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 } −− ˆ Error
```

Records in GHC 7.8

Records are...

Records in GHC 7.8

anticompositional

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?

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

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 :: [∗])
(=:) : 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 :: [∗])
(=:) : s ::: t → t → Rec ’[s ::: t]
(⊕) : Rec ss → Rec ts → Rec (ss ++ ts)
```

Roll Your Own in Haskell

Roll Your Own in Haskell

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

Universes à la Tarski

Universes à la Tarski

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

Universes à la Tarski

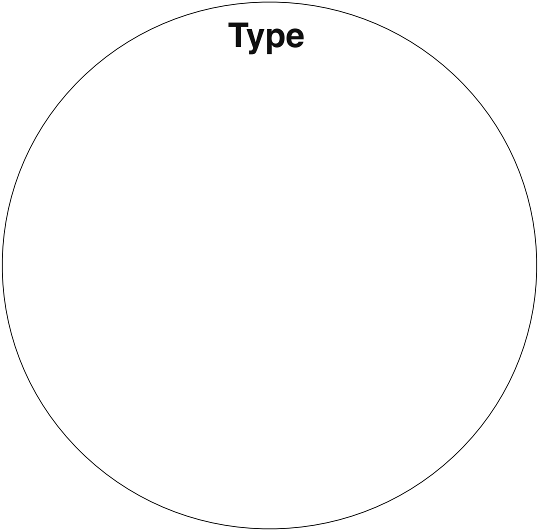
- A type $\mathcal{U}$ of **codes** for types.
- Function $[\![-]\!]_{\mathcal{U}} : \mathcal{U} \to$ **Type**.

Universes à la Tarski

- A type $\mathcal{U}$ of **codes** for types.
- Function $[\![-]\!]_{\mathcal{U}} : \mathcal{U} \to \textbf{Type}$.

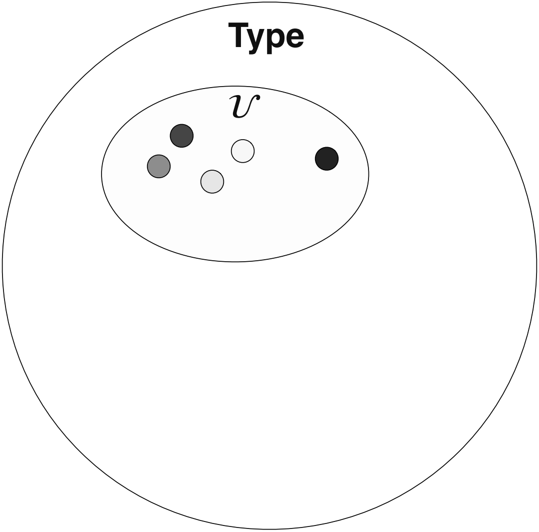
$$\frac{\Gamma \vdash s : \mathcal{U}}{\Gamma \vdash [\![s]\!]_{\mathcal{U}} : \textbf{Type}}$$

Universes à la Tarski

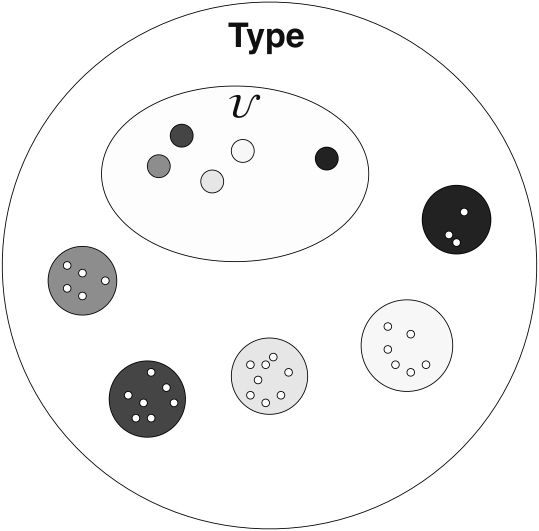


Type

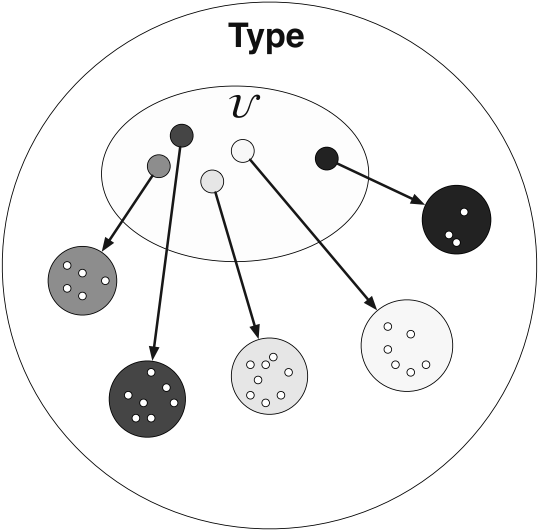
Universes à la Tarski



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:

