Definitional Proof Irrelevance Made Accessible

Thiago Felicissimo
Inria (Gallinette), Nantes, France

Yann Leray
Inria (Gallinette), Nantes, France

Loïc Pujet
ICube, Université de Strasbourg, France

Nicolas Tabareau
Inria (Gallinette), Nantes, France

Éric Tanter
PLEIAD, Computer Science Department, University of Chile, Santiago, Chile
Inria (Gallinette), Nantes, France

Théo Winterhalter
Inria (Deducteam), Gif-sur-Yvette, France

Abstract

A universe of propositions equipped with definitional proof irrelevance constitutes a convenient medium to express properties and proofs in type-theoretic proof assistants such as Lean, Rocq, and Agda. However, allowing accessibility predicates – used to establish semantic termination arguments – to inhabit such a universe yields undecidable typechecking, hampering the predictability and foundational bases of a proof assistant. To effectively reconcile definitional proof irrelevance and accessibility predicates with both theoretical foundations and practicality in mind, we describe a type theory that extends the Calculus of Inductive Constructions featuring observational equality in a universe of strict propositions, with two variants for handling the elimination principle of accessibility predicates: one variant safeguards decidability by sticking to propositional unfolding, and the other variant favors flexibility with definitional unfolding, at the expense of a potentially diverging typechecking procedure. Crucially, the metatheory of this dual approach establishes that any proof made in the definitional variant of the theory can be translated into a proof of the same statement in the propositional variant, all while preserving the decidability of the latter. Moreover, we prove the two variants to be consistent and to satisfy forms of canonicity, ensuring that programs can indeed be properly evaluated. We present an implementation in Rocq and compare it with existing approaches. Overall, this work introduces an effective technique that informs the design of proof assistants with strict propositions, enabling local computation with accessibility predicates without compromising the ambient type theory.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Dependent type theory, proof assistants, Rocq, proof irrelevance, accessibility predicates, observational equality, canonicity, set-theoretic models

Digital Object Identifier 10.4230/LIPIcs.LICS.2026.41

Supplementary Material *Other (Formalization of our results in the Rocq prover)*: `https://doi.org/10.5281/zenodo.20213920`

Acknowledgements The authors would like to thank the reviewers for their helpful comments.

1 Introduction

Proof assistants based on dependent type theory such as Rocq (formerly known as Coq), Lean, and Agda embody the Curry-Howard correspondence, in which propositions are represented as types and proofs as well-typed terms.

41st Annual Symposium on Logic in Computer Science (LICS 2026).
Editors: Claudia Faggian and Joost-Pieter Katoen; Article No. 41; pp. 41:1–41:26
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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.

For instance, the proof assistant Rocq features a special universe for propositions called Prop. All the types in Prop are compatible with *propositional proof irrelevance*, a principle which states that any two proofs of the same statement are propositionally equal. In contrast, this principle is clearly invalid for most datatypes (which are thus put in the Type hierarchy): consider for instance the type of natural numbers $\mathbb{N}$, for which one can show that $0 \neq 1$.

In order to be compatible with proof irrelevance, one must ensure that irrelevant 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 [31].

In the Calculus of Inductive Constructions [37] – the underlying theory of Rocq– this is enforced by a syntactic condition known as the *subsingleton criterion*, which has been introduced in Coq 7.3 in 2002. It states that 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' types are also in Prop. 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 (*a.k.a.* fixpoints) via semantic termination arguments, such as an evaluator for System F terms.

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. As a consequence, working for instance with subset types becomes more unpleasant, even though they are omnipresent in both mathematical and programming developments. Worse, as soon as the user postulates proof irrelevance – or almost any axiom, for that matter – they break the property of *canonicity*, *i.e.,* that any term of type $\mathbb{N}$ computes to a numeral.

To address the aforementioned problems, type-theoretic proof assistants have increasingly added support for *strict propositions*, for which proof irrelevance is not propositional but definitional – allowing, for instance, terms of a subset type be convertible as soon as their data components are convertible. Strict propositions were first introduced in Lean, before being integrated in Agda and in Rocq [20]. 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, strict propositions are also necessary for many important constructions in type theory, such as the setoid model of Altenkirch [8] and the strict presheaves of Pédrot [38].

