IMPREDICATIVITY IN COQ

YOTAM DVIR

TEL-AVIV UNIVERSITY

2019-11-20

TODAY

1. What is Impredicativity
2. Coq Type System
3. Coq Live Demo
4. Justifying Predicativity

IMPREDICATIVITY

Commenting on impredicative developments of real-analysis:

> *[..] a field of possibilities open into infinity has been mistaken for a closed realm of things existing in themselves.* [Weyl, 1949]

IMPREDICATIVITY

A definition is *impredicative* if it generalizes over a totality which includes the very object being defined.

The set of all sets which are not members of themselves

Impredicative because a set is being defined in terms of the collection of all sets of which it is a member.
This impredicativity induces a vicious circle – Russell's paradox.

The least-upper bound of a given ordered set X

Impredicative as it is defined in terms of the set of the upper bounds of X, of which the lub is a member.

THE COQ TYPE SYSTEM

THE COQ TYPE SYSTEM

The Coq system is designed to develop mathematical proofs, and especially to write formal specifications, programs and to verify that programs are correct with respect to their specifications. [..]

THE COQ TYPE SYSTEM

The Coq system is designed to develop mathematical proofs, and especially to write formal specifications, programs and to verify that programs are correct with respect to their specifications. [..] Using the so-called Curry-Howard isomorphism, programs, properties and proofs are formalized in the same language called Calculus of Inductive Constructions, that is a λ*-calculus with a rich type system. [..]*

THE COQ TYPE SYSTEM

The Coq system is designed to develop mathematical proofs, and especially to write formal specifications, programs and to verify that programs are correct with respect to their specifications. [..] Using the so-called Curry-Howard isomorphism, programs, properties and proofs are formalized in the same language called Calculus of Inductive Constructions, that is a λ*-calculus with a rich type system. [..] The very heart of the Coq system is the type checking algorithm that checks the correctness of proofs, in other words that checks that a program complies to its specification.*

[Coq Reference Manual]

THE COQ TYPE SYSTEM

> *The Coq system is designed to develop mathematical proofs, and especially to write formal specifications, programs and to verify that programs are correct with respect to their specifications. [..] Using the so-called Curry-Howard isomorphism, programs, properties and proofs are formalized in the same language called Calculus of Inductive Constructions, that is a λ-calculus with a rich type system. [..] The very heart of the Coq system is the type checking algorithm that checks the correctness of proofs, in other words that checks that a program complies to its specification.*
>
> [Coq Reference Manual]

- The theory underlying Coq is quite complicated
- We will progress in stages towards it

λ-CALCULUS

Recall the λ-calculus – captures the idea of functions by rewriting

$$E[\ (\lambda x.M)N\ ] \mapsto_\beta E[\ M\{N/x\}\ ]$$

For $1 := (\lambda f.\lambda x.fx)$ and $\mathrm{t} := \lambda a.\lambda b.a$ we have

$$1\mathrm{t} \mapsto_\beta \lambda x.\mathrm{t}x \mapsto_\beta \lambda x.\lambda b.x =_\alpha \mathrm{t}$$

For $\Omega := \lambda x.xx$ we have $\Omega\Omega \mapsto_\beta \Omega\Omega$ (does not terminate)

Note the non-determinism of $\mapsto_\beta$:

$$\Omega 1\mathrm{t} \mapsto_\beta (11)\mathrm{t} \qquad \Omega 1\mathrm{t} \mapsto_\beta \Omega\lambda x.\mathrm{t}x$$

TYPING INFORMATION

1. Type systems are usually concerned with extending the λ-calculus with more terms and “type information”
2. Typing information is best thought of as specification

In the simply-typed λ-calculus (that we will see later)

$$M : (\sigma \to \tau) \to \sigma$$

means that M demands its input satisfy the spec $\sigma \to \tau$
& in return guarantees the output will satisfy the spec σ

GRASPING TYPES

> *Note that it is required neither that we should be able to generate somehow all objects of a given type nor that we should so to say know them all individually. It is only a question of understanding what it means to be an arbitrary object of the type in question.*
>
> [Martin-Löf, 1998]

