The Calculus of Inductive Constructions

Hugo Herbelin

10th Oregon Programming Languages Summer School

Eugene, Oregon, June 16-July 1, 2011

Outline

- A bit of history, leading to the Calculus of Inductive Constructions (CIC)
- The first ingredient of the CIC: a Pure Type System with subtyping
- The second ingredient of the CIC: Martin-Löf-style inductive definitions
- The logical strength of the CIC, compared to set theory
- More advanced topics

A few steps of the history of logic

(from formal logic to proof theory)

Boole's *laws of thought* (1854): an algebraic description of propositional reasoning (followed by Peirce, Schröder ...)

Frege's *Begriffsschrift* (1879): formal quantifiers + formal system of proofs (including an axiomatization of Cantor's *naive set theory*)

Peano's *arithmetic* (1889): formal arithmetic on top of Peirce-Schröder "predicate calculus"

Zermelo (1908), Fraenkel (1922): the stabilization of *set theory* as known today (ZF)

Russell and Whitehead's *Principia Mathematica* (1910): type theory as a foundation of mathematics alternative to set theory

Skolem's *Primitive Recursive Arithmetic* (1923): the (quantifier-free) logic of primitive recursive functions (the logic of metamathematics)

Brouwer's *intuitionism* (1923): the view that proofs are "computation methods" + rejection of excluded-middle (classical logic) because not effective as a computation method (followed by Heyting, Kolmogorov, ...)

A few steps of the history of logic

(from formal logic to proof theory)

Gödel's *incompleteness of arithmetic* (1931): consistency cannot be "proved"

Church's λ-*calculus* (1932): a function-based model of computation

Gentzen's *Hauptsatz* (1935, 1936): natural deduction + sequent calculus + consistency of arithmetic (consistency = termination of cut-elimination by induction up to ϵ_0)

Church's *simple theory of types* (1940): higher-order logic

Kleene's *realizability* (1945): extracting programs from intuitionistic proofs (followed by Kreisel)

Gödel's *functional interpretation (Dialectica)* (1958): characterization of the provably total functions of first-order arithmetic (system T)

Prawitz's *normalization* for natural deduction (1965)

Suggested readings: van Heijenoort, From Frege to Gödel + *googling*

A few bits of the history of logic

(towards the genesis of the Coq proof assistant)

Curry's and Howard's *proof-as-program correspondence* (1958, 1969): formal systems of intuitionistic proofs are *structurally identical* to typing systems for programs

Martin-Löf's (extensional) *Intuitionistic Type Theory* (1975): taking the proofs-as-programs correspondence as foundational: a constructive formalism of inductive definitions that is both a logic and a richly-typed functional programming language

Girard and Reynolds' *System F* (1971): characterization of the provably total functions of second-order arithmetic

Coquand's *Calculus of Constructions* (1984): extending system F into an hybrid formalism for both proofs and programs (consistency = termination of evaluation)

Coquand and Huet's implementation of the *Calculus of Constructions (CoC)* (1985)

Coquand and Paulin-Mohring's *Calculus of Inductive Constructions* (1988): mixing the Calculus of Constructions and Intuitionistic Type Theory leading to a new version of CoC called Coq

Coq 8.0 switched to the *Set-Predicative Calculus of Inductive Constructions* (2004): to be compatible with classical choice

Proof assistants: a panel of formalisms

(set theory based)	B tool	Mizar	
(type theory based)	HOL	MLTT	CIC

- CIC = a predicative hierarchy of functional types on top of a propositional System F + dependent proofs + inductive types at all types

- how does set theory compare with CIC? (collection of arbitrary subsets of a big untyped universe vs stratified collections of stand-alone types)

Pure Type Systems

A few elements of the history of pure type systems

De Bruijn's `Automath` systems (1968)

Girard and Reynolds' System F, Girard's F_ω, U^-, U (1970)

Martin-Löf's "$\mathtt{Type} : \mathtt{Type}$" (1971)

Coquand's *Calculus of Constructions* (1984)

Harper-Honsell-Plotkin's *Edinburgh Logical Framework* (1987)

Luo's Extended Calculus of Constructions (ECC) (1989): extension with subtyping

Barendregt's λ-cube (1989)

Berardi's and Terlouw's *~~Generalized~~ Pure Type Systems* (1990): generalizing the cube

+ Pollack, Jutting, Geuvers, McKinna, Barras, Werner, Dowek, Huet, Barthe, Adams, Siles, and many others ...