Crucially, the addition of SProp was shown to preserve all important metatheoretical properties of type theory, in particular canonicity. Moreover, in contrast with Prop axioms, which block computation, canonicity holds even in the presence of consistent SProp axioms, such as *function extensionality* – which states that two functions are equal if they are pointwise equal.

Failure of the subsingleton criterion. The move from Prop to SProp however jeopardizes the good properties of the subsingleton criterion for controlling elimination into Type. Naively extending it to inductive types in SProp 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 [12, 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 [20].

For these reasons, Rocq and Agda use a weakened version of the subsingleton criterion for SProp, which only allows the elimination of False. This restriction limits the applicability of SProp, so in practice Rocq users still resort to the legacy Prop universe to exploit equality rewriting and accessibility predicates for fixpoints, while Agda users only use Type.

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. In practice, Lean implements a *heartbeat* mechanism during type-checking, which 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 which have been accepted previously. In order to mitigate these issues, Lean makes all well-foundedness proofs opaque by default since v4.19 (2024), so that inexperienced users are less likely to run into non-termination, while experienced users are still able to explicitly opt into computing with accessibility.

Observational Equality. Given this less-than-ideal state of affairs, one may wonder if definitional proof irrelevance is doomed to lack well-behaved elimination into Type, or if it can be made to work with Eq and Acc all while preserving the good properties of type theory. For the case of equality, Pujet *et al.* [40, 41, 39] realized that the issue could be addressed by moving to the alternative notion of *observational equality* [10]. Unlike the inductive identity type of Martin-Löf [33], 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 simulate 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 in SProp is the accessibility predicate.

Contributions. The goal of this work is to achieve a graceful marriage of accessibility and definitional proof irrelevance. Of course, we cannot avoid the undecidability result of Gilbert *et al.* [20]: if one allows the unfolding of fixpoints whose accessibility proof starts with a constructor, then proof irrelevance will entail that every accessibility proof may as well start with a constructor, and thus that every fixpoint can be unfolded. If the well-foundedness used for a particular given fixpoint turns out to be unsound – *e.g.,* by relying on an inconsistent assumption – then the automatic unfolding of this fixpoint will blindly diverge.

Therefore, we propose to avoid the problems related to undecidability by first considering an extension of $\mathrm{CIC}_{\mathrm{obs}}$, henceforth named $\mathcal{T}^{=}_{\mathrm{Acc}}$, in which the computation rule for Acc, which specifies how the eliminator interacts with the constructor, only holds *propositionally*. Demoting elimination of accessibility to a propositional computation rule recovers decidability of conversion, but it also breaks the usual statement of canonicity. This can make reasoning about functions defined using accessibility much less natural, given that they will only compute up to a propositional equality: for instance, given such a function $f : \mathbb{N} \to \mathbb{N}$, proving that $f\ k = m$ for k, m concrete natural numbers requires us to perform explicit rewrites (possibly a lot) to compensate for the lack of computation.

Our first technical contribution addresses this difficulty by introducing a second theory $\mathcal{T}^{\equiv}_{\text{Acc}}$ in which the computation rule for accessibility holds *definitionally*. As expected, conversion is undecidable in $\mathcal{T}^{\equiv}_{\text{Acc}}$, but in exchange for this we show that it satisfies canonicity, and thus that it provides a good setting for evaluating programs from $\mathcal{T}^{=}_{\text{Acc}}$. For instance, if we embed the function f from the previous example in $\mathcal{T}^{\equiv}_{\text{Acc}}$, $f\ k$ now evaluates to m, and we can show that $f\ k = m$ by reflexivity. However, this shorter proof is made in the theory $\mathcal{T}^{\equiv}_{\text{Acc}}$, not in $\mathcal{T}^{=}_{\text{Acc}}$, raising the question of how to connect this proof to one in $\mathcal{T}^{=}_{\text{Acc}}$.

