Introduction to Type Theory

August 2007

Types Summer School

Bertinoro, It

Herman Geuvers

Nijmegen NL

Lecture 4: Higher Order Logic, λ-cube, Pure Type Systems

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

to define higher order (predicate) logic

In $\lambda\!\rightarrow$ he adds the following

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

This defines the language of higher order logic.

- Induction

$$\forall_{N\to\mathsf{prop}}(\quad \lambda P{:}N\to\mathsf{prop}.(P0)$$
$$\supset (\forall_N(\lambda x{:}N.(Px \supset P(S\,x)))$$
$$\supset \forall_N(\lambda x{:}N.Px))\ )$$

Notation: $\forall P{:}N\to\mathsf{prop}(\quad (P0)$
$$\supset (\forall x{:}N.(Px \supset P(S\,x)))$$
$$\supset \forall x{:}N.Px\ )$$

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

$$\lambda R : A\to A\to\mathsf{prop}.\ \lambda x, y : A.$$
$$(\forall Q : A\to A\to\mathsf{prop}.\ (\mathsf{trans}(Q) \supset (R \subseteq Q) \supset Q\,x\,y))$$

of type

$$(A\to A\to\mathsf{prop})\to(A\to A\to\mathsf{prop})$$

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

$$\Delta \vdash_\Gamma \varphi$$

  - $\Delta = \psi_1, \ldots, \psi_n$
  - Γ is a $\lambda{\to}$-context
  - $\Gamma \vdash \varphi$: prop, $\Gamma \vdash \psi_1$: prop,. . . , $\Gamma \vdash \psi_n$: 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 $x{:}\sigma \notin \mathsf{FV}(\Delta)$

($\forall$-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 $\varphi =_\beta \psi$

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 $\iota_\sigma : (\sigma \to \mathsf{prop}) \to \sigma$
- Rule for ι:

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

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:

$$\dfrac{\dfrac{\Delta \vdash \forall P{:}N{\to}\mathsf{prop}.(\ldots Pc\ldots)}{\Delta \vdash (\ldots(\lambda y{:}N.y>0)c\ldots)}\,\forall\text{-elim}}{\Delta \vdash (\ldots c>0\ldots)}\,\mathsf{conv}$$

Definability of other connectives (constructively):

$$
\begin{array}{rcl}
\bot & := & \forall\alpha{:}\mathsf{prop}.\alpha \\
\varphi\wedge\psi & := & \forall\alpha{:}\mathsf{prop}.(\varphi \supset \psi \supset \alpha) \supset \alpha \\
\varphi\vee\psi & := & \forall\alpha{:}\mathsf{prop}.(\varphi \supset \alpha) \supset (\psi \supset \alpha) \supset \alpha \\
\exists x{:}\sigma.\varphi & := & \forall\alpha{:}\mathsf{prop}.(\forall x{:}\sigma.\varphi \supset \alpha) \supset \alpha
\end{array}
$$

Idea:

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

Existential quantifier

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

Derivation of the elimination rule in HOL.

$$\dfrac{\exists x{:}\sigma.\varphi \qquad \begin{array}{c}[\varphi]\\ \vdots \\ C\end{array}}{C}\; x \notin \mathsf{FV}(C, \mathsf{ass.}) \qquad \dfrac{\exists x{:}\sigma.\varphi \qquad \begin{array}{c}[\varphi]\\ \vdots \\ C\end{array}}{\dfrac{(\forall x{:}\sigma.\varphi \supset C) \supset C \qquad \forall x{:}\sigma.\varphi \supset C}{C}}$$

Existential quantifier

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

Derivation of the introduction rule in HOL.