PURE TYPE SYSTEMS

1. Pure type systems (PTS) were independently introduced by Stefano Berardi (1988) and Jan Terlouw (1989)
2. Generalize many different type systems (as we shall see)
3. Book recommendation: [Nederpelt and Geuvers, 2014]
   A presentation of an important subset of PTSs called the λ-cube [Barendregt, 1991]
4. Coq is not a PTS, but a large chunk of it almost is and it serves as a good starting point

Pure type systems deal with a single judgement form $\Gamma \vdash M : A$ that is to be read:

"In the context Γ, there is an object M of type A."

PURE TYPE SYSTEMS DETERMINED BY

Every PTS is determined by:

1. a collection $\mathcal{S}$ of *sorts*, sometimes called *universes*
2. a collection $\mathcal{A}$ of pairs of sorts called *axioms*
3. a collection $\mathcal{R}$ of triples of sorts called *rules*

Syntax

Fix some set of variables $\mathcal{V}$. Then:

- $s, s_1, s_2 ::= \mathcal{S}$
- $x, y, z, P, Q, R, S, T ::= \mathcal{V}$
- $A, B, C, D, M, N ::= \mathcal{S} | \mathcal{V} | MN | \lambda \mathcal{V} : A.M | \Pi \mathcal{V} : A.M$
- $\Gamma, \Delta ::= \epsilon | \Gamma, \mathcal{V} : A$ (where ϵ is the empty string)

Π and λ bind variables & we identify terms up to renaming of bound variables (i.e. α-equivalence)

PTS (SORT) (VAR)

An axiom $(s_1 : s_2)$ whenever $\langle s_1, s_2 \rangle$ is in $\mathcal{A}$. There are no other axioms – contexts are built up during the derivation.

$$(s_1 : s_2)\,\frac{}{\vdash s_1 : s_2}$$

The (var) rule corresponds to the axiom scheme of Gentzen single-conclusion systems, but it has an assumption because a type must be so-called “well-formed” in the previous context.

$$(\text{var})\,\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}\; x : \_ \notin \Gamma$$

$$(\text{var})\,\cfrac{(\text{var})\,\cfrac{(\star : \Box)\,\cfrac{}{\star : \Box}}{P : \star \vdash P : \star}}{P : \star, x : P \vdash x : P}$$

PTS (WEAK)

Using (weak) one can extend the context while retaining the state, but again the context must be “well-formed” to extend it.

$$\text{(weak)}\ \frac{\Gamma \vdash M : B \qquad \Gamma \vdash A : s}{\Gamma, x : A \vdash M : B}\ x : \_ \notin \Gamma$$

$$\text{(weak)}\ \frac{\begin{array}{c}\vdots\\ P : \star \vdash P : \star\end{array} \qquad \begin{array}{c}\vdots\\ P : \star \vdash \star : \square\end{array}}{P : \star, Q : \star \vdash P : \star}$$

PTS (FORM)

A formation rules $s_1 \to_s s_2$ whenever $\langle s_1, s_2, s\rangle$ is in $\mathcal{R}$. Tells us what kind of functional dependencies are allowed.

$$(s_1 \to_s s_2)\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.B : s}$$

Set-Theoretic Intuition for Dependent Functions

$$\Pi x : A.B(x) \cong \{f : A \to \bigcup_{x \in A} B(x) \mid \forall a \in A. f(a) \in B(a)\}$$

Conventions

- $A \to B$ instead of $\Pi x : A.B$ when x does not appear free in B
- We write $s_1 \to s_2$ for $s_1 \to_{s_2} s_2$

