```

```
  RNil :: Rec '[]
```

```
  (:&) :: !t → !(Rec rs) → Rec ((s ::: t) ': rs)
```

```
class s ∈ (rs :: [*])
```

```
class ss ⊆ (rs :: [*]) where
```

```
  cast :: Rec rs → Rec ss
```

```
(=:) : s ::: t → t → Rec '[s ::: t]
```

Roll Your Own in Haskell

```
data (s :: Symbol) ::: (t :: *) = Field
```

```
data Rec :: [*] → * where
```

```
  RNil :: Rec '[]
```

```
  (:&) :: !t → !(Rec rs) → Rec ((s ::: t) ': rs)
```

```
class s ∈ (rs :: [*])
```

```
class ss ⊆ (rs :: [*]) where
```

```
  cast :: Rec rs → Rec ss
```

```
(=:) :: s ::: t → t → Rec '[s ::: t]
```

```
(⊕) :: Rec ss → Rec ts → Rec (ss ++ ts)
```

Roll Your Own in Haskell

Roll Your Own in Haskell

$$\begin{aligned} f &:: (("a" :: A \in rs) \Rightarrow \text{Rec } rs) \\ &\rightarrow (("a" :: A \in rs, "b" :: B \in rs) \Rightarrow \text{Rec } rs) \end{aligned}$$

Universes à la Tarski

Universes à la Tarski

- ▶ A type \mathcal{U} of **codes** for types.

Universes à la Tarski

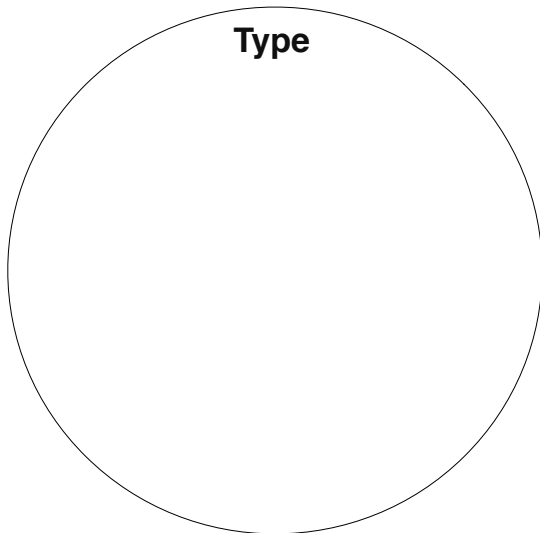
- ▶ A type \mathcal{U} of **codes** for types.
- ▶ Function $El_{\mathcal{U}} : \mathcal{U} \rightarrow \text{Type}$.

Universes à la Tarski

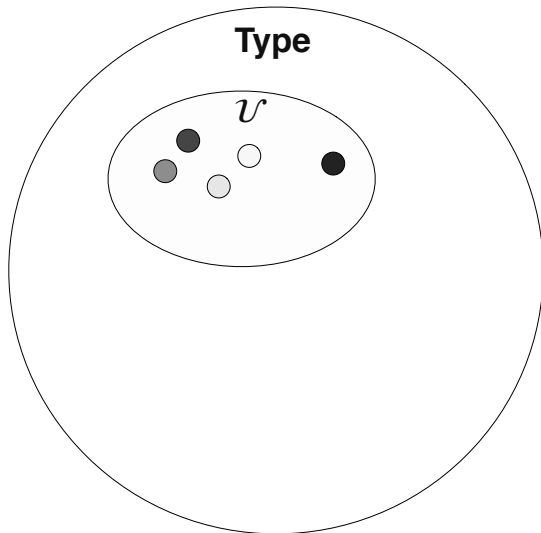
- ▶ A type \mathcal{U} of **codes** for types.
- ▶ Function $El_{\mathcal{U}} : \mathcal{U} \rightarrow \mathbf{Type}$.

$$\frac{\Gamma \vdash s : \mathcal{U}}{\Gamma \vdash El_{\mathcal{U}}(s) : \mathbf{Type}}$$

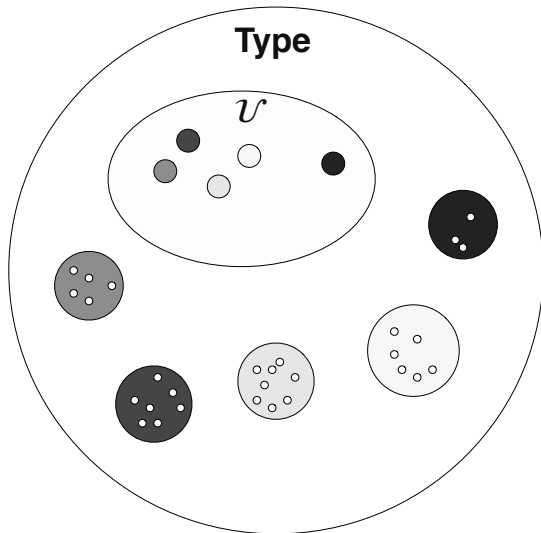
Universes à la Tarski



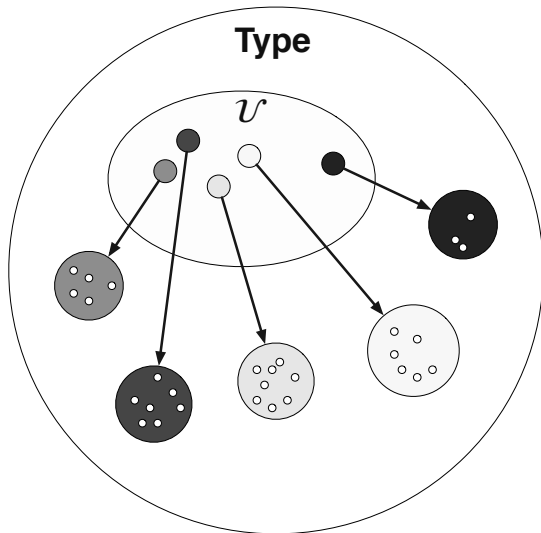
Universes à la Tarski



Universes à la Tarski



Universes à la Tarski



A Closed Universe

A Closed Universe

Let \mathcal{A} be a universe of address books:

A Closed Universe

Let \mathcal{A} be a universe of address books:

- ▶ Statics:

A Closed Universe

Let \mathcal{A} be a universe of address books:

- ▶ Statics:

$\overline{\mathcal{A} : \text{Type}}$

A Closed Universe

Let \mathcal{A} be a universe of address books:

- ▶ Statics:

$\overline{\mathcal{A} : \text{Type}} \quad \overline{\text{Label} : \text{Type}}$

A Closed Universe

Let \mathcal{A} be a universe of address books:

- ▶ Statics:

$\overline{\mathcal{A} : \text{Type}}$

$\overline{\text{Label} : \text{Type}}$

$\overline{\textit{Home, Office} : \text{Label}}$

A Closed Universe

Let \mathcal{A} be a universe of address books:

- ▶ Statics:

$\overline{\mathcal{A} : \text{Type}}$

$\overline{\text{Label} : \text{Type}}$

$\overline{\textit{Home, Office} : \text{Label}}$

$\overline{\text{Name} : \mathcal{A}}$

A Closed Universe

Let \mathcal{A} be a universe of address books:

- ▶ Statics:

$$\overline{\mathcal{A} : \text{Type}}$$
$$\overline{\text{Label} : \text{Type}}$$
$$\overline{\text{Home, Office} : \text{Label}}$$
$$\overline{\text{Name} : \mathcal{A}}$$
$$\overline{\ell : \text{Label} \quad \text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}}$$

