

Introduction to Type Theory
February 2008
Alpha Lernet Summer School
Piriapolis, Uruguay

Herman Geuvers
Nijmegen & Eindhoven, NL

Lecture 5: [Higher Order Logic and the Calculus of Constructions](#)

Church's STT

The original motivation of Church to define introduce simple type theory was:

to define higher order (predicate) logic

In $\lambda \rightarrow$ he adds the following

- **prop** as a basic type
- $\supset : \text{prop} \rightarrow \text{prop} \rightarrow \text{prop}$
- $\forall_{\sigma} : (\sigma \rightarrow \text{prop}) \rightarrow \text{prop}$ (for each type σ)

This defines the language of higher order logic.

Examples

- **Induction**

$$\forall_{N \rightarrow \text{prop}} (\lambda P : N \rightarrow \text{prop}. (P\ 0) \supset (\forall_N (\lambda x : N. (P\ x \supset P(S\ x)))) \supset \forall_N (\lambda x : N. P\ x)))$$

Notation: $\forall P : N \rightarrow \text{prop}. (P\ 0) \supset (\forall x : N. (P\ x \supset P(S\ x))) \supset \forall x : N. P\ x$

- **Higher order predicates/functions:** **transitive closure** of a relation R

$$\lambda R : A \rightarrow A \rightarrow \text{prop}. \lambda x, y : A.$$

$$(\forall Q : A \rightarrow A \rightarrow \text{prop}. (\text{trans}(Q) \supset (R \subseteq Q) \supset Q\ x\ y))$$

of type

$$(A \rightarrow A \rightarrow \text{prop}) \rightarrow (A \rightarrow A \rightarrow \text{prop})$$

Derivation rules for Higher Order Logic (following Church)

- Natural deduction style.
- Rules are 'on top' of the simple type theory.
- Judgements are of the form

$$\Delta \vdash_{\Gamma} \varphi$$

- $\Delta = \psi_1, \dots, \psi_n$
- Γ is a $\lambda \rightarrow$ -context
- $\Gamma \vdash \varphi : \text{prop}, \Gamma \vdash \psi_1 : \text{prop}, \dots, \Gamma \vdash \psi_n : \text{prop}$
- Γ is usually left implicit: $\Delta \vdash \varphi$

(axiom)	$\Delta \vdash \varphi$	if $\varphi \in \Delta$
(\supset -introduction)	$\frac{\Delta \cup \varphi \vdash \psi}{\Delta \vdash \varphi \supset \psi}$	
(\supset -elimination)	$\frac{\Delta \vdash \varphi \supset \psi \quad \Delta \vdash \varphi}{\Delta \vdash \psi}$	
(\forall -introduction)	$\frac{\Delta \vdash \varphi}{\Delta \vdash \forall x:\sigma.\varphi}$	if $x:\sigma \notin \text{FV}(\Delta)$
(\forall -elimination)	$\frac{\Delta \vdash \forall x:\sigma.\varphi}{\Delta \vdash \varphi[t/x]}$	if $t : \sigma$
(conversion)	$\frac{\Delta \vdash \varphi}{\Delta \vdash \psi}$	if $\varphi =_{\beta} \psi$

Full Church higher order logic [1940]

Church has additional things that we will not consider now:

- **Negation** connective with rules
- Classical logic

$$\frac{\Delta \vdash \neg\neg\varphi}{\Delta \vdash \varphi}$$

- Define other connectives in terms of \supset, \forall, \neg (classically).
- **Choice** operator $\iota_\sigma : (\sigma \rightarrow \text{prop}) \rightarrow \sigma$
- Rule for ι :

$$\frac{\Delta \vdash \exists!x:\sigma.P x}{\Delta \vdash P(\iota_\sigma P)}$$

Full Church higher order logic [1940]

Church' original higher order logic is basically the logic of the theorem prover HOL (Gordon, Melham, Harrison) and of Isabelle-HOL (Paulson, Nipkow).

We will here restrict to the basic **constructive** core (\forall, \supset) of HOL.