PTS (FORM)

$$(s_1 \to_s s_2)\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.B : s}$$

$$\text{(weak)}\ \dfrac{\begin{array}{c}\vdots\\ P : \star \vdash P : \star\end{array} \qquad (\star \to \square)\ \dfrac{\begin{array}{c}\vdots\\ P : \star \vdash P : \star\end{array} \qquad \begin{array}{c}\vdots\\ P : \star, x : P \vdash \star : \square\end{array}}{P : \star \vdash P \to \star : \square}}{P : \star, S : P \to \star \vdash P : \star}$$

$$(\star \to \square)\ \dfrac{\begin{array}{c}\vdots\\ P : \star, S : P \to \star \vdash P : \star\end{array} \qquad \begin{array}{c}\vdots\\ P : \star, S : P \to \star, x : P \vdash Sx : \star\end{array}}{P : \star, S : P \to \star \vdash \Pi x : P.Sx : \star}$$

PTS (ABST)

The (abst) rule is for introducing functions. Note that the function type must be "well-formed" to use it.

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

Let $\Gamma \equiv P : \star, S : P \to \star$.

$$(\text{abst})\ \frac{\begin{array}{c}\vdots \\ \Gamma \vdash \Pi x : P.Sx \to Sx : \star\end{array} \qquad \begin{array}{c}\vdots \\ \Gamma, x : P \vdash \lambda y : Sx.y : Sx \to Sx\end{array}}{\Gamma \vdash \lambda x : P.\lambda y : Sx.y : \Pi x : P.Sx \to Sx}$$

Convention

Arrow associates right: $A \to B \to C \to D$ is $A \to (B \to (C \to D))$

PTS (APPL)

The (appl) rule is for eliminating functions.

$$\text{(appl)}\ \frac{\Gamma \vdash M : \Pi x : A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B\{N/x\}}$$

Let $\Gamma \equiv P : \star, S : P \to \star, z : P$.

$$\text{(appl)}\ \frac{\begin{array}{c}\vdots \\ \Gamma \vdash \lambda x : P.\lambda y : Sx.y : \Pi x : P.Sx \to Sx\end{array} \qquad \begin{array}{c}\vdots \\ \Gamma \vdash z : P\end{array}}{\Gamma \vdash (\lambda x : P.\lambda y : Sx.y)z : Sz \to Sz}$$

Convention

Application associates left: $ABCD$ is $((AB)C)D$

PTS (CONV)

The (conv) rule is needed to kick-off computation inside types.

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

Let $\Gamma \equiv P : \star, x : (\lambda Q : \star.Q \rightarrow Q)P$.

$$\frac{\begin{array}{c}\vdots \\ \Gamma \vdash x : (\lambda Q : \star.Q \rightarrow Q)P\end{array} \qquad \begin{array}{c}\vdots \\ \Gamma \vdash P \rightarrow P : \star\end{array}}{\Gamma \vdash x : P \rightarrow P}$$

SIMPLY TYPED λ-CALCULUS

$\mathcal{S} = \{\star, \Box\} \quad \mathcal{A} = \{(\star : \Box)\}$
$\mathcal{R} = \{(\star \to \star)\}$

1. Can encode natural numbers:

$$T : \star \vdash \underbrace{\lambda f : T \to T.\lambda n : T.f(f(n))}_{2} : (T \to T) \to T \to T$$

2. $T_1 : \star, \ldots T_n : \star \vdash M : A$ iff A is a tautology of minimal logic (i.e. classical logic with just $\to$)
3. Not to be confused with Simple Type Theory, which is *based* on STLC but is richer

SYSTEM F

$\mathcal{S} = \{\star, \Box\} \quad \mathcal{A} = \{(\star : \Box)\}$
$\mathcal{R} = \{(\star \to \star), (\Box \to \star)\}$

1. Can encode polymorphic functions:

$$\vdash \underbrace{\lambda T : \star.\lambda x : T.x}_{\text{id}} : \Pi T : \star.T \to T$$

   Can be applied to anything of type $\star$, including its own type!