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

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 $t, q : A$):

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

- 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:

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

$$\begin{aligned}\mathsf{trclos}\, R \quad := \quad & \lambda x, y{:}A. \\ & (\forall Q{:}A{\to}A{\to}\mathsf{Prop}.(\mathsf{trans}(Q){\to}(R \subseteq Q){\to}(Q\, x\, y))).\end{aligned}$$

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 $x{:}\sigma \notin \mathsf{FV}(\Delta)$

($\forall$-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 $\varphi =_\beta \psi$

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

This gives a type theory λHOL

- No 'lifting' of prop to the $\mathbf{type}$ level (via T : prop$\rightarrow\mathbf{type}$).
- Let prop be a new 'universe' of propositional types.
- Direct encoding (shallow embedding) of HOL into the type theory λHOL

(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 $x{:}\sigma \notin \mathsf{FV}(\Delta)$

($\forall$-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 $\varphi =_\beta \psi$

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 λHOL.
Pseudoterms:

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

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

Some of the typing rules are parametrized

(axiom) $\vdash \mathsf{Prop} : \mathsf{Type}$ $\qquad\qquad$ $\vdash \mathsf{Type} : \mathsf{Type}'$

(var) $\dfrac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}$ $\qquad$ (weak) $\dfrac{\Gamma \vdash A : s \quad \Gamma \vdash M : C}{\Gamma, x{:}A \vdash M : C}$

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

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

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

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

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

- The combination (Type,Type) forms the function types $A \to B$ for A, B:Type.
  This comprises the unary predicate types and binary relations types: $A \to$Prop and $A \to A \to$Prop.
  Also: higher order predicate types like $(A \to A \to \text{Prop}) \to$Prop.
  NB A Π-type formed by (Type,Type) is always an $\to$-type.
- (Prop,Prop) forms the propositional types $\varphi \to \psi$ for φ, ψ:Prop; implicational formulas.
  NB A Π-type formed by (Type,Type) is always an $\to$-type.
- (Type,Prop) forms the dependent propositional type $\Pi x{:}A.\varphi$ for A:Type, φ:Prop; universally quantified formulas.

Example: Deriving irreflexivity from anti-symmetry

$$\begin{aligned}
\text{Rel} &:= \lambda X{:}\text{Type}.X \to X \to \text{Prop} \\
\text{AntiSym} &:= \lambda X{:}\text{Type}.\lambda R{:}(\text{Rel}\, X).\forall x, y{:}X.(Rxy) \supset (Ryx) \supset \bot \\
\text{Irrefl} &:= \lambda X{:}\text{Type}.\lambda R{:}(\text{Rel}\, X).\forall x{:}X.(Rxx) \supset \bot
\end{aligned}$$

Derivation in HOL:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall x^A y^A R\,x\,y \supset R\,y\,x \supset \bot}{\forall y^A R\,x\,y \supset R\,y\,x \supset \bot}}{R\,x\,x \supset R\,x\,x \supset \bot} \qquad [R\,x\,x]}{R\,x\,x \supset \bot} \qquad [R\,x\,x]}{\bot}}{R\,x\,x \supset \bot}}{\forall x^A.R\,x\,x \supset \bot}$$

Derivation in HOL, with terms:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{z : \forall x^A y^A R\,x\,y \supset R\,y\,x \supset \bot}{zx : \forall y^A R\,x\,y \supset R\,y\,x \supset \bot}}{zxx : R\,x\,x \supset R\,x\,x \supset \bot} \qquad [q : R\,x\,x]}{zxxq : R\,x\,x \supset \bot} \qquad [q : R\,x\,x]}{zxxqq : \bot}}{\lambda q{:}(R\,x\,x).zxxqq : R\,x\,x \supset \bot}}{\lambda x{:}A.\lambda q{:}(R\,x\,x).zxxqq : \forall x^A.R\,x\,x \supset \bot}$$

Typing judgement in λHOL:

$$A{:}\mathsf{Type}, R{:}A{\to}A{\to}\mathsf{Prop}, \quad z : \Pi x, y{:}A.(R\,x\,y{\to}R\,y\,x{\to}\bot) \vdash$$
$$\lambda x{:}A\lambda q{:}(R\,x\,x).z\,x\,x\,q\,q : (\Pi x{:}A.R\,x\,x{\to}\bot)$$

Question: is the type theory λHOL 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)

λHOL contains $\lambda 2$ and $\lambda\!\rightarrow$.