A Closed Universe

Let \mathcal{A} be a universe of address books:

► Statics:

$$\overline{\mathcal{A} : \text{Type}}$$
$$\overline{\text{Label} : \text{Type}}$$
$$\overline{\text{Home}, \text{Office} : \text{Label}}$$
$$\overline{\text{Name} : \mathcal{A}}$$
$$\overline{\ell : \text{Label} \quad \text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}}$$
$$\overline{s : \mathcal{A} \quad \text{El}_{\mathcal{A}}(s) : \text{Type}}$$

A Closed Universe

Let \mathcal{A} be a universe of address books:

► Statics:

$$\overline{\mathcal{A} : \text{Type}}$$
$$\overline{\text{Label} : \text{Type}}$$
$$\overline{\text{Home}, \text{Office} : \text{Label}}$$
$$\overline{\text{Name} : \mathcal{A}}$$
$$\overline{\ell : \text{Label} \quad \text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}}$$
$$\overline{s : \mathcal{A} \quad \text{El}_{\mathcal{A}}(s) : \text{Type}}$$

► Dynamics:

A Closed Universe

Let \mathcal{A} be a universe of address books:

► Statics:

$$\overline{\mathcal{A} : \text{Type}}$$
$$\overline{\text{Label} : \text{Type}}$$
$$\overline{\text{Home}, \text{Office} : \text{Label}}$$
$$\overline{\text{Name} : \mathcal{A}}$$
$$\overline{\ell : \text{Label} \quad \text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}}$$
$$\overline{s : \mathcal{A} \quad \text{El}_{\mathcal{A}}(s) : \text{Type}}$$

► Dynamics:

$$\overline{\text{El}_{\mathcal{A}}(\text{Name}) \rightsquigarrow \text{string}}$$

A Closed Universe

Let \mathcal{A} be a universe of address books:

► Statics:

$$\overline{\mathcal{A} : \text{Type}}$$
$$\overline{\text{Label} : \text{Type}}$$
$$\overline{\text{Home}, \text{Office} : \text{Label}}$$
$$\overline{\text{Name} : \mathcal{A}}$$
$$\overline{\ell : \text{Label} \quad \text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}}$$
$$\overline{s : \mathcal{A} \quad \text{El}_{\mathcal{A}}(s) : \text{Type}}$$

► Dynamics:

$$\overline{\text{El}_{\mathcal{A}}(\text{Name}) \rightsquigarrow \text{string}}$$
$$\overline{\text{El}_{\mathcal{A}}(\text{Email}[\ell]) \rightsquigarrow \text{string}}$$

A Closed Universe

Let \mathcal{A} be a universe of address books:

► Statics:

$$\overline{\mathcal{A} : \text{Type}}$$
$$\overline{\text{Label} : \text{Type}}$$
$$\overline{\text{Home}, \text{Office} : \text{Label}}$$
$$\overline{\text{Name} : \mathcal{A}}$$
$$\overline{\ell : \text{Label} \quad \text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}}$$
$$\overline{s : \mathcal{A} \quad \text{El}_{\mathcal{A}}(s) : \text{Type}}$$

► Dynamics:

$$\overline{\text{El}_{\mathcal{A}}(\text{Name}) \rightsquigarrow \text{string}}$$
$$\overline{\text{El}_{\mathcal{A}}(\text{Email}[\ell]) \rightsquigarrow \text{string}}$$
$$\overline{\text{El}_{\mathcal{A}}(\text{Phone}[\ell]) \rightsquigarrow \text{list}(\mathbb{N})}$$