From λ-calculus, System F, ... to Pure Type Systems

- Generalize simply-typed λ-calculus...

$$\begin{array}{lll} \textit{terms} \;\; M, N & ::= & x \mid \lambda x : T.M \mid M\, N \\ \textit{types} \;\; T, U & ::= & X \mid T \to U \end{array}$$

... and System F ...

$$\begin{array}{lll} \textit{terms} \;\; M, N & ::= & x \mid \lambda x : T.M \mid M\, N \mid \lambda X : \texttt{Prop}.M \mid M\, T \\ \textit{types} \;\; T, U & ::= & X \mid T \to U \mid \forall X : \texttt{Prop}.T \end{array}$$

From λ-calculus, System F, ... to Pure Type Systems

- Generalize simply-typed λ-calculus...

$$\begin{array}{lll} \textit{terms} \;\; M, N & ::= & x \mid \lambda x : T.M \mid M\,N \\ \textit{types} \;\; T, U & ::= & X \mid T \to U \end{array}$$

... and System F ...

$$\begin{array}{lll} \textit{terms} \;\; M, N & ::= & x \mid \lambda x : T.M \mid M\,N \mid \lambda X : K.M \mid M\,T \\ \textit{types} \;\; T, U & ::= & X \mid T \to U \mid \forall X : K.T \\ \textit{kinds} \;\; K & ::= & \texttt{Prop} \end{array}$$

... and Girard's System F_ω $(= \lambda_{HOL})$ ...

$$\begin{array}{lll} \textit{terms} & M, N & ::= x \mid \lambda x : T.M \mid M\,N \mid \lambda X : \texttt{K}.M \mid M\,T \\ \textit{type constructors} & T, U, F, G & ::= X \mid T \to U \mid \forall X : K.T \mid \lambda X : K.F \mid F\,G \\ \textit{kinds} & K & ::= \texttt{Prop} \mid K \to K \end{array}$$

that now comes, as in Church's HOL, with a conversion rule:

$$\frac{\Gamma \vdash M : T \quad T =_\beta U}{\Gamma \vdash M : U}$$

... the type constructors level is a mono-sorted simply-typed λ-calculus

Exercise: How strong is this system, logically speaking? Show that its consistency can be shown by purely arithmetic means

Pure Type Systems

The previous construction can be generalized by considering a uniform notion of dependent arrow (a.k.a. dependent product, or Π-type) and sorts of a set $\mathcal{S}$ for classifying types:

$$\begin{array}{rcl} M, N, T, U & ::= & x \mid \lambda x : T.M \mid M\ N \\ & \mid & \forall x : T.U \\ & \mid & s \end{array}$$

($\forall x : T.U$ is written $T \to U$ when $x \notin U$; s ranges over $\mathcal{S}$)

In addition to the set of sorts, parametrization is given by a set of *axioms* $\mathcal{A}$ for typing sorts:

$$\frac{s_1 : s_2 \in \mathcal{A}}{\vdash s_1 : s_2}$$

and a set of *rules* $\mathcal{R}$ for typing products

$$\frac{\Gamma \vdash T : s_1 \qquad \Gamma, x : T \vdash U : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \forall x : T.U : s_3}$$

And still the conversion rule:

$$\frac{\Gamma \vdash M : T \qquad \Gamma \vdash U \qquad T =_\beta U}{\Gamma \vdash M : U}$$

The other rules introduce and eliminate the product type:

$$\frac{\Gamma, x : T \vdash M : U \qquad \Gamma \vdash \forall x : T.U : s}{\Gamma \vdash \lambda x : T.M : \forall x : T.U}$$

$$\frac{\Gamma \vdash M : \forall x : T.U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U[N/x]}$$

+ axiom rule + weakening rule

Exercise: What are $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ in the case of simply-typed λ-calculus, System F, System F_ω as PTSs.