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

This rule allows to form

- $\rightarrow$-types on the Type-level (one copie of $\lambda\!\rightarrow$)
- $\rightarrow$-types on the Prop-level (second copie of $\lambda\!\rightarrow$)
- $\Pi\alpha$:Prop.$\alpha\rightarrow\alpha$: polymorphic types on the Prop-level (one copie of $\lambda 2$)

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

Why not extend λHOL to include

- Higher order logic over polymorphic domains?
  like $\Pi A : \text{Type}.A \to A$
- Quantification over all domains?
  like in $\Pi A : \text{Type}.\Pi P{:}A \to \text{Prop}.\Pi x{:}A.P\,x \to P\,x$

This can easily be done by allowing in the Π-rule

- $(s_1, s_2) \in \{\ (\text{Type}', \text{Type})\ \}$ to obtain higher order logic over polymorphic domains $\rightsquigarrow$ system λU^-
- $(s_1, s_2) \in \{\ (\text{Type}', \text{Prop})\ \}$ to allow quantification over all domains $\rightsquigarrow$ system λU

Problem:

- λU (λHOL + (Type',Type) and (Type',Prop)) is inconsistent (Girard)
- λU^- (λHOL + (Type',Type)) is inconsistent (Coquand, Hurkens)

NB λHOL + (Type',Prop) is consistent.

Implications

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

Type Checking in λU^- is still decidable:

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

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

In the type synthesis algorithm we only check equality of types

Variations on the rules of λHOL:

- 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)

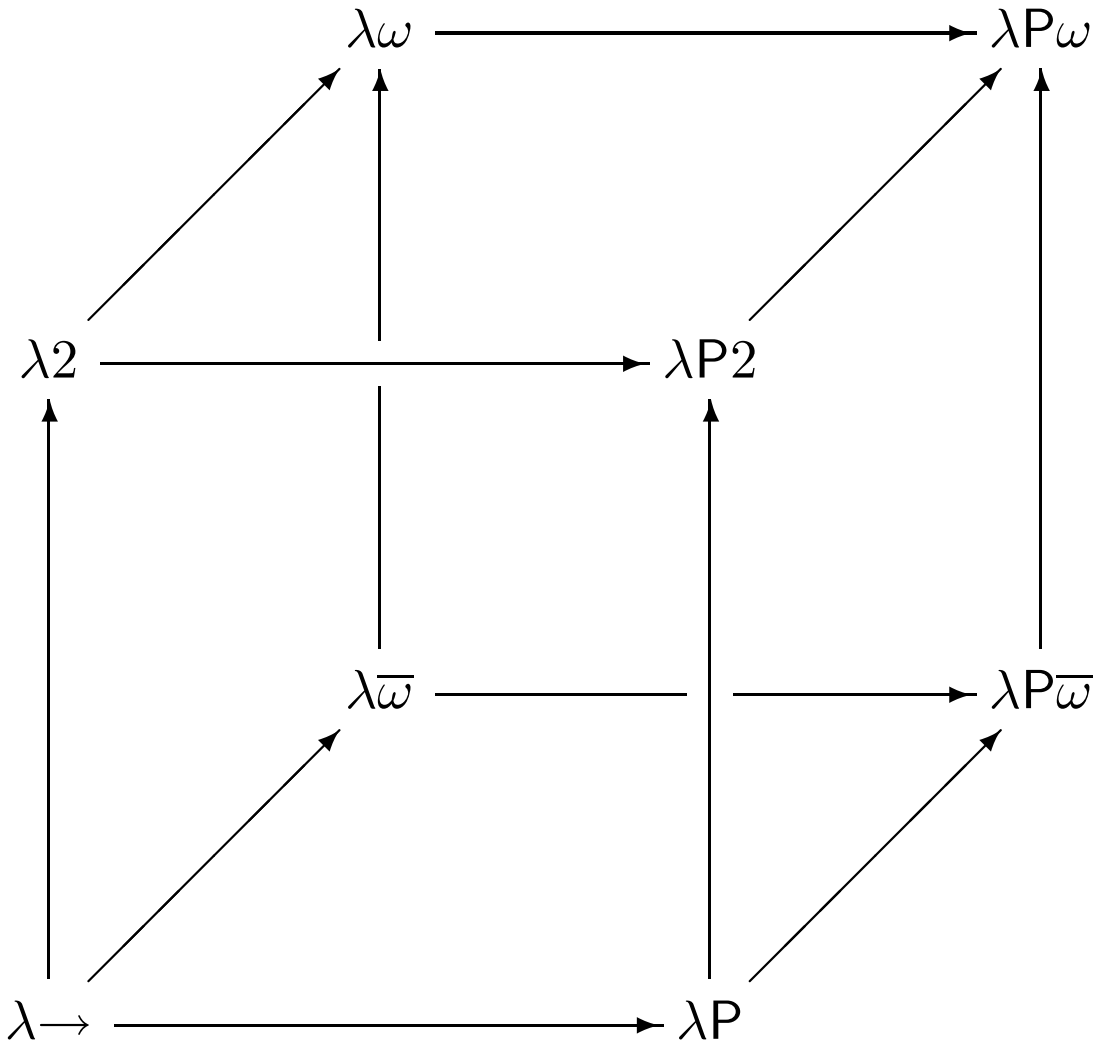
The cube of typed λ-calculi: (forget about Type$'$ for the moment)
Vary on all possible combinations for

