Programming in Vinyl

Jon Sterling
jonmsterling.com

May 18, 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*

Structural Typing

Structural Typing

- Sharing field names and accessors

Structural Typing

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

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?

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

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 :: [$*$] $\to$ $*$ **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 :: [$*$] $\rightarrow$ $*$ **where**
  RNil :: Rec '[]
  (:&) :: !t $\rightarrow$!(Rec rs) $\rightarrow$ Rec ((s ::: t) ': rs)

class s $\in$ (rs :: [$*$])

Roll Your Own in Haskell

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

data Rec :: [$*$] $\to$ $*$ **where**
  RNil :: Rec '[]
  (:&) :: !t $\to$!(Rec rs) $\to$ Rec ((s ::: t) ': rs)

class s $\in$ (rs :: [$*$])
class ss $\subseteq$ (rs :: [$*$]) **where**
  cast :: Rec rs $\to$ Rec ss

Roll Your Own in Haskell

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

data Rec :: [$*$] $\to$ $*$ **where**
  RNil :: Rec '[]
  (:&) :: !t $\to$!(Rec rs) $\to$ Rec ((s ::: t) ': rs)

class s $\in$ (rs :: [$*$])
class ss $\subseteq$ (rs :: [$*$]) **where**
  cast :: Rec rs $\to$ Rec ss
(=:) : s ::: t $\to$ t $\to$ 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

```
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)
lens : s ::: t ∈ rs ⇒ s ::: t → Lens' (Rec rs) t
```

Roll Your Own in Haskell

Roll Your Own in Haskell

f :: Rec ("foo" ::: A ': rs)
  $\rightarrow$ Rec ("bar" ::: B ': "foo" ::: A ': rs)

Universes à la Tarski

Universes à la Tarski

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

Universes à la Tarski

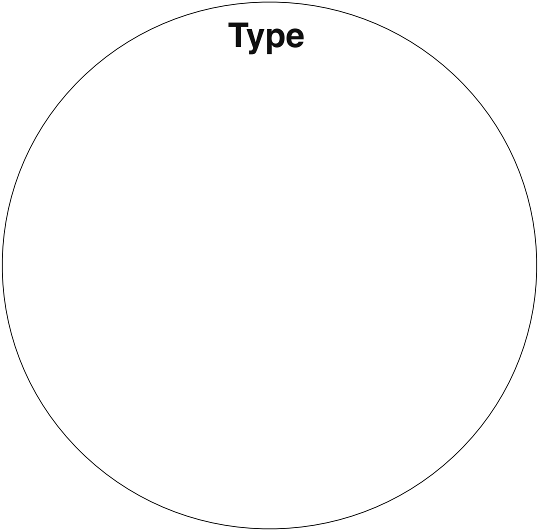
- A type $\mathcal{U}$ of **codes** for types.
- Function $El_{\mathcal{U}} : \mathcal{U} \rightarrow$ Type.

Universes à la Tarski

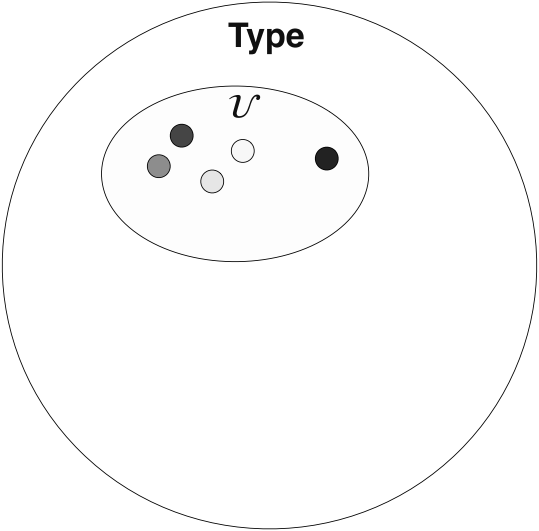
- A type $\mathcal{U}$ of **codes** for types.
- Function $El_{\mathcal{U}} : \mathcal{U} \to \text{Type}$.