Records as Products

Records as Products

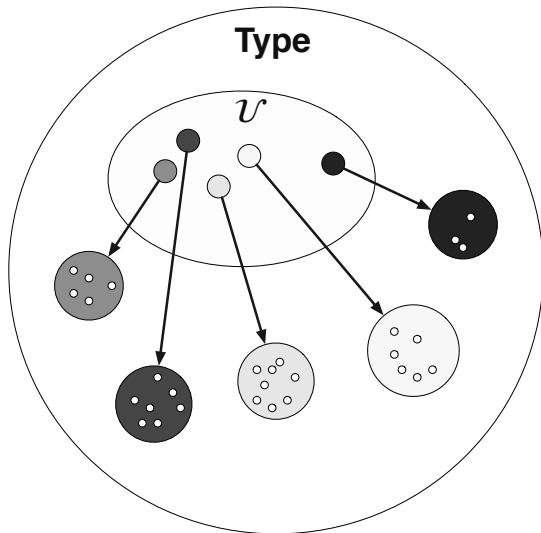
Records: the product of the image of $El_{\mathcal{U}}$ in Type restricted to a subset of \mathcal{U} .

Records as Products

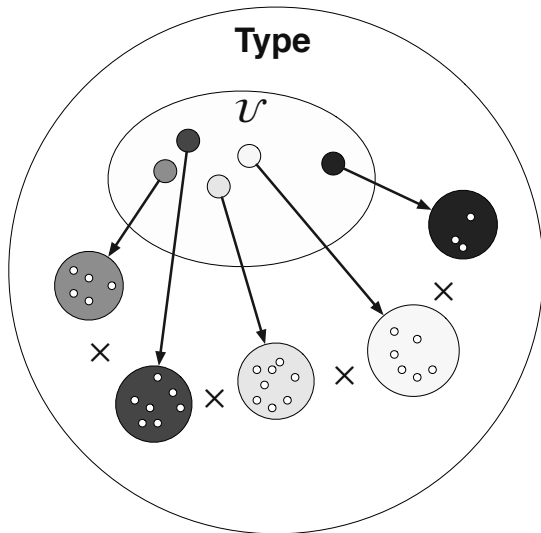
Records: the product of the image of $El_{\mathcal{U}}$ in Type restricted to a subset of \mathcal{U} .

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\nu \subset \mathcal{U}} \prod_{\nu} El_{\mathcal{U}}|_{\nu}$$

Records as Products



Records as Products



Records as Products

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\nu \subset \mathcal{U}} \prod_{\nu} El_{\mathcal{U}}|_{\nu}$$

Example Record

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V} \subset \mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}|\mathcal{V}}$$

Example Record

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V} \subset \mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$
$$\mathcal{A}' \rightsquigarrow \{\text{Name, Email } \textit{Work}\}$$

Example Record

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V} \subset \mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

$$\mathcal{A}' \rightsquigarrow \{\text{Name, Email } \textit{Work}\}$$

$$ex : \text{record}_{\mathcal{U}}$$

Example Record

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V} \subset \mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

$$\mathcal{A}' \rightsquigarrow \{\text{Name}, \text{Email } \textit{Work}\}$$

$$ex : \text{record}_{\mathcal{U}}$$

$$ex \rightsquigarrow \langle \mathcal{A}', \lambda.$$

$$\{\text{Name} \mapsto \text{"Robert Harper"};$$

$$\text{Email } \textit{Work} \mapsto \text{"rwh@cs.cmu.edu"}\}\rangle$$

Corecords as Sums

Corecords as Sums

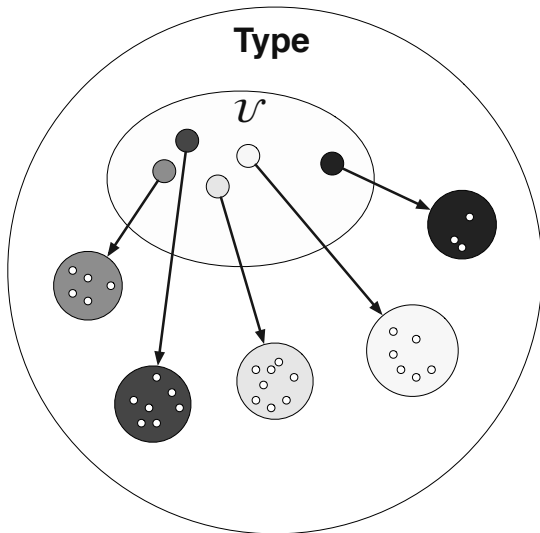
Corecords (extensible variants): the sum of the image of $El_{\mathcal{U}}$ in Type restricted to a subset of \mathcal{U} .

Corecords as Sums

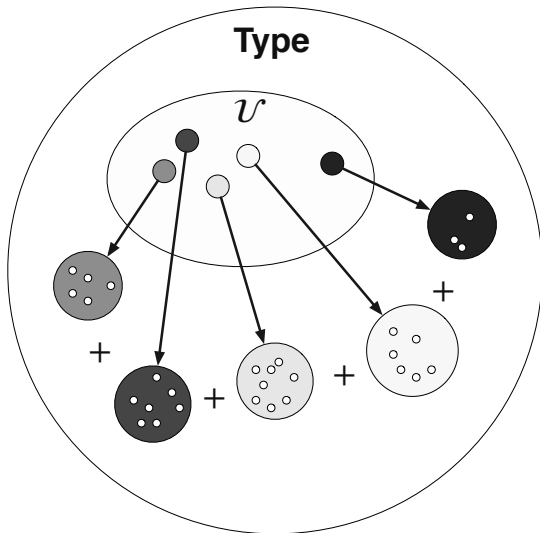
Corecords (extensible variants): the sum of the image of $El_{\mathcal{U}}$ in Type restricted to a subset of \mathcal{U} .

$$\text{corecord}_{\mathcal{U}} \rightsquigarrow \sum_{\nu \subset \mathcal{U}} \sum_{\nu} El_{\mathcal{U}}|_{\nu}$$

Corecords as Sums



Corecords as Sums



Corecords as Sums

$$\text{corecord}_{\mathcal{U}} \rightsquigarrow \sum_{\nu \subset \mathcal{U}} \sum_{\nu} El_{\mathcal{U}|\nu}$$

Doing it in Haskell

Doing it in Haskell

- ▶ Create a universe \mathcal{U} at the type-level

Doing it in Haskell

- ▶ Create a universe \mathcal{U} at the type-level
- ▶ Use type families to approximate $El_{\mathcal{U}}$

Doing it in Haskell

- ▶ Create a universe \mathcal{U} at the type-level
- ▶ Use type families to approximate $El_{\mathcal{U}}$
- ▶ Parameterize `Rec` by \mathcal{U} , $El_{\mathcal{U}}$?

Records in Haskell