Our second contribution is thus to show that the definitional theory $\mathcal{T}^{\equiv}_{\text{Acc}}$ is *conservative* over the propositional theory $\mathcal{T}^{=}_{\text{Acc}}$, by adapting Winterhalter *et al.* [45]'s translation from extensional to intentional type theory, coming from the seminal work of Hofmann [24] and Oury [36]. Concretely, this implies that, when proving a proposition $P : \mathsf{SProp}$ in $\mathcal{T}^{=}_{\text{Acc}}$ (such as $f\ k = m$), one can locally switch to the theory $\mathcal{T}^{\equiv}_{\text{Acc}}$ to complete the proof, with the guarantee that the proof term in $\mathcal{T}^{\equiv}_{\text{Acc}}$ could be elaborated to a proof term in $\mathcal{T}^{=}_{\text{Acc}}$. In practice, this elaboration step can be safely skipped: since proofs of strict propositions are definitionally irrelevant, the proof witness does not matter as far as computation is concerned, and thus the proof system may as well save the completed proof as an opaque constant in $\mathcal{T}^{=}_{\text{Acc}}$. This crucial remark allows us to avoid computing the explicit witness for P in $\mathcal{T}^{=}_{\text{Acc}}$ after performing its proof in $\mathcal{T}^{\equiv}_{\text{Acc}}$, an important optimization for implementations.

Third, as a corollary of canonicity for $\mathcal{T}^{\equiv}_{\text{Acc}}$ and of its conservativity over $\mathcal{T}^{=}_{\text{Acc}}$, we also derive *propositional canonicity*[1] for $\mathcal{T}^{=}_{\text{Acc}}$, stating that any closed term of type $\mathbb{N}$ is propositionally equal to a numeral.

Finally, we also establish the consistency of $\mathcal{T}^{=}_{\text{Acc}}$ by adapting the set-theoretic model of Pujet *et al.* [39], showing that our theory can be used as an internal language for set-level mathematics, all while preserving decidability and a sufficient form of canonicity. Moreover, our canonicity results for $\mathcal{T}^{=}_{\text{Acc}}$ and $\mathcal{T}^{\equiv}_{\text{Acc}}$ also hold in the presence of axioms in SProp, on the condition that they are true in our set-theoretic model. This important aspect ensures for instance that programs whose termination relies on classical principles, such as excluded middle, can still be evaluated.

Practical contribution. Justified by our theoretical results, we have implemented in Rocq the combination of $\mathcal{T}^{=}_{\text{Acc}}$ with a proof mode switch to $\mathcal{T}^{\equiv}_{\text{Acc}}$. This effectively allows users to enjoy the definitional unfolding of well-founded fixpoints inside a proof environment. Using an undecidable theory for proofs can seem scary, yet interactive provers such as Rocq have a long history of safely including potentially diverging mechanisms for making proving easier: for instance, tactics and automation have no termination guarantees.

To illustrate the benefits of our approach, we consider the case of System F and its strong normalization proof, from which one can derive an evaluator, and measure the performance of the evaluator to establish basic arithmetic results over Church numerals by computation. This simple case study shows that evaluating functions through their termination proofs in Prop does not scale, and that automatic rewriting with the propositional computation rule for accessibility ($\mathcal{T}^{=}_{\text{Acc}}$) is orders of magnitude slower than direct computation with the definitional computation rule ($\mathcal{T}^{\equiv}_{\text{Acc}}$).

Outline of the paper. Section 2 describes the two theories $\mathcal{T}^{=}_{\text{Acc}}$ and $\mathcal{T}^{\equiv}_{\text{Acc}}$, Section 3 establishes canonicity of $\mathcal{T}^{\equiv}_{\text{Acc}}$, Section 4 proves that $\mathcal{T}^{\equiv}_{\text{Acc}}$ is conservative over $\mathcal{T}^{=}_{\text{Acc}}$ and that $\mathcal{T}^{=}_{\text{Acc}}$ satisfies propositional canonicity, and Section 5 justifies the consistency of both theories via a set-theoretic model. Finally, Section 6 reports on an implementation of our proposal in Rocq, and Section 7 concludes, discussing related and future work.

[1] Often called homotopy canonicity in the context of *Homotopy Type Theory*.

Supplementary material. Our results have been mechanized in the Rocq prover (version 9.0), the code is available at `https://github.com/thiagofelicissimo/acc-in-sprop` and as a long-term archive on Zenodo (the link is provided in the title page). Definitions and theorems are referred to in the text using the Rocq icon which marks a clickable link that points to the formalization. The examples given in Section 6 as well as the version of Rocq that supports them are available at `https://github.com/Yann-Leray/acc-in-sprop`.