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

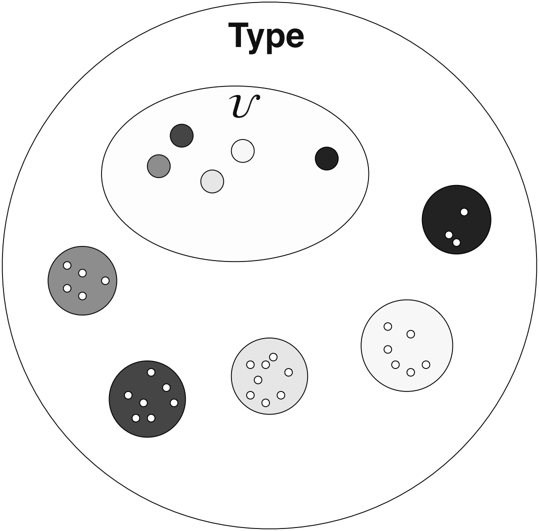
Universes à la Tarski



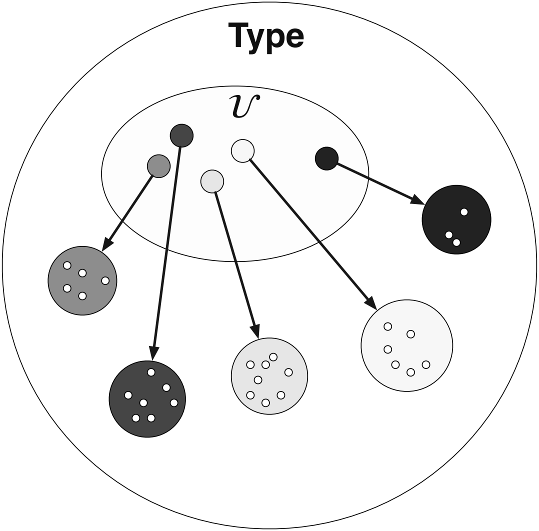
Universes à la Tarski



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:

$$\frac{}{\mathcal{A} : \mathsf{Type}}$$

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \mathsf{Type}} \qquad \frac{}{\mathsf{Label} : \mathsf{Type}}$$

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \mathsf{Type}} \qquad \frac{}{\mathsf{Label} : \mathsf{Type}} \qquad \frac{}{\mathit{Home}, \mathit{Office} : \mathsf{Label}}$$

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \mathsf{Type}} \qquad \frac{}{\mathsf{Label} : \mathsf{Type}} \qquad \frac{}{\mathit{Home}, \mathit{Office} : \mathsf{Label}}$$

$$\frac{}{\mathsf{Name} : \mathcal{A}}$$

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \mathsf{Type}} \qquad \frac{}{\mathsf{Label} : \mathsf{Type}} \qquad \frac{}{\mathit{Home}, \mathit{Office} : \mathsf{Label}}$$

$$\frac{}{\mathsf{Name} : \mathcal{A}} \qquad \frac{\ell : \mathsf{Label}}{\mathsf{Phone}[\ell], \mathsf{Email}[\ell] : \mathcal{A}}$$

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \mathsf{Type}} \qquad \frac{}{\mathsf{Label} : \mathsf{Type}} \qquad \frac{}{\mathit{Home}, \mathit{Office} : \mathsf{Label}}$$

$$\frac{}{\mathsf{Name} : \mathcal{A}} \qquad \frac{\ell : \mathsf{Label}}{\mathsf{Phone}[\ell], \mathsf{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{El_{\mathcal{A}}(s) : \mathsf{Type}}$$

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \mathsf{Type}} \qquad \frac{}{\mathsf{Label} : \mathsf{Type}} \qquad \frac{}{\mathit{Home}, \mathit{Office} : \mathsf{Label}}$$

$$\frac{}{\mathsf{Name} : \mathcal{A}} \qquad \frac{\ell : \mathsf{Label}}{\mathsf{Phone}[\ell], \mathsf{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{El_{\mathcal{A}}(s) : \mathsf{Type}}$$

- Dynamics:

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \text{Type}} \qquad \frac{}{\text{Label} : \text{Type}} \qquad \frac{}{\textit{Home}, \textit{Office} : \text{Label}}$$

$$\frac{}{\text{Name} : \mathcal{A}} \qquad \frac{\ell : \text{Label}}{\text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{El_{\mathcal{A}}(s) : \text{Type}}$$

- Dynamics:

$$\frac{}{El_{\mathcal{A}}(\text{Name}) \rightsquigarrow \mathbf{string}}$$

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \mathsf{Type}} \qquad \frac{}{\mathsf{Label} : \mathsf{Type}} \qquad \frac{}{\mathit{Home}, \mathit{Office} : \mathsf{Label}}$$

$$\frac{}{\mathsf{Name} : \mathcal{A}} \qquad \frac{\ell : \mathsf{Label}}{\mathsf{Phone}[\ell], \mathsf{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{El_{\mathcal{A}}(s) : \mathsf{Type}}$$

- Dynamics:

$$\frac{}{El_{\mathcal{A}}(\mathsf{Name}) \rightsquigarrow \mathbf{string}} \qquad \frac{}{El_{\mathcal{A}}(\mathsf{Email}[\ell]) \rightsquigarrow \mathbf{string}}$$

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \mathsf{Type}} \qquad \frac{}{\mathsf{Label} : \mathsf{Type}} \qquad \frac{}{\mathit{Home}, \mathit{Office} : \mathsf{Label}}$$

$$\frac{}{\mathsf{Name} : \mathcal{A}} \qquad \frac{\ell : \mathsf{Label}}{\mathsf{Phone}[\ell], \mathsf{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{El_{\mathcal{A}}(s) : \mathsf{Type}}$$