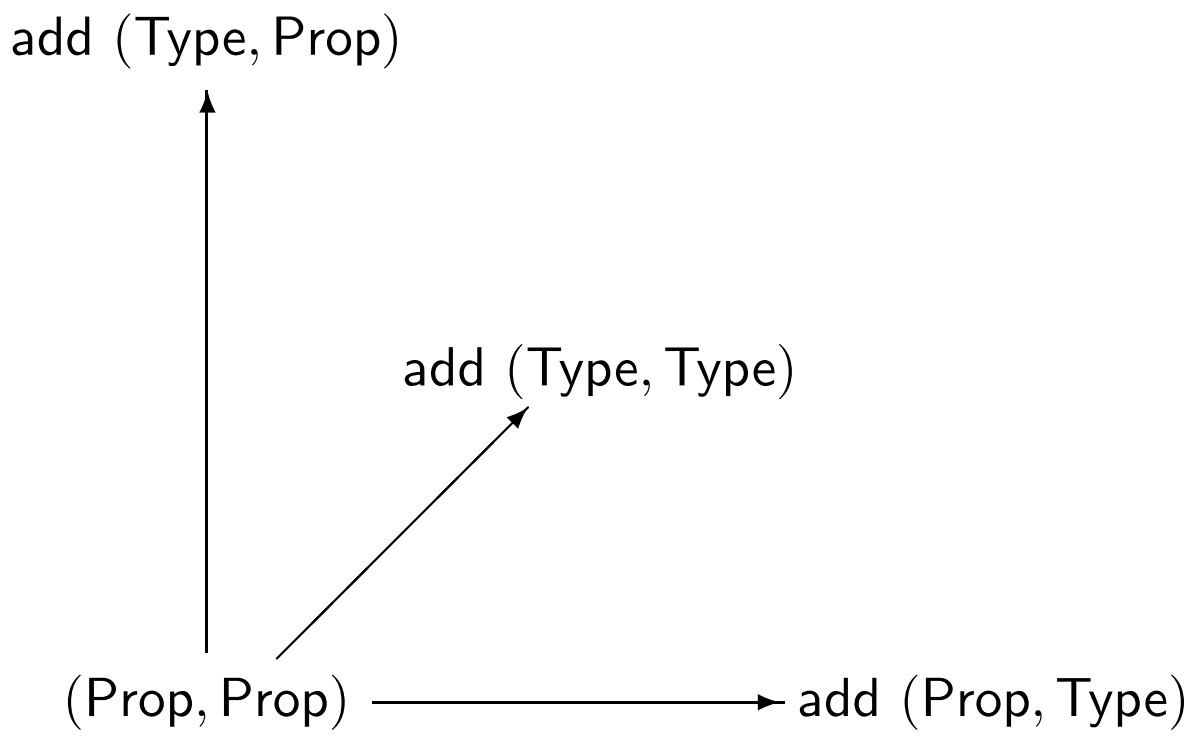
$$\mathcal{R} \subseteq \{\ (\mathsf{Prop}, \mathsf{Prop}), (\mathsf{Type}, \mathsf{Prop}), (\mathsf{Type}, \mathsf{Type}), (\mathsf{Prop}, \mathsf{Type})\ \}$$

