

Formal Proof of Type Preservation of the Dictionary Passing Transform for System F

Marius Weidner

Chair of Programming Languages, University of Freiburg

March 7, 2023

Typeclasses

Overloading Equality in Haskell

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

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

.. eq 42 0 .. eq [42, 0] [42, 0] ..
```

Desugaring Typeclasses

Overloading Equality in System F_0

```
decl eq in
```

```
inst eq : Nat → Nat → Bool
```

```
  = λx. λy. .. in
```

```
inst eq : ∀α. [eq : α → α → Bool] ⇒ [α] → [α] → Bool
```

```
  = Λα. λ(eq : α → α → Bool). λxs. λys. .. in
```

```
.. eq 42 0 .. eq Nat [42, 0] [42, 0] ..
```

Dictionary Passing Transform

Overloading Equality in System F_0

```
decl eq in
inst eq : Nat → Nat → Bool
  = λx. λy. .. in
inst eq : ∀α. [eq : α → α → Bool] ⇒ [α] → [α] → Bool
  = Λα. λ(eq : α → α → Bool). λxs. λys. .. in
.. eq 42 0 .. eq Nat [42, 0] [42, 0] ..
```

System F_0 Transformed to System F

```
let eq1 : Nat → Nat → Bool
  = λx. λy. .. in
let eq2 : ∀α. (α → α → Bool) → [α] → [α] → Bool
  = Λα. λeq1. λxs. λys. .. in

.. eq1 42 0 .. eq2 Nat eq1 [42, 0] [42, 0] ..
```

Elegant Syntax Representations in Agda

System F_0 Syntax

```
data Term : Sorts → Sort r → Set where
  ' _                : s ∈ S → Term S s
  decl 'o' in _       : Term (S ▷ oS) eS → Term S eS
  inst ' _ ' = _ ' in _ : Term S oS → Term S eS → Term S eS → Term S eS
  _ : _              : Term S oS → Term S τS → Term S cS
  λ _ ⇒ _            : Term S cS → Term S eS → Term S eS
  [ _ ] ⇒ _          : Term S cS → Term S τS → Term S τS
```

Substitution & Renaming

Renaming

$\text{Ren} : \text{Sorts} \rightarrow \text{Sorts} \rightarrow \text{Set}$

$\text{Ren } S_1 S_2 = \forall \{s\} \rightarrow \text{Var } S_1 s \rightarrow \text{Var } S_2 s$

$\text{wk} : \text{Term } S s \rightarrow \text{Term } (S \triangleright s') s$

$\text{wk} = \text{ren there}$

Substitution

$\text{Sub} : \text{Sorts} \rightarrow \text{Sorts} \rightarrow \text{Set}$

$\text{Sub } S_1 S_2 = \forall \{s\} \rightarrow \text{Var } S_1 s \rightarrow \text{Term } S_2 s$

$_[_] : \text{Term } (S \triangleright s') s \rightarrow \text{Term } S s' \rightarrow \text{Term } S s$

$t [_ t'] = \text{sub } (\text{single}_s \text{id}_s t') t$

Overloading Formalized

Context

```
data Ctx : Sorts → Set where
  ∅ : Ctx []
  _ ► _ : Ctx S → Term S (item-of s) → Ctx (S ► s)
  _ ► _ : Ctx S → Cstr S → Ctx S
```

Constraint Solving

