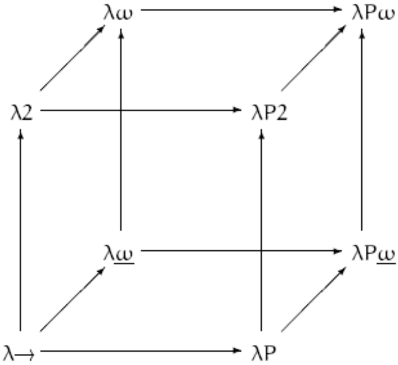
System F

Jim Royer

CIS 352

April 23, 2019



The λ-cube
https://en.wikipedia.org/wiki/Lambda_cube

Reference

Main reference:

- *Practical Foundations for Programming Languages, 1/e, "Girard's System F,"* by Robert Harper, Cambridge University Press, 2013, pages 151–161.

Also see:

- PFPL, 2/e: https://www.cs.cmu.edu/~rwh/pfpl/2nded.pdf
- https://en.wikipedia.org/wiki/Simply_typed_lambda_calculus
- https://en.wikipedia.org/wiki/System_F
- *Lambda Calculus*, by Jeremy Yallop, course notes for *Advanced Functional Programming*, University of Cambridge, 2014. https://www.cl.cam.ac.uk/teaching/1415/L28/lambda.pdf
- *Into the Core Squeezing Haskell into nine constructors* by Simon Peyton Jones http://www.erlang-factory.com/static/upload/media/1488806820775921euc2016intothecoresimonpeytonjones.pdf

The simply typed λ-calculus ($\lambda_{\rightarrow}$)

Type Syntax:

$$T ::= B \mid T \rightarrow T \qquad B ::= \text{base type constants}$$

Expression Raw Syntax:

$$E ::= X \mid \lambda X.E \mid E(E) \qquad X ::= \text{variables}$$

Typing Rules:

$$\frac{}{\Gamma, x:\tau \vdash x:\tau} \qquad \frac{\Gamma, x:\sigma \vdash e:\tau}{\Gamma \vdash (\lambda x.e):\sigma \rightarrow \tau} \qquad \frac{\Gamma \vdash e_0:\sigma \rightarrow \tau \qquad \Gamma \vdash e_1:\sigma}{\Gamma \vdash (e_0\ e_1):\tau}$$

This is ***not*** Turing-complete.

Problems with simple types

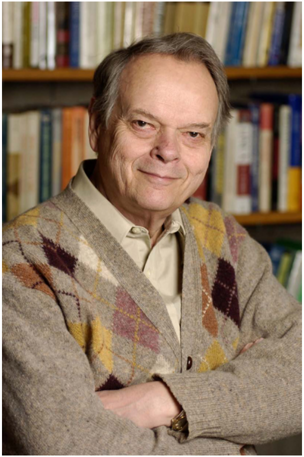
- $\lambda x.x$ has many different typings
  E.g., $\lambda x.x : \mathsf{nat} \to \mathsf{nat}$, $\lambda x.x : \mathsf{bool} \to \mathsf{bool}$, ...
- Similarly, for each choice of types a, b, and c, you need a different program for

$$\lambda f.\lambda g.\lambda x.(f(g(x))) : (b \to c) \to (a \to b) \to (a \to c)$$

- So you have to introduce many of the same functions for different types (no code sharing)
- The types are not that expressive.
- *A possible cure:* Introduce type parameters.

