Definitional Proof Irrelevance Made Accessible

Thiago Felicissimo[1], Yann Leray[1], Loïc Pujet[2], Nicolas Tabareau[1], Eric Tanter[3], and Théo Winterhalter[4]

[1] LS2N, Inria Rennes [2] Université de Strasbourg [3] University of Chile [4] LMF, Inria Saclay

Propositions and proof irrelevance. Although the Curry-Howard philosophy suggests that propositions should be considered as regular types just like $\mathbb{N}$ or $\mathbb{B}$, in practice it is useful to enforce a clear distinction between *relevant* data and *irrelevant* proofs. As such, Rocq features a universe Prop for propositions, compatible with *propositional proof irrelevance*, stating that any two proofs of the same proposition are propositionally equal.

In order to be compatible with proof irrelevance, one must ensure that proof terms cannot be used to produce computationally relevant content in a way that would enable distinguishing two proofs of the same statement. This property is also key for having an *extraction* mechanism, which strips terms of all propositional content in order to compile them to common programming languages [5]. In the Calculus of Inductive Constructions [7]—the underlying theory of Rocq—this is enforced by a syntactic condition known as the *subsingleton criterion*:[1] eliminating a term of an inductive type in Prop to produce a term whose type is in Type is allowed only if the input inductive type has at most one constructor whose non-parameter arguments are also irrelevant proofs. This controlled Prop-to-Type elimination applies in particular to three fundamental inductive types, each corresponding to highly relevant application scenarios in theorem proving:

(a) the empty type False, used to discard impossible branches in pattern matching;

(b) the identity type Eq, used to rewrite provably equal terms;

(c) the accessibility predicate Acc, allowing the definition of recursive functions via semantic termination arguments.

Strict propositions. Unfortunately, the compatibility of Prop with propositional proof irrelevance is often insufficient in practice: the user is still required to perform explicit rewrites when relating terms up to proofs, making working with subset types and quotient types an arduous and unpleasant task. Worse, as soon as the user postulates proof irrelevance—or almost any other axiom—they break *canonicity* (*e.g.,* that every natural number computes to a concrete numeral). Type-theoretic proof assistants have thus increasingly added support for *strict propositions*, for which proof irrelevance is definitional—allowing, for instance, terms of a subset type be convertible as soon as their data components are convertible. They were first introduced in Lean, before being integrated in Agda and in Rocq [4]. Here, we adopt the convention of Rocq and write the universe of strict propositions as SProp, to distinguish it from Prop. Apart from yielding a setting closer to informal mathematical practice, the addition of SProp to Rocq was shown to preserve all important meta-theoretical properties of type theory, in particular canonicity, even in the presence of consistent SProp axioms, in stark contrast with Prop.

Failure of the subsingleton criterion. Naively extending the subsingleton criterion to inductive types in SProp sadly only works safely for False. For Eq, the proof-irrelevant version of the traditional J eliminator with its stronger reduction rule, as implemented in Lean, breaks canonicity in the presence of function extensionality [3, Sec 12.3] and proof normalization in the presence of impredicativity [1]. Worse, elimination of Acc breaks decidability of conversion regardless of function extensionality or (im)predicativity [4]. Agda and Rocq thus restrict the subsingleton criterion for SProp, allowing only the elimination of False. This restriction means that, in practice, Rocq users still resort to the legacy Prop universe to exploit equality rewriting and accessibility predicates for fixpoints, while Agda users typically carry out their development in Type.

[1] This criterion was introduced in Coq 7.3 in 2002.

On the other hand, Lean supports the full subsingleton criterion for SProp, which means that conversion is undecidable and that evaluating a term may trigger an infinite loop. During type-checking, Lean implements a *heartbeat* mechanism that introduces a user-customizable bound on computation steps. While this effectively prevents infinite loops, the implementation of definitional equality becomes different from its specification (*e.g.,* it is no longer transitive) and subtle changes in heuristics to control unfolding might unexpectedly break proofs. In order to mitigate these issues, Lean makes all well-foundedness proofs opaque by default since v4.19 (2024), so that users are not faced with non-termination unless they explicitly opt into computing with accessibility.

Observational Equality. For equality, the problem can be solved using *observational equality* [2, 8, 9, 10]. Unlike the inductive identity type of Martin-Löf [6], whose elimination principle J works uniformly for all predicates in Type, the elimination of observational equality is defined in a type-directed manner via a cast operator. This approach is expressive enough to emulate the traditional J eliminator, and allows elimination of an SProp-valued equality into Type without compromising on decidability, canonicity or consistency. The theory $\mathrm{CIC}_{\mathrm{obs}}$ extends CIC with observational equality, and will be available in a future release of Rocq. Thus, the only part of the subsingleton criterion that remains unavailable with SProp is the accessibility predicate.

Contributions. We propose a novel design combining SProp with elimination of Acc into the Type universe. We avoid the problems related to undecidability by considering an extension of $\mathrm{CIC}_{\mathrm{obs}}$, henceforth named $\mathcal{T}^{=}_{\mathrm{Acc}}$, in which the computation rule for Acc only holds *propositionally*. This aspect quite directly recovers decidability of conversion, yet it breaks the usual statement of canonicity and, in principle, makes reasoning about functions defined with Acc more arduous. For instance, given such a function $f : \mathbb{N} \to \mathbb{N}$, proving that $f\ k = m$ for k, m concrete numerals, requires us to perform explicit rewriting (possibly a lot) to compensate for the lack of computation.

