Recall the definition of System F.

Syntax.

\[
e ::= n \mid x \mid \lambda x : \tau. e \mid e_1 e_2 \mid \Lambda X. e \mid e \tau
\]

\[
v ::= n \mid \lambda x : \tau. e \mid \Lambda X. e
\]

\[
E ::= \[] \mid E \mid v E \mid E \tau
\]

Semantics.

\[
e \Rightarrow e'
\]

\[
E[e] \Rightarrow E[e']
\]

\[
(\lambda x : \tau. e) v \Rightarrow e[v/x]
\]

\[
(\Lambda X. e) \tau \Rightarrow e[\tau/X]
\]

Type System.

\[
\Delta, \Gamma \vdash n : \text{int}
\]

\[
\Delta, \Gamma \vdash x : \tau
\]

\[
\Gamma(x) = \tau
\]

\[
\Delta, \Gamma, x : \tau \vdash e : \tau' \quad \Delta \vdash \tau \text{ ok}
\]

\[
\Delta, \Gamma \vdash \lambda x : \tau. e : \tau \Rightarrow \tau'
\]

\[
\Delta, \Gamma \vdash e_1 : \tau \Rightarrow \tau' \quad \Delta, \Gamma \vdash e_2 : \tau
\]

\[
\Delta, \Gamma \vdash e_1 e_2 : \tau'
\]

\[
\Delta \vdash \tau \text{ ok}
\]

\[
\Delta \vdash \forall X. \tau' \quad \Delta \vdash \tau \text{ ok}
\]

\[
\Delta \vdash [\tau] : \tau' \{\tau/X\}
\]

Type Well Formedness.

\[
\Delta \vdash X \text{ ok}
\]

\[
X \in \Delta
\]

\[
\Delta \vdash \tau \text{ ok}
\]

\[
\Delta \vdash \tau \text{ ok}
\]

\[
\Delta \vdash \tau \text{ ok}
\]

\[
\Delta \vdash \forall X. \tau \text{ ok}
\]

Sums and Products

We can encode sums and products in System F without adding additional types! The encodings are based on the Church encodings from untyped λ-calculus.
Erasure

The semantics of System F presented above explicitly passes type. In an implementation, one often wants to eliminate types for efficiency. The following translation “erases” the types from a System F expression.

\[
\begin{align*}
erase(x) & = x \\
erase(\lambda x : \tau. e) & = \lambda x. \text{erase}(e) \\
erase(e_1 e_2) & = \text{erase}(e_1) \text{erase}(e_2) \\
erase(\Lambda X. e) & = \lambda z. \text{erase}(e) \quad \text{where } z \text{ is fresh for } e \\
erase(e [\tau]) & = \text{erase}(e) (\lambda x. x)
\end{align*}
\]

The following theorem states that the translation is adequate.

**Theorem (Adequacy).** For all expressions \(e\) and \(e'\), we have \(e \rightarrow e'\) iff \(\text{erase}(e) \rightarrow \text{erase}(e')\).

The type reconstruction problem asks whether, for a given untyped \(\lambda\)-calculus expression \(e'\) there exists a well-typed System F expression \(e\) such that \(\text{erase}(e) = e'\). It was shown to be undecidable by Wells in 1994, by showing that type checking is undecidable for a variant of untyped \(\lambda\)-calculus without annotations. See Pierce Chapter 23 for further discussion, and restrictions of System F for which type reconstruction is decidable.