

A Second Look at Overloading

```

inst eq : Nat → Nat → Bool
  eq Z      Z      = True
  eq (S x) (S y) = eq x y
  eq _      _      = False

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

let isEq = eq [Zero] [Zero]

```

Pseudocode — Example

$e ::= x$ $| \lambda x. e$ $| e_1 e_2$ $| \mathbf{let} \ x = e_2 \ \mathbf{in} \ e_1$ $\tau ::= \alpha$ $| \tau_1 \rightarrow \tau_2$ $\sigma ::= \tau$ $| \forall \alpha. \sigma$

```
let id :: ∀a. a → a = λx. x in .. (id 42) .. (id "foo")
let cons :: ∀a. a → [a] → [a] = λx. λlst. x : lst in ..
```

```
let evil :: Nat → (∀a. a → a) → Nat = λi. λid. id i in ..
(λid. .. (id 42) .. (id "foo") ..) (λx. x)
```

$e := x$

| o (if overloaded)

| k ($k \in \{\text{unit}, 42, [e_1, \dots, e_n], \dots\}$)

| $\lambda x. e$

| $e_1 e_2$

| **let** $x = e_2$ **in** e_1

$p := e$

| **inst** $o : \sigma_T = e$ **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]
```

$$\begin{aligned}
\tau &::= \alpha \\
&| \tau_1 \rightarrow \tau_2 \\
&| D \ \tau_1 \ \dots \ \tau_n \quad (D \in \{\text{Unit}, \text{Nat}, \text{List } \tau, ..\}, \text{arity}(D) = n) \\
\pi_\alpha &::= o_1 : \alpha \rightarrow \tau_1, \ \dots \ , o_n : \alpha \rightarrow \tau_n \quad (n \in \mathbb{N}, o_i \neq o_j) \\
\sigma &::= \tau \\
&| \forall \alpha. \pi_\alpha \Rightarrow \sigma_T \\
\sigma_T &::= T \ \alpha_1 \ \dots \ \alpha_n \rightarrow \tau \quad (T \in D \cup \{\rightarrow\}, \text{tv}(\tau) \subseteq \{\alpha_1, \dots, \alpha_n\}) \\
&| \forall \alpha. \pi_\alpha \Rightarrow \sigma_T \quad (\text{tv}(\pi_\alpha) \subseteq \text{tv}(\sigma_T))
\end{aligned}$$

$$(\text{VAR}) \quad \frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

$$(\rightarrow \text{I}) \quad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x. e : \tau_1 \rightarrow \tau_2}$$

$$(\rightarrow \text{E}) \quad \frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}$$

System 0 — Typing

$$\begin{array}{c}
(\text{LET}) \quad \dfrac{\Gamma \vdash e_2 : \sigma \quad \Gamma, x : \sigma \vdash e_1 : \tau}{\Gamma \vdash \mathbf{let} \ x = e_2 \ \mathbf{in} \ e_1 : \tau}
\end{array}$$

$$\begin{array}{c}
(\text{INST}) \quad \dfrac{\Gamma \vdash e : \sigma_T \quad \Gamma, o : \sigma_T \vdash p : \sigma \quad \forall (o : \sigma_{T'}) \in \Gamma : T \neq T'}{\Gamma \vdash \mathbf{inst} \ o : \sigma_T = e \ \mathbf{in} \ p : \sigma}
\end{array}$$

$$\begin{array}{c}
(\forall \text{I}) \quad \dfrac{\Gamma, \pi_\alpha \vdash e : \sigma \quad \text{fresh } \alpha}{\Gamma \vdash e : \forall \alpha. \pi_\alpha \Rightarrow \sigma}
\end{array}$$

$$\begin{array}{c}
(\forall \text{E}) \quad \dfrac{\Gamma \vdash e : \forall \alpha. \pi_\alpha \Rightarrow \sigma \quad \Gamma \vdash [\tau / \alpha] \pi_\alpha}{\Gamma \vdash e : [\tau / \alpha] \sigma}
\end{array}$$

$$\Gamma = \{\text{eq} : \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Bool}, \\ \text{eq} : \forall \alpha. (\text{eq} : \alpha \rightarrow \alpha \rightarrow \text{Bool}) \Rightarrow [\alpha] \rightarrow [\alpha] \rightarrow \text{Bool}\}$$

$$\frac{\text{eq} : \forall \alpha. (\text{eq} : \alpha \rightarrow \alpha \rightarrow \text{Bool}) \Rightarrow [\alpha] \rightarrow [\alpha] \rightarrow \text{Bool} \in \Gamma}{\Gamma \vdash \text{eq} : \forall \alpha. (\text{eq} : \alpha \rightarrow \alpha \rightarrow \text{Bool}) \Rightarrow [\alpha] \rightarrow [\alpha] \rightarrow \text{Bool}} \quad \frac{\text{Nat} \rightarrow \text{Nat} \rightarrow \text{Bool} \in \Gamma}{\Gamma \vdash \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Bool}} \quad \dots$$

$$\Gamma \vdash \text{eq} : [\text{Nat}] \rightarrow [\text{Nat}] \rightarrow \text{Bool} \quad \dots$$

$$\Gamma \vdash \text{eq} [0] : [\text{Nat}] \rightarrow \text{Bool} \quad \dots$$

$$\Gamma \vdash \text{eq} [0] [0]$$

System 0 — Constraint Solving

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

```

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

```

System 0

```

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

```

Hindley Milner

System 0 — 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

Repository

github.com/Mari-W/pop1

References

- [A Second Look at Overloading](#) 1995
Martin Odersky, Philip Wadler, Martin Wehr
- [A Theory of Type Polymorphism in Programming](#) 1978
Hindley Milner