Properties of PTS

Substitution property

If $\Gamma, x : B, \Delta \vdash M : A$ and $\Gamma \vdash N : B$, then $\Gamma, \Delta[x := N] \vdash M[x := N] : A[x := N]$.

Correctness of types

If $\Gamma \vdash A : B$, then either $B \in \mathcal{S}$ or $\exists s \in \mathcal{S}. \Gamma \vdash B : s$.

Thinning

If $\Gamma \vdash A : B$ is legal and $\Gamma \subseteq \Delta$, then $\Delta \vdash A : B$.

Strengthening

If $\Gamma_1, x : A, \Gamma_2 \vdash M : B$ and $x \notin \mathsf{FV}(\Gamma_2) \cup \mathsf{FV}(M) \cup \mathsf{FV}(B)$, then $\Gamma_1, \Gamma_2 \vdash M : B$.

Properties of PTS (cont.)

Confluence

Let $M, N \in \mathcal{T}$. If $M =_\beta N$, then $M \twoheadrightarrow_\beta P$ and $N \twoheadrightarrow_\beta P$ for some $P \in \mathcal{T}$.

Subject Reduction

If $\Gamma \vdash M : A$ and $M \twoheadrightarrow_\beta N$, then $\Gamma \vdash N : A$.

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.

Type Checking, Type Inference and Type Inhabitation

Problems one would like to have an algorithm for:

Type Checking Problem (TCP)	$\Gamma \vdash_T M : A$?
Type Synthesis Problem (TSP)	$\Gamma \vdash_T M : ?$
Type Inhabitation Problem (TIP)	$\Gamma \vdash_T ? : A$

In practice, TCP is often reduced to TSP:

To solve $M\,N : \sigma$ one has to solve $N : ?$ and if this gives answer τ, solve $M : \tau \to \sigma$.

- For $\lambda\!\to$ all these problems are decidable.
- TIP is undecidable for extensions of $\lambda\!\to$ (as it corresponds to the provability in some logic).

Strong Normalization and Decidability of Type Checking

Normalization and Type Checking are intimately connected due to conversion rule.

Strong Normalization (SN)

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

SN holds for some PTS (e.g., all subsystems of **λC**) and for some not (e.g., λU⁻, λ∗).

A PTS is (weakly or strongly) **normalizing** if all its legal terms are (weakly or strongly) normalizing.

Decidability of Type Checking

In a PTS that is (weakly or strongly) normalizing and with S finite, the problems of type checking and type synthesis are decidable.

Barendregt's λ-Cube

Barendregt's λ-Cube was proposed as a fine-grained analysis of the Calculus of Constructions.

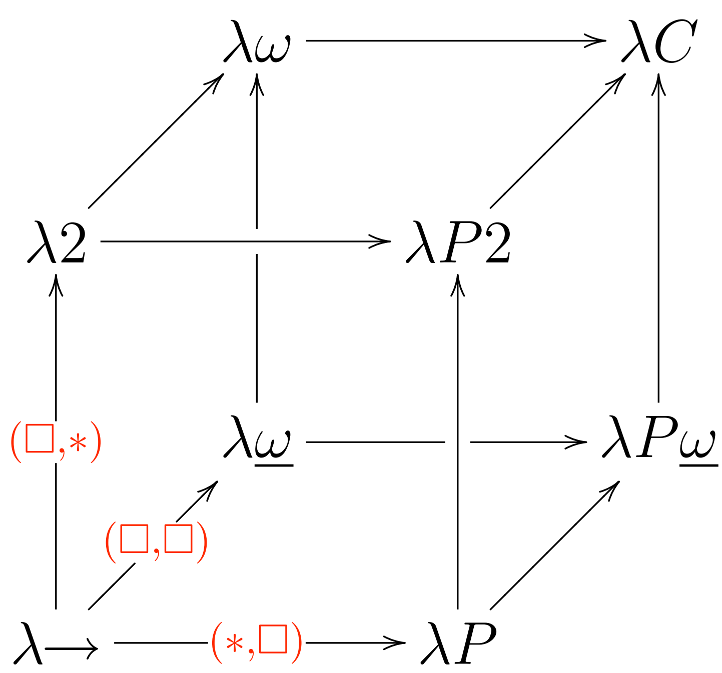
The λ-Cube

The cube of typed lambda calculi consists of eight PTS all of them having $\mathcal{S} = \{*, \Box\}$, and $\mathcal{A} = \{* : \Box\}$ and the rules for each system as follows:

System	$\mathcal{R}$			
$\lambda\!\rightarrow$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
λC	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

The λ-Cube

Note that arrows denote inclusion of one system in another.