```
data [_] ∈ _ : Cstr S → Ctx S → Set where
  here : [ (' o : τ) ] ∈ (Γ ► (' o : τ))
  under-bind : {l : Term S (item-of s')} → [ (' o : τ) ] ∈ Γ → [ (' there o : wk τ) ] ∈ (Γ ► l)
  under-inst : [ c ] ∈ Γ → [ c ] ∈ (Γ ► c')
```

Extrinsic Typing Rules

System F_0 Typing

`data` $_ \vdash _ : _ : \text{Ctx } S \rightarrow \text{Term } S \ s \rightarrow \text{Term } S \ (\text{kind-of } s) \rightarrow \text{Set}$ `where`

$\vdash_{\text{inst}} :$

$\Gamma \vdash e_2 : \tau \rightarrow$

$\Gamma \blacktriangleright ('o : \tau) \vdash e_1 : \tau' \rightarrow$

$\Gamma \vdash \text{inst}' 'o' = e_2 \text{ 'in } e_1 : \tau'$

$\vdash_{\text{'o}} :$

$['o : \tau] \in \Gamma \rightarrow$

$\Gamma \vdash 'o : \tau$

$\vdash_{\lambda} :$

$\Gamma \blacktriangleright c \vdash e : \tau \rightarrow$

$\Gamma \vdash \lambda c \Rightarrow e : [c] \Rightarrow \tau$

$\vdash_{\emptyset} :$

$\Gamma \vdash e : ['o : \tau] \Rightarrow \tau' \rightarrow$

$['o : \tau] \in \Gamma \rightarrow$

$\Gamma \vdash e : \tau'$

Fun Lemmas on Our Way to Type Preservation

Type Transform Preserves Renaming

$$\text{F.ren } (\vdash \rho \rightsquigarrow \rho \vdash \rho) (\tau \rightsquigarrow \tau \tau) \equiv \tau \rightsquigarrow \tau (\text{F}^O.\text{ren } \rho \tau)$$

Type Transform Preserves Substitution

$$\text{F.sub } (\vdash \sigma \rightsquigarrow \sigma \vdash \sigma) (\tau \rightsquigarrow \tau \tau) \equiv \tau \rightsquigarrow \tau (\text{F}^O.\text{sub } \sigma \tau)$$

Instance Resolution Transforms to Unique Variable

$$\begin{aligned} o:\tau \in \Gamma \rightsquigarrow \Gamma x \equiv \tau : \forall \{ \Gamma : \text{F}^O.\text{Ctx } \text{F}^O.S \} \rightarrow (o:\tau \in \Gamma : [\text{F}^O.o : \text{F}^O.\tau] \in \Gamma) \rightarrow \\ \text{F.lookup } (\Gamma \rightsquigarrow \Gamma \Gamma) (o:\tau \in \Gamma \rightsquigarrow x o:\tau \in \Gamma) \equiv (\tau \rightsquigarrow \tau \text{F}^O.\tau) \end{aligned}$$

Type Preservation of the Dictionary Passing Transform

Typed System F^O transforms to typed System F

$$\begin{aligned} \vdash t \rightsquigarrow \vdash t &: \{\Gamma : F^O.\text{Ctx } F^O.S\} \{t : F^O.\text{Term } F^O.S \ F^O.s\} \{T : F^O.\text{Term } F^O.S \ (F^O.\text{kind-of } F^O.s)\} \rightarrow \\ &(\vdash t : \Gamma \ F^O.\vdash t : T) \rightarrow \\ &(\Gamma \rightsquigarrow \Gamma' \ \Gamma) \ F.\vdash (\vdash t \rightsquigarrow \vdash t) : (T \rightsquigarrow T' \ T) \\ \vdash t \rightsquigarrow \vdash t \ (\vdash 'o \ o : \tau \in \Gamma) &= \vdash 'x \ (o : \tau \in \Gamma \rightsquigarrow \Gamma' x \equiv \tau \ o : \tau \in \Gamma) \\ \vdash t \rightsquigarrow \vdash t \ (\vdash \lambda \{c = ('o : \tau)\} \vdash e) &= \vdash \lambda \ (\text{subst} \ (_ \ F.\vdash \vdash t \rightsquigarrow \vdash t \vdash e : _) \\ &\quad \tau \rightsquigarrow \text{wk-inst}.\tau \equiv \text{wk-inst}.\tau \rightsquigarrow \tau \ (\vdash t \rightsquigarrow \vdash t \vdash e)) \\ \vdash t \rightsquigarrow \vdash t \ (\vdash \odot \vdash e \ o : \tau \in \Gamma) &= \vdash \cdot \ (\vdash t \rightsquigarrow \vdash t \vdash e) \ (\vdash 'x \ (o : \tau \in \Gamma \rightsquigarrow \Gamma' x \equiv \tau \ o : \tau \in \Gamma)) \\ - \dots \end{aligned}$$

Further Work: Hindley Milner & Semantic Preservation

Overloading in Hindley Milner

- Constraint abstractions cannot require poly types
 - ① Introduce sorts m_s and p_s in favour of single sort τ_s
- All instances must differ in the type of their first argument for each overloaded variable
 - ① Preserves Algorithm W

Proving Semantic Preservation

- Overloaded languages require typed semantics
- Prove that, if $\vdash e \hookrightarrow \vdash e'$ then
$$\exists [e''] (\vdash e \hookrightarrow e' \rightsquigarrow e \hookrightarrow e' \vdash e \hookrightarrow^* e'') \times (\vdash e \hookrightarrow e' \rightsquigarrow e \hookrightarrow e' \vdash e' \hookrightarrow^* e'')$$