A Second Look at Overloading

Marius Weidner – Chair of Programming Languages – Seminar '22

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

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


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

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

Pseudocode

$$
\begin{aligned}
e :=\ & x \\
|\ & \lambda x.\ e \\
|\ & e\ e \\
|\ & \textbf{let}\ x = e\ \textbf{in}\ e
\end{aligned}
\qquad\qquad
\begin{aligned}
\tau :=\ & \alpha \\
|\ & \tau \to \tau \\
\sigma :=\ & \tau \\
|\ & \forall \alpha.\ \sigma
\end{aligned}
$$

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

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

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

$$
\begin{aligned}
p :=\ & e \\
\mid\ & \textbf{inst}\ o : \sigma_T = \sigma\ \textbf{in}\ p
\end{aligned}
$$

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

$$
\begin{aligned}
\tau &:= \alpha \\
&\mid \tau \to \tau \\
&\mid D\ \tau_1\ \ldots\ \tau_n \qquad (D \in \{\mathrm{Unit},\ \mathrm{Nat},\ \mathrm{List}\ \tau,\ ..\},\ \mathrm{arity}(D) = n) \\
\pi_\alpha &:= o_1 : \alpha \to \tau_1,\ \ldots\ , o_n : \alpha \to \tau_n \qquad (n \in \mathbb{N},\ o_i \neq o_j) \\
\sigma &:= \tau \\
&\mid \forall \alpha . \pi_\alpha \Rightarrow \sigma_T \\
\sigma_T &:= T\ \alpha_1\ \ldots\ \alpha_n \to \tau \qquad (T \in D \cup \{\to\},\ \mathrm{tv}(\tau) \subseteq \{\alpha_1, .., \alpha_n\}) \\
&\mid \forall \alpha . \pi_\alpha \Rightarrow \sigma_T \qquad (\mathrm{tv}(\pi_\alpha) \subseteq \mathrm{tv}(\sigma_T))
\end{aligned}
$$

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

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

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

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

$$
\begin{aligned}
\Gamma = \{&\mathrm{eq} : \mathrm{Nat} \to \mathrm{Nat} \to \mathrm{Bool}, \\
&\mathrm{eq} : \forall \alpha.\ (\mathrm{eq} : \alpha \to \alpha \to \mathrm{Bool}) \Rightarrow [\alpha] \to [\alpha] \to \mathrm{Bool}\}
\end{aligned}
$$

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{\begin{array}{c}\mathrm{eq} : \forall \alpha.\ (\mathrm{eq} : \alpha \to \alpha \to \mathrm{Bool}) \\ \Rightarrow [\alpha] \to [\alpha] \to \mathrm{Bool} \in \Gamma\end{array}}
            {\begin{array}{c}\Gamma \vdash \mathrm{eq} : \forall \alpha.\ (\mathrm{eq} : \alpha \to \alpha \to \mathrm{Bool}) \\ \Rightarrow [\alpha] \to [\alpha] \to \mathrm{Bool}\end{array}}
      \quad
      \dfrac{\mathrm{Nat} \to \mathrm{Nat} \to \mathrm{Bool} \in \Gamma}
            {\Gamma \vdash \mathrm{Nat} \to \mathrm{Nat} \to \mathrm{Bool}}
      \quad \ldots
    }{\Gamma \vdash \mathrm{eq} : [\mathrm{Nat}] \to [\mathrm{Nat}] \to \mathrm{Bool}}
    \quad \ldots
  }{\Gamma \vdash \mathrm{eq}\ [0] : [\mathrm{Nat}] \to \mathrm{Bool}}
  \quad \ldots
}{\Gamma \vdash \mathrm{eq}\ [0]\ [0]}
$$

$$
\begin{aligned}
&[\![\texttt{inst}\ eq : \mathbb{N} \to \mathbb{N} \to \mathbb{B} = e_1\ \texttt{in} \\
&\quad \texttt{inst}\ eq : \forall\alpha.\ (\mathrm{eq} : \alpha \to \alpha \to \mathbb{B}) \Rightarrow [\alpha] \to [\alpha] \to \mathbb{B} = e_2\ \texttt{in} \\
&\quad eq\ [0]\ [0]]\!]_{\emptyset} \\
= {} &[\![\texttt{inst}\ eq : \forall\alpha.\ (\mathrm{eq} : \alpha \to \alpha \to \mathbb{B}) \Rightarrow [\alpha] \to [\alpha] \to \mathbb{B} = ..\ \texttt{in} \\
&\quad eq\ [0]\ [0]]\!]_{\{eq := \lambda x.\ \texttt{if}\ x\ \texttt{is}\ \mathbb{N}\ \texttt{then}\ [\![e_1]\!]\ x\}} \\
= {} &[\![eq\ [0]\ [0]]\!]_{\{eq := \lambda x.\ \texttt{if}\ x\ \texttt{is}\ \texttt{List}\ \texttt{then}\ [\![e_2]\!]\ x\ \texttt{else}\ \lambda x.\ \texttt{if}\ x\ \texttt{is}\ \mathbb{N}\ \texttt{then}\ [\![e_1]\!]\ x\}} \\
= {} &(\lambda x.\ \texttt{if}\ x\ \texttt{is}\ [\alpha]\ \texttt{then}\ [\![e_2]\!]\ x\ \texttt{else}\ \lambda x.\ \texttt{if}\ x\ \texttt{is}\ \mathbb{N}\ \texttt{then}\ [\![e_1]\!]\ x)\ [0]\ [0] \\
= {} &[\![e_2]\!]\ [0]\ [0]
\end{aligned}
$$

System O — 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 O

System O — Relationship with Record Typing

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