The need for a **conversion** rule

$$\frac{\Delta \vdash \forall P:N \rightarrow \text{prop}.\dots Pc\dots}{\Delta \vdash (\dots (\lambda y:N.y > 0)c\dots)} \forall\text{-elim}$$
$$\frac{\Delta \vdash (\dots (\lambda y:N.y > 0)c\dots)}{\Delta \vdash (\dots c > 0\dots)} \text{conv}$$

Definability of other connectives (constructively)

$$\begin{aligned}\perp & := \forall \alpha:\text{prop}.\alpha \\ \varphi \wedge \psi & := \forall \alpha:\text{prop} . (\varphi \supset \psi \supset \alpha) \supset \alpha \\ \varphi \vee \psi & := \forall \alpha:\text{prop} . (\varphi \supset \alpha) \supset (\psi \supset \alpha) \supset \alpha \\ \exists x:\sigma . \varphi & := \forall \alpha:\text{prop} . (\forall x:\sigma . \varphi \supset \alpha) \supset \alpha\end{aligned}$$

Idea:

The definition of a connective is an encoding of the **elimination** rule.

Existential quantifier

$$\exists x:\sigma.\varphi := \forall \alpha:\text{prop}.\left(\forall x:\sigma.\varphi \supset \alpha\right) \supset \alpha$$

Derivation of the elimination rule in HOL.

$$\frac{\frac{\frac{[\varphi] \quad \vdots}{\exists x:\sigma.\varphi} \quad C}{C} \quad x \notin \text{FV}(C, \text{ass.})}{C} \quad \frac{\frac{\frac{\exists x:\sigma.\varphi}{(\forall x:\sigma.\varphi \supset C) \supset C} \quad C}{\forall x:\sigma.\varphi \supset C} \quad \frac{[\varphi] \quad \vdots \quad C}{C}}{C}$$

Existential quantifier

$$\exists x:\sigma.\varphi := \forall \alpha:\text{prop}.(\forall x:\sigma.\varphi \supset \alpha) \supset \alpha$$

Derivation of the introduction rule in HOL.

$$\frac{\frac{\varphi[t/x]}{\exists x:\sigma.\varphi} \quad \frac{\frac{\varphi[t/x] \quad \frac{[\forall x:\sigma.\varphi \supset \alpha]}{\varphi[t/x] \supset \alpha}}{\alpha}}{(\forall x:\sigma.\varphi \supset \alpha) \supset \alpha}}{\exists x:\sigma.\varphi}$$

Equality

Equality is **definable** in higher order logic:

t and q terms are equal if they share the same properties
(**Leibniz** equality)

Definition in HOL (for $t, q : A$):

$$t =_A q := \forall P:A \rightarrow \text{prop}. (Pt \supset Pq)$$

- This equality is **reflexive** and **transitive** (easy)
- It is also **symmetric**(!) Trick: find a “smart” predicate P

Exercise: Prove reflexivity, transitivity and symmetry of $=_A$.

Exercise: Proof of symmetry of $=_A$

(Trick: take $\lambda y:A.y =_A t$ for P .)

$$\frac{\frac{\frac{\Delta \vdash t =_A q}{\Delta \vdash \forall P:A \rightarrow \text{prop.}(Pt \supset Pq)}}{\Delta \vdash (t =_A t) \supset (q =_A t)}}{\Delta \vdash q =_A t} \quad \frac{\dots}{\Delta \vdash t =_A t}$$

Exercise

The **transitive closure** of a binary relation R on A has been defined as follows.

$$\text{trclos } R \quad := \quad \lambda x, y: A. \\ (\forall Q: A \rightarrow A \rightarrow \text{Prop}. (\text{trans}(Q) \rightarrow (R \subseteq Q) \rightarrow (Q \ x \ y))).$$

1. Prove that the **transitive closure** is **transitive**.
2. Prove that the **transitive closure of R** contains R .

(axiom)	$\Delta \vdash \varphi$	if $\varphi \in \Delta$
(\supset -introduction)	$\frac{\Delta \cup \varphi \vdash \psi}{\Delta \vdash \varphi \supset \psi}$	
(\supset -elimination)	$\frac{\Delta \vdash \varphi \supset \psi \quad \Delta \vdash \varphi}{\Delta \vdash \psi}$	
(\forall -introduction)	$\frac{\Delta \vdash \varphi}{\Delta \vdash \forall x:\sigma.\varphi}$	if $x:\sigma \notin \text{FV}(\Delta)$
(\forall -elimination)	$\frac{\Delta \vdash \forall x:\sigma.\varphi}{\Delta \vdash \varphi[t/x]}$	if $t : \sigma$
(conversion)	$\frac{\Delta \vdash \varphi}{\Delta \vdash \psi}$	if $\varphi =_{\beta} \psi$

Proofs-as-terms

Why not introduce a λ -term notation for the derivations?