System U^-: two levels of polymorphisms

Replace the mono-sorted simply-typed λ-calculus with a new copy of System F:

terms	M, N	$::=$	$x \mid \lambda x : T.M \mid M\,N \mid \lambda X : K.M \mid M\,T$
type constructors	T, U, F, G	$::=$	$X \mid T \rightarrow U \mid \forall X : K.T \mid \lambda X : K.F \mid F\,G \mid \lambda \mathcal{X} : \mathtt{Type}.F \mid F\,K$
kinds	K	$::=$	$\mathtt{Prop} \mid \mathcal{X} \mid K \rightarrow K \mid \forall \mathcal{X} : \mathtt{Type}.\,K$

... this is inconsistent (Girard's adaptation of Burali-Forti's paradox, Miquel's adaptation of Russell's paradox, Coquand's exploitation of Reynolds' polymorphism-is-not-set-theoretic result, Hurkens' paradox)!

(and a good tool to know what to avoid to turn CIC into an inconsistent system)

Exercise: Describe the above as a Pure Type System (what are $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$?)

A predicative extension of System F_ω

Replace the mono-sorted simply-typed λ-calculus with a polymorphic but predicative λ-calculus:

terms	M, N	$::=$	$x \mid \lambda x : T.M \mid M\,N \mid \lambda X : K.M \mid M\,T$
type constructors	T, U, F, G	$::=$	$X \mid T \to U \mid \forall X : K.T \mid \lambda X : K.F \mid F\,G \mid \lambda \mathcal{X} : \mathtt{Type}.F \mid F\,K$
level 1 kinds	K	$::=$	$\mathtt{Prop} \mid \mathcal{X} \mid K \to K$
level 2 kinds	K_2	$::=$	$\mathtt{Type} \mid \forall \mathcal{X} : \mathtt{Type}.\,K$

Exercise: Describe the above as a Pure Type Systems (what are $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$?)

Generalizing the second-level of polymorphic to a polymorphic over higher-order levels: $F_{\omega.2}$

terms	M, N	$::=$	$x \mid \lambda x : T.M \mid M\,N \mid \lambda X : K.M \mid M\,T \mid \lambda \mathcal{X} : K_2.M \mid M\,K$
type constructors	T, U, F, G	$::=$	$X \mid T \to U \mid \forall X : K.T \mid \lambda X : K.F \mid F\,G \mid \lambda \mathcal{X} : K_2.F \mid F\,K \mid \forall \mathcal{X} : K_2.T$
level 1 kinds	K, P, Q	$::=$	$\texttt{Prop} \mid \mathcal{X} \mid \forall X : K.K \mid \lambda X : K.P \mid P\,T \mid \lambda \mathcal{X} : K_2.P \mid P\,Q$
level 2 kinds	K_2	$::=$	$\texttt{Type} \mid \forall \mathcal{X} : K_2.\,K \mid \forall X : K.\,K_2 \mid \forall \mathcal{X} : K_2.\,K_2$

Exercise: Describe the above as a Pure Type Systems (what are $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$?)

Adding variables at level 2 kinds: λ_Z

terms	M, N	$::=$	$x \mid \lambda x : T.M \mid M\,N \mid \lambda X : K.M \mid M\,T \mid \lambda\mathcal{X} : K_2.M \mid M\,K \mid \lambda\mathcal{X}_2 : K_3.M \mid M\,K_2$
type cstr.	T, U, F, G	$::=$	$X \mid T \to U \mid \forall X : K.T \mid \lambda X : K.F \mid F\,G \mid \lambda\mathcal{X} : K_2.F \mid F\,K \mid \forall\mathcal{X} : K_2.T \mid \forall\mathcal{X}_2 : K_3.T$
level 1 kinds	K, P, Q	$::=$	$\texttt{Prop} \mid \mathcal{X} \mid \forall X : K.K \mid \lambda X : K.P \mid P\,T \mid \lambda\mathcal{X} : K_2.P \mid P\,Q$
level 2 kinds	K_2	$::=$	$\texttt{Type} \mid \mathcal{X}_2 \mid \forall\mathcal{X} : K_2\,K \mid \forall X : K\,K_2 \mid \forall\mathcal{X} : K_2\,K_2$
level 3 kinds	K_3	$::=$	$\texttt{Kind}$

