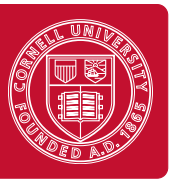
CS 4110 – Programming Languages and Logics
Lecture #23: Programming in System F



Recall the definition of System F.

Syntax.

$$
\begin{aligned}
e &::= n \mid x \mid \lambda x{:}\tau.\, e \mid e_1\ e_2 \mid \Lambda X.\, e \mid e\ [\tau] \\
v &::= n \mid \lambda x{:}\tau.\, e \mid \Lambda X.\, e \\
E &::= [\cdot] \mid E\ e \mid v\ E \mid E\ [\tau]
\end{aligned}
$$

Semantics.

$$\frac{e \to e'}{E[e] \to E[e']} \qquad \frac{}{(\lambda x{:}\tau.\, e)\ v \to e\{v/x\}} \qquad \frac{}{(\Lambda X.\, e)\ [\tau] \to e\{\tau/X\}}$$

Type System.

$$\frac{}{\Delta, \Gamma \vdash n : \textsf{int}} \qquad \frac{}{\Delta, \Gamma \vdash x : \tau}\ \Gamma(x) = \tau \qquad \frac{\Delta, \Gamma, x{:}\tau \vdash e{:}\tau' \quad \Delta \vdash \tau\ \textsf{ok}}{\Delta, \Gamma \vdash \lambda x{:}\tau.\, e : \tau \to \tau'}$$

$$\frac{\Delta, \Gamma \vdash e_1 : \tau \to \tau' \quad \Delta, \Gamma \vdash e_2 : \tau}{\Delta, \Gamma \vdash e_1\ e_2 : \tau'} \qquad \frac{\Delta \cup \{X\}, \Gamma \vdash e : \tau}{\Delta, \Gamma \vdash \Lambda X.\, e : \forall X.\, \tau} \qquad \frac{\Delta, \Gamma \vdash e : \forall X.\, \tau' \quad \Delta \vdash \tau\ \textsf{ok}}{\Delta, \Gamma \vdash e\ [\tau] : \tau'\{\tau/X\}}$$

Type Well Formedness.

$$\frac{}{\Delta \vdash X\ \textsf{ok}}\ X \in \Delta \qquad \frac{}{\Delta \vdash \textsf{int ok}} \qquad \frac{\Delta \vdash \tau_1\ \textsf{ok} \quad \Delta \vdash \tau_2\ \textsf{ok}}{\Delta \vdash \tau_1 \to \tau_2\ \textsf{ok}} \qquad \frac{\Delta \cup \{X\} \vdash \tau\ \textsf{ok}}{\Delta \vdash \forall X.\, \tau\ \textsf{ok}}$$

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.

$$
\begin{array}{rl}
\tau_1 \times \tau_2 & \triangleq \forall R.\, (\tau_1 \to \tau_2 \to R) \to R \\
(\cdot, \cdot) & \triangleq \Lambda T_1.\, \Lambda T_2.\, \lambda v_1 : T_1, \lambda v_2 : T_2.\, \Lambda R.\, \lambda p : (T_1 \to T_2 \to R).\, p\ v_1\ v\ 2 \\
\pi_1 & \triangleq \Lambda T_1.\, \Lambda T_2.\, \lambda v : T_1 \times T_2.\, v\ [T_1]\ (\lambda x : T_1.\, \lambda y : T_2.\, x) \\
\pi_2 & \triangleq \Lambda T_1.\, \Lambda T_2.\, \lambda v : T_1 \times T_2.\, v\ [T_2]\ (\lambda x : T_1.\, \lambda y : T_2.\, y)
\end{array}
$$

$$
\begin{array}{rl}
\mathsf{unit} & \triangleq \forall R.\, R \to R \\
() & \triangleq \Lambda R.\, \lambda x : R.\, x
\end{array}
$$

$$
\begin{array}{rl}
\tau_1 + \tau_2 & \triangleq \forall R.(\tau_1 \to R) \to (\tau_2 \to R) \to R \\
\mathsf{inl} & \triangleq \Lambda T_1.\, \Lambda T_2.\, \lambda v_1 : T_1.\, \Lambda R.\, \lambda b_1 : T_1 \to R.\, \lambda b_2 : T_2 \to R.\, b_1\ v_1 \\
\mathsf{inr} & \triangleq \Lambda T_1.\, \Lambda T_2.\, \lambda v_2 : T_2.\, \Lambda R.\, \lambda b_1 : T_1 \to R.\, \lambda b_2 : T_2 \to R.\, b_2\ v_2 \\
\mathsf{case} & \triangleq \Lambda T_1.\, \Lambda T_2.\, .\, \Lambda R.\, \lambda v : T_1 + T_2.\, \lambda b_1 : T_1 \to R.\, \lambda b_2 : T_2 \to R.\, v\ [R]\ b_1\ b_2
\end{array}
$$

$$
\mathsf{void} \triangleq \forall R.\, R
$$

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.

$$
\begin{array}{rcll}
\mathit{erase}(x) & = & x & \\
\mathit{erase}(\lambda x\!:\!\tau.\, e) & = & \lambda x.\, \mathit{erase}(e) & \\
\mathit{erase}(e_1\ e_2) & = & \mathit{erase}(e_1)\ \mathit{erase}(e_2) & \\
\mathit{erase}(\Lambda X.\, e) & = & \lambda z.\, \mathit{erase}(e) & \text{where } z \text{ is fresh for } e \\
\mathit{erase}(e\ [\tau]) & = & \mathit{erase}(e)\ (\lambda x.\, x) &
\end{array}
$$

The following theorem states that the translation is adequate.

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

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