This gives a type theory λ HOL

- No ‘lifting’ of prop to the type level (via $T : \text{prop} \rightarrow \text{type}$).
- Let prop be a new ‘universe’ of propositional types.
- Direct representation (shallow embedding) of HOL into the type theory λ HOL

(axiom)	$\frac{}{\Delta \vdash x : \varphi}$	if $x:\varphi \in \Delta$
(\supset -introduction)	$\frac{\Delta, x:\varphi \vdash M : \psi}{\Delta \vdash \lambda x:\varphi.M : \varphi \supset \psi}$	
(\supset -elimination)	$\frac{\Delta \vdash M : \varphi \supset \psi \quad \Delta \vdash N : \varphi}{\Delta \vdash M N \psi}$	
(\forall -introduction)	$\frac{\Delta \vdash M : \varphi}{\Delta \vdash \lambda x:\sigma.M : \forall x:\sigma.\varphi}$	if $x:\sigma \notin \text{FV}(\Delta)$
(\forall -elimination)	$\frac{\Delta \vdash M : \forall x:\sigma.\varphi}{\Delta \vdash M t : \varphi[t/x]}$	if $t : \sigma$
(conversion)	$\frac{\Delta \vdash M : \varphi}{\Delta \vdash M : \psi}$	if $\varphi =_{\beta} \psi$

Two levels

Now we have **two** 'levels' of type theories

- The (simple) type theory describing the **language** of HOL
- The type theory for the **proof-terms** of HOL

NB Many rules, many **similar** rules.

We put these levels together into one type theory **λ HOL**.

Pseudoterms:

$$T ::= \text{Prop} \mid \text{Type} \mid \text{Type}' \mid \text{Var} \mid (\Pi \text{Var}:T.T) \mid (\lambda \text{Var}:T.T) \mid \text{TT}$$

$\{\text{Prop}, \text{Type}, \text{Type}'\}$ is the set of **sorts**, \mathcal{S} .

Some of the typing rules are **parametrized**

(**axiom**) $\vdash \text{Prop} : \text{Type}$

$\vdash \text{Type} : \text{Type}'$

(**var**)
$$\frac{\Gamma \vdash A : s}{\Gamma, x:A \vdash x : A}$$

(**weak**)
$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash M : C}{\Gamma, x:A \vdash M : C}$$

$$(II) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2 \quad \text{if } (s_1, s_2) \in \{ (\text{Type}, \text{Type}), (\text{Prop}, \text{Prop}), (\text{Type}, \text{Prop}) \}}{\Gamma \vdash \Pi x:A. B : s_2}$$

$$(\lambda) \quad \frac{\Gamma, x:A \vdash M : B \quad \Gamma \vdash \Pi x:A. B : s}{\Gamma \vdash \lambda x:A. M : \Pi x:A. B}$$

$$(\text{app}) \quad \frac{\Gamma \vdash M : \Pi x:A. B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$$

$$(\text{conv}) \quad \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma \vdash M : B} \text{ if } A =_{\beta} B$$

$$(II) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2}{\Gamma \vdash \Pi x:A. B : s_2} \quad \text{if } (s_1, s_2) \in \{ (\text{Type}, \text{Type}), (\text{Prop}, \text{Prop}), (\text{Type}, \text{Prop}) \}$$

- The combination **(Type, Type)** forms the **function types** $A \rightarrow B$ for $A, B:\text{Type}$.

This comprises the **unary predicate types** and **binary relations types**:
 $A \rightarrow \text{Prop}$ and $A \rightarrow A \rightarrow \text{Prop}$.

Also: **higher order predicate types** like $(A \rightarrow A \rightarrow \text{Prop}) \rightarrow \text{Prop}$.

NB A Π -type formed by **(Type, Type)** is always an \rightarrow -type.

- **(Prop, Prop)** forms the **propositional types** $\varphi \rightarrow \psi$ for $\varphi, \psi:\text{Prop}$; **implicational formulas**.

NB A Π -type formed by **(Type, Type)** is always an \rightarrow -type.

