Introduction to Type Theory — Herman Geuvers

Lecture 4: Calculus of Constructions and Homotopy Type Theory
OPLSS 2026

1 The Barendregt cube

As in λP, $*$ classifies types and $\Box$ classifies $*$. Thus $A : *$ means that A is a type, while $A : \Box$ means that A is a kind or universe-level object.

The Barendregt cube organizes eight typed lambda calculi by varying which dependencies are allowed in product formation. It is also called the lambda cube. Starting from ordinary functions, the three axes add:

- types depending on terms, $(*, \Box)$, which gives dependent types;
- terms depending on types, $(\Box, *)$, which gives polymorphism;
- types depending on types, $(\Box, \Box)$, which gives type operators.

The general product rule has the form

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

with sorts $*$ and $\Box$. The four possible dependencies are:

$(*, *)$	terms may depend on terms, ordinary functions
$(\Box, *)$	terms may depend on types, polymorphism
$(*, \Box)$	types may depend on terms, dependent types
$(\Box, \Box)$	types may depend on types, type operators

The simply typed calculus $\lambda^{\to}$ has only $(*, *)$; System F adds $(\Box, *)$; λP adds $(*, \Box)$; System F_ω adds $(\Box, \Box)$; and the Calculus of Constructions has all four. Berardi and Terlouw generalized this presentation into pure type systems.

The kind-forming corner $* \to *$ is visible in typed functional programming as type constructors such as `List : Type -> Type` (historically written `List : * -> *` in Haskell presentations). A term-level function such as $A \to \mathit{List}\, A$ still lives at the ordinary $*$ level once A is fixed.

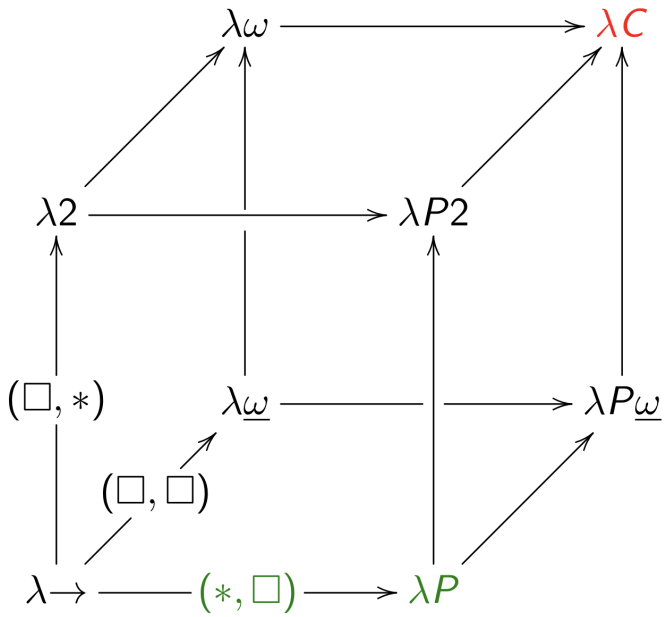

Figure 1: The lambda cube. The Calculus of Constructions is the corner with all three dependency axes.

2 Calculus of Constructions

The Calculus of Constructions (CC), introduced by Coquand and Huet, is the top corner of the cube. It combines polymorphism, dependent types, higher-order type operators, and a formulas-as-types interpretation of higher-order logic. In CC one can express polymorphic data encodings such as

$$\Pi\alpha : *.\, \alpha \to (\alpha \to \alpha) \to \alpha : *,$$

predicate domains such as $\mathbb{N} \to \mathbb{N} \to * : \Box$, propositional connectives such as

$$\varphi \wedge \psi := \Pi\alpha : *.\, (\varphi \to \psi \to \alpha) \to \alpha,$$

and higher-order quantification such as

$$\Pi P : \mathbb{N} \to *.\, \Pi x : \mathbb{N}.\, P\, x \to P\, x : *.$$

Induction can be written as a higher-order proposition,

$$\forall P : \mathbb{N} \to *.\, P\, 0 \to (\forall x : \mathbb{N}.\, P\, x \to P(Sx)) \to \forall x : \mathbb{N}.\, P\, x,$$

but in pure CC it is not derivable as a general principle. Algebraic data types and iterators can be encoded, but the system does not provide primitive recursion or inductive types as basic constructs. Rocq therefore extends this core with inductive types and structurally recursive definitions.