- Dynamics:

$$\frac{}{El_{\mathcal{A}}(\mathsf{Name}) \rightsquigarrow \mathbf{string}} \qquad \frac{}{El_{\mathcal{A}}(\mathsf{Email}[\ell]) \rightsquigarrow \mathbf{string}}$$

$$\frac{}{El_{\mathcal{A}}(\mathsf{Phone}[\ell]) \rightsquigarrow \mathsf{list}(\mathbb{N})}$$

Records as Products

Records as Products

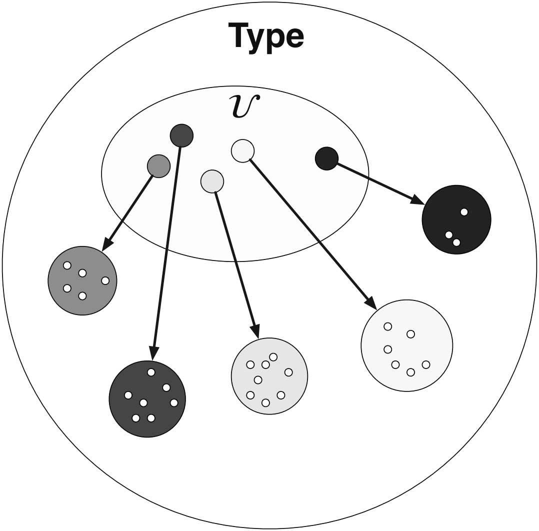
Records: the product of the image of $El_{\mathcal{U}}$ in Type restricted to a subset of $\mathcal{U}$.

Records as Products

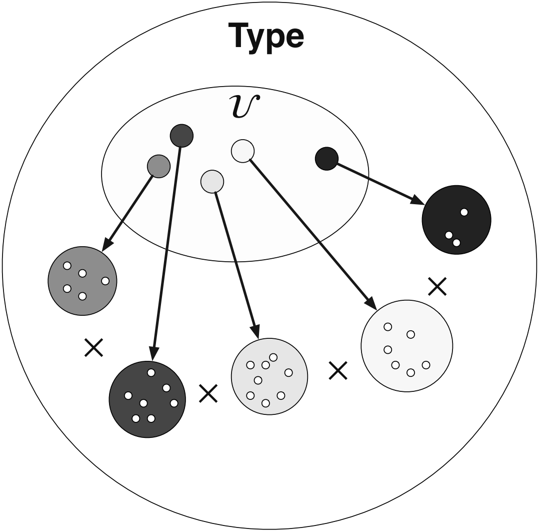
Records: the product of the image of $El_{\mathcal{U}}$ in Type restricted to a subset of $\mathcal{U}$.

$$\mathsf{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V} \subseteq \mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

Records as Products



Records as Products



Records as Products

$$\mathsf{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V} \subseteq \mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

Example Record

$$\mathsf{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}\subseteq\mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

Example Record

$$\mathsf{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}\subseteq\mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

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

Example Record

$$\mathsf{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V} \subseteq \mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

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

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

Example Record

$$\mathsf{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}\subseteq\mathcal{U}} \prod_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

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

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

$$ex \rightsquigarrow \langle \mathcal{A}', \lambda. \{\mathsf{Name} \mapsto \text{"Robert Harper"}; \mathsf{Email}\ \textit{Work} \mapsto \text{"rwh@cs.cmu.edu"}\}\rangle$$

Presheaves

Presheaves

A presheaf on some space X is a functor $\mathcal{O}(X)^{\mathrm{op}} \to \mathbf{Type}$, where $\mathcal{O}$ is the category of open sets of X for whatever topology you have chosen.

Topologies on some space X

Topologies on some space X

- What are the open sets on X?

Topologies on some space X

- What are the open sets on X?
- The empty set and X are open sets
- The union of open sets is open
- Finite intersections of open sets are open

Records are Presheaves

Records are Presheaves

- Let $\mathcal{O} = \mathcal{P}$, the discrete topology

Records are Presheaves

- Let $\mathcal{O} = \mathcal{P}$, the discrete topology
- Then records on a universe X give rise to a presheaf $\mathcal{R}$: subset inclusions are taken to casts from larger to smaller records

Records are Presheaves

- Let $\mathcal{O} = \mathcal{P}$, the discrete topology
- Then records on a universe X give rise to a presheaf $\mathcal{R}$: subset inclusions are taken to casts from larger to smaller records

$$\text{for } V \subseteq X \quad \mathcal{R}(V) :\equiv \prod_{V} El_X|_V : \mathbf{Type}$$

Records are Presheaves

- Let $\mathcal{O} = \mathcal{P}$, the discrete topology
- Then records on a universe X give rise to a presheaf $\mathcal{R}$: subset inclusions are taken to casts from larger to smaller records

$$\text{for } V \subseteq X \quad \mathcal{R}(V) :\equiv \prod_{V} El_X|_V : \mathbf{Type}$$

$$\text{for } i : V \hookrightarrow U \quad \mathcal{R}(i) :\equiv \mathsf{cast} : \mathcal{R}(U) \to \mathcal{R}(V)$$