- **(Type, Prop)** forms the **dependent propositional type** $\Pi x:A. \varphi$ for $A:\text{Type}, \varphi:\text{Prop}$; **universally quantified formulas**.

Example

Deriving **irreflexivity** from **anti-symmetry**

Rel := $\lambda X:\text{Type}. X \rightarrow X \rightarrow \text{Prop}$

AntiSym := $\lambda X:\text{Type}. \lambda R:(\text{Rel } X). \forall x, y:X. (Rxy) \supset (Ryx) \supset \perp$

Irrefl := $\lambda X:\text{Type}. \lambda R:(\text{Rel } X). \forall x:X. (Rxx) \supset \perp$

Derivation in HOL

$$\frac{\frac{\frac{\forall x^A y^A R x y \supset R y x \supset \perp}{\forall y^A R x y \supset R y x \supset \perp}}{R x x \supset R x x \supset \perp} [R x x]}{R x x \supset \perp} [R x x]}{\perp} \\ \frac{R x x \supset \perp}{\forall x^A . R x x \supset \perp}$$

Derivation in HOL, with terms

$$\begin{array}{c}
 z : \forall x^A y^A R x y \supset R y x \supset \perp \\
 \hline
 zx : \forall y^A R x y \supset R y x \supset \perp \\
 \hline
 zxx : R x x \supset R x x \supset \perp \quad [q : R x x] \\
 \hline
 zxxq : R x x \supset \perp \quad [q : R x x] \\
 \hline
 zxxqq : \perp \\
 \hline
 \lambda q : (R x x). zxxqq : R x x \supset \perp \\
 \hline
 \lambda x : A. \lambda q : (R x x). zxxqq : \forall x^A. R x x \supset \perp
 \end{array}$$

Typing judgement in λ HOL:

$$\begin{array}{l}
 A : \text{Type}, R : A \rightarrow A \rightarrow \text{Prop}, \quad z : \Pi x, y : A. (R x y \rightarrow R y x \rightarrow \perp) \vdash \\
 \lambda x : A \lambda q : (R x x). z x x q q : (\Pi x : A. R x x \rightarrow \perp)
 \end{array}$$

λ HOL versus HOL

Question: is the type theory λ HOL really isomorphic with HOL?

Yes: we can **disambiguate** the syntax. [No details.]

Properties of λHOL

- **Uniqueness of types**

If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then $A =_{\beta} B$.

- **Subject Reduction**

If $\Gamma \vdash M : A$ and $M \longrightarrow_{\beta} N$, then $\Gamma \vdash N : A$.

- **Strong Normalization**

If $\Gamma \vdash M : A$, then all β -reductions from M terminate.

Proof of SN is a **higher order** extension of the one for $\lambda 2$ (using the **saturated sets**).

Decidability Questions

$\Gamma \vdash M : \sigma?$ TCP

$\Gamma \vdash M : ?$ TSP

$\Gamma \vdash ? : \sigma$ TIP

For λHOL :

- TIP is **undecidable**
- TCP/TSP: simultaneously.

The type checking algorithm is close to the one for λP . (In λP we had a judgement of **correct** context; this form of judgement could also be introduced for λHOL)

λ HOL contains $\lambda 2$ and $\lambda \rightarrow$

$$(II) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2}{\Gamma \vdash \Pi x:A. B : s_2} \quad \text{if } (s_1, s_2) \in \{ (\text{Type}, \text{Type}), (\text{Prop}, \text{Prop}), (\text{Type}, \text{Prop}) \}$$

This rule allows to form

- \rightarrow -types on the **Type-level** (one copy of $\lambda \rightarrow$)
- \rightarrow -types on the **Prop-level** (second copie of $\lambda \rightarrow$)
- $\Pi \alpha:\text{Prop}.\alpha \rightarrow \alpha$: **polymorphic types** on the **Prop-level** (one copie of $\lambda 2$)

Extending λHOL

$$(II) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2}{\Gamma \vdash \Pi x:A. B : s_2} \quad \text{if } (s_1, s_2) \in \{ (\text{Type}, \text{Type}), (\text{Prop}, \text{Prop}), (\text{Type}, \text{Prop}) \}$$