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 $(\mathsf{Prop}, \mathsf{Prop})$ in every $\mathcal{R}$


$\lambda\omega$
$\lambda\mathsf{P}\omega$
$\lambda 2$
$\lambda\mathsf{P}2$
$\lambda\overline{\omega}$
$\lambda\mathsf{P}\overline{\omega}$
$\lambda\!\rightarrow$
$\lambda\mathsf{P}$


add (Type, Prop)
add (Type, Type)
(Prop, Prop)
add (Prop, Type)

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

System	$\mathcal{R}$			
$\lambda\!\!\rightarrow$	(Prop, Prop)			
$\lambda 2$ (system F)	(Prop, Prop)	(Type, Prop)		
λP (LF)	(Prop, Prop)		(Prop, Type)	
$\lambda\overline{\omega}$	(Prop, Prop)			(Type, Type)
λP2	(Prop, Prop)	(Type, Prop)	(Prop, Type)	
$\lambda\omega$ (system Fω)	(Prop, Prop)	(Type, Prop)		(Type, Type)
$\lambda P\overline{\omega}$	(Prop, Prop)		(Prop, Type)	(Type, Type)
λPω (CC)	(Prop, Prop)	(Type, Prop)	(Prop, Type)	(Type, Type)

$\lambda\!\rightarrow$ in this presentation is equivalent to $\lambda\!\rightarrow$ in the way we've presented before. Similarly for $\lambda 2$, λP, ... This cube also gives a fine structure for the

Calculus of Constructions, CC (Coquand and Huet)

CC has:

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

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 propostions:

$$\mathsf{EXT} := \forall \alpha, \beta{:}\mathsf{prop}.(\alpha \Leftrightarrow \beta) \Rightarrow (\alpha =_{\mathsf{prop}} \beta)$$

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

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	$A =_{\mathsf{Prop}} B$
	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: we may try to improve on this: taking the sets and the propositions apart:

System λPREDω:

- 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

Pure Type Systems

Determined by a triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ with

- $\mathcal{S}$ the set of sorts
- $\mathcal{A}$ the set of axioms, $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$
- $\mathcal{R}$ the set of rules, $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$

If $s_2 = s_3$ in $(s_1, s_2, s_3) \in \mathcal{R}$, we write $(s_1, s_2) \in \mathcal{R}$.

pseudoterms:

$$\mathsf{T} ::= \mathcal{S} \mid \mathsf{Var} \mid (\Pi \mathsf{Var}{:}\mathsf{T}.\mathsf{T}) \mid (\lambda \mathsf{Var}{:}\mathsf{T}.\mathsf{T}) \mid \mathsf{TT}.$$

$$\text{(sort)} \quad \vdash s_1 : s_2 \ \text{ if } (s_1, s_2) \in \mathcal{A} \qquad \text{(var)} \ \frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A} \ \text{ if } x \notin \Gamma$$

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

$$(\Pi) \ \frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_3} \ \text{ if } (s_1, s_2, s_3) \in \mathcal{R}$$

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

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

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

Examples of PTSs

CC	
$\mathcal{S}$	Prop, Type
$\mathcal{A}$	Prop : Type
$\mathcal{R}$	(Prop, Prop), (Prop, Type), (Type, Prop), (Type, Type)

λPREDω	
$\mathcal{S}$	Set, Type^s, Prop, Type
$\mathcal{A}$	Set : Type^s, Prop : Type
$\mathcal{R}$	(Set, Set), (Set, Type), (Type, Type), (Prop, Prop), (Set, Prop), (Type, Prop)

λHOL	
$\mathcal{S}$	Prop, Type, Type$'$
$\mathcal{A}$	Prop : Type, Type : Type$'$
$\mathcal{R}$	(Prop, Prop), (Type, Type), (Type, Prop)

