# Resizing Prop down to an axiom

Stefan Monnier[1]

Université de Montréal - DIRO, Montréal, Canada
monnier@iro.umontreal.ca

**Abstract**

We present an axiomatization of the impredicative Prop universe of the Calculus of Constructions. More specifically, we present two extensions of a predicative CC$\omega$, one with an impredicative quantification in the bottom universe, and the other with a set of axioms and related reduction rules, and then we prove equivalence of the two calculi.

As the title suggests, the axioms are closely related to HoTT's propositional resizing axiom, thus giving concrete evidence for the intuition that this form of impredicativity is equivalent to that offered by the Prop universe. This can also help interoperability between proof systems with a Prop universe and those without.

We finish by discussing how this result can be extended to a calculus with inductive types.

## 1 Introduction

Impredicativity, in the context of type theory, is the ability for a proposition to be quantified over a type which can be instantiated with that very same proposition. The kind of circularity it introduces is more subtle than that of recursion, but just as dangerous. While many systems disallow this principle outright, favoring the simpler and cleaner metatheory of predicative type theories, it is quite popular both in the context of proof assistants and in the context of functional programming languages.

There are a few different ways to introduce impredicativity, such as: via $\mathsf{Type} : \mathsf{Type}$, as used in Dependent Haskell [7]; via the traditional impredicative bottom universe, as used in System F and in the Calculus of Constructions [3]; via resizing axioms, as used in the HoTT book [6]; or via universe polymorphism, as proposed in [4].

Of those, the impredicative bottom universe as implemented in Coq's $\mathsf{Prop}$ is the most widely used in proof assistants and it is generally accepted that the propositional resizing axiom is an alternative to it of comparable power. Yet there is as yet no formal investigation to show precisely *how* comparable they are.

We intend to remedy this situation in the present work in the following way: we show a proof of equivalence between two calculi we call $\mathsf{iCC}\omega$ and $\mathsf{rCC}\omega$: both are pure type system (PTS) [1] with the usual tower of universes; the first comes with an impredicative quantification in universe $\mathsf{Type}_0$ while the second replaces it with a set of axioms which provides a form of propositional resizing. More specifically our contributions are:

- A type-preserving translation of any term of $\mathsf{iCC}\omega$ into a term of $\mathsf{rCC}\omega$, following the approach of syntactic models used in [2].
- A type-preserving translation of any term of $\mathsf{rCC}\omega$ into a term of $\mathsf{iCC}\omega$, simply by providing definitions for the axioms.
- An extension of these results to calculi with inductive types. This extension is not fully satisfactory because the equivalence is not fully satisfied.

## 2 The encoding

The calculi we use in this paper are pure type systems [1], using the following syntax:

$$\begin{array}{llll} \textit{(var)} & x, y, t & \in & \mathcal{V} \\ \textit{(sort)} & s & \in & \mathcal{S} \\ \textit{(term)} & e, \tau & ::= & s \mid x \mid (x : \tau_1) \to \tau_2 \mid \lambda x : \tau . e \mid e_1 @ e_2 \end{array}$$

A specific PTS is then defined by providing the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ which defines respectively the sorts, axioms, and rules of this system. All our calculi are extensions of the following base predicative calculus we call $\mathsf{pCC}\omega$:

$$\begin{array}{lcll} \mathcal{S} & = & \{\ \mathsf{Type}_\ell & \mid\ \ell \in \mathbb{N}\ \} \\ \mathcal{A} & = & \{\ (\mathsf{Type}_\ell : \mathsf{Type}_{\ell+1}) & \mid\ \ell \in \mathbb{N}\ \} \\ \mathcal{R} & = & \{\ (\mathsf{Type}_{\ell_1}, \mathsf{Type}_{\ell_2}, \mathsf{Type}_{\mathsf{max}(\ell_1,\ell_2)}) & \mid\ \ell_1, \ell_2 \in \mathbb{N}\ \} \end{array}$$

$\mathsf{iCC}\omega$ is defined as the extension of $\mathsf{pCC}\omega$ with the following rules:

$$\mathcal{R} \quad = \quad \ldots \cup \{\ (\mathsf{Type}_\ell, \mathsf{Type}_0, \mathsf{Type}_0)\ \mid\ \ell \in \mathbb{N} \wedge \ell > 0\ \}$$