Why not extend λHOL to include

- Higher order logic over **polymorphic domains**?
like $\Pi A : \text{Type}. A \rightarrow A$
- Quantification over **all domains**?
like in $\Pi A : \text{Type}. \Pi P : A \rightarrow \text{Prop}. \Pi x : A. P x \rightarrow P x$
- Add $(s_1, s_2) \in \{ (\text{Type}', \text{Type}) \}$ to obtain higher order logic over **polymorphic domains** \rightsquigarrow system λU^-
- Add $(s_1, s_2) \in \{ (\text{Type}', \text{Prop}) \}$ to allow **quantification over all domains** \rightsquigarrow system λU

Problem

- λU ($\lambda\text{HOL} + (\text{Type}', \text{Type})$ and $(\text{Type}', \text{Prop})$) is **inconsistent** (Girard)
- λU^- ($\lambda\text{HOL} + (\text{Type}', \text{Type})$) is **inconsistent** (Coquand, Hurkens)

NB $\lambda\text{HOL} + (\text{Type}', \text{Prop})$ is consistent.

Implications

- λU^- can't be used as a logic.
- In λU^- , there is a **closed** term M with $\vdash M : \perp$
- This M can not be in **normal form** (by some syntactic reasoning)
- So, λU^- is **not SN**

Type Checking

Type Checking in λU^- is still **decidable**:

All **types** (terms of type Prop, Type or Type') are **strongly normalizing**

$$\begin{aligned} \text{Type}_\Gamma(MN) &= \text{if } \text{Type}_\Gamma(M) = C \text{ and } \text{Type}_\Gamma(N) = D \\ &\quad \text{then } \text{if } C \twoheadrightarrow_\beta \Pi x:A.B \text{ and } A =_\beta D \\ &\quad \quad \text{then } B[N/x] \text{ else 'false'} \\ &\quad \text{else 'false'}, \end{aligned}$$

In the type synthesis algorithm we only check equality of **types**

Variations

Variations on the rules of λ HOL:

- There are many type system with (slightly) different rules
- Many (proofs of) properties are similar
- **Plan**: Study these type systems in one general framework:
 - The **cube** of typed λ -calculi (Barendregt)
 - **Pure Type Systems** (Terlouw, Berardi)

The λ -cube

The **cube** of typed λ -calculi: (forget about Type' for the moment)