Metatheory. Although we formalize the results in Rocq, our metatheory is taken to be set theory. We thus rely on the fact that CIC can be used as an internal language for sets [42, 27].

2 Type Theories with Accessibility

In this section we formally introduce the systems $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ mentioned in the introduction, which are both extensions of $\mathrm{CIC}_{\mathrm{obs}}$ as introduced by Pujet *et al.* [39]. Since $\mathcal{T}^{=}_{\mathrm{Acc}}$ is included in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$, we start by describing their common features before precisely explaining the extra feature of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$.

Typing rules. Figures 1–3 give the common typing and conversion rules for $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$– congruence rules are omitted, we refer to the formalization for the complete set of rules (). The main typing judgments are $\Gamma \vdash^{\ell}_{=} t : A$ for $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\Gamma \vdash^{\ell}_{\equiv} t : A$ for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$– we drop the subscript when referring to features and rules common to both systems. They specify that, in the context Γ, the term t is typed by the term A, which lives in the *type universe* $\mathcal{U}_\ell$. Here, ℓ is a *universe level*, and can be either the impredicative level Ω, or a predicative level $\mathfrak{n} \in \{0, 1, 2, \dots\}$.[2] As specified by rule Proof-Irr, the universe $\mathcal{U}_\Omega$ is definitionally proof-irrelevant: any two terms t and u of type A are convertible as soon as A is in $\mathcal{U}_\Omega$.

Given a level ℓ, we define its next level ℓ^+ by $\Omega^+ := 0$ and $\mathfrak{n}^+ := 1 + \mathfrak{n}$, and rule Univ specifies that $\mathcal{U}_\ell$ has type $\mathcal{U}_{\ell^+}$. Note that, in the rules of Figures 1–3, we omit the level ℓ in $\Gamma \vdash^{\ell} t : A$ when A is of the form $\mathcal{U}_{\ell'}$ to avoid the redundant annotation ℓ'^+.

Aside from observational equality and accessibility, which we explain later, the theory includes dependent functions (with η-conversion) as well as natural numbers, in order to illustrate how general inductive types can be handled. While the type of natural numbers $\mathbb{N}$ lives in $\mathcal{U}_0$, the dependent function type $\Pi_{\ell,\ell'}(x : A).\, B$ formed with $A : \mathcal{U}_\ell$ and $B : \mathcal{U}_{\ell'}$ lives in the universe $\mathcal{U}_{\ell \ltimes \ell'}$, where $\ell \ltimes \ell'$ is defined by $\ell \ltimes \Omega := \Omega$, $\Omega \ltimes \ell := \ell$ and $\mathfrak{n} \ltimes \mathfrak{n}' := \mathsf{max}(\mathfrak{n}, \mathfrak{n}')$.

Fully annotated syntax. As can be seen in the typing rules, we adopt a *fully annotated* syntax, by writing application as $t @^{x:A.B}_{\ell,\ell'} u$, abstraction as $\lambda^{x:A.B}_{\ell,\ell'} x.\, t$, etc. We do so as the annotations ensure that well-typed terms can be organized into initial models for algebraic specifications of the two type theories [43, 15] – one could in principle work directly at this algebraic level, but formalization of such developments in proof assistants is still a challenge [9]. Nevertheless, in order to avoid clutter, we allow ourselves to informally omit these annotations in some cases, *e.g.,* writing $t\ u$ for application and $\lambda x.\, t$ for abstraction. In particular, for equality $\mathrm{Eq}^{\mathfrak{n}}(A, x, y)$, we omit the $\mathfrak{n}$ superscript when the carrier type A is of the form $\mathcal{U}_\ell$, similarly to our convention for level annotations on the typing judgment.

