The road to dependent types

Outline

- Untyped lambda calculus
- Simply typed lambda calculus
- Polymorphic lambda calculus (System F)
- Higher-order polymorphic lambda calculus (System F_ω)
- First-order dependent types
- The lambda cube

Untyped λ calculus

$$E \overset{\text{def}}{=} \lambda x.E \quad | \quad E_1\ E_2 \quad | \quad x$$

$$x \in Var$$

Untyped λ calculus with boolean constants

$$E \stackrel{\text{def}}{=} \lambda x.E \quad | \quad E_1\ E_2 \quad | \quad x \quad | \quad \text{true} \quad | \quad \text{false}$$

$$x \in \text{Var}$$

Untyped λ calculus with boolean constants

Example:

(λx.x) true

Untyped λ calculus with boolean constants

A troublesome example:

true (λx.x)

Introduce a type system to rule out such "meaningless" terms.

Simply typed λ calculus ($\lambda_{\rightarrow}$)

$$E \stackrel{\text{def}}{=} \lambda x{:}T.E \quad | \quad E_1\ E_2 \quad | \quad x$$

$$T \stackrel{\text{def}}{=} T \rightarrow T$$

$$x \in Var$$

Simply typed λ calculus

$$E \stackrel{\text{def}}{=} \lambda x{:}T.E \quad | \quad E_1\ E_2 \quad | \quad x$$

$$T \stackrel{\text{def}}{=} T \to T$$

$$x \in Var$$

terms indexed by terms

Simply typed λ calculus with boolean constants

$$E \stackrel{\text{def}}{=} \lambda x{:}T.E \quad | \quad E_1\ E_2 \quad | \quad x \quad | \quad \text{true} \quad | \quad \text{false}$$

$$T \stackrel{\text{def}}{=} T \rightarrow T \quad | \quad \text{Bool}$$

$$x \in \text{Var}$$

Simply typed λ calculus with boolean constants

Example:

(λx:Bool.x) true

Simply typed λ calculus with boolean constants

Example:

(λx:Bool.x) true

Where:

λx:Bool.x : Bool → Bool

true : Bool

Simply typed λ calculus with boolean constants

Example:

(λx:Bool.x) true

Problem: We have to define a new version of the identity function for each type of value we want to apply it to. Poor code reuse. Need polymorphism.

System F (polymorphic λ calculus)

$$E \stackrel{\text{def}}{=} \lambda x{:}T.E \quad | \quad E_1\ E_2 \quad | \quad x \quad | \quad \Lambda\alpha.E \quad | \quad E\,[\,T\,]$$

$$T \stackrel{\text{def}}{=} T \to T \quad | \quad \alpha \quad | \quad \forall\alpha.T$$

$$x \in \text{Var}$$

$$\alpha \in \text{TypeVar}$$

System F (polymorphic λ calculus)

$$E \stackrel{\text{def}}{=} \lambda x{:}T.E \quad | \quad E_1\ E_2 \quad | \quad x \quad | \quad \Lambda\alpha.E \quad | \quad E\,[\,T\,]$$

$$T \stackrel{\text{def}}{=} T \to T \quad | \quad \alpha \quad | \quad \forall\alpha.T$$

$$x \in \text{Var}$$

$$\alpha \in \text{TypeVar}$$

terms indexed by types

System F (assuming boolean constants)

Example:

$(\Lambda\alpha.\lambda x{:}\alpha.x)\ [\mathrm{Bool}]\ \mathrm{true}$

System F (assuming boolean constants)

Example:

$(\Lambda\alpha.\lambda x{:}\alpha.x)\ [Bool]\ true$

Where:

$\Lambda\alpha.\lambda x{:}\alpha.x \quad : \quad \forall\alpha.\ \alpha \to \alpha$

$(\Lambda\alpha.\lambda x{:}\alpha.x)\ [Bool] \quad : \quad Bool \to Bool$

System F (assuming boolean constants)

Example:

$(\Lambda\alpha.\lambda x{:}\alpha.x)$ [Bool] true

Problem: No way to express parametric data types (eg. List[T]). Need type functions.

System F_ω (higher-order polymorphic λ calculus)

$$E \stackrel{def}{=} \lambda x{:}T.E \quad | \quad E_1\ E_2 \quad | \quad x \quad | \quad \Lambda\alpha{:}K.E \quad | \quad E\,[\,T\,]$$

$$T \stackrel{def}{=} T \rightarrow T \quad | \quad \alpha \quad | \quad \forall\alpha{:}K.T \quad | \quad \lambda\alpha{:}K.T \quad | \quad T_1\ T_2$$

$$K \stackrel{def}{=} \star \quad | \quad K \rightarrow K$$

$x \in \mathrm{Var}$

$\alpha \in \mathrm{TypeVar}$

System F_ω (higher-order polymorphic λ calculus)

$$E \stackrel{def}{=} \lambda x{:}T.E \;\mid\; E_1\,E_2 \;\mid\; x \;\mid\; \Lambda\alpha{:}K.E \;\mid\; E\,[\,T\,]$$

$$T \stackrel{def}{=} T \to T \;\mid\; \alpha \;\mid\; \forall\alpha{:}K.T \;\mid\; \lambda\alpha{:}K.T \;\mid\; T_1\,T_2$$

$$K \stackrel{def}{=} \star \;\mid\; K \to K$$

$x \in Var$

$\alpha \in TypeVar$

types indexed by types

System F_ω (assuming list constants)

