Chapter 23: Universal Types

System F
Power of System F
Properties (Soundness, decidability, parametericity, impredicativity)
System F

- First discovered by Jean-Yves Girard (1972)
- Independently developed by John Reynolds (1974) as polymorphic lambda-calculus
- A natural extension of $\lambda \to$ with a new form of abstract and application over types:

$$\begin{align*}
(\lambda X.t_{12}) [T_2] & \rightarrow [X \to T_2]t_{12} \\
(\lambda x:T_{11}.t_{12}) v_2 & \rightarrow [x \to v_2]t_{12}
\end{align*}$$

$$\frac{
\begin{array}{c}
\Gamma, X \vdash t_2 : T_2 \\
\end{array}
}{
\Gamma \vdash \lambda X.t_2 : \forall X.T_2
}$$

$$\frac{
\begin{array}{c}
\Gamma \vdash t_1 : \forall X.T_{12} \\
\end{array}
}{
\Gamma \vdash t_1 [T_2] : [X \to T_2]T_{12}
}$$
Syntax and Evaluation

Syntax
\[ \begin{align*}
t & ::= x \\
   & \quad \lambda x : T . t \\
   & \quad t \; t \\
   & \quad \lambda X . t \\
   & \quad t [T] \\
\end{align*} \]

values:
\[ \begin{align*}
\lambda x : T . t \\
\lambda X . t
\end{align*} \]

Evaluation

terms:
\[ \begin{align*}
t_1 \rightarrow t'_1 \\
t_1 \; t_2 \rightarrow t'_1 \; t_2 \\
t_2 \rightarrow t'_2 \\
v_1 \; t_2 \rightarrow v_1 \; t'_2 \\
(\lambda x : T_{11} . t_{12}) \; v_2 \rightarrow [x \rightarrow v_2] \; t_{12} \quad \text{(E-APPABS)}
\end{align*} \]

values:
\[ \begin{align*}
t_1 \rightarrow t'_1 \\
t_1 \; [T_2] \rightarrow t'_1 \; [T_2] \\
(\lambda X . t_{12}) \; [T_2] \rightarrow [X \rightarrow T_2] \; t_{12} \quad \text{(E-TAPPTAB)}
\end{align*} \]
Types and Type Context

\[ \begin{align*}
T & ::= \\
& \quad X \\
& \quad T \rightarrow T \\
& \quad \forall X. T \\
\Gamma & ::= \\
& \quad \emptyset \\
& \quad \Gamma, x : T \\
& \quad \Gamma, X \\
\end{align*} \]

types:
- type variable
- type of functions
- universal type

contexts:
- empty context
- term variable binding
- type variable binding
Typing

\[\Gamma \vdash t : T\]

(T-VAR)

\[\frac{x : T \in \Gamma}{\Gamma \vdash x : T}\]

(T-ABS)

\[\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1 \cdot t_2 : T_1 \to T_2}\]

(T-APP)

\[\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 \; t_2 : T_{12}}\]

(T-TABS)

\[\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \lambda X \cdot t_2 : \forall X \cdot T_2}\]

(T-TAPP)

\[\frac{\Gamma \vdash t_1 : \forall X \cdot T_{12}}{\Gamma \vdash t_1 \; [T_2] : [X \to T_2]T_{12}}\]
Ex.: Defining Polymorphic Functions

- **id** = $\lambda X. \lambda x:X. x$
  - $id : \forall X. X \rightarrow X$
  - $id \ [\text{Nat}] 0 \rightarrow 0$

- **double** = $\lambda X. \lambda f:X\rightarrow X. \lambda a:X. f (f \ a)$
  - $double : \forall X. (X\rightarrow X) \rightarrow X \rightarrow X$
  - $double \ [\text{Nat}] (\lambda x: \text{Nat. succ(succ(x)))} 3 \rightarrow 7$
  - $quadruple = \lambda X. double \ [X\rightarrow X] (double \ [X])$

- **selfApp** = $\lambda x: \forall X. X \rightarrow X. x \ [\forall X. X \rightarrow X] \ x$
  - $selfApp : (\forall X. X \rightarrow X) \rightarrow (\forall X. X \rightarrow X)$
Ex.: Polymorphic Lists

- nil : $\forall X. \text{List } X$
- cons : $\forall X. X \rightarrow \text{List } X \rightarrow \text{List } X$
- isnil : $\forall X. \text{List } X \rightarrow \text{Bool}$
- head : $\forall X. \text{List } X \rightarrow X$
- tail : $\forall X. \text{List } X \rightarrow \text{List } X$

- map : $\forall X. \forall Y. (X \rightarrow Y) \rightarrow \text{List } X \rightarrow \text{List } Y$

\[
\text{map} = \lambda X. \lambda Y. \lambda f: X \rightarrow Y.
\]
\[
(\text{fix } (\lambda m: (\text{List } X) \rightarrow (\text{List } Y). \lambda l: \text{List } X.
\]
\[
\text{if isnil } [X] l \text{ then nil } [Y]
\]
\[
\text{else cons } [Y] (f \text{ (head } [X] l)) (m \text{ (tail } [X] l)))
\]

Exercise: Can you write reverse?
Ex.: Church Encoding

- Church encodings can be carried out in System F.