```
data Rec :: ( $\mathcal{U} \rightarrow *$ )  $\rightarrow$  [ $\mathcal{U}$ ]  $\rightarrow *$  where
```


Records in Haskell

```
data Rec :: ( $\mathcal{U} \rightarrow *$ )  $\rightarrow$  [ $\mathcal{U}$ ]  $\rightarrow *$  where  
  RNil :: Rec  $\text{el}_{\mathcal{U}}$  '[]
```

Records in Haskell

```
data Rec :: ( $\mathcal{U} \rightarrow *$ )  $\rightarrow$  [ $\mathcal{U}$ ]  $\rightarrow *$  where  
  RNil :: Rec  $\text{el}_{\mathcal{U}}$  '[]  
  (:&)amp :: !( $\text{el}_{\mathcal{U}}$  r)  $\rightarrow$  !(Rec  $\text{el}_{\mathcal{U}}$  rs)  $\rightarrow$  Rec  $\text{el}_{\mathcal{U}}$  (r ': rs)
```

Recovering HList

Recovering HList

```
type HList rs = Rec ( $\Lambda\tau. \tau$ ) rs
```

Recovering HList

type HList rs = Rec ($\Lambda\tau. \tau$) rs

ex :: HList [\mathbb{Z} , **Bool**, **String**]

Recovering HList

```
type HList rs = Rec ( $\Lambda\tau. \tau$ ) rs
```

```
ex :: HList [ $\mathbb{Z}$ , Bool, String]
```

```
ex = 34 :& True :& "vinyl" :& RNil
```

Validating Records

bob :: Rec El _{\mathcal{A}} [Name, Email Work]

Validating Records

bob :: Rec El_A [Name, Email Work]

bob = Name =: "Robert_Harper"

⊕ Email Work =: "rwh@cs.cmu.edu"

Validating Records

```
bob :: Rec ElA [Name, Email Work]  
bob = Name =: "Robert_LHarper"  
      ⊕ Email Work =: "rwh@cs.cmu.edu"
```

```
validateName :: String → Either Error String  
validateEmail :: String → Either Error String  
validatePhone :: [N] → Either Error [N]
```

Validating Records

```
bob :: Rec ElA [Name, Email Work]  
bob = Name =: "Robert_LHarper"  
      ⊕ Email Work =: "rwh@cs.cmu.edu"
```

```
validateName :: String → Either Error String  
validateEmail :: String → Either Error String  
validatePhone :: [N] → Either Error [N]
```

unnnnnnhhh...

Validating Records

```
bob :: Rec ElA [Name, Email Work]
bob = Name =: "Robert_LHarper"
      ⊕ Email Work =: "rwh@cs.cmu.edu"
```

```
validateName :: String → Either Error String
```

```
validateEmail :: String → Either Error String
```

```
validatePhone :: [N] → Either Error [N]
```

unnnnnnhhh...

```
validateContact
```

```
  :: Rec ElA [Name, Email Work]
```

```
  → Either Error (Rec ElA [Name, Email Work])
```

Welp.

Effects inside records

```
data Rec :: ( $\mathcal{U} \rightarrow *$ )  $\rightarrow$  [ $\mathcal{U}$ ]  $\rightarrow *$  where  
  RNil :: Rec  $\text{el}_{\mathcal{U}}$  '[]  
  (:&) :: !( $\text{el}_{\mathcal{U}}$  r)  $\rightarrow$  !(Rec  $\text{el}_{\mathcal{U}}$  rs)  $\rightarrow$  Rec  $\text{el}_{\mathcal{U}}$  (r ': rs)
```

Effects inside records

```
data Rec :: ( $\mathcal{U} \rightarrow *$ )  $\rightarrow$  ( $* \rightarrow *$ )  $\rightarrow$  [ $\mathcal{U}$ ]  $\rightarrow *$  where  
  RNil :: Rec  $\text{el}_{\mathcal{U}}$  f '[]  
  (:&)amp :: !(f (el $_{\mathcal{U}}$  r))  $\rightarrow$  !(Rec el $_{\mathcal{U}}$  f rs)  $\rightarrow$  Rec el $_{\mathcal{U}}$  f (r ': rs)
```

Effects inside records

```
data Rec :: ( $\mathcal{U} \rightarrow *$ )  $\rightarrow$  ( $* \rightarrow *$ )  $\rightarrow$  [ $\mathcal{U}$ ]  $\rightarrow *$  where  
  RNil :: Rec  $\text{el}_{\mathcal{U}}$  f '[]  
  (:&) :: !(f (el $\mathcal{U}$  r))  $\rightarrow$  !(Rec el $\mathcal{U}$  f rs)  $\rightarrow$  Rec el $\mathcal{U}$  f (r ': rs)  
  
