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 *\(\lambda\)-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 \(\varepsilon_0\))

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 $F_{\omega}$, $U^{-}, U$ (1970)
Martin-Löf’s “Type : Type” (1971)
Coquand’s *Calculus of Constructions* (1984)
Luo’s Extended Calculus of Constructions (ECC) (1989): extension with subtyping
Barendregt’s $\lambda$-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 $\lambda$-calculus, System $F$, ... to Pure Type Systems

- Generalize simply-typed $\lambda$-calculus...

  \[
  \begin{align*}
  \text{terms } M, N & ::= x \mid \lambda x : T.M \mid M N \\
  \text{types } T, U & ::= X \mid T \rightarrow U
  \end{align*}
  \]

... and System $F$ ...

  \[
  \begin{align*}
  \text{terms } M, N & ::= x \mid \lambda x : T.M \mid M N \mid \lambda X : \text{Prop}.M \mid M T \\
  \text{types } T, U & ::= X \mid T \rightarrow U \mid \forall X : \text{Prop}.T
  \end{align*}
  \]
From $\lambda$-calculus, System $F$, ... to Pure Type Systems

- Generalize simply-typed $\lambda$-calculus...

$$\textit{terms} \; M, N :: = \; x \mid \lambda x : T . M \mid M \; N$$

$$\textit{types} \; T, U :: = \; X \mid T \rightarrow U$$

... and System $F$ ...

$$\textit{terms} \; M, N :: = \; x \mid \lambda x : T . M \mid M \; N \mid \lambda X : K . M \mid M \; T$$

$$\textit{types} \; T, U :: = \; X \mid T \rightarrow U \mid \forall X : K . T$$

$$\textit{kinds} \; K :: = \; \text{Prop}$$

... and Girard’s System $F_\omega \ (= \lambda_{\text{HOL}}) ...$

$$\textit{terms} \; M, N :: = \; x \mid \lambda x : T . M \mid M \; N \mid \lambda X : K . M \mid M \; T$$

$$\textit{type constructors} \; T, U, F, G :: = \; X \mid T \rightarrow U \mid \forall X : K . T \mid \lambda X : K . F \mid F \; G$$

$$\textit{kinds} \; K :: = \; \text{Prop} \mid K \rightarrow K$$

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

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

$$\Gamma \vdash M : U$$

... the type constructors level is a mono-sorted simply-typed $\lambda$-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 \( \Pi \)-type) and sorts of a set \( S \) for classifying types:

\[
M, N, T, U ::= x \mid \lambda x : T. M \mid M N \\
| \forall x : T. U \\
| s
\]

(\( \forall x : T. U \) is written \( T \rightarrow U \) when \( x \notin U \); \( s \) ranges over \( S \))

In addition to the set of sorts, parametrization is given by a set of axioms \( A \) for typing sorts:

\[
\frac{s_1 : s_2 \in A}{\vdash s_1 : s_2}
\]

and a set of rules \( R \) for typing products

\[
\frac{\Gamma \vdash T : s_1 \quad \Gamma, x : T \vdash U : s_2 \quad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \forall x : T. U : s_3}
\]
And still the conversion rule:

\[
\Gamma \vdash M : T \quad \Gamma \vdash U \quad T =_\beta U
\]

\[
\Gamma \vdash M : U
\]

The other rules introduce and eliminate the product type:

\[
\Gamma, x : T \vdash M : U \quad \Gamma \vdash \forall x : T.U : s
\]

\[
\Gamma \vdash \lambda x : T.M : \forall x : T.U
\]

\[
\Gamma \vdash M : \forall x : T.U \quad \Gamma \vdash N : T
\]

\[
\Gamma \vdash MN : U[N/x]
\]

+ axiom rule + weakening rule

Exercise: What are \( S \), \( A \) and \( R \) in the case of simply-typed \( \lambda \)-calculus, System \( F \), System \( F_\omega \) as PTSs.
System $U^-$: two levels of polymorphisms

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

\[
\begin{align*}
\text{terms} & \quad M, N ::= x \mid \lambda x : T.M \mid M \, N \mid \lambda X : K.M \mid M \, T \\
\text{type constructors} & \quad T, U, F, G ::= X \mid T \rightarrow U \mid \forall X : K.T \mid \lambda X : K.F \mid F \, G \mid \lambda X : \text{Type}.F \mid F \, K \\
\text{kinds} & \quad K ::= \text{Prop} \mid \mathcal{X} \mid K \rightarrow K \mid \forall \mathcal{X} : \text{Type}.K
\end{align*}
\]

... 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 $F_\omega$

Replace the mono-sorted simply-typed $\lambda$-calculus with a polymorphic but predicative $\lambda$-calculus:

**terms**

\[ M, N ::= x | \lambda x : T. M | M N | \lambda X : K. M | M T \]

**type constructors**

\[ T, U, F, G ::= X | T \to U | \forall X : K. T | \lambda X : K. F | F G | \lambda X : \text{Type}. F | F K \]

**level 1 kinds**

\[ K ::= \text{Prop} | \forall X : K. K \]

**level 2 kinds**

\[ K_2 ::= \text{Type} | \forall X : \text{Type}. K \]