Records are Sheaves

Records are Sheaves

For a cover $U = \bigcup_i U_i$ on X, then:

$$\mathcal{R}(U) \xrightarrow{\quad e \quad} \prod_i \mathcal{R}(U_i) \overset{p}{\underset{q}{\rightrightarrows}} \prod_{i,j} \mathcal{R}(U_i \cap U_j)$$

is an equalizer, where

$$\begin{aligned} e &= \lambda r.\lambda i.\ \mathsf{cast}_{U_i}(r) \\ p &= \lambda f.\lambda i.\lambda j.\ \mathsf{cast}_{U_i \cap U_j}(f(i)) \\ q &= \lambda f.\lambda i.\lambda j.\ \mathsf{cast}_{U_i \cap U_j}(f(j)) \end{aligned}$$

Records are Sheaves

For a cover $U = \bigcup_i U_i$ on X, then:

$$
\begin{tikzcd}
\mathcal{R}(U) \arrow[rr, "e"] & & \prod_i \mathcal{R}(U_i) \arrow[r, shift left, "p"] \arrow[r, shift right, "q"'] & \prod_{i,j} \mathcal{R}(U_i \cap U_j) \\
& & & \\
& & \Gamma \arrow[uu, "m"'] \arrow[uull, dashed, "!u"] &
\end{tikzcd}
$$

where

$$
\begin{aligned}
e &= \lambda r.\lambda i.\ \mathsf{cast}_{U_i}(r) \\
p &= \lambda f.\lambda i.\lambda j.\ \mathsf{cast}_{U_i \cap U_j}(f(i)) \\
q &= \lambda f.\lambda i.\lambda j.\ \mathsf{cast}_{U_i \cap U_j}(f(j))
\end{aligned}
$$

Corecords as Sums

Corecords as Sums

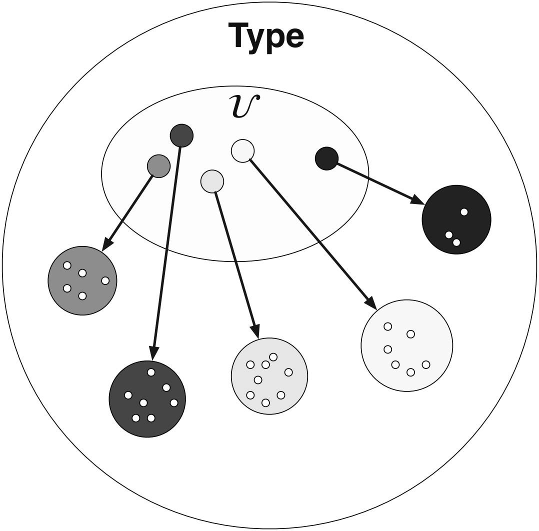
Corecords (extensible variants): the sum of the image of $El_{\mathcal{U}}$ in Type restricted to a subset of $\mathcal{U}$.

Corecords as Sums

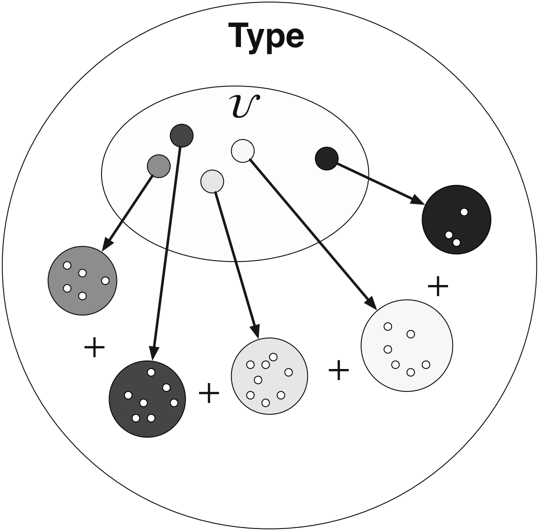
Corecords (extensible variants): the sum of the image of $El_{\mathcal{U}}$ in Type restricted to a subset of $\mathcal{U}$.

$$\mathsf{corecord}_{\mathcal{U}} \leadsto \sum_{\mathcal{V} \subseteq \mathcal{U}} \sum_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

Corecords as Sums



Corecords as Sums



Corecords as Sums

$$\mathsf{corecord}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V} \subseteq \mathcal{U}} \sum_{\mathcal{V}} El_{\mathcal{U}}|_{\mathcal{V}}$$

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

Records in Haskell

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

Records in Haskell

$$
\begin{array}{l}
\textbf{data}\ \mathrm{Rec} :: (\mathcal{U} \to *) \to [\ \mathcal{U}\ ] \to *\ \textbf{where} \\
\quad \mathrm{RNil} :: \mathrm{Rec}\ \mathrm{el}_{\mathcal{U}}\ \text{'[]}
\end{array}
$$