(=:) : Applicative f  $\Rightarrow$  sing r  $\rightarrow$  el $\mathcal{U}$  r  $\rightarrow$  Rec el $\mathcal{U}$  f '[r]
```

Effects inside records

data Rec :: ($\mathcal{U} \rightarrow *$) \rightarrow ($* \rightarrow *$) \rightarrow [\mathcal{U}] $\rightarrow *$ **where**
 RNil :: Rec $\text{el}_{\mathcal{U}}$ f '[]
 (:&) :: !(f (el _{\mathcal{U}} r)) \rightarrow !(Rec el _{\mathcal{U}} f rs) \rightarrow Rec el _{\mathcal{U}} f (r ': rs)

(=:) : Applicative f \Rightarrow sing r \rightarrow el _{\mathcal{U}} r \rightarrow Rec el _{\mathcal{U}} f '[r]
k =: x = pure x :& RNil

Effects inside records

data Rec :: ($\mathcal{U} \rightarrow *$) \rightarrow ($* \rightarrow *$) \rightarrow [\mathcal{U}] $\rightarrow *$ **where**
 RNil :: Rec el _{\mathcal{U}} f '[]
 (:&) :: !(f (el _{\mathcal{U}} r)) \rightarrow !(Rec el _{\mathcal{U}} f rs) \rightarrow Rec el _{\mathcal{U}} f (r ': rs)

(=:) : Applicative f \Rightarrow sing r \rightarrow el _{\mathcal{U}} r \rightarrow Rec el _{\mathcal{U}} f '[r]
k =: x = pure x :& RNil

(\Leftarrow): sing r \rightarrow f (el _{\mathcal{U}} r) \rightarrow Rec el _{\mathcal{U}} f '[r]

Effects inside records

data Rec :: ($\mathcal{U} \rightarrow *$) \rightarrow ($* \rightarrow *$) \rightarrow [\mathcal{U}] $\rightarrow *$ **where**
 RNil :: Rec $\text{el}_{\mathcal{U}}$ f '[]
 (:&) :: !(f (el _{\mathcal{U}} r)) \rightarrow !(Rec el _{\mathcal{U}} f rs) \rightarrow Rec el _{\mathcal{U}} f (r' : rs)

(=:) : Applicative f \Rightarrow sing r \rightarrow el _{\mathcal{U}} r \rightarrow Rec el _{\mathcal{U}} f '[r]
k =: x = pure x :& RNil

(\Leftarrow) : sing r \rightarrow f (el _{\mathcal{U}} r) \rightarrow Rec el _{\mathcal{U}} f '[r]
k \Leftarrow x = x :& RNil

Compositional Validation

type $\text{Rec}_{\mathcal{A}} = \text{Rec El}_{\mathcal{A}}$

Compositional Validation

type $\text{Rec}_{\mathcal{A}} = \text{Rec El}_{\mathcal{A}}$

$\text{bob} :: \text{Rec}_{\mathcal{A}} \text{ Identity } [\text{Name}, \text{Email Work}]$

Compositional Validation

```
type RecA = Rec ElA  
bob :: RecA Identity [Name, Email Work]  
bob = Name =: "Robert_Harper"  
      ⊕ Email Work =: "rwh@cs.cmu.edu"
```

Compositional Validation

```
type RecA = Rec ElA  
bob :: RecA Identity [Name, Email Work]  
bob = Name =: "Robert_Harper"  
      ⊕ Email Work =: "rwh@cs.cmu.edu"
```

Compositional Validation

type Validator a = a \rightarrow **Either** Error a

Compositional Validation

```
type Validator a = a → Either Error a  
validateName :: RecA Validator '[Name]  
validatePhone :: ∀ℓ. RecA Validator '[Phone ℓ]  
validateEmail :: ∀ℓ. RecA Validator '[Email ℓ]
```


Compositional Validation

```
type Validator a = a → Either Error a  
validateName :: RecA Validator '[Name]  
validatePhone :: ∀ℓ. RecA Validator '[Phone ℓ]  
validateEmail :: ∀ℓ. RecA Validator '[Email ℓ]
```

```
type TotalContact =  
  [ Name, Email Home, Email Work  
    , Phone Home, Phone Work ]
```

Compositional Validation

type Validator a = a \rightarrow **Either** Error a
validateName :: Rec_A Validator '[Name]
validatePhone :: $\forall \ell$. Rec_A Validator '[Phone ℓ]
validateEmail :: $\forall \ell$. Rec_A Validator '[Email ℓ]

type TotalContact =
[Name, Email Home, Email Work
 , Phone Home, Phone Work]

validateContact :: Rec_A Validator TotalContact
validateContact = validateName
 \oplus validateEmail
 \oplus validateEmail
 \oplus validatePhone
 \oplus validatePhone

Record Operators

Record Operators

```
newtype Lift o f g x = Lift { runLift :: f x 'o' g x }
```

Record Operators

```
newtype Lift o f g x = Lift { runLift :: f x 'o' g x }
```

```
type Validator = Lift (→) Identity (Either Error)
```

Record Operators

newtype Lift o f g x = Lift { runLift :: f x 'o' g x }

type Validator = Lift (→) Identity (**Either** Error)

(\odot) :: Rec_U (Lift (→) f g) rs → Rec_U f rs → Rec_U g rs

Record Operators

newtype Lift o f g x = Lift { runLift :: f x 'o' g x }

type Validator = Lift (→) Identity (**Either** Error)

(\odot) :: Rec_U (Lift (→) f g) rs → Rec_U f rs → Rec_U g rs

rdist :: Applicative f ⇒ Rec_U f rs → f (Rec_U Identity rs)

Compositional Validation

```
newtype Lift o f g x = Lift { runLift :: f x 'o' g x }  
type Validator = Lift (→) Identity (Either Error)  
(⊛) :: RecU (Lift (→) f g) rs → RecU f rs → RecU g rs  
rdist :: Applicative f ⇒ RecU f rs → f (RecU Identity rs)
```


Compositional Validation

```
newtype Lift o f g x = Lift { runLift :: f x 'o' g x }  
type Validator = Lift (→) Identity (Either Error)  
(⊙) :: RecU (Lift (→) f g) rs → RecU f rs → RecU g rs  
rdist :: Applicative f ⇒ RecU f rs → f (RecU Identity rs)  
  
validateContact :: RecA Validator TotalContact
```

Compositional Validation

```
newtype Lift o f g x = Lift { runLift :: f x 'o' g x }  
type Validator = Lift (→) Identity (Either Error)  
(⊙) :: RecU (Lift (→) f g) rs → RecU f rs → RecU g rs  
rdist :: Applicative f ⇒ RecU f rs → f (RecU Identity rs)  
  
validateContact :: RecA Validator TotalContact  
  
bobValid :: RecA (Either Error) [Name, Email Work]
```

Compositional Validation

```
newtype Lift o f g x = Lift { runLift :: f x 'o' g x }  
type Validator = Lift (→) Identity (Either Error)  
(⊛) :: RecU (Lift (→) f g) rs → RecU f rs → RecU g rs  
rdist :: Applicative f ⇒ RecU f rs → f (RecU Identity rs)  
  
validateContact :: RecA Validator TotalContact  
  
bobValid :: RecA (Either Error) [Name, Email Work]  
bobValid = cast validateContact ⊛ bob
```

Compositional Validation