2. Can encode various inductive types:

$$T : \star \vdash \underbrace{\Pi Q : \star.Q \to (T \to Q \to Q) \to Q}_{\text{List } T} : \star$$

3. Impredicative because there are $\star$'s that are defined by quantifying over all $\star$'s.

SYSTEM F

4. The impredicativity is apparently harmless. Arguably justified because of Parametricity – the $\star$'s quantified cannot be inspected and case split upon (see Abstraction Thm).
5. System F captures the impredicative core present in Coq.
6. An extension of $\mathcal{R}$ by $(\Box \to \Box)$ called Fω can encode type families: $\vdash \lambda T : \star.\ \mathrm{List}\ T : \star \to \star$.

[Girard, 1989]

An arithmetic function can be represented in System F if and only if it can be proved total in second order Peano Arithmetic.

[Reynolds, 1983] Abstraction Theorem

There is a semantic interpretation that shows that functions in system F take related inputs to related outputs.

DEPENDENT TYPES (λP)

$\mathcal{S} = \{\star, \Box\} \quad \mathcal{A} = \{(\star : \Box)\}$
$\mathcal{R} = \{(\star \to \star), (\star \to \Box)\}$

1. Can encode propositions as types that depends on terms:

$$T : \star, Q : T \to T \to \star \vdash \underbrace{(\Pi x : T.\Pi y : T.Qxy) \to \Pi x : T.Qxx}_{H} : \star$$

$$T : \star, Q : T \to T \to \star \vdash \lambda z : (\Pi x : T.\Pi y : T.Qxy).\lambda x : T.zxx : H$$

2. Here we get a much broader so-called
   Curry-Howard isomorphism
   AKA *propositions-as-types*
   AKA *proofs-as-programs*

CALCULUS OF CONSTRUCTIONS (λC)

$\mathcal{S} = \{\star, \Box\} \quad \mathcal{A} = \{(\star : \Box)\}$
$\mathcal{R} = \{(\star \to \star), (\star \to \Box), (\Box \to \star), (\Box \to \Box)\}$
The calculus of construction (λC) combines Fω with λP.

$$\vdash \underbrace{\lambda T : \star.\lambda P : T \to \star.\Pi Q : \star.(\Pi x : T.P \to Q) \to Q}_{\exists} : \star$$

$$\vdash \underbrace{\lambda T : \star.\lambda x : T.\lambda y : T.\Pi P : \star.Px \to Py}_{=} : \Pi T : \star.T \to T \to \star$$

NAÏVE TT

$\mathcal{S} = \{\star\} \quad \mathcal{A} = \{(\star : \star)\}$
$\mathcal{R} = \{(\star \to \star)\}$

1. Matrin-Löf's original formulation included these rules
2. Collapses $\star$ and $\Box$ from λC
3. The bad kind of impredicativity: inconsistent, i.e. every type in inhabited, in particular $\Pi T : \star . T$

SYSTEM U^-

$\mathcal{S} = \{\star, \Box, \triangle\} \quad \mathcal{A} = \{(\star : \Box), (\Box : \triangle)\}$
$\mathcal{R} = \{(\star \to \star), (\Box \to \star), (\Box \to \Box), (\triangle \to \Box)\}$

1. Also impredicative, this time at a not-the-lowest level
2. Seems less suspicious that $\star : \star$ because there is no circularity in terms of the axioms, but still, it is inconsistent [Girard, 1972]

On this problem and suggested solution:

> *This seems actually to show that the predicativity and non-predicativity are not contradictory concepts: simply, the level of proposition may be non-predicative and the level of type must be predicative.* [Coquand, 1986]

NICE PROPERTIES THAT PTSS ENJOY

Thinning (refined Weakening)

If $\Gamma \vdash A : B$ and $\Delta \supseteq \Gamma$ is well-formed ($\Delta \vdash \_$), then $\Delta \vdash A : B$.

Permutation (refined Exchange)

If $\Gamma \vdash A : B$ and Δ is a well-formed permutation of Γ, then $\Delta \vdash A : B$.

Condensing

If $\Gamma, x : C, \Delta \vdash A : B$ and x is not free in Δ, A, B then $\Gamma \vdash A : B$.