System F: Discoverers

Jean-Yves Girard in 1972



John Reynolds in 1974

System F

Type Syntax:

$$T ::= V \mid T \to T \mid (\forall V)T \qquad V ::= \text{type variables}$$

Expression Syntax:

$$E ::= X \mid \lambda(X : T).E \mid E(E) \mid \Lambda V.E \mid E[T] \qquad X ::= \text{variables}$$

- $\Lambda t.e$ defines a polymorphic function with type parameter t.
- $e[\tau]$ applies polymorphic function e to a type τ.
- *Example:* The polymorphic identity function

  $$\vdash \Lambda t.(\lambda(x : t).x) : (\forall t)[t \to t]$$

  and applied to a type nat

  $$\mathsf{nat\ type} \vdash (\Lambda t.(\lambda(x : t).x))[\mathsf{nat}] : \mathbb{N} \to \mathbb{N}$$

System F Statics

Type formation judgments: $\Delta \vdash \tau$ type

where $\Delta = t_1 \text{ type}, \ldots, t_k \text{ type}$ and $t_1, \ldots, t_k$ are type variables.
Intuitively $\Delta \vdash \tau$ type says: if $t_1, \ldots, t_k$ are types, so is τ.

Typing judgments: $\Delta\,\Gamma \vdash e : \tau$

Intuitively says, given the type variables of Δ and the types assigned variables in Γ, then e has type τ.

Type formation rules

$$\frac{}{\Delta, t \text{ type} \vdash t \text{ type}} \qquad \frac{\Delta \vdash t_1 \text{ type} \qquad \Delta \vdash t_2 \text{ type}}{\Delta \vdash t_1 \to t_2 \text{ type}} \qquad \frac{\Delta, t \text{ type} \vdash \tau \text{ type}}{\Delta \vdash (\forall t.\tau) \text{ type}}$$

Typing judgment rules

$$\frac{}{\Delta\ \ \Gamma, x : \tau \vdash x : \tau} \qquad \frac{\Delta \vdash \tau_1 \text{ type} \qquad \Delta\ \ \Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta\ \ \Gamma \vdash \lambda(x : \tau_1).e : \tau_1 \to \tau_2}$$

$$\frac{\Delta \vdash e_1 : \tau_2 \to \tau \qquad \Delta\ \Gamma \vdash e_2 : \tau_2}{\Delta\ \Gamma \vdash e_1(e_2) : \tau}$$

$$\frac{\Delta, t \text{ type}\ \ \Gamma \vdash e : \tau}{\Delta\ \ \Gamma \vdash \Lambda t.e : (\forall t.\tau)} \qquad \frac{\Delta\ \ \Gamma \vdash e : (\forall t.\tau') \qquad \Delta \vdash \tau \text{ type}}{\Delta\ \ \Gamma \vdash e[\tau] : \tau'[\tau/t]}$$

System F Dynamics

$$\frac{}{\lambda(x:\tau).e \;\; \mathsf{val}} \qquad \frac{}{\Lambda\tau.e \;\; \mathsf{val}}$$

$$\frac{e_2 \;\; \mathsf{val}}{(\lambda(x:\tau_1).e\; e_2) \mapsto e[e_2/x]} \qquad \frac{e_1 \mapsto e_1'}{(e_1\; e_2) \mapsto (e_1'\; e_2)} \qquad \frac{e_1 \;\; \mathsf{val} \qquad e_2 \mapsto e_2'}{(e_1\; e_2) \mapsto (e_1\; e_2')}$$

$$\frac{}{(\Lambda t.e)[\tau] \mapsto e[\tau/t]} \qquad \frac{e \mapsto e'}{\Lambda t.e \mapsto \Lambda t.e'}$$

The yellow parts for call-by-value.

Theorem (Preservation)

If $e : \tau$ and $e \mapsto e'$, then $e' : \tau$.

Theorem (Progress)

If $e : \tau$, then either $e \;\; \mathsf{val}$ or there is an e' such that $e \mapsto e'$.

System F: So why is System F neat?

I'll let other people explain:

- *An Introduction to System F*
  by Alexandre Miquel
  https://www.cs.rice.edu/~javaplt/411/11-fall/Readings/IntroToSystemF.pdf
- *Into the Core Squeezing Haskell into nine constructors*
  by Simon Peyton Jones
  http://www.erlang-factory.com/static/upload/media/1488806820775921euc2016intothecoresimonpeytonjones.pdf