Records in Haskell

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

Records in Haskell (Actually)

Records in Haskell (Actually)

data TyFun :: $* \to * \to *$

Records in Haskell (Actually)

data TyFun :: $* \to * \to *$
type family (f :: TyFun k l $\to *$) \$ (x :: k) :: l

Records in Haskell (Actually)

data TyFun :: $* \to * \to *$
type family (f :: TyFun k l $\to *$) \$ (x :: k) :: l

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

Recovering HList

Recovering HList

data Id :: (TyFun k k) $\rightarrow$ $*$ **where**
type instance Id $ x = x

Recovering HList

```
data Id :: (TyFun k k) → * where
type instance Id $ x = x

type HList rs = Rec Id rs
```

Recovering HList

data Id :: (TyFun k k) $\rightarrow *$ **where**
type instance Id \$ x = x

type HList rs = Rec Id rs

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

Recovering HList

```
data Id :: (TyFun k k) → ∗ where
type instance Id $ x = x

type HList rs = Rec Id rs

ex :: HList [ℤ, Bool, String]
ex = 34 :& True :& "vinyl" :& RNil
```

Validating Records

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

Validating Records

bob :: Rec $\mathsf{El}_{\mathcal{A}}$ [Name, Email Work]
bob = Name =: "Robert␣Harper"
  $\oplus$ Email Work =: "rwh@cs.cmu.edu"

Validating Records

bob :: Rec $\text{El}_{\mathcal{A}}$ [Name, Email Work]
bob = Name =: "Robert␣Harper"
    $\oplus$ Email Work =: "rwh@cs.cmu.edu"

validateName :: **String** $\rightarrow$ **Either** Error **String**
validateEmail :: **String** $\rightarrow$ **Either** Error **String**
validatePhone :: [$\mathbb{N}$] $\rightarrow$ **Either** Error [$\mathbb{N}$]

Validating Records

bob :: Rec El$_{\mathcal{A}}$ [Name, Email Work]
bob = Name =: "Robert␣Harper"
    $\oplus$ Email Work =: "rwh@cs.cmu.edu"

validateName :: **String** $\rightarrow$ **Either** Error **String**
validateEmail :: **String** $\rightarrow$ **Either** Error **String**
validatePhone :: [$\mathbb{N}$] $\rightarrow$ **Either** Error [$\mathbb{N}$]

unnnnnhhh...

Validating Records

bob :: Rec $\mathrm{El}_{\mathcal{A}}$ [Name, Email Work]
bob = Name =: "Robert␣Harper"
  $\oplus$ Email Work =: "rwh@cs.cmu.edu"

validateName :: **String** $\rightarrow$ **Either** Error **String**
validateEmail :: **String** $\rightarrow$ **Either** Error **String**
validatePhone :: [$\mathbb{N}$] $\rightarrow$ **Either** Error [$\mathbb{N}$]

unnnnnhhh...

validateContact
  :: Rec $\mathrm{El}_{\mathcal{A}}$ [Name, Email Work]
  $\rightarrow$ **Either** Error (Rec $\mathrm{El}_{\mathcal{A}}$ [Name, Email Work])

Welp.

Effects inside records

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

Effects inside records

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

Effects inside records

```
data Rec :: (TyFun 𝒰 * → *) → (* → *) → [ 𝒰 ] → * where
   RNil :: Rec el_𝒰 f '[]
   (:&) :: !(f (el_𝒰 $ r)) → !(Rec el_𝒰 f rs) → Rec el_𝒰 f (r ': rs)

(=:) : Applicative f ⇒ sing r → el_𝒰 $ r → Rec el_𝒰 f '[r]
```

Effects inside records

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

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

Effects inside records

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

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

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

Effects inside records

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

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

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

Compositional Validation

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

Compositional Validation

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

Compositional Validation

```
type Rec_A = Rec El_A
bob :: Rec_A Identity [Name, Email Work]
bob = Name =: "Robert␣Harper"
    ⊕ Email Work =: "rwh@cs.cmu.edu"
```

Compositional Validation

type $\mathsf{Rec}_{\mathcal{A}}$ = Rec $\mathsf{El}_{\mathcal{A}}$
bob :: $\mathsf{Rec}_{\mathcal{A}}$ Identity [Name, Email Work]
bob = Name =: "Robert␣Harper"
   $\oplus$ Email Work =: "rwh@cs.cmu.edu"

Compositional Validation

$\textbf{type}\ \text{Validator a} = \text{a} \rightarrow \textbf{Either}\ \text{Error a}$

Compositional Validation

type Validator a = a $\rightarrow$ **Either** Error a
validateName :: Rec$_{\mathcal{A}}$ Validator '[Name]
validatePhone :: $\forall \ell$. Rec$_{\mathcal{A}}$ Validator '[Phone ℓ]
validateEmail :: $\forall \ell$. Rec$_{\mathcal{A}}$ Validator '[Email ℓ]