```
newtype Lift o f g x = Lift { runLift :: f x 'o' g x }  
type Validator = Lift (→) Identity (Either Error)  
(⊛) :: RecU (Lift (→) f g) rs → RecU f rs → RecU g rs  
rdist :: Applicative f ⇒ RecU f rs → f (RecU Identity rs)
```

```
validateContact :: RecA Validator TotalContact
```

```
bobValid :: RecA (Either Error) [Name, Email Work]  
bobValid = cast validateContact ⊛ bob
```

```
validBob :: Either Error (RecA Identity [Name, Email Work])
```

Compositional Validation

```
newtype Lift o f g x = Lift { runLift :: f x 'o' g x }  
type Validator = Lift (→) Identity (Either Error)  
(⊛) :: RecU (Lift (→) f g) rs → RecU f rs → RecU g rs  
rdist :: Applicative f ⇒ RecU f rs → f (RecU Identity rs)
```

```
validateContact :: RecA Validator TotalContact
```

```
bobValid :: RecA (Either Error) [Name, Email Work]  
bobValid = cast validateContact ⊛ bob
```

```
validBob :: Either Error (RecA Identity [Name, Email Work])  
validBob = rdist bobValid
```

Laziness as an effect

Laziness as an effect

- ▶ Laziness considered harmful in the small

Laziness as an effect

- ▶ Laziness considered harmful in the small
- ▶ Vinyl records are strict in their constructors

Laziness as an effect

- ▶ Laziness considered harmful in the small
- ▶ Vinyl records are strict in their constructors
- ▶ Lazy variants usually accomplished through duplication

Laziness as an effect

- ▶ Laziness considered harmful in the small
 - ▶ Vinyl records are strict in their constructors
 - ▶ Lazy variants usually accomplished through duplication
- ↑ **Utterly unacceptable**

Laziness as an effect

Laziness as an effect

```
newtype Identity a = Identity { runIdentity :: a }
```

Laziness as an effect

```
newtype Identity a = Identity { runIdentity :: a }  
data Thunk a = Thunk { unThunk :: a }
```

Laziness as an effect

newtype Identity a = Identity { runIdentity :: a }

data Thunk a = Thunk { unThunk :: a }

type PlainRec_{*U*} rs = Rec_{*U*} Identity rs

Laziness as an effect

```
newtype Identity a = Identity { runIdentity :: a }
```

```
data Thunk a = Thunk { unThunk :: a }
```

```
type PlainRecU rs = RecU Identity rs
```

```
type LazyRecU rs = RecU Thunk rs
```

Concurrent Records with Async

Concurrent Records with Async

`fetchName :: Rec \mathcal{A} IO '[Name]`

Concurrent Records with Async

```
fetchName :: RecA IO '[Name]  
fetchName = Name  $\Leftarrow$  someOperation
```

Concurrent Records with Async

$\text{fetchName} :: \text{Rec}_{\mathcal{A}} \text{IO } '[\text{Name}]$

$\text{fetchName} = \text{Name} \Leftarrow \text{someOperation}$

$\text{fetchWorkEmail} :: \text{Rec}_{\mathcal{A}} \text{IO } '[\text{Email Work}]$

Concurrent Records with Async

`fetchName :: RecA IO '[Name]`

`fetchName = Name \Leftarrow someOperation`

`fetchWorkEmail :: RecA IO '[Email Work]`

`fetchWorkEmail = Email Work \Leftarrow anotherOperation`

Concurrent Records with Async

$\text{fetchName} :: \text{Rec}_{\mathcal{A}} \text{IO } '[\text{Name}]$

$\text{fetchName} = \text{Name} \Leftarrow \text{someOperation}$

$\text{fetchWorkEmail} :: \text{Rec}_{\mathcal{A}} \text{IO } '[\text{Email Work}]$

$\text{fetchWorkEmail} = \text{Email Work} \Leftarrow \text{anotherOperation}$

$\text{fetchBob} :: \text{Rec}_{\mathcal{A}} \text{IO } [\text{Name}, \text{Email Work}]$

Concurrent Records with Async

$\text{fetchName} :: \text{Rec}_{\mathcal{A}} \text{IO } '[\text{Name}]$

$\text{fetchName} = \text{Name} \Leftarrow \text{someOperation}$

$\text{fetchWorkEmail} :: \text{Rec}_{\mathcal{A}} \text{IO } '[\text{Email Work}]$

$\text{fetchWorkEmail} = \text{Email Work} \Leftarrow \text{anotherOperation}$

$\text{fetchBob} :: \text{Rec}_{\mathcal{A}} \text{IO } [\text{Name}, \text{Email Work}]$

$\text{fetchBob} = \text{fetchName} \oplus \text{fetchWorkEmail}$

Concurrent Records with Async

Concurrent Records with Async