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

A Closed Universe

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

- Statics:

$$\frac{}{\mathcal{A} : \textbf{Type}} \qquad \frac{}{\text{Label} : \textbf{Type}}$$

A Closed Universe

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

- Statics:

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

A Closed Universe

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

- Statics:

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

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

A Closed Universe

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

- Statics:

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

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

A Closed Universe

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

- Statics:

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

$$\frac{}{\text{Name} : \mathcal{A}} \qquad \frac{\ell : \text{Label}}{\text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{[\![s]\!]_{\mathcal{A}} : \textbf{Type}}$$

A Closed Universe

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

- Statics:

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

$$\frac{}{\text{Name} : \mathcal{A}} \qquad \frac{\ell : \text{Label}}{\text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{[\![s]\!]_{\mathcal{A}} : \textbf{Type}}$$

- Dynamics:

A Closed Universe

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

- Statics:

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

$$\frac{}{\text{Name} : \mathcal{A}} \qquad \frac{\ell : \text{Label}}{\text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{[\![s]\!]_{\mathcal{A}} : \textbf{Type}}$$

- Dynamics:

$$\frac{}{[\![\text{Name}]\!]_{\mathcal{A}} \rightsquigarrow \textbf{string}}$$

A Closed Universe

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

- Statics:

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

$$\frac{}{\mathrm{Name} : \mathcal{A}} \qquad \frac{\ell : \mathrm{Label}}{\mathrm{Phone}[\ell], \mathrm{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{[\![s]\!]_{\mathcal{A}} : \textbf{Type}}$$

- Dynamics:

$$\frac{}{[\![\mathrm{Name}]\!]_{\mathcal{A}} \rightsquigarrow \textbf{string}} \qquad \frac{}{[\![\mathrm{Email}[\ell]]\!]_{\mathcal{A}} \rightsquigarrow \textbf{string}}$$

A Closed Universe

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

- Statics:

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

$$\frac{}{\text{Name} : \mathcal{A}} \qquad \frac{\ell : \text{Label}}{\text{Phone}[\ell], \text{Email}[\ell] : \mathcal{A}} \qquad \frac{s : \mathcal{A}}{\llbracket s \rrbracket_{\mathcal{A}} : \textbf{Type}}$$

- Dynamics:

$$\frac{}{\llbracket \text{Name} \rrbracket_{\mathcal{A}} \leadsto \textbf{string}} \qquad \frac{}{\llbracket \text{Email}[\ell] \rrbracket_{\mathcal{A}} \leadsto \textbf{string}}$$

$$\frac{}{\llbracket \text{Phone}[\ell] \rrbracket_{\mathcal{A}} \leadsto \mathbb{N}\ \textbf{list}}$$

Records as Products

Records as Products

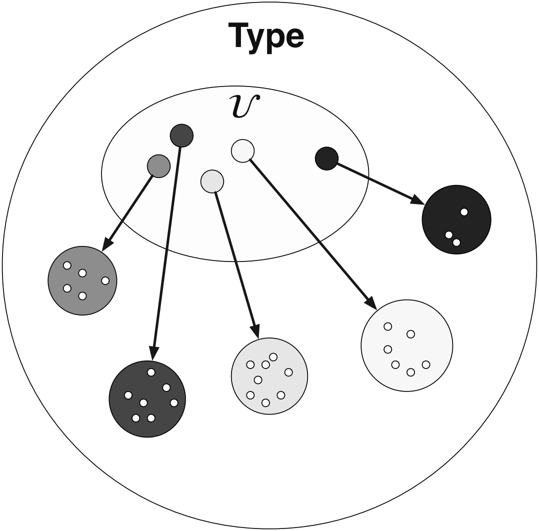
Records: the product of the image of $[\![-]\!]_{\mathcal{U}}$ in **Type** restricted to a subset of the domain.