[2] In Rocq syntax, one writes $\mathsf{Type}_{\mathfrak{n}}$ for $\mathcal{U}_{\mathfrak{n}}$ and SProp for $\mathcal{U}_\Omega$.

$$\textsc{Ctx-Nil}\ \frac{}{\vdash \cdot} \qquad \textsc{Ctx-Cons}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A : \mathcal{U}_\ell}{\vdash \Gamma, x :^\ell A} \qquad \textsc{Var}\ \frac{\vdash \Gamma \qquad (x :^\ell A) \in \Gamma}{\Gamma \vdash^\ell x : A} \qquad \textsc{Ax}\ \frac{\vdash \Gamma \qquad (c : A) \in \Sigma \qquad \cdot \vdash A : \mathcal{U}_\Omega}{\Gamma \vdash^\Omega c : A}$$

$$\textsc{Conv}\ \frac{\Gamma \vdash^\ell t : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}_\ell}{\Gamma \vdash^\ell t : B} \qquad \textsc{Univ}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathcal{U}_\ell : \mathcal{U}_{\ell+}} \qquad \Pi\textsc{-Form}\ \frac{\Gamma \vdash A : \mathcal{U}_\ell \qquad \Gamma, x : A \vdash B : \mathcal{U}_{\ell'}}{\Gamma \vdash \Pi_{\ell,\ell'}(x : A).\, B : \mathcal{U}_{\ell \ltimes \ell'}}$$

$$\textsc{Fun}\ \frac{\Gamma \vdash A : \mathcal{U}_\ell \qquad \Gamma, x : A \vdash B : \mathcal{U}_{\ell'} \qquad \Gamma, x : A \vdash^{\ell'} t : B}{\Gamma \vdash^{\ell \ltimes \ell'} \lambda^{x:A.B}_{\ell,\ell'} x.\, t : \Pi_{\ell,\ell'}(x : A).\, B} \qquad \textsc{App}\ \frac{\Gamma \vdash A : \mathcal{U}_\ell \qquad \Gamma, x : A \vdash B : \mathcal{U}_{\ell'} \qquad \Gamma \vdash^{\ell \ltimes \ell'} t : \Pi_{\ell,\ell'}(x : A).\, B \qquad \Gamma \vdash^\ell u : A}{\Gamma \vdash^{\ell'} t @^{x:A.B}_{\ell,\ell'} u : B[x := u]}$$

$$\mathbb{N}\textsc{-Form}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N} : \mathcal{U}_0} \qquad \textsc{Zero}\ \frac{\vdash \Gamma}{\Gamma \vdash^0 0 : \mathbb{N}} \qquad \textsc{Succ}\ \frac{\Gamma \vdash^0 n : \mathbb{N}}{\Gamma \vdash^0 \mathrm{S}(n) : \mathbb{N}} \qquad \textsc{Eq-Form}\ \frac{\Gamma \vdash A : \mathcal{U}_\mathtt{n} \qquad \Gamma \vdash^\mathtt{n} a : A \qquad \Gamma \vdash^\mathtt{n} b : A}{\Gamma \vdash \mathrm{Eq}^\mathtt{n}(A, a, b) : \mathcal{U}_\Omega}$$

$$\mathbb{N}\textsc{-Elim}\ \frac{\Gamma, m : \mathbb{N} \vdash P : \mathcal{U}_\ell \qquad \Gamma \vdash^\ell t_0 : P[m := 0] \qquad \Gamma, n : \mathbb{N}, x : P[m := n] \vdash^\ell t_\mathrm{S} : P[m := \mathrm{S}(n)] \qquad \Gamma \vdash^0 n : \mathbb{N}}{\Gamma \vdash^\ell \mathbb{N}\text{-el}^\ell(m.P, t_0, nx.t_\mathrm{S}, n) : P[m := n]} \qquad \textsc{Refl}\ \frac{\Gamma \vdash A : \mathcal{U}_\mathtt{n} \qquad \Gamma \vdash^\mathtt{n} a : A}{\Gamma \vdash^\Omega \mathrm{refl}^\mathtt{n}(a, A) : \mathrm{Eq}^\mathtt{n}(A, a, a)}$$

$$\textsc{Cast}\ \frac{\Gamma \vdash A : \mathcal{U}_\ell \qquad \Gamma \vdash B : \mathcal{U}_\ell \qquad \Gamma \vdash^\Omega e : \mathrm{Eq}(\mathcal{U}_\ell, A, B) \qquad \Gamma \vdash^\ell a : A}{\Gamma \vdash^\ell \mathrm{cast}^\ell(A, B, e, a) : B}$$