Compositional Validation

type Validator a = a $\rightarrow$ **Either** Error a
validateName :: Rec$_{\mathcal{A}}$ Validator '[Name]
validatePhone :: $\forall \ell$. Rec$_{\mathcal{A}}$ Validator '[Phone ℓ]
validateEmail :: $\forall \ell$. Rec$_{\mathcal{A}}$ Validator '[Email ℓ]

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

Compositional Validation

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

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

validateContact :: Rec_𝒜 Validator TotalContact
validateContact = validateName
                ⊕ validateEmail
                ⊕ validateEmail
                ⊕ validatePhone
                ⊕ 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 $(\rightarrow)$ Identity (**Either** Error)

Record Operators

newtype Lift o f g x = Lift $\{$ runLift :: f x ‘o‘ g x $\}$

type Validator = Lift $(\rightarrow)$ Identity (**Either** Error)

($\circledcirc$) :: $\mathrm{Rec}_{\mathcal{U}}$ (Lift $(\rightarrow)$ f g) rs $\rightarrow$ $\mathrm{Rec}_{\mathcal{U}}$ f rs $\rightarrow$ $\mathrm{Rec}_{\mathcal{U}}$ g rs

Record Operators

newtype Lift o f g x = Lift $\{$ runLift :: f x ‘o‘ g x $\}$

type Validator = Lift $(\rightarrow)$ Identity (**Either** Error)

($\circledast$) :: Rec$_{\mathcal{U}}$ (Lift $(\rightarrow)$ f g) rs $\rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ Rec$_{\mathcal{U}}$ g rs

rdist :: Applicative f $\Rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ f (Rec$_{\mathcal{U}}$ Identity rs)

Compositional Validation

newtype Lift o f g x = Lift $\{$ runLift :: f x ‘o‘ g x $\}$
type Validator = Lift $(\rightarrow)$ Identity (**Either** Error)
($\circledast$) :: Rec$_{\mathcal{U}}$ (Lift $(\rightarrow)$ f g) rs $\rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ Rec$_{\mathcal{U}}$ g rs
rdist :: Applicative f $\Rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ f (Rec$_{\mathcal{U}}$ Identity rs)

Compositional Validation

newtype Lift o f g x = Lift $\{$ runLift :: f x ‘o‘ g x $\}$
type Validator = Lift $(\rightarrow)$ Identity (**Either** Error)
($\circledast$) :: Rec$_{\mathcal{U}}$ (Lift $(\rightarrow)$ f g) rs $\rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ Rec$_{\mathcal{U}}$ g rs
rdist :: Applicative f $\Rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ f (Rec$_{\mathcal{U}}$ Identity rs)

validateContact :: Rec$_{\mathcal{A}}$ Validator TotalContact

Compositional Validation

newtype Lift o f g x = Lift $\{$ runLift :: f x ‘o‘ g x $\}$
type Validator = Lift $(\rightarrow)$ Identity (**Either** Error)
($\circledast$) :: Rec$_{\mathcal{U}}$ (Lift $(\rightarrow)$ f g) rs $\rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ Rec$_{\mathcal{U}}$ g rs
rdist :: Applicative f $\Rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ f (Rec$_{\mathcal{U}}$ Identity rs)

validateContact :: Rec$_{\mathcal{A}}$ Validator TotalContact

bobValid :: Rec$_{\mathcal{A}}$ (**Either** Error) [Name, Email Work]

Compositional Validation

newtype Lift o f g x = Lift $\{$ runLift :: f x ‘o‘ g x $\}$
type Validator = Lift $(\rightarrow)$ Identity (**Either** Error)
($\circledast$) :: Rec$_{\mathcal{U}}$ (Lift $(\rightarrow)$ f g) rs $\rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ Rec$_{\mathcal{U}}$ g rs
rdist :: Applicative f $\Rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ f (Rec$_{\mathcal{U}}$ Identity rs)

validateContact :: Rec$_{\mathcal{A}}$ Validator TotalContact

bobValid :: Rec$_{\mathcal{A}}$ (**Either** Error) [Name, Email Work]
bobValid = cast validateContact $\circledast$ bob

Compositional Validation

newtype Lift o f g x = Lift $\{$ runLift :: f x ‘o‘ g x $\}$
type Validator = Lift $(\rightarrow)$ Identity (**Either** Error)
($\circledast$) :: Rec$_{\mathcal{U}}$ (Lift $(\rightarrow)$ f g) rs $\rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ Rec$_{\mathcal{U}}$ g rs
rdist :: Applicative f $\Rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ f (Rec$_{\mathcal{U}}$ Identity rs)

validateContact :: Rec$_{\mathcal{A}}$ Validator TotalContact

bobValid :: Rec$_{\mathcal{A}}$ (**Either** Error) [Name, Email Work]
bobValid = cast validateContact $\circledast$ bob

validBob :: **Either** Error (Rec$_{\mathcal{A}}$ 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)
(⊛) :: 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)