Dependencies

Let us call "types" to the pseudo-terms of type $*$ and "kinds" to the pseudo-terms of type $\Box$.

term : type : kind

- $(*, *)$ Terms depending on terms. (functions)

$$\vdash (\lambda x{:}\sigma.\, x) : \sigma \to \sigma$$

- $(\Box, *)$ Terms depending on types. (polymorphism)

$$\vdash (\lambda \alpha{:}*. \lambda x{:}\alpha.\, \alpha) : \Pi \alpha{:}*.\, \alpha \to \alpha$$

- $(*, \Box)$ Types depending on terms. (dependent functions)

$$A : *, P : A \to * \vdash (\lambda a{:}A. \lambda x{:}Pa.\, x) : \Pi a{:}A.\, Pa \to Pa$$

- $(\Box, \Box)$ Types depending on types. (constructors of a kind)

$$\vdash (\lambda \alpha{:}*. \alpha \to \alpha) : * \to *$$

Logics as PTS

Other examples of PTS were given by Berardi who defined logical systems as PTS.

Eight systems of intuitionistic logic will be introduced that correspond in some sense to the systems in the λ-cube. Four systems of proposition logic and four systems of many-sorted predicate logic.

λPROP	proposition logic
λPROP2	second-order proposition logic
λPROP$\underline{\omega}$	weakly higher-order proposition logic
λPROPω	higher-order proposition logic
λPRED	predicate logic
λPRED2	second-order predicate logic
λPRED$\underline{\omega}$	weakly higher-order predicate logic
λPREDω	higher-order predicate logic

Salient features

- All the systems are minimal logics in the sense that the only logical operators are $\supset$ and $\forall$.
- However, for the second and higher-order systems the operators $\neg$, $\wedge$, $\vee$ and $\exists$, as well as Leibeniz's equality are all definable.
- Classical versions of the logics in the upper-plane (of the cube) are obtained easily (by adding the axiom $\forall \alpha . \neg\neg\alpha \rightarrow \alpha$).

Berardi's Logic Cube

The Logic Cube

The cube of logical typed lambda calculi consists of the following eight PTS. Each of them has

$$\begin{aligned} \mathcal{S} &= \mathrm{Prop},\ \mathrm{Set},\ \mathrm{Type}^p,\ \mathrm{Type}^s \\ \mathcal{A} &= (\mathrm{Prop} : \mathrm{Type}^p),\ (\mathrm{Set} : \mathrm{Type}^s) \end{aligned}$$

and the rules for each of the systems are

System	$\mathcal{R}$			
λPROP	(Prop, Prop)			
λPROP2	(Prop, Prop)		$(\mathrm{Type}^p, \mathrm{Prop})$	
$\lambda\mathrm{PROP}\underline{\omega}$	(Prop, Prop)		$(\mathrm{Type}^p, \mathrm{Type}^p)$	
$\lambda\mathrm{PROP}\omega$	(Prop, Prop)		$(\mathrm{Type}^p, \mathrm{Type}^p)$ $(\mathrm{Type}^p, \mathrm{Prop})$	
λPRED	(Set, Set) (Prop, Prop)	$(\mathrm{Set}, \mathrm{Type}^p)$ (Set, Prop)		
λPRED2	(Set, Set) (Prop, Prop)	$(\mathrm{Set}, \mathrm{Type}^p)$ (Set, Prop)	$(\mathrm{Type}^p, \mathrm{Prop})$	
$\lambda\mathrm{PRED}\underline{\omega}$	(Set, Set) (Prop, Prop)	$(\mathrm{Set}, \mathrm{Type}^p)$ (Set, Prop)	$(\mathrm{Type}^p, \mathrm{Set})$	$(\mathrm{Type}^p, \mathrm{Type}^p)$
$\lambda\mathrm{PRED}\omega$	(Set, Set) (Prop, Prop)	$(\mathrm{Set}, \mathrm{Type}^p)$ (Set, Prop)	$(\mathrm{Type}^p, \mathrm{Set})$ $(\mathrm{Type}^p, \mathrm{Prop})$	$(\mathrm{Type}^p, \mathrm{Type}^p)$