Records as Products

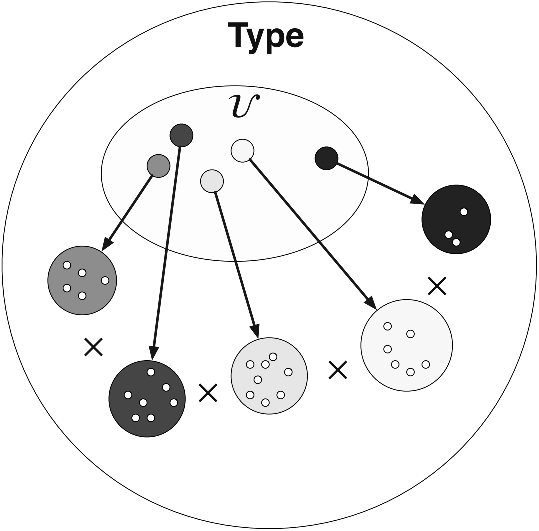
Records: the product of the image of $[\![-]\!]_{\mathcal{U}}$ in **Type** restricted to a subset of the domain.

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}:\textbf{Type}} \sum_{i:\mathcal{V}\hookrightarrow\mathcal{U}} \prod_{\mathcal{V}} [\![-]\!]_{\mathcal{U}} \circ i$$

Records as Products



Records as Products



Records as Products

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}:\textbf{Type}} \sum_{i:\mathcal{V}\hookrightarrow\mathcal{U}} \prod_{\mathcal{V}} [\![-]\!]_{\mathcal{U}} \circ i$$

Example Record

$$\mathrm{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}:\textbf{Type}} \sum_{i:\mathcal{V}\hookrightarrow\mathcal{U}} \prod_{\mathcal{V}} [\![-]\!]_{\mathcal{U}} \circ i$$

Example Record

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}:\textbf{Type}} \sum_{i:\mathcal{V}\hookrightarrow\mathcal{U}} \prod_{\mathcal{V}} [\![-]\!]_{\mathcal{U}} \circ i$$

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

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

Example Record

$$\text{record}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}:\textbf{Type}} \sum_{i:\mathcal{V}\hookrightarrow\mathcal{U}} \prod_{\mathcal{V}} [\![-]\!]_{\mathcal{U}} \circ i$$

$$\begin{aligned}
\mathcal{A}' &\rightsquigarrow \{\text{Name}, \text{Email } \textit{Work}\} \\
\textit{ex} &: \text{record}_{\mathcal{U}} \\
\textit{ex} &\rightsquigarrow \langle \mathcal{A}', \lambda x.x, \lambda. \\
&\qquad \{\text{Name} \mapsto \text{"Robert Harper"}; \\
&\qquad\ \text{Email } \textit{Work} \mapsto \text{"rwh@cs.cmu.edu"}\}\rangle
\end{aligned}$$

Corecords as Sums

Corecords as Sums

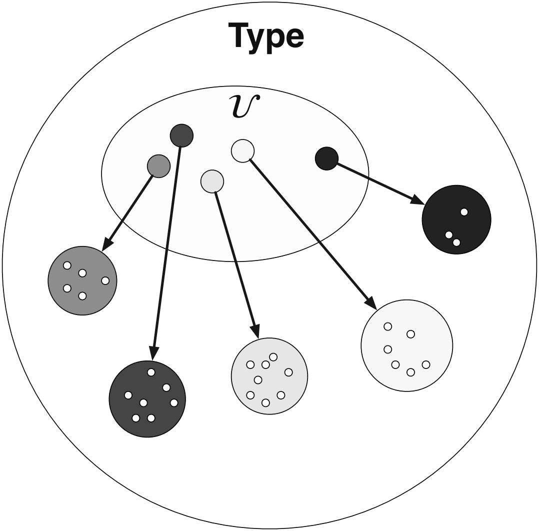
Corecords (extensible variants): the sum of the image of $[\![-]\!]_{\mathcal{U}}$ in **Type** restricted to a subset of the domain.