λU	
$\mathcal{S}$	Prop, Type, Type$'$
$\mathcal{A}$	Prop : Type, Type : Type$'$
$\mathcal{R}$	(Prop, Prop), (Type, Type), (Type$'$, Type), (Type$'$, Prop), (Type, Prop)

$\lambda\star$	
$\mathcal{S}$	$\star$
$\mathcal{A}$	$\star : \star$
$\mathcal{R}$	$(\star, \star)$

A PTS-morphism from $\lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$ to $\lambda(\mathcal{S}', \mathcal{A}', \mathcal{R}')$ is an $f : \mathcal{S} \to \mathcal{S}'$ that preserves the axioms and rules:

- if $(s_1, s_2) \in \mathcal{A}$ then $(f(s_1), f(s_2)) \in \mathcal{A}'$
- if $(s_1, s_2, s_3) \in \mathcal{R}$ then $(f(s_1), f(s_2), f(s_3)) \in \mathcal{R}'$

f extends the pseudoterms and contexts:

If $\Gamma \vdash M : A$ then $f(\Gamma) \vdash f(M) : f(A)$

There are now two type systems for higher order predicate logic: λPREDω and λHOL.

λPREDω	
$\mathcal{S}$	Set, Type$^{\mathcal{S}}$, Prop, Type
$\mathcal{A}$	Set : Type$^{\mathcal{S}}$, Prop : Type
$\mathcal{R}$	(Set, Set), (Set, Type), (Type, Type), (Prop, Prop), (Set, Prop), (Type, Prop)

λHOL	
$\mathcal{S}$	Prop, Type, Type$'$
$\mathcal{A}$	Prop : Type, Type : Type$'$
$\mathcal{R}$	(Prop, Prop), (Type, Type), (Type, Prop)

They are equivalent:

The PTS-morphism $h : \lambda\text{PRED}\omega \to \lambda\text{HOL}$, given by

$$h(\text{Prop}) := \text{Prop} \qquad h(\text{Set}) := \text{Type}$$
$$h(\text{Type}^p) := \text{Type} \qquad h(\text{Type}^s) := \text{Type}'$$

constitutes an isomorphism between the derivable sequents.

What is the use of the abstract framework of PTSs?

- Present (the kernel of) systems in a uniform way
- Compare systems (e.g. λHOL, λPREDω, CC) within one framework
- Prove properties for all systems at once.

Properties of PTSs.

- Uniqueness of types
  If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$, then $A =_\beta B$.
  Holds if $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ and $\mathcal{R} \subseteq (\mathcal{S} \times \mathcal{S}) \times \mathcal{S}$ are functions.
  Definition A PTS where $\mathcal{A}$ and $\mathcal{R}$ are functions is called a functional PTS (or singly sorted PTS).

- Subject Reduction
  If $\Gamma \vdash M : A$ and $M \longrightarrow_\beta N$, then $\Gamma \vdash N : A$.

- Substitution property
  If $\Gamma, x : B, \Delta \vdash M : A$, $\Gamma \vdash P : B$, then
  $\Gamma, \Delta[P/x] \vdash M[P/x] : A[P/x]$.

Properties of PTSs ctd.

- **Thinning**
  If $\Gamma \vdash M : A$ and $\Gamma \subseteq \Delta$, Δ well-formed, then $\Delta \vdash M : A$.

- **Strengthening**
  If $\Gamma, x : B, \Delta \vdash M : A$ and $x \notin \mathsf{FV}(M, A, \Delta)$, then
  $\Gamma, \Delta \vdash M : A$.

Strong Normalization (SN)
If $\Gamma \vdash M : A$, then all β-reductions from M terminate.

SN holds for some PTSs, and for some not.
SN for CC is proved by a higher order extension of the saturated sets argument (for $\lambda 2$).