Set is the class of sets and Prop is the class of propositions.

The Logic Cube

$$
\begin{tikzcd}
 & \lambda\mathrm{PROP}\omega \arrow[rr] & & \lambda\mathrm{PRED}\omega \\
\lambda\mathrm{PROP}2 \arrow[ur] \arrow[rr] & & \lambda\mathrm{PRED}2 \arrow[ur] & \\
 & \lambda\mathrm{PROP}\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda\mathrm{PRED}\underline{\omega} \arrow[uu] \\
\lambda\mathrm{PROP} \arrow[uu, "{(\Box,*)}"] \arrow[ur, "{(\Box,\Box)}"] \arrow[rr, "{(*,\Box)}"] & & \lambda\mathrm{PRED} \arrow[uu] \arrow[ur] &
\end{tikzcd}
$$

Dependencies

The sorts Set and Type^p form the universes of domains.

- $A_1 \to \ldots \to A_n \to \alpha$ with α : Set are functional types.
- $A_1 \to \ldots \to A_n \to \mathrm{Prop}$ are predicate types.

The sort Type^s allows the introduction of variables of type Set.

- (Prop, Prop) allows the formation of implication of two formulae

$$\phi : \mathrm{Prop}, \psi : \mathrm{Prop} \vdash \phi \to \psi : \mathrm{Prop}$$

- (Set, Prop) allows quantification over sets

$$A : \mathrm{Set}, \phi : \mathrm{Prop} \vdash \underbrace{(\Pi x : A.\, \phi)}_{\forall x : A. \phi} : \mathrm{Prop}$$

Dependencies (cont.)

- (Set, Type^p) allows the formation of first-order predicates

$$A : \mathsf{Set} \ \vdash \ A \to \mathsf{Prop} : \mathsf{Type}^p$$

hence $$A : \mathsf{Set}, P : A \to \mathsf{Prop}, x : A \ \vdash \ Px : \mathsf{Prop}$$

P is a predicate over a set A.

- (Type^p , Prop) allows quantification over predicate types

$$A : \mathsf{Set} \ \vdash \ \underbrace{(\Pi P{:}A \to \mathsf{Prop}.\, \Pi x{:}A.\, Px \to Px)}_{\forall P{:}A\to\mathsf{Prop}.\, \forall x{:}A.\, Px\to Px} : \mathsf{Prop}$$

Dependencies (cont.)

- (Set, Set) allows function types

$$A : \mathsf{Set}, B : \mathsf{Set} \ \vdash \ A \to B : \mathsf{Set}$$

$$\dfrac{\dfrac{\vdots}{A : \mathsf{Set}, B : \mathsf{Set} \ \vdash \ A : \mathsf{Set}} \quad \dfrac{\vdots}{A : \mathsf{Set}, B : \mathsf{Set}, x : A \ \vdash \ B : \mathsf{Set}}}{A : \mathsf{Set}, B : \mathsf{Set} \ \vdash \ \underbrace{A \to B}_{\Pi x:A.B} : \mathsf{Set}} \ (\mathsf{Set}, \mathsf{Set})$$

- (Type^p , Type^p) allows higher order types

$$A : \mathsf{Set} \ \vdash \ (\Pi P{:}A \to \mathsf{Prop}.\, \mathsf{Prop}) : \mathsf{Type}^p$$

$$\dfrac{\dfrac{\vdots}{A : \mathsf{Set} \ \vdash \ A \to \mathsf{Prop} : \mathsf{Type}^p} \quad \dfrac{\vdots}{A : \mathsf{Set}, P : A \to \mathsf{Prop} \ \vdash \ \mathsf{Prop} : \mathsf{Type}^p}}{A : \mathsf{Set} \ \vdash \ (\Pi P{:}A \to \mathsf{Prop}.\, \mathsf{Prop}) : \mathsf{Type}^p} \ (\mathsf{Type}^p, \mathsf{Type}^p)$$