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

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

terms

$M, N ::= x \mid \lambda x : T. M \mid M N \mid \lambda X : K. M \mid M T \mid \lambda X : K_2. M \mid M K$

type constructors

$T, U, F, G ::= X \mid T \rightarrow U \mid \forall X : K. T \mid \lambda X : K. F \mid F G \mid \lambda X : K_2. F \mid F K \mid \forall X : K_2. T$

level 1 kinds

$K, P, Q ::= \text{Prop} \mid X \mid \forall X : K. K \mid \lambda X : K. P \mid P T \mid \lambda X : K_2. P \mid P Q$

level 2 kinds

$K_2 ::= \text{Type} \mid \forall X : K_2. K \mid \forall X : K. K_2 \mid \forall X : K_2. K_2$

Exercise: Describe the above as a Pure Type Systems (what are $S$, $A$ and $R$?)
Adding variables at level 2 kinds: $\lambda_Z$

**Terms**

$$M, N ::= x | \lambda x : T . M | M \cdot N | \lambda X : K . M | M \cdot T | \lambda \alpha : K_2 . M | M K | \lambda \alpha_2 : K_3 . M | M K_2$$

**Type Constructors**

$$T, U, F, G ::= X \mid T \rightarrow U \mid \forall X : K . T \mid \lambda X : K . F \mid F G \mid \lambda \alpha : K_2 . F \mid F K \mid \forall \alpha : K_2 . T \mid \forall \alpha_2 : K_3 . T$$

**Level 1 Kinds**

$$K, P, Q ::= \text{Prop} \mid \mathcal{X} \mid \forall X : K . K \mid \lambda X : K . P \mid P T \mid \lambda \alpha : K_2 . P \mid P Q$$

**Level 2 Kinds**

$$K_2 ::= \text{Type} \mid \mathcal{X}_2 \mid \forall \alpha : K_2 K \mid \forall X : K K_2 \mid \forall \alpha : K_2 K_2$$

**Level 3 Kinds**

$$K_3 ::= \text{Kind}$$

*Theorem* (Miquel, 2001): $\lambda_Z$ is equiconsistent with Zermelo’s set theory

Note: cardinal strength is $V_{\omega,2}$ ($\omega$ iterations of the power-set from the natural numbers)
Adding a hierarchy of universes: $F_{\omega^2}$

**terms**

$$M, N ::= x | \lambda x : T.M | M N | \lambda X_{n-1} : K_n.M | M F_{n-1}$$

**type constructors**

$$T, U, F_0 ::= X_0 | T \rightarrow U | \forall X_{n-1} : K_n.T | \lambda X_{n-1} : K_n.F_0 | F_0 F_{n-1}$$

**level 1 kinds**

$$K_1, F_1 ::= \text{Prop} | X_1 | \forall X_0 : K_1.K_1 | \lambda X_{n-1} : K_n.F_1 | F_1 F_{n-1}$$

**level 2 kinds**

$$K_2, F_2 ::= \text{Type}_1 | X_2 | \forall X_{i-1} : K_i \leq 2.K_j \leq 2 | \lambda X_{n-1} : K_n.F_2 | F_2 F_{n-1}$$

... 

**level n+1 kinds**

$$K_{n+1}, F_{n+1} ::= \text{Type}_n | X_{n+1} | \forall X_{i-1} : K_i \leq n+1.K_j \leq n+1 | \lambda X_{p-1} : K_p.F_{n+1} | F_{n+1} F_{p-1}$$

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

Last step: let proofs be dependent in types!

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

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

\[
\forall C : \text{Type}.(\forall a : X, P(a) \rightarrow C) \rightarrow C
\]

informally \(\{a : A \mid P(a)\}\)
The (original) Calculus of Constructions

That is $F_\omega$ (one level) extended with proof dependencies

terms $M, N ::= x \mid \lambda x : T.M \mid M N \mid \lambda X : K.M \mid M F$

type constructors $T, U, F, G ::= X \mid \forall x : T.U \mid \forall X : K.T \mid \lambda X : K.F \mid F G \mid \lambda x : T.F \mid F N$

kinds $K, P ::= \text{Prop} \mid \forall X : K.K \mid \forall x : T.K$

Note that now, the product of $T$ over $U$ might be dependent and has to be written $\forall x : T.U$
$S = \{\text{Prop, Type}\}$
$A = \{(\text{Prop} : \text{Type})\}$

$\lambda \rightarrow = \text{simply-typed } \lambda\text{-calculus}$
$\lambda P = \text{LF}$
$\lambda F = \text{System } F$
$\lambda F \omega = \text{System } F_\omega$
$\lambda C = \text{Calculus of Constructions}$
Back to the non-inductive part of Coq
Adding Set: an alias for $\text{Type}_0$

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: $\text{CC}_\omega$

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

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

$\hookrightarrow$ it does not work

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

$\hookrightarrow$ this lacks uniformity ...

The good solution is to replace conversion by subtyping:

\[
\frac{T =_{\beta} T' \quad U \leq_{\beta} U'}{\forall x : T.U \leq_{\beta} \forall x : T'.U'} \quad \frac{p \leq n}{\text{Type}_p \leq_{\beta} \text{Type}_n} \quad \frac{\text{Prop} \leq_{\beta} \text{Type}_n}{T \leq_{\beta} U \quad T =_{\beta} T' \quad U =_{\beta} U'}{T' \leq_{\beta} U'}
\]
Miscellaneous issues?

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