Some more examples of PTSs

CC^ω	
$\mathcal{S}$	$\mathsf{Prop}, \{\mathsf{Type}_i\}_{i\in\mathbb{N}}$
$\mathcal{A}$	$\mathsf{Prop} : \mathsf{Type}, \mathsf{Type}_i : \mathsf{Type}_{i+1}$
$\mathcal{R}$	$(\mathsf{Prop}, \mathsf{Prop}), (\mathsf{Prop}, \mathsf{Type}_i), (\mathsf{Type}_i, \mathsf{Prop})$
	$(\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_{\max(i,j)})$

Recall that $(\mathsf{Type}_1, \mathsf{Type}_0, \mathsf{Type}_0)$ is inconsistent (λU)
Similarly $(\mathsf{Type}_{i+1}\,\mathsf{Type}_i, \mathsf{Type}_i))$ would be inconsistent.

The Extended Calculus of Constructions has in addition

- Cumulativity: $\mathsf{Prop} \subseteq \mathsf{Type}_0 \subseteq \mathsf{Type}_1 \subseteq \ldots$, so

$$\frac{\Gamma \vdash A : \mathsf{Prop}}{\Gamma \vdash A : \mathsf{Type}_0} \qquad \frac{\Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash A : \mathsf{Type}_{i+1}}$$

- Σ-types:

$$\frac{\Gamma \vdash A : \mathsf{Prop} \;\; \Gamma, x{:}A \vdash B : \mathsf{Prop}}{\Gamma \vdash \Sigma x{:}A.B : \mathsf{Prop}} \qquad \frac{\Gamma \vdash A : \mathsf{Type}_i \;\; \Gamma, x{:}A \vdash B : \mathsf{Type}_j}{\Gamma \vdash \Sigma x{:}A.B : \mathsf{Type}_{\max(i,j)}}$$

For $\varphi : \mathsf{Prop}$

- We have $\Pi A{:}\mathsf{Type}_i.\varphi : \mathsf{Prop}$, but
- We do not have $\Sigma A{:}\mathsf{Type}_i.\varphi : \mathsf{Prop}$.

Note: The type theory of Coq has in addition $\mathsf{Set} : \mathsf{Type}$ and rules $(\mathsf{Set}, \mathsf{Set}), (\mathsf{Type}_i, \mathsf{Set}), (\mathsf{Set}, \mathsf{Prop})$.

Σ-types

$$\frac{\Gamma \vdash A : \mathsf{Type}\ \ \Gamma, x{:}A \vdash B : \mathsf{Prop}}{\Gamma \vdash \Sigma x{:}A.B : \mathsf{Prop}}$$

leads to inconsistency:

- Define $\Omega := \Sigma A{:}\mathsf{Set}.\Sigma R{:}A{\to}A{\to}\mathsf{Prop}.\mathrm{wf}(R)$
  $\mathrm{wf}(R)$ denotes that R is well-founded. (No infinite descending R-chains).
- Define $<$ on Ω by

  $$(A, R) < (B, Q) := R \text{ can be embedded into } Q \text{ under some } b : B$$

  Then
  - $<$ is well-founded on Ω
  - If (A, R) well-founded, then $(A, R) < (\Omega, <)$

  so contradiction: $\ldots < (\Omega, <) < (\Omega, <) < (\Omega, <)$.

Intensionality versus Extensionality

The equality in the side condition in the conv rule is intensional and decidable. It can also be extensional.

Extensional equality amounts to the following rules:

$$(\text{ext}) \qquad \frac{\Gamma \vdash M, N : A \to B \quad \Gamma \vdash p : \Pi x{:}A.(Mx = Nx)}{\Gamma \vdash M = N : A \to B}$$

$$(\text{conv}) \qquad \frac{\Gamma \vdash P : A \quad \Gamma \vdash A = B : s}{\Gamma \vdash P : B}$$

- Intensional equality of functions = equality of algorithms
  (the way the function is presented to us (syntax))
- Extensional equality of functions = equality of graphs
  (the (set-theoretic) meaning of the function (semantics))

Adding the rule (ext) renders TCP undecidable:

Suppose $H : (A \to B) \to$ Prop and $x : (H\ f)$; then

$$x : (H\ g) \text{ iff there is a } p : \Pi x{:}A.f\ x = g\ x$$

So, to solve this TCP, we need to solve a TIP.

The interactive theorem prover Nuprl is based on extensional type theory.