Example of a derivation tree

$$\dfrac{\dfrac{\vdash \mathsf{Set} : \mathsf{Type}^s}{A : \mathsf{Set} \vdash A : \mathsf{Set}} \qquad \dfrac{\dfrac{\vdash \mathsf{Prop} : \mathsf{Type}^p \quad \vdash \mathsf{Set} : \mathsf{Type}^s}{A : \mathsf{Set} \vdash \mathsf{Prop} : \mathsf{Type}^p} \quad \dfrac{\vdash \mathsf{Set} : \mathsf{Type}^s}{A : \mathsf{Set} \vdash A : \mathsf{Set}}}{A : \mathsf{Set}, y : A \vdash \mathsf{Prop} : \mathsf{Type}^p}\,(\mathsf{Set}, \mathsf{Type}^p)}{A : \mathsf{Set} \vdash A \to \mathsf{Prop} : \mathsf{Type}^p} \tag{2.1}$$

$$\dfrac{\dfrac{\vdots}{A : \mathsf{Set}, P : A \to \mathsf{Prop}, x : A \vdash P : A \to \mathsf{Prop}} \quad \dfrac{\vdots}{A : \mathsf{Set}, P : A \to \mathsf{Prop}, x : A \vdash x : A}}{A : \mathsf{Set}, P : A \to \mathsf{Prop}, x : A \vdash Px : \mathsf{Prop}} \tag{2.2}$$

$$\dfrac{(2.2) \quad (2.2)}{A : \mathsf{Set}, P : A \to \mathsf{Prop}, x : A, q : Px \vdash Px : \mathsf{Prop}} \tag{2.3}$$

$$\dfrac{(2.1) \qquad \dfrac{A : \mathsf{Set}, P : A \to \mathsf{Prop}, x : A \vdash A : \mathsf{Set} \quad \dfrac{(2.2) \quad (2.3)}{A : \mathsf{Set}, P : A \to \mathsf{Prop}, x : A \vdash Px \to Px : \mathsf{Prop}}\,(\mathsf{Prop}, \mathsf{Prop})}{A : \mathsf{Set}, P : A \to \mathsf{Prop} \vdash (\Pi x : A.\, Px \to Px) : \mathsf{Prop}}\,(\mathsf{Set}, \mathsf{Prop})}{A : \mathsf{Set} \vdash (\Pi P : A \to \mathsf{Prop}.\, \Pi x : A.\, Px \to Px) : \mathsf{Prop}}\,(\mathsf{Type}^p, \mathsf{Prop})$$

Second-order definability of the logical operations

Despite the logical construction directly encoded in PTS are implication and universal quantification, it is a well known fact in that the upper-plane of the cube the logic connectives $\wedge$, $\vee$, $\bot$, $\neg$ and $\exists$ are definable in terms of $\supset$ and $\forall$.

- For $A, B : \mathsf{Prop}$ define

$$\begin{array}{rcl} \bot & \equiv & \Pi\alpha : \mathsf{Prop}.\, \alpha \\ \neg A & \equiv & A \to \bot \\ A \wedge B & \equiv & \Pi\alpha : \mathsf{Prop}.\, (A \to B \to \alpha) \to \alpha \\ A \vee B & \equiv & \Pi\alpha : \mathsf{Prop}.\, (A \to \alpha) \to (B \to \alpha) \to \alpha \end{array}$$

- For $A : \mathsf{Prop}$ and $X : \mathsf{Set}$ define

$$\exists x : X.A \equiv \Pi\alpha : \mathsf{Prop}.\, (\Pi x : X.\, A \to \alpha) \to \alpha$$

- For $X : \mathsf{Set}$ and $x, y : X$ define the equality predicate $=_L$ called Leibniz equality.

$$(x =_L y) \equiv \Pi P : X \to \mathsf{Prop}.\, Px \to Py$$

Examples

It is not difficult to check that the intuitionistic elimination and introduction rules for the logic connectives (∧, ∨, ⊥, ¬ and ∃) are sound.