We address this difficulty by introducing a second theory $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ in which the computation rule for accessibility holds *definitionally*. Although $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ has an undecidable conversion [4], we prove it satisfies canonicity, meaning that it provides a good setting to evaluate programs from $\mathcal{T}^{=}_{\mathrm{Acc}}$. For instance, if we embed the function f from the previous example in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$, $f\ k$ now evaluates to m, and we can show that $f\ k = m$ by reflexivity. Second, we also show that the definitional theory $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ is *conservative* over the propositional one $\mathcal{T}^{=}_{\mathrm{Acc}}$, by adapting Winterhalter et al. [11]'s translation from extensional to intentional type theory. Concretely, this implies that, when proving a proposition $P : \mathsf{SProp}$ in $\mathcal{T}^{=}_{\mathrm{Acc}}$ (such as $f\ k = m$), one can locally switch to the theory $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ to complete the proof, with the guarantee that the proof term in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ could be elaborated to $\mathcal{T}^{=}_{\mathrm{Acc}}$. Yet, because proofs of in SProp are irrelevant, this proof term translation never has to be performed in practice, and one can instead just add P in $\mathcal{T}^{=}_{\mathrm{Acc}}$ as a (justified) axiom. This remark allows us to avoid computing the explicit witness for P in $\mathcal{T}^{=}_{\mathrm{Acc}}$, an important optimization for implementations.

As a direct corollary of canonicity for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ and of its conservativity over $\mathcal{T}^{=}_{\mathrm{Acc}}$, we also derive *propositional canonicity* for $\mathcal{T}^{=}_{\mathrm{Acc}}$, stating that any closed term of type $\mathbb{N}$ is propositionally equal to a numeral. Moreover, our canonicity results for $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ are preserved in the presence of consistent axioms in SProp. This important aspect ensures for instance that programs whose termination relies on classical principles, such as excluded middle, can still be properly evaluated.

Finally, we also establish the consistency of $\mathcal{T}^{=}_{\mathrm{Acc}}$ by adapting the set-theoretic model of Pujet and Tabareau [10], showing that our theory can be adequately used as an internal language for set-level mathematics, all while preserving decidability and a sufficient form of canonicity.

Practical contribution. We implement in Rocq the combination of $\mathcal{T}^{=}_{\mathrm{Acc}}$ with a proof mode switch to $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$. This effectively allows users to enjoy the definitional unfolding of well-founded fixpoints within a proof. Using an undecidable theory isn't as scary as it seems, given that interactive provers routinely include potentially diverging mechanisms for making proving easier: *e.g.,* automation tactics with no termination guarantees. We test our approach with a case study of the normalization proof of System F, from which we derive an evaluator. We show that evaluating functions through their termination proofs in `Prop` does not scale, and that automatic rewriting with the propositional computation rule for accessibility in `SProp` ($\mathcal{T}^{=}_{\mathrm{Acc}}$) is orders of magnitude slower than direct computation with the definitional computation rule for accessibility ($\mathcal{T}^{\equiv}_{\mathrm{Acc}}$).

References

[1] Andreas Abel and Thierry Coquand. Failure of Normalization in Impredicative Type Theory with Proof-Irrelevant Propositional Equality. *Logical Methods in Computer Science*, Volume 16, Issue 2, June 2020. doi: 10.23638/LMCS-16(2:14)2020. URL https://lmcs.episciences.org/6606.

[2] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *Proceedings of the Workshop on Programming Languages meets Program Verification (PLPV 2007)*, pages 57–68, 2007. doi: 10.1145/1292597.1292608.

[3] Jeremy Avigad, Leonardo de Moura, Soonho Kong, Sebastian Ullrich, and with contributions from the Lean Community. Theorem Proving in Lean 4, 2025. URL https://lean-lang.org/theorem_proving_in_lean4/.

[4] Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional Proof-Irrelevance without K. *Proceedings of the ACM on Programming Languages*, 3:1–28, January 2019. doi: 10.1145/3290316. URL https://hal.inria.fr/hal-01859964.

[5] P. Letouzey. *Programmation fonctionnelle certifiée – L'extraction de programmes dans l'assistant Coq*. PhD thesis, Université Paris-Sud, July 2004.

[6] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73 – 118. Elsevier, 1975. doi: https://doi.org/10.1016/S0049-237X(08)71945-1. URL http://www.sciencedirect.com/science/article/pii/S0049237X08719451.

[7] Christine Paulin-Mohring. Introduction to the Calculus of Inductive Constructions. In Bruno Woltzenlogel Paleo and David Delahaye, editors, *All about Proofs, Proofs for All*, volume 55 of *Studies in Logic (Mathematical logic and foundations)*. College Publications, January 2015. ISBN 978-1-84890-166-7. URL https://hal.inria.fr/hal-01094195.

[8] Loïc Pujet and Nicolas Tabareau. Observational Equality: Now For Good. *Proceedings of the ACM on Programming Languages*, 6(POPL):1–29, January 2022. doi: 10.1145/3498693. URL https://hal.inria.fr/hal-03367052.

[9] Loïc Pujet and Nicolas Tabareau. Impredicative Observational Equality. *Proceedings of the ACM on Programming Languages*, 7(POPL):74, January 2023. doi: 10.1145/3571739. URL https://hal.science/hal-03857705.

[10] Loïc Pujet and Nicolas Tabareau. Observational Equality Meets CIC. In *Lecture Notes in Computer Science*, volume 14576 of *Lecture Notes in Computer Science*, pages 275–301, Luxembourg, Luxembourg, April 2024. Springer Nature Switzerland. doi: 10.1007/978-3-031-57262-3_12. URL https://hal.science/hal-04535982.

[11] Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. Eliminating reflection from type theory. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2019, page 91–103, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362221. doi: 10.1145/3293880.3294095. URL https://doi.org/10.1145/3293880.3294095.