$$\textsc{Transp}\ \frac{\Gamma \vdash A : \mathcal{U}_\mathtt{n} \qquad \Gamma \vdash^\mathtt{n} a : A \qquad \Gamma, x : A \vdash P : \mathcal{U}_\Omega \qquad \Gamma \vdash^\Omega p : P[x := a] \qquad \Gamma \vdash^\mathtt{n} b : A \qquad \Gamma \vdash^\Omega e : \mathrm{Eq}^\mathtt{n}(A, a, b)}{\Gamma \vdash^\Omega \mathrm{transp}^\mathtt{n}(A, a, x.P, p, b, e) : P[x := b]}$$

$$\textsc{Eq-}\Pi_1\ \frac{\Gamma \vdash A_i : \mathcal{U}_\ell \qquad \Gamma, x : A_i \vdash B_i : \mathcal{U}_\mathtt{n} \quad (\text{for } i = 1, 2) \qquad \Gamma \vdash^\Omega e : \mathrm{Eq}(\mathcal{U}_{\ell \ltimes \mathtt{n}}, \Pi(x : A_1).\, B_1, \Pi(x : A_2).\, B_2)}{\Gamma \vdash^\Omega \Pi^{\ell,\mathtt{n}}_{\mathrm{inj1}}(A_1, x.B_1, A_2, x.B_2, e) : \mathrm{Eq}(\mathcal{U}_\ell, A_2, A_1)}$$

$$\textsc{Eq-}\Pi_2\ \frac{\begin{array}{c}\Gamma \vdash A_i : \mathcal{U}_\ell \qquad \Gamma, x : A_i \vdash B_i : \mathcal{U}_\mathtt{n} \quad (\text{for } i = 1, 2) \\ \Gamma \vdash^\Omega e : \mathrm{Eq}(\mathcal{U}_{\ell \ltimes \mathtt{n}}, \Pi(x : A_1).\, B_1, \Pi(x : A_2).\, B_2) \qquad \Gamma \vdash^\ell a_2 : A_2 \\ a_1 := \mathrm{cast}^\ell(A_2, A_1, \Pi^{\ell,\mathtt{n}}_{\mathrm{inj1}}(A_1, x.B_1, A_2, x.B_2, e), a_2)\end{array}}{\Gamma \vdash^\Omega \Pi^{\ell,\mathtt{n}}_{\mathrm{inj2}}(A_1, x.B_1, A_2, x.B_2, e, a_2) : \mathrm{Eq}(\mathcal{U}_\mathtt{n}, B_1[x := a_1], B_2[x := a_2])}$$

$$\textsc{Acc-Form}\ \frac{\Gamma \vdash A : \mathcal{U}_\mathtt{n} \qquad \Gamma \vdash^\mathtt{n} a : A \qquad \Gamma, x : A, y : A \vdash R : \mathcal{U}_\Omega}{\Gamma \vdash \mathrm{Acc}^\mathtt{n}(A, xy.R, a) : \mathcal{U}_\Omega} \qquad \textsc{Acc-in}\ \frac{\Gamma \vdash A : \mathcal{U}_\mathtt{n} \qquad \Gamma \vdash^\mathtt{n} a : A \qquad \Gamma, x : A, y : A \vdash R : \mathcal{U}_\Omega \qquad \Gamma \vdash^\Omega p : \Pi(b : A).\, b \prec_R a \to \mathrm{Acc}^\mathtt{n}(A, xy.R, b)}{\Gamma \vdash^\Omega \mathrm{acc\text{-}in}^\mathtt{n}(A, xy.R, a, p) : \mathrm{Acc}^\mathtt{n}(A, xy.R, a)}$$

$$\textsc{Acc-inv}\ \frac{\Gamma \vdash A : \mathcal{U}_\mathtt{n} \qquad \Gamma \vdash^\mathtt{n} a : A \qquad \Gamma, x : A, y : A \vdash R : \mathcal{U}_\Omega \qquad \Gamma \vdash^\Omega p : \mathrm{Acc}^\mathtt{n}(A, xy.R, a) \qquad \Gamma \vdash^\mathtt{n} b : A \qquad \Gamma \vdash^\Omega r : b \prec_R a}{\Gamma \vdash^\Omega \mathrm{acc\text{-}inv}^\mathtt{n}(A, xy.R, a, p, b, r) : \mathrm{Acc}^\mathtt{n}(A, xy.R, b)}$$