Remember $A \wedge B \;\equiv\; \Pi\alpha\!:\!\mathsf{Prop}.\,(A \to B \to \alpha) \to \alpha$

Elimination rules

$$\frac{A \wedge B}{A}\ (\wedge\mathsf{E}_1) \qquad A : \mathsf{Prop}, B : \mathsf{Prop}, p : A \wedge B \;\vdash\; pA(\lambda x\!:\!A.\,\lambda y\!:\!B.\,x) : A$$

$$\frac{A \wedge B}{B}\ (\wedge\mathsf{E}_2) \qquad A : \mathsf{Prop}, B : \mathsf{Prop}, p : A \wedge B \;\vdash\; pB(\lambda x\!:\!A.\,\lambda y\!:\!B.\,y) : B$$

Introduction rule

$$\frac{A \quad B}{A \wedge B}\ (\wedge\mathsf{I}) \qquad A : \mathsf{Prop}, B : \mathsf{Prop}, a : A, b : B \;\vdash\; (\lambda\alpha\!:\!\mathsf{Prop}.\,\lambda p\!:\!(A \to B \to \alpha).\,pab) : A \wedge B$$

Examples (cont.)

Note that $A : \mathsf{Prop}, B : \mathsf{Prop} \;\vdash\; A \wedge B : \mathsf{Prop}$ can be derived in λPROP2,

but the term $\mathsf{AND} \;\equiv\; \lambda A\!:\!\mathsf{Prop}.\,\lambda B\!:\!\mathsf{Prop}.\,A \wedge B$ cannot.

One has to be in λPROPω to derive $\vdash\; \mathsf{AND} : \mathsf{Prop} \to \mathsf{Prop} \to \mathsf{Prop}$

ex falso sequitur quodlibet

$$\frac{\bot}{A}\ (\textit{ex falso}) \qquad A : \mathsf{Prop}, p : \Pi\alpha\!:\!\mathsf{Prop}.\alpha \;\vdash\; pA : A$$

Examples (cont.)

Let us now prove reflexivity and symmetry for the Leibniz equality. Remember that for X : Set, x, $y : X$

$$(x =_L y) \;\equiv\; \Pi P\!:\!X \to \mathsf{Prop}.\, Px \to Py$$

Reflexivity $\quad X : \mathsf{Set}, x : X \;\vdash\; \underbrace{(\lambda P\!:\!X \to \mathsf{Prop}.\, \lambda q\!:\!Px.\, q)}_{\mathsf{w}} : (x =_L x)$

Symmetry

Let $\quad \Gamma \equiv X : \mathsf{Set}, x : X, y : X, t : (x =_L y)$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash t : (x =_L y) \quad \Gamma \vdash (\lambda z\!:\!X.\, z =_L x) : X \to \mathsf{Prop}}{\Gamma \vdash t(\lambda z\!:\!X.\, z =_L x) : (\lambda z\!:\!X.\, z =_L x)x \to (\lambda z\!:\!X.\, z =_L x)y} \;\vdots}{\Gamma \vdash t(\lambda z\!:\!X.\, z =_L x) : (x =_L x) \to (y =_L x)}\,(=_\beta) \qquad \Gamma \vdash \mathsf{w} : (x =_L x)}{\Gamma \vdash t(\lambda z\!:\!X.\, z =_L x)\mathsf{w} : (y =_L x)}$$

So,

$$X : \mathsf{Set}, x : X, y : X, t : (x =_L y) \;\vdash\; t(\lambda z\!:\!X.\, z =_L x)(\lambda P\!:\!X \to \mathsf{Prop}.\, \lambda q\!:\!Px.\, q) : (y =_L x)$$

Exercices

- Check the soundness of intuitionistic elimination and introduction rules for the other logic connectives.
- Check that the Leibniz equality is transitive.