Example:

list_type = $\lambda\alpha : \star . \text{List } \alpha$

we can deduce:

list_type : $\star \rightarrow \star$

First-order dependent types (LF)

$$E \stackrel{\text{def}}{=} \lambda x{:}T.E \quad | \quad E_1\ E_2 \quad | \quad x$$

$$T \stackrel{\text{def}}{=} T \rightarrow T \quad | \quad \Pi x{:}T.T \quad | \quad T\ E$$

$$x \in \text{Var}$$

First-order dependent types

$$E \stackrel{\text{def}}{=} \lambda x{:}T.E \quad | \quad E_1\ E_2 \quad | \quad x$$

$$T \stackrel{\text{def}}{=} T \rightarrow T \quad | \quad \Pi x{:}T.T \quad | \quad T\ E$$

$$x \in Var$$

types indexed by terms

First-order dependent types (with Vec and Nat)

Example:

append : $\Pi m{:}\mathrm{Nat}.\Pi n{:}\mathrm{Nat}.\ \mathrm{Vec}\ m \rightarrow \mathrm{Vec}\ n \rightarrow \mathrm{Vec}\ (m+n)$

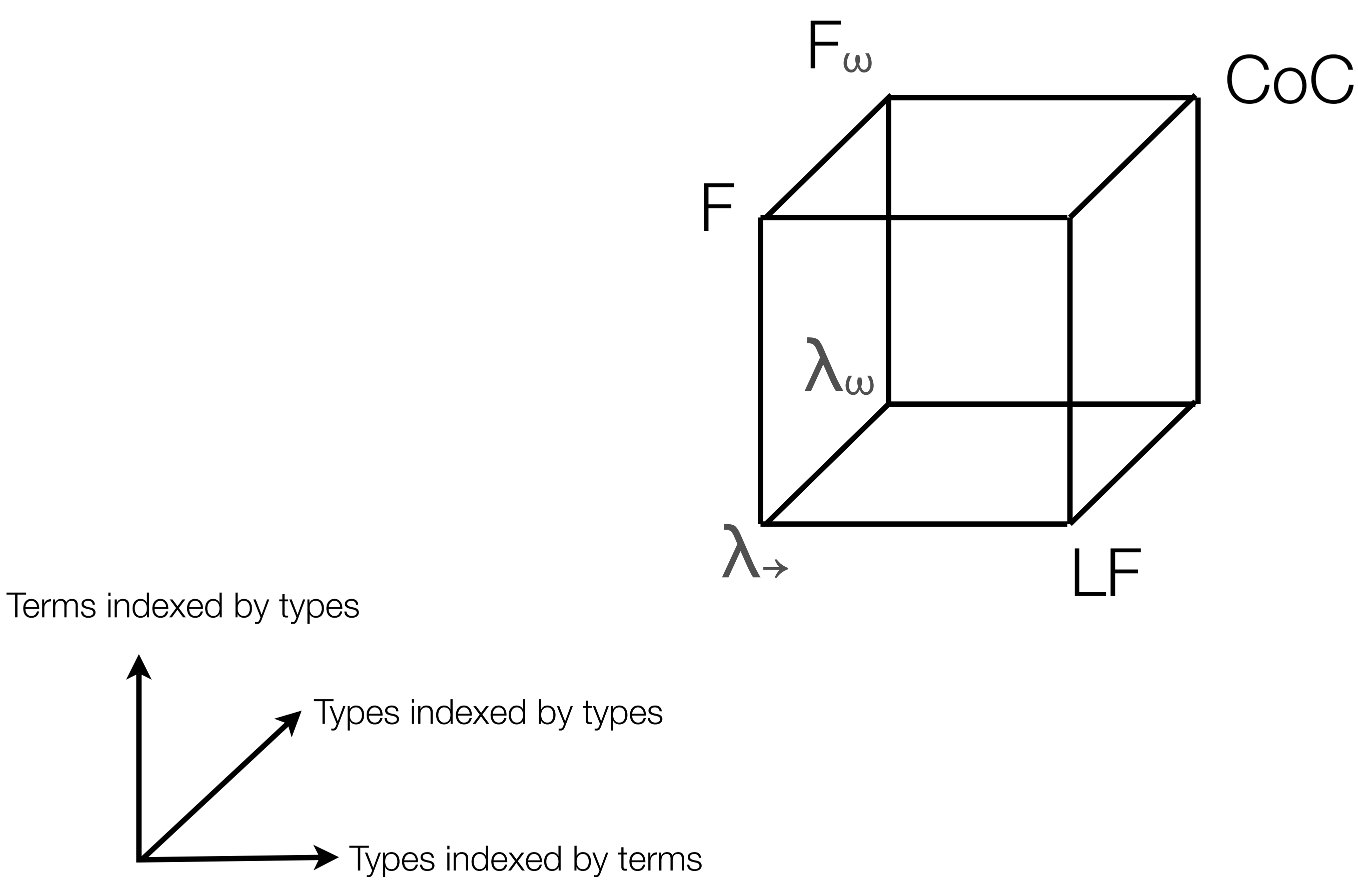
First-order dependent types (with Vec and Nat)

Test for type equality may
require term evaluation:

Vec (3+2) = Vec (1+4)

(parts of) programs evaluated at
type-checking time. How to
handle non-termination?

The Lambda Cube



The Lambda Cube