Figure 1 Common typing rules for $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ ( ), part 1.

$$\textsc{Acc-el}\quad \frac{\Gamma \vdash A : \mathcal{U}_{\mathtt{n}} \qquad \Gamma \vdash^{\mathtt{n}} a : A \qquad \Gamma, x : A, y : A \vdash R : \mathcal{U}_{\Omega} \qquad \Gamma \vdash^{\Omega} q : \mathrm{Acc}^{\mathtt{n}}(A, xy.R, a) \qquad \Gamma, x : A \vdash P : \mathcal{U}_{\ell} \qquad \Gamma, a : A, z : \Pi(b : A).\, b \prec_R a \to P[x := b] \vdash^{\ell} p : P[x := a]}{\Gamma \vdash^{\ell} \mathrm{acc\text{-}el}^{\mathtt{n},\ell}(A, xy.R, a, x.P, az.p, q) : P[x := a]}$$

$$\textsc{Acc-el-prop}\quad \frac{\Gamma \vdash A : \mathcal{U}_{\mathtt{n}} \qquad \Gamma \vdash^{\mathtt{n}} a : A \qquad \Gamma, x : A, y : A \vdash R : \mathcal{U}_{\Omega} \qquad \Gamma \vdash^{\Omega} q : \mathrm{Acc}(A, xy.R, a) \qquad \Gamma, x : A \vdash P : \mathcal{U}_{\mathtt{n}'} \qquad \Gamma, a : A, z : \Pi(b : A).\, b \prec_R a \to P[x := b] \vdash^{\mathtt{n}'} p : P[x := a] \qquad f := \lambda br.\, \mathrm{acc\text{-}el}^{\mathtt{n},\mathtt{n}'}(A, xy.R, b, x.P, az.p, \mathrm{acc\text{-}inv}^{\mathtt{n}}(A, xy.R, a, q, b, r))}{\Gamma \vdash^{\Omega} \mathrm{acc\text{-}el\text{-}comp}^{\mathtt{n},\mathtt{n}'}(A, xy.R, a, x.P, az.p, q) : \mathrm{Eq}(P[x := a], \mathrm{acc\text{-}el}^{\mathtt{n},\mathtt{n}'}(A, xy.R, a, x.P, az.p, q), p[z := f, a := a])}$$

Figure 2 Common typing rules for $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ (), part 2.

$$\textsc{Sym}\quad \frac{\Gamma \vdash^{\ell} t \equiv u : A}{\Gamma \vdash^{\ell} u \equiv t : A} \qquad\qquad \textsc{Trans}\quad \frac{\Gamma \vdash^{\ell} t \equiv u : A \qquad \Gamma \vdash^{\ell} u \equiv v : A}{\Gamma \vdash^{\ell} t \equiv v : A}$$

$$\textsc{Proof-Irr}\quad \frac{\Gamma \vdash^{\Omega} t : A \qquad \Gamma \vdash^{\Omega} u : A}{\Gamma \vdash^{\Omega} t \equiv u : A} \qquad\qquad \textsc{Conv-Conv}\quad \frac{\Gamma \vdash^{\ell} t \equiv u : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}_{\ell}}{\Gamma \vdash^{\ell} t \equiv u : B}$$

$$\eta\textsc{-Eq}\quad \frac{\Gamma \vdash A : \mathcal{U}_{\ell} \qquad \Gamma, x : A \vdash B : \mathcal{U}_{\ell'} \qquad \Gamma \vdash^{\ell \ltimes \ell'} f_i : \Pi(x : A).\, B \quad (\text{for } i = 1, 2) \qquad \Gamma, x : A \vdash^{\ell'} f_1\ x \equiv f_2\ x : B}{\Gamma \vdash^{\ell \ltimes \ell'} f_1 \equiv f_2 : \Pi(x : A).\, B}$$

