Universal Polymorphism and System F

Consider the identity function: How do we write it?

\( \lambda x : \text{Bool}. \ x \)

\( \lambda y : \text{Nat}. \ y \)

\( \lambda z : \text{Bool} \rightarrow \text{Nat}. \ z \)

It is tedious to write this... every time we wish to use an identity.

Thus we use Universal Polymorphism/ Parametric Polymorphism!

\( \forall X. \lambda (z : X \rightarrow X). \ z : \forall X. X \rightarrow X \)

Expression

Type

\((\forall X. \lambda (z : X). \ z)[\text{int}] \rightarrow \)
\( X_2 : \text{int}, z \)

From a high level, just like we normally have "lambda abstractions" we now have "type abstractions"

Just like we normally have "function applications" we now have "type applications"

The STLC with type polymorphism is called System F

Why F? No reason!

Independently discovered by Girard and Reynolds

With polymorphism, we can encode sum types and product types, so in this language we will only formalize functions (we will also remove basic types like Bool and Nat)

System F:

\[ e ::= x \]
\[ f \]
\[ \Gamma \vdash e_1 : \tau_1, e_2 : \tau_2 \]
\[ \Gamma \vdash \lambda x. e : \tau \]
\[ \Gamma \vdash e_2 [x] : \tau \]

\[ \tau \vdash x : \tau \]
\[ \tau \vdash \lambda x. e : \tau \]
\[ \tau \vdash \forall x. \tau \]

To finish our language, we need to define typing, values, and semantics.

**Typing:** \[ \Gamma : \text{Var} \to \text{Type + Unit} \]

\[ \frac{\Gamma (\alpha) = \tau}{\Gamma \vdash \alpha : \tau} \]

\[ \frac{\Gamma \vdash x_1 : \tau_1, \Gamma \vdash x_2 : \tau_2}{\Gamma \vdash x_1 \cdot e : \tau_2} \]

\[ \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2, \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash \lambda x. e : \tau_1, e : \tau_2} \]

\[ \frac{\Gamma \vdash e_1 : \tau_1, \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 \cdot e_2 : \tau_2} \]
\[ \Gamma, X : \tau \vdash e : \tau \]
\[ \Gamma \vdash \forall X : \tau, \Gamma \vdash e \vdash \forall X : \tau \]

\[ \tau, [\tau / x] : \tau' \]
\[ \tau, \tau' \vdash [\tau / x] : \tau' \]
\[ \tau, \tau' \vdash \tau' \]
\[ \tau, \tau' \vdash \tau' \]

\[ X [\tau / x] : \tau \]
\[ \gamma \neq X \]
\[ \gamma \vdash X \]
\[ \gamma [\tau / x] = \gamma \]

\[ \forall X : \tau \vdash \forall X : \tau \]
\[ X \forall Y : \tau \vdash \forall Y : \tau \]
\[ (\forall Y : \tau) [\tau / x] \]

\[ \lambda (x : \tau), e \text{ value} \]
\[ \forall X : e \text{ value} \]

\[ e_1 \rightarrow e_1' \]
\[ e_1, e_2 \rightarrow e_1', e_2' \]

\[ v_1 \text{ value} \]
\[ v_1, e_1 \rightarrow v_1, e_1' \]
\[ \text{value} \]

\[ (\lambda(x:e).e) \; v_e \to e[v_e/x] \]

\[ e \to e' \]

\[ e[v_e] \to e'[v_e] \]

\[ (\lambda x.e)[v_e] \to e[v_e/x] \]

\[ e_i[v_{\gamma x}] \to e'_i \quad e_i[v_{\gamma x}] = e'_i \]

\[ (e_i \; e) \; v_{\gamma x} \to e'_i \; e'_n \]

\[ x \in \gamma \]

\[ x + y \]

\[ y \in \gamma \]

\[ (\lambda x.e)[v_{\gamma x}] = \lambda x.e' \]

\[ (\lambda(x:e).e)[v_{\gamma x}] \rightarrow \lambda(x:e).e'[v_{\gamma x}] \]

\[ (x(y:z).e)[v_{\gamma x}] = \lambda(x:y:z).e'[v_{\gamma x}] \]

\[ y + x \]

That was very unpleasant! But notice that if you ignore variable replacement, it's actually quite easy.

Progress: maintained!
Preservation: maintained!
Normalization: maintained!
Encoding Sums:
\[ x_1 + x_2 := \forall X. (x_1 \to X) \to (x_2 \to X) \to X \]

\[ \text{inl } e : \tau_1 \to \tau_2 := \lambda X. \lambda (f : \tau_1 \to X), \lambda (g : \tau_2 \to X). \text{f} e \]

\[ \text{inr } e : \tau_1 \to \tau_2 := \lambda X. \lambda (f : \tau_1 \to X), \lambda (g : \tau_2 \to X). \text{g} e \]

\[
\text{match } e_0 : \tau_1 \to \tau_2 \text{ with } \\
\{ \text{inl } x \Rightarrow e_1 : \tau_3 \\
\{ \text{inr } y \Rightarrow e_2 : \tau_3 \}
\}
\]

\[ e_0 \left[ x_3 \right] \left( \lambda (x : \tau_1). e_1 \right) \left( \lambda (y : \tau_2). e_2 \right) \]

**Type Inference Problem.**

Define \( \text{erase } (e) \) as

\[
\begin{align*}
\text{erase } (x) &= x \\
\text{erase } e_1 e_2' &= \text{erase } e_1 = e_1' \\
\text{erase } (\lambda x. e) &= \lambda x. e'
\end{align*}
\]

Q1. Given \( e' \), does there exist \( e \) such that \( \text{erase } (e) = e' \) and \( \text{type } e \)?

(Indiscernability of type inference: there does
not exist an algorithm that can always correctly answer Q1

Existential Types:

\( \exists x : X . \exists y : Y . e \)

\( e : \ldots \quad \exists x_1, e_3 \text{ as } e_2 \)
\( \quad \text{let } \exists x, x^2 = e_1 \text{ in } e_2 \)

\( \forall \text{ value} \)

\( \exists x_1, v_3 \text{ as } e_2 \text{ value} \)

\( e_1 \rightarrow e_1' \)

\( \exists x_1, e_1, v_3 \text{ as } e \rightarrow \exists x_1, e_1, v_3 \text{ as } e \)

let \( \exists x, x^3 = \exists x_1, v_3 \text{ as } e_2 \) in \( e \)

\( e_2 [x_1 \mapsto ] [v_3 \mapsto ] \)
\[ e_1 \rightarrow e_1 \]

Let \( \{X, x\} = e_1 \) in \( e_2 \) \( \Rightarrow \) let \( \{X, x\} = e_1 \) in \( e_2 \)

\[ \Gamma \vdash e : \tau_2 \left[ \tau_1 / x \right] \]

\[ \Gamma \vdash e_1, e_3 \text{ as } \exists x. \tau_1 : \exists x. \tau_2 \]

\[ \Gamma \vdash e_1 : \exists \exists x. \tau_1, \exists \tau_2 \]

\[ \Gamma, X : x, x : \tau_1 + e_1 : \tau_2 \]

\[ \Gamma \vdash \text{let } \exists x. x^3 = e_1 \text{ in } e_2 : \tau_2 \]

One can encode existentials as:

\[ \exists x, \tau_2 : \exists \exists x, \tau_2 \Rightarrow A \forall x. (A x : \tau_2) \Rightarrow A \]