Substitution (refined Cut)

If $\Gamma, x : C, \Delta \vdash A : B$ and $\Gamma \vdash D : C$, then $\Gamma, \Delta\{D/x\} \vdash A\{D/x\} : B\{D/x\}$.

Type Correctness

If $\Gamma \vdash M : A$ then $A \in \mathcal{S}$ or $\Gamma \vdash A : s$ for some $s \in \mathcal{S}$.

Type Preservation

If $\Gamma \vdash M : A$ and $M =_\beta N$ then $\Gamma \vdash N : A$.

Confluence

If $\Gamma \vdash M : A$ and $M \mapsto_\beta^* R$ and $M \mapsto_\beta^* S$ then they can converge to some N, i.e. $R \mapsto_\beta^* N$ and $S \mapsto_\beta^* N$.

Decidable Type Checking

Strong Normalization implies decidability of $\Gamma \vdash A : B$.

Defn. Strong Normalization

If $\Gamma \vdash M : A$ then every sequence of $\mapsto_\beta$ from M eventually terminates with an irreducible term.

COQ TYPE SYSTEM

i ranges over $\mathbb{N}_+$.
$\mathcal{S} = \{\text{Prop}, \text{Type}_i\}$ $\quad \mathcal{A} = \{(\text{Prop} : \text{Type}_1), (\text{Type}_i : \text{Type}_{i+1})\}$
$\mathcal{R} = \{(\text{Prop} \to \text{Prop}), (\text{Type}_i \to \text{Prop}), (\text{Type}_i \to \text{Type}_i)\}$

The (conv) rule is strengthened:

$$\text{(conv)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B}\ A \leq B$$

The $\leq$ relation is transitive and closed under

1. $=_\beta$
2. $\text{Prop} \leq \text{Type}_1 \leq \text{Type}_2 \ldots$ (Cumulativity)
3. If $A =_\beta M$ and $B \leq N$ then $\Pi x : A.B \leq \Pi x : M.N$

MORE STUFF IN COQ TYPE SYSTEM

Things in CIC we've ignored:

1. Global environments, definitions, and δ reductions
2. Let expressions and ζ reductions
3. η expansions
4. The sort Set of small types
5. The sort Sprop of strict-propositions (experimental feature)
6. (Co)Inductive types and ι reductions

PROOF IRRELEVANCE

1. The impredicativity of Prop is closely related to the concept of proof irrelevance – any two proofs of the same Prop are equal:
$$\Pi P : \mathrm{Prop}.\Pi x, y : P.x =_P y$$
2. Coq cannot prove this theorem; however, it is provable assuming excluded-middle:
$$\Pi P : \mathrm{Prop}.P \vee \neg P$$

PROGRAM EXTRACTION

1. Proof irrelevance is a means to control information flow
2. If data is declared irrelevant, it can be ignored when extracting a program
3. Using irrelevance is somewhat a design decision

[Bauer, 2014]

Reveal the remainder $\Pi n : \mathbb{N}. \Sigma k : \mathbb{N}. \Sigma b : \{0, 1\}. n = 2k + b$

Hide the remainder $\Pi n : \mathbb{N}. \Sigma k : \mathbb{N}. \exists b : \{0, 1\}. n = 2k + b$

LIVE DEMONSTRATION

1. Inductive types (because formal treatment is exhausting)
2. Equality: Leibniz vs. Inductive
3. Impredicativity is related to Proof Irrelevance
4. Proof Irrelevance is useful in program extraction
5. Stratification of Type enables data abstractions

JUSTIFYING IMPREDICATIVITY

If the collection is not closed, as is $\star$ in Coq, what can justify its impredicativity?
In [Longo et al., 1992] the innocuous C axiom is added to their formulation of system F:

Axiom C

If $\Gamma \vdash M : \Pi x : \star . C$ and x does not appear free in B, then for all $\Gamma \vdash A, B : \star$ it holds that $MA = MB$.