```
newtype Concurrently a  
  = Concurrently { runConcurrently :: IO a }
```


Concurrent Records with Async

newtype Concurrently a
= Concurrently { runConcurrently :: **IO** a }

$(\textcircled{\$}) :: (\forall a. f\ a \rightarrow g\ a) \rightarrow \text{Rec}_{\mathcal{U}}\ f\ rs \rightarrow \text{Rec}_{\mathcal{U}}\ g\ rs$

Concurrent Records with Async

newtype Concurrently a
= Concurrently { runConcurrently :: **IO** a }

$(\textcircled{\$}) :: (\forall a. f\ a \rightarrow g\ a) \rightarrow \text{Rec}_{\mathcal{U}}\ f\ rs \rightarrow \text{Rec}_{\mathcal{U}}\ g\ rs$

bobConcurrently :: $\text{Rec}_{\mathcal{A}}\ \text{Concurrently}\ [\text{Name}, \text{Email Work}]$

Concurrent Records with Async

newtype Concurrently a
= Concurrently { runConcurrently :: **IO** a }

$(\textcircled{\$}) :: (\forall a. f\ a \rightarrow g\ a) \rightarrow \text{Rec}_{\mathcal{U}}\ f\ rs \rightarrow \text{Rec}_{\mathcal{U}}\ g\ rs$

bobConcurrently :: $\text{Rec}_{\mathcal{A}}\ \text{Concurrently}\ [\text{Name}, \text{Email}\ \text{Work}]$

bobConcurrently = Concurrently $(\textcircled{\$})\ \text{fetchBob}$

Concurrent Records with Async

newtype Concurrently a
= Concurrently { runConcurrently :: **IO** a }

$(\textcircled{\$}) :: (\forall a. f\ a \rightarrow g\ a) \rightarrow \text{Rec}_{\mathcal{U}}\ f\ rs \rightarrow \text{Rec}_{\mathcal{U}}\ g\ rs$

bobConcurrently :: $\text{Rec}_{\mathcal{A}}$ Concurrently [Name, Email Work]

bobConcurrently = Concurrently $(\textcircled{\$})$ fetchBob

concurrentBob :: Concurrently ($\text{Rec}_{\mathcal{A}}$ Identity [...])

Concurrent Records with Async

newtype Concurrently a
= Concurrently { runConcurrently :: **IO** a }

$(\textcircled{\$}) :: (\forall a. f\ a \rightarrow g\ a) \rightarrow \text{Rec}_{\mathcal{U}}\ f\ rs \rightarrow \text{Rec}_{\mathcal{U}}\ g\ rs$

bobConcurrently :: $\text{Rec}_{\mathcal{A}}$ Concurrently [Name, Email Work]

bobConcurrently = Concurrently $(\textcircled{\$})$ fetchBob

concurrentBob :: Concurrently ($\text{Rec}_{\mathcal{A}}$ Identity [...])

concurrentBob = rdist bobConcurrently

Concurrent Records with Async

Concurrent Records with Async

```
fetchBob :: RecA IO [Name, Email Work]  
bobConcurrently :: RecA Concurrently [Name, Email Work]  
concurrentBob :: Concurrently (RecA Identity [...])
```

Concurrent Records with Async

```
fetchBob :: RecA IO [Name, Email Work]
bobConcurrently :: RecA Concurrently [Name, Email Work]
concurrentBob :: Concurrently (RecA Identity [...])

bob :: IO (RecA Identity [Name, Email Work])
```


Concurrent Records with Async

```
fetchBob :: RecA IO [Name, Email Work]
bobConcurrently :: RecA Concurrently [Name, Email Work]
concurrentBob :: Concurrently (RecA Identity [...])

