A Second Look at Overloading

Marius Weidner — Chair of Programming Languages — Seminar '22

```haskell
class Eq α where
  eq :: α → α → Bool

instance Eq Nat where
  eq Zero    Zero    = True
  eq (Suc x) (Suc y) = eq x y
  eq _       _       = False


instance Eq α ⇒ Eq [α] where
  eq []      []      = True
  eq (x : xs) (y : ys) = eq x y &&
                         eq xs ys
  eq _        _      = False


isEq :: Bool
isEq = eq [Zero] [Zero]
```

Haskell

```rust
trait Eq
  fn eq(&self, rhs: &Self) → Bool

impl Eq for Nat
  fn eq(&self, rhs: &Self) → Bool
    match (self, rhs)
      (Zero, Zero)     ⇒ True,
      (Suc(x), Suc(y)) ⇒ x.eq(y),
      (_, _)           ⇒ False

impl<A: Eq> Eq for [A]
  fn eq(&self, rhs: &Self) → Bool
    match (self, rhs)
      ([], []) ⇒ True,
      ([x, xs@..], [y, ys@..])
           ⇒ x.eq(y) && xs.eq(ys),
      (_, _)  ⇒ False

fn is_eq() → Bool
  [Zero].eq(&[Zero])
```

Rust

```
inst eq :: Nat → Nat → Bool
  eq Zero     Zero    = True
  eq (Suc x) (Suc y) = eq x y
  eq _        _       = False

inst eq :: (eq :: α → α → Bool) ⇒ [α] → [α] → Bool
  eq []      []       = True
  eq (x:xs) (y:ys)   = eq x y && eq xs ys
  eq _       _        = False

let isEq = [Zero] == [Zero]
```

Pseudocode

$$e := x$$
$$\mid \lambda x.\, e$$
$$\mid e\, e$$
$$\mid \textbf{let}\ x = e\ \textbf{in}\ e$$

$$\tau := \alpha$$
$$\mid \tau \to \tau$$
$$\sigma := \tau$$
$$\mid \forall \alpha.\, \sigma$$

```
let id   = λx. x                in .. :: ∀a. a → a
let cons = λx. λlst. x : lst in .. :: ∀a. a → [a] → [a]
```

```
let evil = λi. λid. id i    in .. :: Int → (∀a. a → a) → Int
```

Hindley Milner — Syntax

$$e := \quad x$$
$$\mid \; o \quad \text{(if overloaded)}$$
$$\mid \; k \quad (k \in \{\texttt{unit}, \; 42, \; [e_1, .., e_n], \; ..\})$$
$$\mid \; \lambda x. \; e$$
$$\mid \; e \; e$$
$$\mid \; \textbf{let} \; x = e \; \textbf{in} \; e$$

$$p := \quad e$$
$$\mid \; \textbf{inst} \; o : \sigma_T = \sigma \; \textbf{in} \; p$$

```
inst eq : Nat → Nat → Bool = λx. λy. x ≐ y in
inst eq : ∀a. (eq : a → a → Bool) ⟹ [a] → [a] → Bool =
      | λ[]. λ[]. True
      | λ[x : xs].λ[y : ys]. eq x y && eq xs ys in
eq [0] [0]
```

$$\tau ::= \alpha$$

$$\mid \tau \rightarrow \tau$$

$$\mid D \ \tau_1 \ ... \ \tau_n \quad (D \in \{\mathrm{Unit, \ Nat, \ List} \ \tau, ..\}, \ \mathrm{arity}(D) = n)$$

$$\pi_\alpha ::= o_1 : \alpha \rightarrow \tau_1, \ ... \ , o_n : \alpha \rightarrow \tau_n \quad (n \in \mathbb{N}, \ o_i \neq o_j)$$

$$\sigma ::= \tau$$

$$\mid \forall \alpha . \pi_\alpha \Rightarrow \sigma_T$$

$$\sigma_T ::= T \ \alpha_1 \ ... \ \alpha_n \rightarrow \tau \quad (T \in D \cup \{\rightarrow\}, \ \mathrm{tv}(\tau) \subseteq \{\alpha_1, .., \alpha_n\})$$

$$\mid \forall \alpha . \pi_\alpha \Rightarrow \sigma_T \quad (\mathrm{tv}(\pi_\alpha) \subseteq \mathrm{tv}(\sigma_T))$$

System 0 — Syntax

$$\text{(LET)} \qquad \frac{\Gamma \vdash e_1 : \sigma \qquad \Gamma, \; x : \sigma \vdash e_2 : \tau}{\Gamma \vdash \mathbf{let} \; x = e_1 \; \mathbf{in} \; e_2 : \tau}$$

$$\text{(INST)} \qquad \frac{\Gamma \vdash e : \sigma_T \quad \Gamma, \; o : \sigma_T \vdash p : \sigma \quad \forall (o : \sigma_{T'}) \in \Gamma : T \neq T'}{\Gamma \vdash \mathbf{inst} \; o : \sigma_T = e \; \mathbf{in} \; p : \sigma}$$