validateContact :: Rec_A Validator TotalContact

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

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

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 PlainRec_U rs = Rec_U Identity rs
type LazyRec_U rs = Rec_U Thunk rs
```

Concurrent Records with Async

Concurrent Records with Async

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

Concurrent Records with Async

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

Concurrent Records with Async

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

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

Concurrent Records with Async

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

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

Concurrent Records with Async

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

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

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

Concurrent Records with Async

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

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

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

$(\circledcirc\$) :: (\forall\, a.\ f\ a \rightarrow g\ a) \rightarrow \mathrm{Rec}_{\mathcal{U}}\ f\ rs \rightarrow \mathrm{Rec}_{\mathcal{U}}\ g\ rs$

Concurrent Records with Async

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

$(\circledS) :: (\forall\, a.\ f\ a \to g\ a) \to \mathrm{Rec}_{\mathcal{U}}\ f\ rs \to \mathrm{Rec}_{\mathcal{U}}\ g\ rs$

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

Concurrent Records with Async

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

(\$) :: ($\forall$ 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 \$ fetchBob

Concurrent Records with Async

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

(⊚$⊚) :: (∀ a. f a → g a) → $\text{Rec}_{\mathcal{U}}$ f rs → $\text{Rec}_{\mathcal{U}}$ g rs

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

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

Concurrent Records with Async

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

$(\circledS) :: (\forall\, a.\ f\ a \rightarrow g\ a) \rightarrow \mathrm{Rec}_{\mathcal{U}}\ f\ rs \rightarrow \mathrm{Rec}_{\mathcal{U}}\ g\ rs$

bobConcurrently :: $\mathrm{Rec}_{\mathcal{A}}$ Concurrently [Name, Email Work]
bobConcurrently = Concurrently $\circledS$ fetchBob

concurrentBob :: Concurrently ($\mathrm{Rec}_{\mathcal{A}}$ Identity [...])
concurrentBob = rdist bobConcurrently

Concurrent Records with Async

Concurrent Records with Async

fetchBob :: Rec$_{\mathcal{A}}$ **IO** [Name, Email Work]
bobConcurrently :: Rec$_{\mathcal{A}}$ Concurrently [Name, Email Work]
concurrentBob :: Concurrently (Rec$_{\mathcal{A}}$ Identity [...])

Concurrent Records with Async

fetchBob :: $\text{Rec}_{\mathcal{A}}$ **IO** [Name, Email Work]
bobConcurrently :: $\text{Rec}_{\mathcal{A}}$ Concurrently [Name, Email Work]
concurrentBob :: Concurrently ($\text{Rec}_{\mathcal{A}}$ Identity [...])

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

Concurrent Records with Async

fetchBob :: $\text{Rec}_{\mathcal{A}}$ **IO** [Name, Email Work]
bobConcurrently :: $\text{Rec}_{\mathcal{A}}$ Concurrently [Name, Email Work]
concurrentBob :: Concurrently ($\text{Rec}_{\mathcal{A}}$ Identity [...])

bob :: **IO** ($\text{Rec}_{\mathcal{A}}$ Identity [Name, Email Work])
bob = runConcurrently concurrentBob

Containers: The Syntax for Data Types

$$\frac{}{\textsf{container} : \mathsf{Type}}$$

Containers: The Syntax for Data Types

$$\frac{}{\textsf{container} : \textsf{Type}} \qquad \frac{\mathcal{U} : \textsf{Type} \quad El_{\mathcal{U}} : \mathcal{U} \to \textsf{Type}}{\mathcal{U} \lhd El_{\mathcal{U}} : \textsf{container}}$$

Containers: The Syntax for Data Types

$$\frac{}{\textsf{container} : \textsf{Type}} \qquad \frac{\mathcal{U} : \textsf{Type} \quad El_{\mathcal{U}} : \mathcal{U} \to \textsf{Type}}{\mathcal{U} \triangleleft El_{\mathcal{U}} : \textsf{container}}$$

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

Containers: The Syntax for Data Types

$$\frac{}{\text{container} : \mathsf{Type}} \qquad \frac{\mathcal{U} : \mathsf{Type} \quad El_{\mathcal{U}} : \mathcal{U} \to \mathsf{Type}}{\mathcal{U} \lhd El_{\mathcal{U}} : \text{container}}$$

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

Containers: The Syntax for Data Types

$$\frac{}{\text{container} : \mathsf{Type}} \qquad \frac{\mathcal{U} : \mathsf{Type} \quad El_{\mathcal{U}} : \mathcal{U} \to \mathsf{Type}}{\mathcal{U} \lhd El_{\mathcal{U}} : \text{container}}$$

