

A Second Look at Overloading

```

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

```

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

isEq :: Bool
isEq = [Zero] == [Zero]

```

Pseudocode

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

```
let id = λx. x : ∀a. a → a in foo (id "Answer: ") (id 42)
```

Hindley Milner — Syntax

(TAUT)	$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$	$\frac{\Gamma \vdash e' : \sigma \quad \Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \mathbf{let} \ x = e' \ \mathbf{in} \ e : \tau}$	(LET)
(\rightarrow I)	$\frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash \lambda x. e : \tau \rightarrow \tau'}$	$\frac{\Gamma \vdash e : \tau \rightarrow \tau' \quad \Gamma \vdash e' : \tau}{\Gamma \vdash e \ e' : \tau'}$	(\rightarrow E)
(\forall I)	$\frac{\Gamma \vdash e : \sigma \quad \text{fresh } \alpha}{\Gamma \vdash e : \forall \alpha. \sigma}$	$\frac{\Gamma \vdash e : \sigma \quad \sigma \sqsubseteq \sigma'}{\Gamma \vdash e : \sigma'}$	(\forall E)

Hindley Milner — Typing

Let $\Gamma = \{\text{id} : \forall \alpha. \alpha \rightarrow \alpha, \text{n} : \text{Nat}\}.$

$$\frac{\frac{\text{id} : \forall \alpha. \alpha \rightarrow \alpha \in \Gamma \quad \forall \alpha. \alpha \rightarrow \alpha \sqsubseteq \text{Nat} \rightarrow \text{Nat}}{\Gamma \vdash \text{id} : \text{Nat} \rightarrow \text{Nat}} \quad \frac{\text{n} : \text{Nat} \in \Gamma}{\Gamma \vdash \text{n} : \text{Nat}}}{\Gamma \vdash \text{id n}}$$

Hindley Milner — Instantiation

$$\frac{
\frac{
\frac{x : \alpha \in \{x : \alpha\}}{x : \alpha \vdash x : \alpha}
}{\vdash \lambda x. x : \alpha \rightarrow \alpha}
\quad
\frac{
\text{id} : \forall \alpha. \alpha \rightarrow \alpha \in \{\text{id} : \forall \alpha. \alpha \rightarrow \alpha\}}
{\text{id} : \forall \alpha. \alpha \rightarrow \alpha \vdash \text{id} : \forall \alpha. \alpha \rightarrow \alpha}
}{
\vdash \lambda x. x : \forall \alpha. \alpha \rightarrow \alpha \quad \text{fresh } \alpha \quad \vdash \mathbf{let} \text{ id} = \lambda x. x \mathbf{in} \text{id} : \forall \alpha. \alpha \rightarrow \alpha
}$$

Hindley Milner — Generalization

$e ::= x$ (o if overloaded)

$p ::= e$

| $e\ e$

| **inst** $o : \sigma_T = \sigma$ **in** p

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

| $\lambda x. e$

| **let** $x = e$ **in** e

```
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
  | λ_. λ_. False
in eq [4, 2] [4, 2]
```


$$\tau := \alpha$$

$$| \tau \rightarrow \tau$$

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

$$\sigma := \tau$$

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

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

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

$$| \forall \alpha. \pi_\alpha \Rightarrow \sigma_T \quad (\text{tv}(\pi_\alpha) \subseteq \text{tv}(\sigma_T))$$

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

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

$$(\text{SET}) \quad \frac{\Gamma \vdash x_1 : \sigma_1 \quad \dots \quad \Gamma \vdash x_n : \sigma_n}{\Gamma \vdash x_1 : \sigma_1 \quad \dots \quad x_n : \sigma_n}$$

$$(\text{INST}) \quad \frac{\Gamma \vdash e : \sigma_T \quad \Gamma, o : \sigma_T \vdash p : \sigma \quad \forall (o : \sigma'_T) \in \Gamma \Rightarrow \sigma'_T \neq \sigma_T}{\Gamma \vdash \mathbf{inst} \ o : \sigma_T = e \ \mathbf{in} \ p : \sigma}$$

System 0 — Derivation

System 0 — Semantics

System 0 — Translation to Hindley Milner

Folien & Code

github.com/Mari-W/pop1

Literatur

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