$$\text{(}\forall\text{I)} \qquad \frac{\Gamma, \; \pi_\alpha \vdash e : \sigma \qquad \text{fresh } \alpha}{\Gamma \vdash e : \forall \alpha . \pi_\alpha \Rightarrow \sigma}$$

$$\text{(}\forall\text{E)} \qquad \frac{\Gamma \vdash e : \forall \alpha . \; \pi_\alpha \Rightarrow \sigma \qquad \Gamma \vdash [\tau/\alpha]\pi_\alpha}{\Gamma \vdash e : [\tau/\alpha]\sigma}$$

$$\Gamma = \{\text{eq} : \text{Nat} \to \text{Nat} \to \text{Bool},$$

$$\text{eq} : \forall \alpha. \ (\text{eq} : \alpha \to \alpha \to \text{Bool}) \Rightarrow [\alpha] \to [\alpha] \to \text{Bool}\}$$

$$\cfrac{\cfrac{\cfrac{\text{eq} : \forall \alpha. \ (\text{eq} : \alpha \to \alpha \to \text{Bool}) \Rightarrow [\alpha] \to [\alpha] \to \text{Bool} \in \Gamma}{\Gamma \vdash \text{eq} : \forall \alpha. \ (\text{eq} : \alpha \to \alpha \to \text{Bool}) \Rightarrow [\alpha] \to [\alpha] \to \text{Bool}} \quad \cfrac{\text{Nat} \to \text{Nat} \to \text{Bool} \in \Gamma}{\Gamma \vdash \text{Nat} \to \text{Nat} \to \text{Bool}} \quad \ldots}{\cfrac{\Gamma \vdash \text{eq} : [\text{Nat}] \to [\text{Nat}] \to \text{Bool}}{\cfrac{\Gamma \vdash \text{eq} \ [0] : [\text{Nat}] \to \text{Bool}}{\Gamma \vdash \text{eq} \ [0] \ [0]} \quad \ldots} \quad \ldots}$$

System 0 — Constraint Solving

$$\llbracket \text{inst } eq : \mathbb{N} \to \mathbb{N} \to \mathbb{B} = e_1 \text{ in}$$
$$\quad \text{inst } eq : \forall \alpha. \ (\text{eq} : \alpha \to \alpha \to \mathbb{B}) \Rightarrow [\alpha] \to [\alpha] \to \mathbb{B} = e_2 \text{ in}$$
$$\quad eq \ [0] \ [0] \rrbracket_\emptyset$$
$$= \llbracket \text{inst } eq : \forall \alpha. \ (\text{eq} : \alpha \to \alpha \to \mathbb{B}) \Rightarrow [\alpha] \to [\alpha] \to \mathbb{B} = .. \text{ in}$$
$$\quad eq \ [0] \ [0] \rrbracket_{\{eq := \lambda x. \ \text{if } x \text{ is } \mathbb{N} \text{ then } \llbracket e_1 \rrbracket \ x\}}$$
$$= \llbracket eq \ [0] \ [0] \rrbracket_{\{eq := \lambda x. \ \text{if } x \text{ is List then } \llbracket e_2 \rrbracket \ x \text{ else } \lambda x. \ \text{if } x \text{ is } \mathbb{N} \text{ then } \llbracket e_1 \rrbracket \ x\}}$$
$$= (\lambda x. \ \text{if } x \text{ is } [\alpha] \text{ then } \llbracket e_2 \rrbracket \ x \text{ else } \lambda x. \ \text{if } x \text{ is } \mathbb{N} \text{ then } \llbracket e_1 \rrbracket \ x) \ [0] \ [0]$$
$$= \llbracket e_2 \rrbracket \ [0] \ [0]$$

System 0 — Semantics

```
inst eq : Nat → Nat → Bool
  = λ.. in
inst eq : ∀α. (eq : α → α → Bool) ⇒ [α] → [α] → Bool
  = λ.. in
eq [0] [0]
```

System O

```
let eq₀ :: Nat → Nat → Bool
  = λ.. in
let eq₁ :: ∀α. (α → α → Bool) → [α] → [α] → Bool
  = λeq₀. λ.. in
eq₁ eq₀ [0] [0]
```

Hindley Milner

System O — Translation to Hindley Milner

```
let max :: ∀β. (gte : β → β → Bool) ⇒
            ∀α. (α ≤ {key: β}) ⇒ α → α → α
  = λx. λy. if gte x.key y.key then x else y in
max {field: "a", key: 1} {field: "b", key: 2}
```

Records + Subtyping

```
inst field : ∀α. ∀β. R₀ α β → α = λR₀ x y. x in
inst key : ∀α. ∀β. R₀ α β → β = λR₀ x y. y in
let max :: ∀β. (gte : β → β → Bool) ⇒
            ∀α. (key : α → β) ⇒ α → α → α
  = λx. λy. if gte (key x) (key y) then x else y in
max (R₀ "a" 1) (R₀ "b" 2)
```

System 0

System 0 — Relationship with Record Typing

11

Repository

github.com/Mari-W/popl

References

- A Second Look at Overloading `1995`
  Martin Odersky, Philip Wadler, Martin Wehr
- A Theory of Type Polymorphism in Programming `1978`
  Hindley Milner