CC has the expected metatheoretic properties: uniqueness of types up to beta-conversion, subject reduction, and strong normalization. For decision problems, inhabitation/type inhabitation is undecidable, while type checking and type synthesis are decidable together, using an algorithm close to the one for λP.

The conservativity issue concerns the embedding of higher-order logic into CC. Without separating propositions from computational types, HOL axioms about all propositions can accidentally apply to types. For example, propositional extensionality may imply $\mathbb{N} = \mathbb{N} \to \mathbb{N}$, which would make $\mathbb{N}$ a lambda-model and force all functions on $\mathbb{N}$ to have fixed points. Rocq avoids this by distinguishing Prop from Set/Type.

3 Higher-order logic in CC

Church's higher-order logic quantifies over domains such as D, $D \to \mathrm{Prop}$, $(D \to \mathrm{Prop}) \to \mathrm{Prop}$, and function spaces such as $(D \to D) \to D$. In a constructive type-theoretic setting, many connectives can be encoded by their eliminators:

$$\bot := \forall \alpha : *.\, \alpha, \qquad \varphi \vee \psi := \forall \alpha : *.\, (\varphi \to \alpha) \to (\psi \to \alpha) \to \alpha,$$

$$\exists x : A.\, \varphi := \forall \alpha : *.\, (\forall x : A.\, \varphi \to \alpha) \to \alpha.$$

Leibniz equality is also definable:

$$t =_A q := \forall P : A \to *.\, P\, t \to P\, q.$$

Reflexivity and transitivity are direct; symmetry follows by choosing a predicate that mentions equality to the left endpoint.

As an example of higher-order definition, the transitive closure of a relation $R : A \to A \to *$ can be expressed as the intersection of all transitive relations containing R:

$$\mathsf{Trclos}\, R := \lambda x\, y : A.\, \forall Q : A \to A \to *.\, \mathsf{Trans}\, Q \to (R \subseteq Q) \to Q\, x\, y.$$

Rocq often permits both this higher-order encoding and an inductive definition. The standard library generally favors inductive definitions because their induction and recursion principles are computationally explicit.

Church's 1940 paper is "A Formulation of the Simple Theory of Types." The system is usually described as simple type theory, historically a simpler alternative to Russell and Whitehead's ramified theory of types rather than merely an extension of it.

4 Extended Calculus of Constructions

The Extended Calculus of Constructions adds universe levels, cumulativity, and dependent sums. Cumulativity arranges universes in a hierarchy

$$* \subseteq \Box_0 \subseteq \Box_1 \subseteq \cdots,$$

so a type in a lower universe may be used in a higher one. Dependent sums are formed at the $*$ level and at universe levels:

$$\frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Sigma x : A.B : *},$$

$$\frac{\Gamma \vdash A : \Box_i \qquad \Gamma, x : A \vdash B : \Box_j}{\Gamma \vdash \Sigma x : A.B : \Box_{\max(i,j)}}.$$

For $\varphi : *$, one has $\Pi A : \Box_i.\varphi : *$, but not $\Sigma A : \Box_i.\varphi : *$. Rocq further refines the universe structure with Set, Prop, and rules for their interaction.

5 Identity types and paths

Homotopy type theory starts from the intensional identity type. In Rocq-style notation:

$$\begin{aligned}&\mathsf{Inductive}\ \mathit{identity}\ (A : \mathrm{Type}) : A \to A \to \mathrm{Type} :=\\ &\quad \mathsf{refl} : \forall a : A,\ a = a.\end{aligned}$$

The eliminator is the J-rule:

$$\frac{P : \forall a\, b : A.\, a = b \to \mathrm{Prop} \qquad r : \forall a : A.\, P\, a\, a\, \mathsf{refl}}{J\, r : \forall x\, y : A.\, \forall i : x = y.\, P\, x\, y\, i},$$

with computation $J\, r\, a\, a\, \mathsf{refl} \to r\, a$.

