Programming in Vinyl

Jon Sterling
FOBO

March 31, 2014

Records in GHC 7.8

Haskell Records are nominally typed.

Records in GHC 7.8

Haskell Records are nominally typed.

$$\frac{\Gamma \vdash M.S \rightsquigarrow \{\vec{rs}\} \quad \Gamma \vdash N.T \rightsquigarrow \{\vec{rs}\} \quad \Gamma \vdash x : M.S}{\Gamma \nvdash x : N.T}$$

Records in GHC 7.8

Records may not share field names.

Records in GHC 7.8

Records may not share field names.

data R = R { x :: X }

Records in GHC 7.8

Records may not share field names.

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

Records in GHC 7.8

Records are...

Records in GHC 7.8

anticompositional

Records in Standard ML

Records in Standard ML

slightly better

Records in Standard ML

Records in Standard ML

Records are permutative, and not nominal.

Records in Standard ML

Records are permutative, and not nominal.

$$\frac{\Gamma \vdash x : \{\vec{ss}\} \quad \Gamma \vdash ts \in \text{permutations}(\vec{ss})}{\Gamma \vdash x : \{\vec{ts}\}}$$

Records in OCaml

OCaml records are

Records in OCaml

OCaml records are

- structural

Records in OCaml

OCaml records are

- structural
- endowed with a subtyping relation

Records in OCaml

OCaml records are

- structural
- endowed with a subtyping relation
- but more importantly...

Records in OCaml

row polymorphic

Row Polymorphism

Row Polymorphism

Pick out the parts you care about, and quantify the rest.
— Someone wise

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?

$$\frac{x : \{a : A\}}{f(x) : \{a : A, b : B\}}$$

Row Polymorphism

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

NOPE

Row Polymorphism

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

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

Row Polymorphism

Row Polymorphism

- supports type inference

Row Polymorphism

- supports type inference
- is compositional

Row Polymorphism

- supports type inference
- is compositional
- is **modular**

Row Polymorphism

- supports type inference
- is compositional
- is **modular**

Roll Your Own in Haskell

Roll Your Own in Haskell

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

Roll Your Own in Haskell

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

data Rec :: [$*$] $\rightarrow$ $*$ **where**

Roll Your Own in Haskell

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

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

Roll Your Own in Haskell

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

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

Roll Your Own in Haskell

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

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

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

Roll Your Own in Haskell

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

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

class s $\in$ (rs :: [$*$])
(=:) : k ::: t $\rightarrow$ t $\rightarrow$ Rec '[k ::: t]

Roll Your Own in Haskell

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

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

class s $\in$ (rs :: [$*$])
(=:) : k ::: t $\rightarrow$ t $\rightarrow$ Rec '[k ::: t]
($<$+$>$) : Rec ss $\rightarrow$ Rec ts $\rightarrow$ Rec (ss ++ ts)

Roll Your Own in Haskell

Roll Your Own in Haskell

$$\frac{x :: \text{"a"} ::: A \in \vec{rs} \implies \text{Rec}\ \vec{rs}}{f\ x :: (\text{"a"} ::: A \in \vec{rs}, \text{"b"} ::: B \in \vec{rs}) \implies \text{Rec}\ \vec{rs}}$$