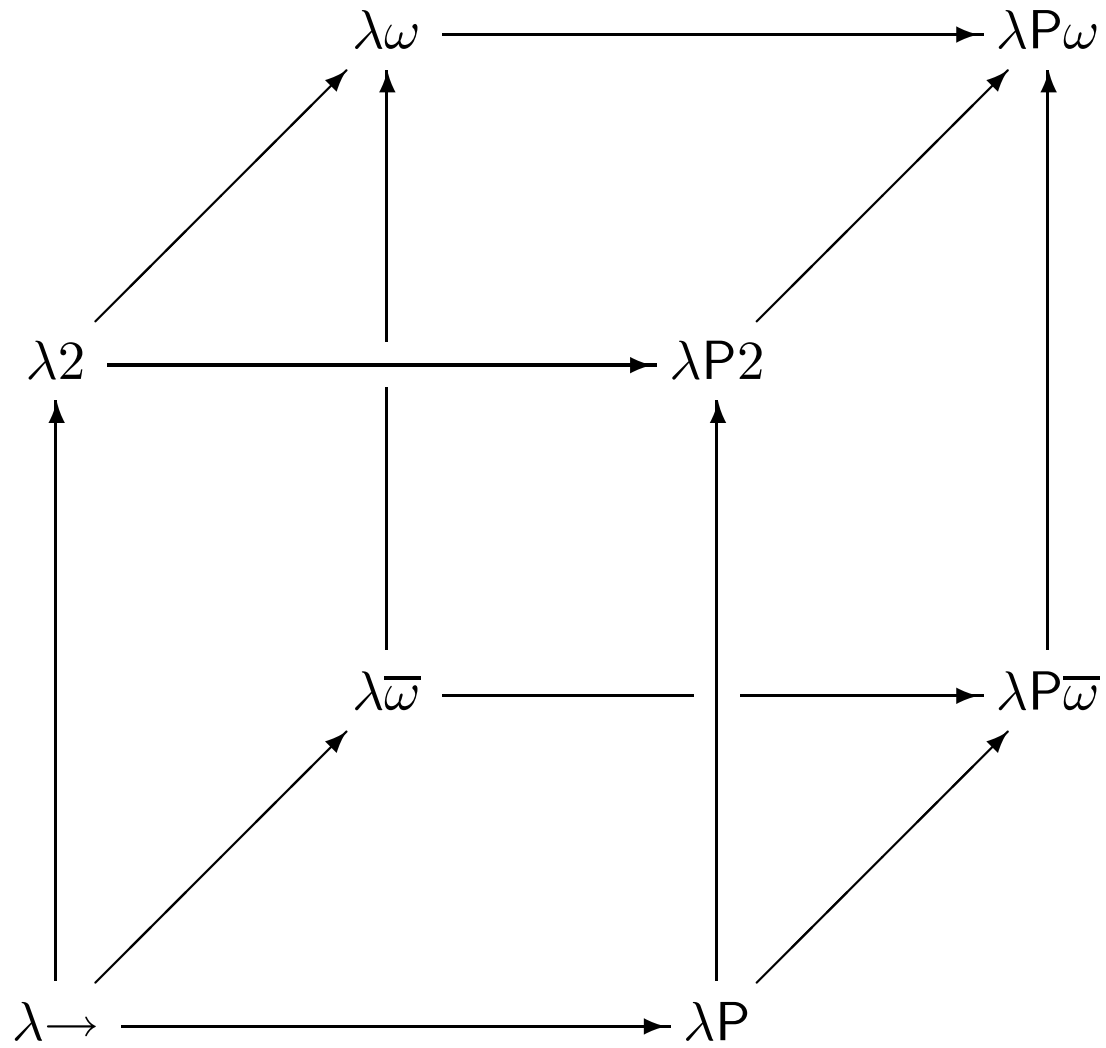
Vary on all possible combinations for

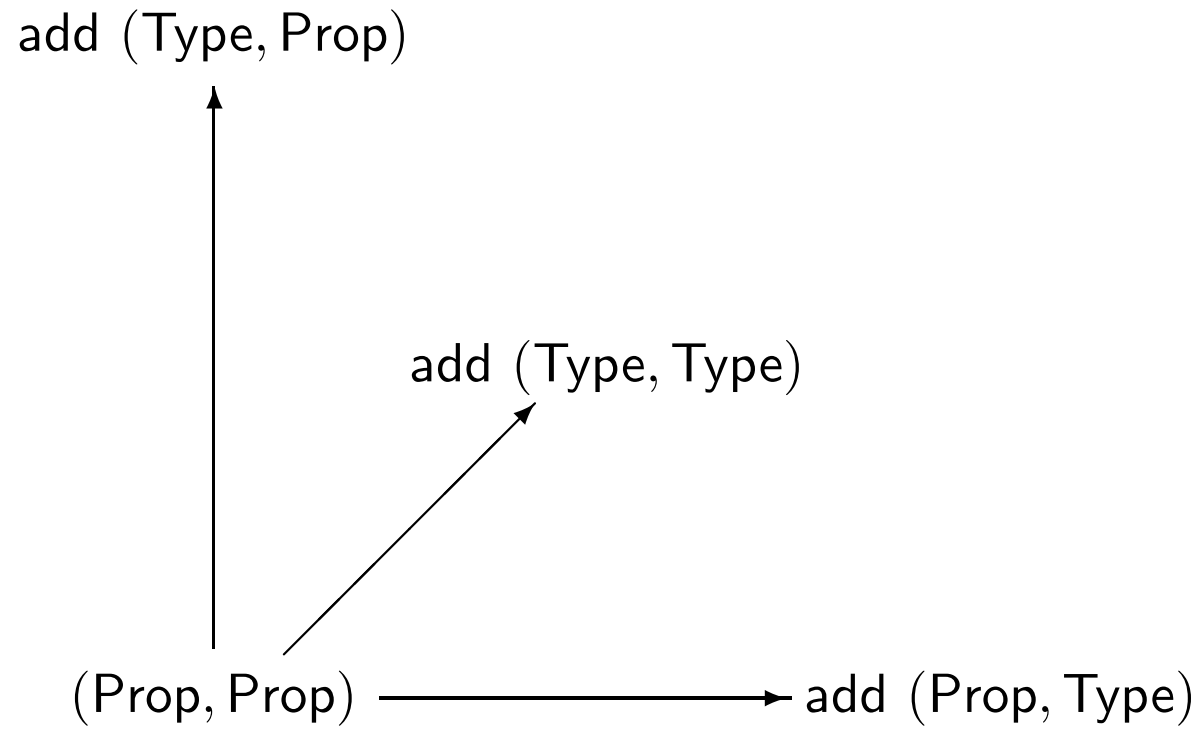
$$\mathcal{R} \subseteq \{ (\text{Prop}, \text{Prop}), (\text{Type}, \text{Prop}), (\text{Type}, \text{Type}), (\text{Prop}, \text{Type}) \}$$