$$\frac{C : \text{container}}{C.\text{Sh} : \mathsf{Type}} \qquad \frac{C \leadsto \mathcal{U} \lhd El_{\mathcal{U}}}{C.\text{Sh} \leadsto \mathcal{U}}$$

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

Containers: The Syntax for Data Types

$$\frac{}{\mathsf{container} : \mathsf{Type}} \qquad \frac{\mathcal{U} : \mathsf{Type} \quad El_{\mathcal{U}} : \mathcal{U} \to \mathsf{Type}}{\mathcal{U} \lhd El_{\mathcal{U}} : \mathsf{container}}$$

$$\frac{C : \mathsf{container}}{C.\mathsf{Sh} : \mathsf{Type}} \qquad \frac{C \leadsto \mathcal{U} \lhd El_{\mathcal{U}}}{C.\mathsf{Sh} \leadsto \mathcal{U}}$$

$$\frac{C : \mathsf{container}}{C.\mathsf{Po} : C.\mathsf{Sh} \to \mathsf{Type}} \qquad \frac{C \leadsto \mathcal{U} \lhd El_{\mathcal{U}}}{C.\mathsf{Po} \leadsto El_{\mathcal{U}}}$$

Restricting Containers

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

Restricting Containers

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

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

Container Lifting

$$\frac{C : \mathsf{container} \quad F : \mathsf{Type} \to \mathsf{Type}}{C \uparrow F : \mathsf{container}}$$

Container Lifting

$$\frac{C : \textsf{container} \quad F : \textsf{Type} \to \textsf{Type}}{C \uparrow F : \textsf{container}}$$

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

A Menagerie of Quantifiers

A Menagerie of Quantifiers

Dependent Products:

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

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

A Menagerie of Quantifiers

Dependent Sums:

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

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

A Menagerie of Quantifiers

Inductive Types:

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, x : A \vdash B : \mathsf{Type}}{\Gamma \vdash \mathrm{W}_A\, B : \mathsf{Type}}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma, v : B[a] \vdash b : \mathrm{W}_A\, B}{\Gamma \vdash \mathsf{sup}(a; v.\, b) : \mathrm{W}_A\, B}$$

A Menagerie of Quantifiers

Coinductive Types:

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, x : A \vdash B : \mathsf{Type}}{\Gamma \vdash \mathrm{M}_A\, B : \mathsf{Type}}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma, v : B[a] \vdash b : \infty\,(\mathrm{M}_A\, B)}{\Gamma \vdash \mathsf{inf}(a; v.\, b) : \mathrm{M}_A\, B}$$

A Scheme for Quantifiers

A Scheme for Quantifiers

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

Quantifiers Give Containers Semantics

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

Quantifiers Give Containers Semantics

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

$$\frac{C : \text{container} \quad Q \text{ quantifier}}{[\![C]\!]_Q : \mathsf{Type}}$$

Quantifiers Give Containers Semantics

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

$$\frac{C : \text{container} \quad Q \text{ quantifier}}{[\![C]\!]_Q : \mathsf{Type}}$$

$$\frac{C \rightsquigarrow \mathcal{U} \lhd El_{\mathcal{U}}}{[\![C]\!]_Q \rightsquigarrow Q_{\mathcal{U}} El_{\mathcal{U}}}$$

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.

$$\mathsf{Rec}\ El_{\mathcal{U}}\ F\ rs \cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs \ni -} \uparrow F]\!]_{\Pi}$$

Vinyl Records as Containers

Records and corecords are finite products and sums respectively.

$$\begin{aligned}\mathsf{Rec}\ El_{\mathcal{U}}\ F\ rs &\cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs\ni -} \uparrow F]\!]_{\Pi} \\ \mathsf{CoRec}\ El_{\mathcal{U}}\ F\ rs &\cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs\ni -} \uparrow F]\!]_{\Sigma}\end{aligned}$$

Vinyl Records as Containers

Records and corecords are finite products and sums respectively.

$$\begin{aligned}\mathsf{Rec}\ El_{\mathcal{U}}\ F\ rs &\cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs\ni -} \uparrow F]\!]_{\Pi} \\ \mathsf{CoRec}\ El_{\mathcal{U}}\ F\ rs &\cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs\ni -} \uparrow F]\!]_{\Sigma}\end{aligned}$$

$$???\ El_{\mathcal{U}}\ F\ rs \cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs\ni -} \uparrow F]\!]_{\mathrm{W}}$$

Vinyl Records as Containers

Records and corecords are finite products and sums respectively.

$$\mathsf{Rec}\ El_{\mathcal{U}}\ F\ rs \cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs\ni -} \uparrow F]\!]_{\Pi}$$
$$\mathsf{CoRec}\ El_{\mathcal{U}}\ F\ rs \cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs\ni -} \uparrow F]\!]_{\Sigma}$$

$$???\ El_{\mathcal{U}}\ F\ rs \cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs\ni -} \uparrow F]\!]_{\mathrm{W}}$$
$$???\ El_{\mathcal{U}}\ F\ rs \cong [\![(\mathcal{U} \triangleleft El_{\mathcal{U}})|_{rs\ni -} \uparrow F]\!]_{\mathrm{M}}$$

Questions