JUSTIFYING IMPREDICATIVITY

If the collection is not closed, as is $\star$ in Coq, what can justify its impredicativity?
In [Longo et al., 1992] the innocuous C axiom is added to their formulation of system F:

Axiom C

If $\Gamma \vdash M : \Pi x : \star . C$ and x does not appear free in B, then for all $\Gamma \vdash A, B : \star$ it holds that $MA = MB$.

Then the Genericity theorem is proven for the resulting system:

Genericity Theorem

In the system Fc, let $\Gamma \vdash M, N : \Pi x : \star . C$. If there exists $\Gamma \vdash A : \star$ such that $MA = NA$, then $M = N$.

So the terms must only be equal at a particular instance to be equal everywhere.

JUSTIFYING IMPREDICATIVITY

The logical ramifications are detailed in a later paper:

> *Consider [..] a proposition [..] such as* $\forall x P(x)$*, where x ranges on some intended collection of individuals. [..] the proof does not depend on the specific [individual] chosen, but only on the assumption that x is [an individual from the range]. In type-theoretic terms, a sound proof would only depend on the* type *of x, not on its value. [..] Herbrand called this kind of "uniform" proofs* **prototype**.
>
> [Longo, 2000]

JUSTIFYING IMPREDICATIVITY

The logical ramifications are detailed in a later paper:

> *Consider [..] a proposition [..] such as* $\forall x P(x)$*, where x ranges on some intended collection of individuals. [..] the proof does not depend on the specific [individual] chosen, but only on the assumption that x is [an individual from the range]. In type-theoretic terms, a sound proof would only depend on the* type *of x, not on its value. [..] Herbrand called this kind of "uniform" proofs* **prototype**.
>
> [Longo, 2000]

In that paper a much earlier one is quoted:

> *If we reject the belief that it is necessary to run through individual cases and rather make it clear to ourselves that the complete verification of a statement means nothing more than its logical validity for an arbitrary property, we will come to the conclusion that impredicative definitions are logically admissible.* [Carnap, 1931]

The End

BARENDREGT, H. (1991).
INTRODUCTION TO GENERALIZED TYPE SYSTEMS.
Journal of Functional Programming, 1(2):125–154.

BAUER, A. (2014).
WHY DOES COQ HAVE PROP?
Published: Theoretical Computer Science Stack Exchange.

CARNAP, R. (1931).
THE LOGICIST FOUNDATIONS OF MATHEMATICS.

COQUAND, T. (1986).
AN ANALYSIS OF GIRARD'S PARADOX.
In *In Symposium on Logic in Computer Science*, pages 227–236. IEEE Computer Society Press.

GIRARD, J.-Y. (1972).
INTERPRÉTATION FONCTIONNELLE ET ÉLIMINATION DES COUPURES DE L'ARITHMÉTIQUE D'ORDRE SUPÉRIEUR.
PhD thesis, Éditeur inconnu.

GIRARD, J.-Y. (1989).
PROOFS AND TYPES.

Number 7 in Cambridge tracts in theoretical computer science.
Cambridge University Press, Cambridge [England] ; New York.

LONGO, G. (2000).
PROTOTYPE PROOFS IN TYPE THEORY.
MLQ, 46(2):257–266.

LONGO, G., MILSTED, K., AND SOLOVIEV, S. (1992).
THE GENERICITY THEOREM AND THE NOTION OF PARAMETRICITY IN THE POLYMORPHIC-CALCULUS.
Technical report.

MARTIN-LÖF, P. (1998).
AN INTUITIONISTIC THEORY OF TYPES.
Twenty-five years of constructive type theory, 36:127–172.

NEDERPELT, R. AND GEUVERS, H. (2014).
TYPE THEORY AND FORMAL PROOF: AN INTRODUCTION.
Cambridge University Press.

REYNOLDS, J. C. (1983).
TYPES, ABSTRACTION AND PARAMETRIC POLYMORPHISM.

WEYL, H. (1949).

Philosophie der Mathematik und Naturwissenschaft (Philosophy of Mathematics and Natural Science).
R. Oldenbourg, Munich. Traduit et réédité par Princeton University Press.