bob :: IO (RecA Identity [Name, Email Work])
bob = runConcurrently concurrentBob
```

Containers: The Syntax for Data Types

container : Type

Containers: The Syntax for Data Types

$$\frac{}{\text{container} : \text{Type}} \qquad \frac{\mathcal{U} : \text{Type} \quad El_{\mathcal{U}} : \mathcal{U} \rightarrow \text{Type}}{\mathcal{U} \triangleleft El_{\mathcal{U}} : \text{container}}$$

Containers: The Syntax for Data Types

$$\frac{}{\text{container} : \text{Type}} \qquad \frac{\mathcal{U} : \text{Type} \quad El_{\mathcal{U}} : \mathcal{U} \rightarrow \text{Type}}{\mathcal{U} \triangleleft El_{\mathcal{U}} : \text{container}}$$

$$\frac{C : \text{container}}{C.\text{Sh} : \text{Type}}$$

Containers: The Syntax for Data Types

$$\frac{}{\text{container} : \text{Type}} \qquad \frac{\mathcal{U} : \text{Type} \quad El_{\mathcal{U}} : \mathcal{U} \rightarrow \text{Type}}{\mathcal{U} \triangleleft El_{\mathcal{U}} : \text{container}}$$

$$\frac{C : \text{container}}{C.\text{Sh} : \text{Type}} \qquad \frac{C \rightsquigarrow \mathcal{U} \triangleleft El_{\mathcal{U}}}{C.\text{Sh} \rightsquigarrow \mathcal{U}}$$

Containers: The Syntax for Data Types

$$\frac{}{\text{container} : \text{Type}} \qquad \frac{\mathcal{U} : \text{Type} \quad El_{\mathcal{U}} : \mathcal{U} \rightarrow \text{Type}}{\mathcal{U} \triangleleft El_{\mathcal{U}} : \text{container}}$$

$$\frac{C : \text{container}}{C.\text{Sh} : \text{Type}} \qquad \frac{C \rightsquigarrow \mathcal{U} \triangleleft El_{\mathcal{U}}}{C.\text{Sh} \rightsquigarrow \mathcal{U}}$$

$$\frac{C : \text{container}}{C.\text{Po} : C.\text{Sh} \rightarrow \text{Type}}$$

Containers: The Syntax for Data Types

$$\frac{}{\text{container} : \mathbf{Type}} \qquad \frac{\mathcal{U} : \mathbf{Type} \quad El_{\mathcal{U}} : \mathcal{U} \rightarrow \mathbf{Type}}{\mathcal{U} \triangleleft El_{\mathcal{U}} : \text{container}}$$

$$\frac{C : \text{container}}{C.Sh : \mathbf{Type}} \qquad \frac{C \rightsquigarrow \mathcal{U} \triangleleft El_{\mathcal{U}}}{C.Sh \rightsquigarrow \mathcal{U}}$$

$$\frac{C : \text{container}}{C.Po : C.Sh \rightarrow \mathbf{Type}} \qquad \frac{C \rightsquigarrow \mathcal{U} \triangleleft El_{\mathcal{U}}}{C.Po \rightsquigarrow El_{\mathcal{U}}}$$

Restricting Containers

$$\frac{C : \text{container} \quad \mathcal{V} \subseteq C.\text{Sh}}{C|_{\mathcal{V}} : \text{container}}$$

Restricting Containers

$$\frac{C : \text{container} \quad \mathcal{V} \subseteq C.\text{Sh}}{C|_{\mathcal{V}} : \text{container}}$$

$$\frac{C \rightsquigarrow \mathcal{U} \triangleleft El_{\mathcal{U}}}{C|_{\mathcal{V}} \rightsquigarrow \mathcal{V} \triangleleft El_{\mathcal{U}}}$$

Container Lifting

$$\frac{C : \text{container} \quad F : \text{Type} \rightarrow \text{Type}}{C \uparrow F : \text{container}}$$

Container Lifting

$$\frac{C : \text{container} \quad F : \text{Type} \rightarrow \text{Type}}{C \uparrow F : \text{container}}$$

$$\frac{C \rightsquigarrow \mathcal{U} \triangleleft El_{\mathcal{U}}}{C \uparrow F \rightsquigarrow \mathcal{U} \triangleleft F \circ El_{\mathcal{U}}}$$

A Menagerie of Quantifiers

A Menagerie of Quantifiers

Dependent Products:

$$\frac{\Gamma \vdash A : \mathbf{Type} \quad \Gamma, x : A \vdash B : \mathbf{Type}}{\prod_A B : \mathbf{Type}}$$

$$\frac{\Gamma, x : A \vdash e : B[x]}{\Gamma \vdash \lambda x. e : \prod_A B}$$

A Menagerie of Quantifiers

Dependent Sums:

$$\frac{\Gamma \vdash A : \mathbf{Type} \quad \Gamma, x : A \vdash B : \mathbf{Type}}{\sum_A B : \mathbf{Type}}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[a]}{\langle a, b \rangle : \sum_A B}$$

A Menagerie of Quantifiers

Inductive Types:

$$\frac{\Gamma \vdash A : \mathbf{Type} \quad \Gamma, x : A \vdash B : \mathbf{Type}}{W_A B : \mathbf{Type}}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma, v : B[a] \vdash b : W_A B}{\text{sup}(a; v. b) : W_A B}$$

A Menagerie of Quantifiers

Coinductive Types:

$$\frac{\Gamma \vdash A : \mathbf{Type} \quad \Gamma, x : A \vdash B : \mathbf{Type}}{M_A B : \mathbf{Type}}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma, v : B[a] \vdash b : \infty (M_A B)}{\text{inf}(a; v. b) : M_A B}$$

A Scheme for Quantifiers

A Scheme for Quantifiers

$$\frac{\Gamma, A : \mathbf{Type}, (x : A \vdash B : \mathbf{Type}) \vdash Q_A B : \mathbf{Type}}{\Gamma \vdash Q \text{ quantifier}}$$

Quantifiers Give Containers Semantics

$$\frac{\Gamma, A : \mathbf{Type}, (x : A \vdash B : \mathbf{Type}) \vdash Q_A B : \mathbf{Type}}{\Gamma \vdash Q \text{ quantifier}}$$

Quantifiers Give Containers Semantics

$$\frac{\Gamma, A : \mathbf{Type}, (x : A \vdash B : \mathbf{Type}) \vdash Q_A B : \mathbf{Type}}{\Gamma \vdash Q \text{ quantifier}}$$

$$\frac{C : \text{container} \quad Q \text{ quantifier}}{\llbracket C \rrbracket_Q : \mathbf{Type}}$$

Quantifiers Give Containers Semantics

$$\frac{\Gamma, A : \mathbf{Type}, (x : A \vdash B : \mathbf{Type}) \vdash Q_A B : \mathbf{Type}}{\Gamma \vdash Q \text{ quantifier}}$$

$$\frac{C : \text{container} \quad Q \text{ quantifier}}{\llbracket C \rrbracket_Q : \mathbf{Type}}$$

$$\frac{C \rightsquigarrow \mathcal{U} \triangleleft El_{\mathcal{U}}}{\llbracket C \rrbracket_Q \rightsquigarrow Q_{\mathcal{U}} El_{\mathcal{U}}}$$

Vinyl Records as Containers

Vinyl Records as Containers

Records and corecords are finite products and sums respectively.

Vinyl Records as Containers

Records and corecords are finite products and sums respectively.

$$\text{Rec } El_{\mathcal{U}} F rs \cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\Pi}$$

Vinyl Records as Containers

Records and corecords are finite products and sums respectively.

$$\begin{aligned}\text{Rec } El_{\mathcal{U}} F rs &\cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\Pi} \\ \text{CoRec } El_{\mathcal{U}} F rs &\cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\Sigma}\end{aligned}$$

Vinyl Records as Containers

Records and corecords are finite products and sums respectively.

$$\text{Rec } El_{\mathcal{U}} F rs \cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\Pi}$$

$$\text{CoRec } El_{\mathcal{U}} F rs \cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\Sigma}$$

$$??? El_{\mathcal{U}} F rs \cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\mathbf{W}}$$

Vinyl Records as Containers

Records and corecords are finite products and sums respectively.

$$\begin{aligned}\text{Rec } El_{\mathcal{U}} F rs &\cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\Pi} \\ \text{CoRec } El_{\mathcal{U}} F rs &\cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\Sigma} \\ \\ ??? El_{\mathcal{U}} F rs &\cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\mathbf{W}} \\ ??? El_{\mathcal{U}} F rs &\cong \llbracket (\mathcal{U} \triangleleft El_{\mathcal{U}}) |_{rs \ni -} \uparrow F \rrbracket_{\mathbf{M}}\end{aligned}$$

Questions