Theorem (Miquel, 2001): λ_Z is equiconsistent with Zermelo's set theory

Note: cardinal strength is $V_{\omega.2}$ (ω iterations of the power-set from the natural numbers)

Adding a hierarchy of universes: F_{ω^2}

terms	M, N	$::=$	$x \mid \lambda x : T.M \mid M\,N \mid \lambda X_{n-1} : K_n.M \mid M\,F_{n-1}$
type constructors	T, U, F_0	$::=$	$X_0 \mid T \to U \mid \forall X_{n-1} : K_n.T \mid \lambda X_{n-1} : K_n.F_0 \mid F_0\,F_{n-1}$
level 1 kinds	K_1, F_1	$::=$	$\mathtt{Prop} \mid X_1 \mid \forall X_0 : K_1.\,K_1 \mid \lambda X_{n-1} : K_n.F_1 \mid F_1\,F_{n-1}$
level 2 kinds	K_2, F_2	$::=$	$\mathtt{Type}_1 \mid X_2 \mid \forall X_{i-1} : K_{i\leq 2}.\,K_{j\leq 2} \mid \lambda X_{n-1} : K_n.F_2 \mid F_2\,F_{n-1}$
		$\vdots$	
level n+1 kinds	K_{n+1}, F_{n+1}	$::=$	$\mathtt{Type_n} \mid X_{n+1} \mid \forall X_{i-1} : K_{i\leq n+1}.\,K_{j\leq n+1} \mid \lambda X_{p-1} : K_p.F_{n+1} \mid F_{n+1}\,F_{p-1}$
		$\vdots$	

Further readings: H. Barendregt, H. Geuvers, A. Miquel

Adding dependencies

Last step: let proofs be dependent in types!

$$
\begin{array}{llcl}
\textit{terms} & M, N & ::= & \ldots \\
\textit{type constructors} & T, U, F_0 & ::= & \ldots \mid \lambda X : T.F_0 \mid F_0\, M \\
\textit{level 1 kinds} & K_1, F_1 & ::= & \ldots \mid \forall x : T.K \mid \lambda x : T.K \mid K\, M \\
\textit{level 2 kinds} & K_2, F_2 & ::= & \ldots \mid \forall x : T.K_2 \mid \lambda x : T.F_2 \mid F_2\, M \\
 & & \vdots & \\
\textit{level n+1 kinds} & K_{n+1}, F_{n+1} & ::= & \ldots \mid \forall x : T.K_{n+1} \mid \lambda x : T.F_{n+1} \mid F_{n+1}\, M \\
 & & \vdots &
\end{array}
$$

This adds nothing to the logical expressiveness but this allows for example to form subset types:

$$\forall C : \mathtt{Type}.(\forall a : X, P(a) \to C) \to C$$

informally $\{a : A | P(a)\}$

The (original) Calculus of Constructions

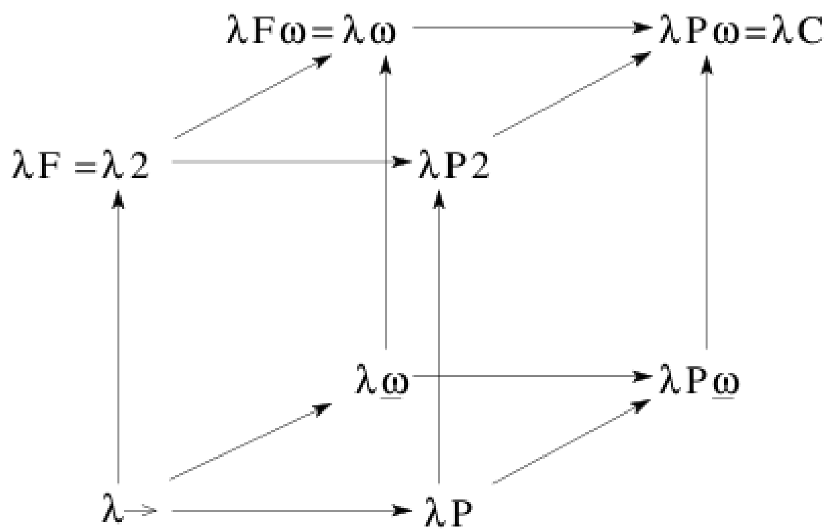
That is F_ω (one level) extended with proof dependencies