As a first approximation, $\mathsf{rCC}\omega$ is defined as the extension of $\mathsf{pCC}\omega$ with the following axioms and rewrite rules:

$$\begin{array}{lcll} ||\cdot|| & : & \mathsf{Type}_\ell \to \mathsf{Type}_0 & \text{for all } \ell \in \mathbb{N} \\ |\cdot| & : & (t : \mathsf{Type}_\ell) \to t \to ||t|| & \text{for all } \ell \in \mathbb{N} \\ \mathsf{bind} & : & (t_1 : \mathsf{Type}_{\ell_1}) \to (t_2 : \mathsf{Type}_{\ell_2}) \to ||t_1|| \to (t_1 \to ||t_2||) \to ||t_2|| & \text{for all } \ell_1, \ell_2 \in \mathbb{N} \end{array}$$

$$\mathsf{bind}\ |e_1|\ \lambda x : \tau . e_2 \quad \rightsquigarrow \quad e_2\{e_1/x\}$$

Where $||\cdot||$ is pronounced "erased" and $|\cdot|$ is pronounced "erase". The crux of the matter to encode (written $[\cdot]$) an impredicative quantification $(x : \tau_1) \to \tau_2$ of $\mathsf{iCC}\omega$ into $\mathsf{rCC}\omega$ is to erase it with $||\cdot||$ so as to bring it back down to $\mathsf{Type}_0$: $[(x : \tau_1) \to \tau_2] = ||(x : [\tau_1]) \to [\tau_2]||$
As a consequence, the encoding needs to erase systematically all inhabitants of the $\mathsf{Type}_0$ universe. A first cut of the encoding looks like the following:

$$\begin{array}{lcl} [x] & = & x \\ [\mathsf{Type}_\ell] & = & \mathsf{Type}_\ell \\ [(x : \tau_1) \to \tau_2] & = & \begin{cases} ||(x : [\tau_1]) \to [\tau_2]|| & \text{if in the } \mathsf{Type}_0 \text{ universe} \\ (x : [\tau_1]) \to [\tau_2] & \text{otherwise} \end{cases} \\ [e_1 @ e_2] & = & \begin{cases} \mathsf{bind}\ [e_1]\ \lambda f . f @ [e_2] & \text{if in the } \mathsf{Type}_0 \text{ universe} \\ [e_1] @ [e_2] & \text{otherwise} \end{cases} \\ [\lambda x : \tau . e] & = & \begin{cases} |\lambda x : [\tau] . [e]| & \text{if in the } \mathsf{Type}_0 \text{ universe} \\ \lambda x : [\tau] . [e] & \text{otherwise} \end{cases} \end{array}$$

Sadly, this encoding is a bit too naïve to preserve types. While the monadic erasure axioms proposed above are rather enticing (and similar to those used in [5]) we will need to tweak them a bit in order to be able to refine the encoding into one that is type preserving.

### Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) grant N$^o$ 298311/2012 and RGPIN-2018-06225. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSERC.

# References

[1] Henk P. Barendregt. Introduction to generalized type systems. *Journal of Functional Programming*, 1(2):121–154, April 1991. doi:10.1017/S0956796800020025.

[2] Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. The next 700 syntactical models of type theory. In *Certified Programs and Proofs*, page 182–194, 2017. doi:10.1145/3018610.3018620.

[3] Thierry Coquand and Gérard P. Huet. The calculus of constructions. Technical Report RR-0530, INRIA, 1986.

[4] Stefan Monnier and Nathaniel Bos. Is impredicativity implicitly implicit? In *Types for Proofs and Programs*, Leibniz International Proceedings in Informatics (LIPIcs), pages 9:1–9:19, 2019. doi:10.4230/LIPIcs.TYPES.2019.9.

[5] Arnaud Spiwack. Notes on axiomatising hurkens's paradox, 2015. URL: https://arxiv.org/abs/1507.04577.

[6] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. URL: https://arxiv.org/abs/1308.0729.

[7] Stephanie Weirich, Antoine Voizard, Pedro Henrique Azevedo de Amorim, and Richard A. Eisenberg. A specification for dependent types in Haskell. In *International Conference on Functional Programming*, page 1–29, 2017. doi:10.1145/3110275.