- CBool = $\forall X. X \rightarrow X \rightarrow X$;
  - $\operatorname{tru} = \lambda X. \lambda t: X. \lambda f: X. t$;
  - $\operatorname{fls} = \lambda X. \lambda t: X. \lambda f: X. f$;
  - $\operatorname{and} = ?$

- CNat = $\forall X. (X \rightarrow X) \rightarrow X \rightarrow X$
  - $c0 = \lambda X. \lambda s: X \rightarrow X. \lambda z: X. z$
  - $c1 = \lambda X. \lambda s: X \rightarrow X. \lambda z: X. s \ z$
  - $\operatorname{csucc} = \lambda n: \text{CNat}. \lambda X. \lambda s: X \rightarrow X. \lambda z: X. s \ (n \ [X] \ s \ z)$
  - $\operatorname{cplus} = \lambda m: \text{CNat}. \lambda n: \text{CNat}. \lambda X. \lambda s: X \rightarrow X. \lambda z: X. m \ [X] \ s \ (n \ [X] \ s \ z)$
Ex.: Encoding Lists

• List X = ∀R. (X → R → R) → R → R
  - nil = λX. (λR. λc:X→R→R. λn:R. n) as List X
  - cons = λX. λhd:X. λtl:List X.
    (λR. λc:X→R→R. λn:R. c hd (tl [R] c n)) as List X;
  - isnil = λX. λl:List X.
    l [Bool] (λhd:X. λtl:Bool. false) true
  - head = λX. λl:List X.
    l [X] (λhd:X. λtl:X. hd) (diverge [X] unit)
  - sum : List Nat → Nat
    sum = ⋯ definition without using fix ⋯?
Basic Properties of System F

Very similar to those of the simply typed $\lambda$-calculus.

**Theorem [Preservation]:** If $\Gamma \vdash t : T$ and $t \rightarrow t'$, then $\Gamma \vdash t' : T$.

**Theorem [Progress]:** If $t$ is a closed, well-typed term, then either $t$ is a value or there is some $t'$ with $t \rightarrow t'$.

**Theorem [Normalization]:** Well-typed System F terms are normalizing.
Erasure and Type Construction

Theorem [Wells, 1994]: It is undecidable whether, given a closed term \( m \) of the untyped lambda-calculus, there is some well-typed term \( t \) in System F such that \( \text{erase}(t) = m \).
Partial Erasure and Type Construction

\[
\begin{align*}
erase_p(x) &= x \\
erase_p(\lambda x : T_1 . t_2) &= \lambda x : T_1 . erase_p(t_2) \\
erase_p(t_1 \ t_2) &= erase_p(t_1) \ erase_p(t_2) \\
erase_p(\lambda X . t_2) &= \lambda X . erase_p(t_2) \\
erase_p(t_1 [T_2]) &= erase_p(t_1) []
\end{align*}
\]

**Theorem** [Boehm 1985, 1989]: It is **undecidable** whether, given a closed term \( s \) in which type applications are marked but the arguments are omitted, there is some well-typed System F term \( t \) such that \( erase_p(t) = s \).

Type reconstruction is as hard as **higher-order unification**. (But many practical algorithms have been developed)
Theorem: If \( erase_v(t) = u \), then either (1) both \( t \) and \( u \) are normal forms according to their respective evaluation relations, or (2) \( t \rightarrow t' \) and \( u \rightarrow u' \), with \( erase_v(t') = u' \).
Fragments of System F

- **Rank-1 (prenex) polymorphism**
  - type variables may not be instantiated with polymorphic types

- **Rank-2 polymorphism**
  - A type is said to be of rank 2 if no path from its root to a $\forall$ quantifier passes to the left of 2 or more arrows.

\[
(\forall X. X \to X) \to \text{Nat} \quad \text{OK}
\]
\[
\text{Nat} \to (\forall X. X \to X) \to \text{Nat} \to \text{Nat} \quad \text{OK}
\]
\[
((\forall X. X \to X) \to \text{Nat}) \to \text{Nat} \quad X
\]

Type reconstruction for ranks 2 and lower is decidable, and that for rank 3 and higher of System F is undecidable.
Parametricity

- Uniform behavior of polymorphic programs

\[
\text{CBool} = \forall X. X \rightarrow X \rightarrow X;
\]
\[
\text{tru} = \lambda X. \lambda t: X. \lambda f: X. t;
\]
\[
\text{fls} = \lambda X. \lambda t: X. \lambda f: X. f;
\]

(1) Tru and fls are the only two basic inhabitants of CBool.

(2) Free Theorem:

e.g., for reverse: \( \forall X. \text{List} X \rightarrow \text{List} X \), we have

\[
\text{map f . reverse} = \text{reverse . map f}
\]
Impredicativity

Definition. A definition (of a set, a type, etc.) is called “impredicative” if it involves a quantifier whose domain includes the very thing being defined.

System F is impredicative, because the type variable $X$ in the type

$$T = \forall X. X \rightarrow X$$

ranges over all types, including $T$ itself.

Russell’s paradox: let $A = \{ x \mid x \text{ is not in } x \}$, then is “$A$ in $A$”? 
Homework

23.5.1 **Theorem** [Preservation]: If $\Gamma \vdash t : T$ and $t \rightarrow t'$, then $\Gamma \vdash t' : T$.

*Proof:* Exercise [Recommended, ***].

23.5.2 **Theorem** [Progress]: If $t$ is a closed, well-typed term, then either $t$ is a value or else there is some $t'$ with $t \rightarrow t'$.

*Proof:* Exercise [Recommended, ***].