$$
\begin{array}{lll}
\textit{terms} & M, N & ::= x \mid \lambda x : T.M \mid M\,N \mid \lambda X : K.M \mid M\,F \\
\textit{type constructors} & T, U, F, G & ::= X \mid \forall x : T.U \mid \forall X : K.T \mid \lambda X : K.F \mid F\,G \mid \lambda x : T.F \mid F\,N \\
\textit{kinds} & K, P & ::= \mathtt{Prop} \mid \forall X : K.\,K \mid \forall x : T.\,K
\end{array}
$$

Note that now, the product of T over U might be dependent and has to be written $\forall x : T.U$

Barendregt's cube

$\mathcal{S} = \{\texttt{Prop}, \texttt{Type}\}$
$\mathcal{A} = \{(\texttt{Prop} : \texttt{Type})\}$



$\lambda \rightarrow$	=	simply-typed λ-calculus	$\mathcal{R}_{st} = \{\texttt{Prop}, \texttt{Prop}, \texttt{Prop}\}$
λP	=	LF	$\mathcal{R}_{LF} = \mathcal{R}_{st} \cup \{\texttt{Prop}, \texttt{Type}, \texttt{Type}\}$
λF	=	System F	$\mathcal{R}_{F} = \mathcal{R}_{st} \cup \{\texttt{Type}, \texttt{Prop}, \texttt{Prop}\}$
$\lambda F\omega$	=	System F_ω	$\mathcal{R}_{F_\omega} = \mathcal{R}_{F} \cup \{\texttt{Type}, \texttt{Type}, \texttt{Type}\}$
λC	=	Calculus of Constructions	$\mathcal{R}_{CC} = \mathcal{R}_{F_\omega} \cup \mathcal{R}_{LF}$

Back to the non-inductive part of Coq

Adding `Set`: an alias for $\texttt{Type}_0$

Since Coq 8.0, `Set` behaves like a `Type` except that it does not contain `Prop`.

Before, `Set` was a copy, impredicative, of `Prop`

The distinction between `Prop` and `Set` was motivated by extraction (realizability)

In practice, it also emphasizes the difference of intended meaning: `Prop` is thought as "proof-irrelevant", `Set` is thought as "computationally-relevant"

Adding universes subtyping: CC_ω

One wants that if $\vdash M : \mathtt{Type}_n$ then $\vdash M : \mathtt{Type}_m$ for all $m > n$

The naive way: add all rules $(\mathtt{Type}_n, \mathtt{Type}_p, \mathtt{Type}_q)$ for $q \geq max(n,p)$

$\hookrightarrow$ it does not work

The less naive way: goes out PTSs and explicitly add inference rules $\dfrac{\Gamma \vdash M : \mathtt{Type}_n}{\Gamma \vdash M : \mathtt{Type}_{n+p}}$

$\hookrightarrow$ this lacks uniformity ...

The good solution is to replace conversion by subtyping:

$$\frac{T =_\beta T' \qquad U \leq_\beta U'}{\forall x : T.U \leq_\beta \forall x : T'.U'} \qquad \frac{p \leq n}{\mathtt{Type}_p \leq_\beta \mathtt{Type}_n} \qquad \frac{}{\mathtt{Prop} \leq_\beta \mathtt{Type}_n}$$

$$\frac{T \leq_\beta U \qquad T =_\beta T' \qquad U =_\beta U'}{T' \leq_\beta U'}$$

Miscellaneous issues?

- η-conversion
- judgmental equality vs untyped conversion
- terminology: functional, full, semi-full, injective
- syntax-directed presentation and expansion postponement
- basic metatheory