The J-rule gives transport: from $t : Q\, a$ and $p : a = b$, one obtains a transported term $p_* t : Q\, b$. This transported term is not definitionally the same as t; intensional type theory keeps such information explicit to preserve decidable type checking. The J-rule also gives symmetry and transitivity of identity, commonly written p^{-1} and $p \cdot q$.

What J does not give is function extensionality, proof irrelevance, or uniqueness of identity proofs (UIP). UIP is equivalent to the K-rule, which says every self-identity proof $q : a = a$ is equal to refl. Hofmann and Streicher's groupoid interpretation showed that K and UIP are not consequences of ordinary intensional type theory.

6 Homotopy type theory

The homotopical reading interprets a type as a space, a term as a point, and a proof $p : a = b$ as a path from a to b. A proof $h : p = q$ is then a path between paths, or a homotopy. This makes the algebra of equality proofs look like the algebra of a groupoid: paths compose, invert, and satisfy identity and associativity laws up to higher equality.

Some types still have collapsed equality structure. A type A is an h-proposition when all its terms are equal:

$$\Pi x\, y : A.\, x = y.$$

It is an h-set when all equality proofs between its terms are equal:

$$\Pi x\, y : A.\, \Pi p\, q : x = y.\, p = q.$$

Ordinary data types such as natural numbers and booleans are h-sets, even though general types need not satisfy UIP.

The fundamental group of the HoTT circle is $\mathbb{Z}$. A higher sphere is not just another circle; as a higher inductive type, S^1 has a point and a loop, S^2 has a point and a two-dimensional path, and higher spheres can be presented by higher path constructors or recursively as suspensions.

7 Univalence and higher inductive types

Voevodsky's univalence axiom identifies equality of types with equivalence of types:

$$(A = B) \simeq (A \simeq B).$$

This validates the structure identity principle: equivalent structures can be transported across as equal structures. Univalence also implies function extensionality. A central challenge is to give univalence a computational interpretation; cubical type theories are one important response to that challenge.

Higher inductive types extend ordinary inductive types with path constructors. The circle is generated by a point and a loop:

$$\begin{aligned} &\mathsf{Inductive}\ S^1 : \mathrm{Type} := \\ &\quad base : S^1 \\ &\quad loop : base = base. \end{aligned}$$

The torus adds two loops and a surface equating the two orders of composition. HITs can also define quotient-like data structures. For Kuratowski finite sets $K(A)$, constructors include empty set, singleton, union, laws for union, and a truncation constructor forcing the result to be an h-set. For listed finite sets $L(A)$, constructors include nil, cons, duplicate removal, commutation, and truncation. One can prove $K(A) \simeq L(A)$; by univalence this equivalence becomes an equality, so definitions and proofs can be transported between the two presentations.

Voevodsky's motivation for formal foundations is often tied to errors and uncertainty in highly abstract mathematics, including flaws in work around motivic cohomology and higher-groupoid ideas. These experiences pushed him toward computer-checked foundations and ultimately univalent foundations.

References

[1] H. P. Barendregt. *Introduction to generalized type systems*. Journal of Functional Programming, 1991. https://doi.org/10.1017/S0956796800020025

[2] T. Coquand and G. Huet. *The calculus of constructions*. Information and Computation 76(2–3):95–120, 1988. https://doi.org/10.1016/0890-5401(88)90005-3

[3] A. Church. *A formulation of the simple theory of types*. Journal of Symbolic Logic 5(2):56–68, 1940. https://doi.org/10.2307/2266170

[4] M. Hofmann and T. Streicher. *The groupoid interpretation of type theory*. In *Twenty-five years of constructive type theory*, 1998.

[5] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. https://homotopytypetheory.org/book/

[6] V. Voevodsky. *The Origins and Motivations of Univalent Foundations*. Institute for Advanced Study, 2014. https://www.ias.edu/ideas/2014/voevodsky-origins