Corecords as Sums

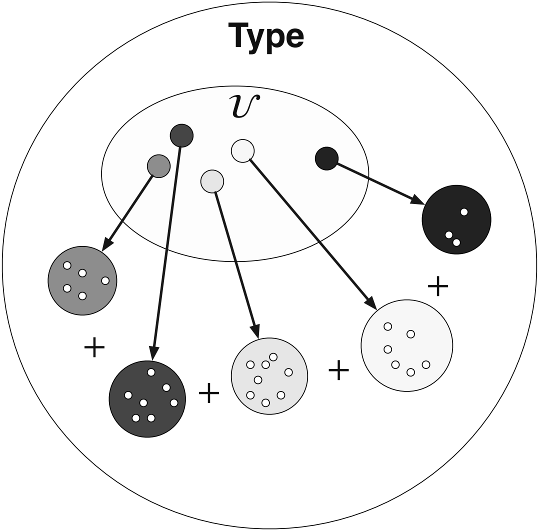
Corecords (extensible variants): the sum of the image of $[\![-]\!]_{\mathcal{U}}$ in **Type** restricted to a subset of the domain.

$$\text{corecord}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}:\textbf{Type}} \sum_{i:\mathcal{V}\hookrightarrow\mathcal{U}} \sum_{\mathcal{V}} [\![-]\!]_{\mathcal{U}} \circ i$$

Corecords as Sums



Corecords as Sums



Corecords as Sums

$$\text{corecord}_{\mathcal{U}} \rightsquigarrow \sum_{\mathcal{V}:\textbf{Type}} \sum_{i:\mathcal{V}\hookrightarrow\mathcal{U}} \sum_{\mathcal{V}} [\![-]\!]_{\mathcal{U}} \circ i$$

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 $[\![-]\!]_{\mathcal{U}}$

Doing it in Haskell

- Create a universe $\mathcal{U}$ at the type-level
- Use type families to approximate $[\![-]\!]_{\mathcal{U}}$
- Parameterize Rec by $\mathcal{U}$, $[\![-]\!]_{\mathcal{U}}$?

Records in Haskell

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

Records in Haskell

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

  RNil :: Rec $[\![-]\!]_{\mathcal{U}}$ '[]

Records in Haskell

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

Recovering HList

Recovering HList

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

Recovering HList

type HList rs = Rec (Λτ. τ) rs

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

Recovering HList

```
type HList rs = Rec (Λτ. τ) rs

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

Validating Records

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

Validating Records

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

Validating Records

bob :: Rec $[\![-]\!]_{\mathcal{A}}$ [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 $[\![-]\!]_{\mathcal{A}}$ [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 $\llbracket - \rrbracket_{\mathcal{A}}$ [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 $\llbracket - \rrbracket_{\mathcal{A}}$ [Name, Email Work]
  $\rightarrow$ **Either** Error (Rec $\llbracket - \rrbracket_{\mathcal{A}}$ [Name, Email Work])

Welp.

Effects inside records

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

Effects inside records

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

Compositional Validation

type $\text{Rec}_{\mathcal{A}}$ = Rec $[\![-]\!]_{\mathcal{A}}$

Compositional Validation

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

Compositional Validation

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

Compositional Validation

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

Compositional Validation

type Validator a = a → **Either** 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 $\rightarrow$ **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]

validateContact :: Rec$_{\mathcal{A}}$ 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 ($\rightarrow$) Identity (**Either** Error)

($\circledcirc\!\!\!\!\star$) :: Rec$_{\mathcal{U}}$ (Lift ($\rightarrow$) f g) rs $\rightarrow$ Rec$_{\mathcal{U}}$ f rs $\rightarrow$ 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)
(⊛) :: 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 ($\rightarrow$) 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])
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$_{\mathcal{U}}$ rs = Rec$_{\mathcal{U}}$ Identity rs

Laziness as an effect

newtype Identity a = Identity $\{$ runIdentity :: a $\}$

data Thunk a = Thunk $\{$ unThunk :: a $\}$

type PlainRec$_{\mathcal{U}}$ rs = Rec$_{\mathcal{U}}$ Identity rs

type LazyRec$_{\mathcal{U}}$ rs = Rec$_{\mathcal{U}}$ Thunk rs

Concurrent Records with Async

Concurrent Records with Async

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

Concurrent Records with Async

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

Concurrent Records with Async

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

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

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

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

Concurrent Records with Async

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

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

fetchBob :: $\mathrm{Rec}_{\mathcal{A}}$ **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 }

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

Concurrent Records with Async

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

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

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

Concurrent Records with Async

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

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

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

Concurrent Records with Async

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

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

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

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

Concurrent Records with Async

$$\textbf{newtype}\ \text{Concurrently a}$$
$$\quad = \text{Concurrently}\ \{\ \text{runConcurrently} :: \textbf{IO}\ \text{a}\ \}$$

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

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

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

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

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

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

bob :: **IO** (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