in the Π -rule:

$$(\Pi) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2}{\Gamma \vdash \Pi x:A. B : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{R}$$

We take $(\text{Prop}, \text{Prop})$ in every \mathcal{R}





$$(II) \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2}{\Gamma \vdash \Pi x:A. B : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{R}$$

System	\mathcal{R}
$\lambda \rightarrow$	(Prop, Prop)
$\lambda 2$ (system F)	(Prop, Prop) (Type, Prop)
λP (LF)	(Prop, Prop) (Prop, Type)
$\lambda \bar{\omega}$	(Prop, Prop) (Type, Type)
$\lambda P 2$	(Prop, Prop) (Type, Prop) (Prop, Type)
$\lambda \omega$ (system Fω)	(Prop, Prop) (Type, Prop) (Type, Type)
$\lambda P \bar{\omega}$	(Prop, Prop) (Prop, Type) (Type, Type)
$\lambda P \omega$ (CC)	(Prop, Prop) (Type, Prop) (Prop, Type) (Type, Type)

Calculus of Constructions

$\lambda \rightarrow$ in this presentation is equivalent to $\lambda \rightarrow$ as presented before.
Similarly for $\lambda 2$, λP , ... This **cube** also gives a **fine structure** for the

Calculus of Constructions, CC (Coquand and Huet)

- Polymorphic **data types** on the Prop-level,
e.g. $\Pi \alpha : \text{Prop}. \alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$.
- **Predicate domains** on the Type-level,
e.g. $N \rightarrow N \rightarrow \text{Prop}$
- **Logic** on the Prop-level,
e.g. $\varphi \wedge \psi := \Pi \alpha : \text{Prop}. (\varphi \rightarrow \psi \rightarrow \alpha) \rightarrow \alpha$.
- **Universal quantification** (first and higher order),
e.g. $\Pi P : N \rightarrow \text{Prop}. \Pi x : N. Px \rightarrow Px$.

HOL in CC

One can do **higher order predicate logic** in CC, in a slightly unusual way:

- ‘**propositions**’ and first order ‘**sets**’ are both of type Prop
- propositions and sets are **completely mixed**

Is it **faithful** to do higher order predicate logic in CC??

Answer: **No!**

There are **non-provable** formulas of HOL that become **inhabited** in CC

Consider **extensionality** of propositions:

$$\mathbf{EXT} := \forall \alpha, \beta: \text{prop}. (\alpha \leftrightarrow \beta) \Rightarrow (\alpha =_{\text{prop}} \beta)$$

In CC, this becomes $\prod \alpha, \beta: \text{Prop}. (\alpha \leftrightarrow \beta) \rightarrow (\alpha =_{\text{Prop}} \beta)$

Suppose two base domains A and B and constants $a : A$, $b : B$.

In HOL, the following formulas are consistent.

- $\varphi := \forall x:A. x = a$, $\psi := \forall x:B. \exists y:B. x \neq y$

But in CC, **EXT** also applies to the base sets A and B .

$A \leftrightarrow B$ (both are non-empty) so $A =_{\text{Prop}} B$

so property ψ (of B) also applies to A

so $\forall x:A. \exists y:A. x \neq y$

contradicting φ

So, in CC, φ and ψ are **inconsistent**

We have to be **careful** when doing higher order logic in CC

Or: system $\lambda\text{PRED}\omega$ (taking the **sets** and the **propositions apart**):

- **Sorts:** Prop, Set, Type^p , Type^s
- **Axioms** for these sorts: Prop : Type^p , Set : Type^s
- **Rules R:**
 - (Prop, Prop): implication
 - (Set, Prop): first order quantification
 - (Type^p , Prop): higher order quantification
 - (Set, Set): function types
 - (Set, Type^p): predicate types
 - (Type^p , Type^p): higher order types