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

Marius Weidner

Chair of Programming Languages, University of Freiburg

February 10, 2023

Type Classes in Haskell

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 Type Classes

Overloading Equality in System F_O

```
decl eq in

inst eq : Nat → Nat → Bool
  = λx. λy. .. in
inst eq : ∀α. [eq : α → α → Bool] ⇒ [α] → [α] → Bool
  = Λα. ƛ(eq : α → α → Bool). λxs. λys. .. in

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

Dictionary Passing Transform

Overloading Equality in System F_O

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

System F_O 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] ..
```

Agda Formalization of System F_O

Syntax Representation in Agda

```
data Term : Sorts → Sort r → Set where
  `_             : Var S s → Term S s
  tt             : Term S eₛ
  λ`x→_          : Term (S ▷ eₛ) eₛ → Term S eₛ
  Λ`α→_          : Term (S ▷ τₛ) eₛ → Term S eₛ
  ƛ_⇒_           : Term S cₛ → Term S eₛ → Term S eₛ
  _·_            : Term S eₛ → Term S eₛ → Term S eₛ
  _•_            : Term S eₛ → Term S τₛ → Term S eₛ
  let`x=_`in_    : Term S eₛ → Term (S ▷ eₛ) eₛ → Term S eₛ
  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
  `⊤             : Term S τₛ
  _⇒_            : Term S τₛ → Term S τₛ → Term S τₛ
  ∀`α_           : Term (S ▷ τₛ) τₛ → Term S τₛ
  [_]⇒_          : Term S cₛ → Term S τₛ → Term S τₛ
```

Agda Formalization of System F_O

Context

```
data Ctx : Sorts → Set where
  ∅ : Ctx []
  _▶_ : Ctx S → Stores S 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 : {ST : Stores S s’} →
    [ (‘ o : τ) ]∈ Γ → [ (‘ there o : wk τ) ]∈ (Γ ▶ ST)
  under-inst : [ c ]∈ Γ → [ c ]∈ (Γ ▸ c’)
```

The Dectionary Passing Transform

Fun Lemmas on Our Way to Type Preservation

Type Preservation of the Dictionary Passing Transform