$$\beta\textsc{-conv}\quad \frac{\Gamma \vdash A : \mathcal{U}_{\ell} \qquad \Gamma, x : A \vdash B : \mathcal{U}_{\ell'} \qquad \Gamma, x : A \vdash^{\ell'} t : B \qquad \Gamma \vdash^{\ell} u : A}{\Gamma \vdash^{\ell'} (\lambda^{x:A.B}_{\ell,\ell'} x.\, t) @^{x:A.B}_{\ell,\ell'} u \equiv t[x := u] : B[x := u]}$$

$$\mathbb{N}\textsc{-Elim-Succ}\quad \frac{\Gamma, m : \mathbb{N} \vdash P : \mathcal{U}_{\ell} \qquad \Gamma \vdash^{\ell} t_0 : P[m := 0] \qquad \Gamma, n : \mathbb{N}, x : P[m := n] \vdash^{\ell} t_{\mathrm{S}} : P[m := \mathrm{S}(n)] \qquad \Gamma \vdash n : \mathbb{N} \qquad t_n := \mathbb{N}\text{-}\mathrm{el}^{\ell}(m.P, t_0, nx.t_{\mathrm{S}}, n)}{\Gamma \vdash^{\ell} \mathbb{N}\text{-}\mathrm{el}^{\ell}(m.P, t_0, nx.t_{\mathrm{S}}, \mathrm{S}(n)) \equiv t_{\mathrm{S}}[n := n, x := t_n] : P[m := \mathrm{S}(n)]}$$

$$\mathbb{N}\textsc{-Elim-Zero}\quad \frac{\Gamma, m : \mathbb{N} \vdash P : \mathcal{U}_{\ell} \qquad \Gamma \vdash^{\ell} t_0 : P[m := 0] \qquad \Gamma, n : \mathbb{N}, x : P[m := n] \vdash^{\ell} t_{\mathrm{S}} : P[m := \mathrm{S}(n)]}{\Gamma \vdash^{\ell} \mathbb{N}\text{-}\mathrm{el}^{\ell}(m.P, t_0, nx.t_{\mathrm{S}}, 0) \equiv t_0 : P[m := 0]}$$

$$\textsc{Cast-Refl}\quad \frac{\Gamma \vdash A : \mathcal{U}_{\ell} \qquad \Gamma \vdash^{\Omega} e : \mathrm{Eq}(\mathcal{U}_{\ell}, A, A) \qquad \Gamma \vdash^{\ell} t : A}{\Gamma \vdash^{\ell} \mathrm{cast}^{\ell}(A, A, e, t) \equiv t : A}$$

$$\textsc{Cast-}\Pi\quad \frac{\Gamma \vdash A_i : \mathcal{U}_{\ell} \qquad \Gamma, x : A_i \vdash B_i : \mathcal{U}_{\mathtt{n}} \quad (\text{for } i = 1, 2) \qquad \Gamma \vdash^{\Omega} e : \mathrm{Eq}(\mathcal{U}_{\ell \ltimes \mathtt{n}}, \Pi(x : A_1).\, B_1, \Pi(x : A_2).\, B_2) \qquad \Gamma \vdash^{\ell \ltimes \mathtt{n}} f : \Pi(x : A_1).\, B_1 \qquad a_1 := \mathrm{cast}^{\ell}(A_2, A_1, \Pi_{\mathrm{inj1}}(A_1, x.B_1, A_2, x.B_2, e), a_2) \qquad e' := \Pi_{\mathrm{inj2}}(A_1, x.B_1, A_2, x.B_2, e, a_2)}{\Gamma \vdash^{\ell \ltimes \mathtt{n}} \mathrm{cast}^{\ell \ltimes \mathtt{n}}(\Pi(x : A_1).\, B_1, \Pi(x : A_2).\, B_2, e, f) \equiv \lambda a_2.\mathrm{cast}^{\mathtt{n}}(B_1[x := a_1], B_2[x := a_2], e', f\ a_1) : \Pi(x : A_2).\, B_2}$$

Figure 3 Common conversion rules for $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ (congruence rules omitted) ().

Axioms. The type system is actually parametrized by a *signature* Σ (), containing the declaration of axioms $c : P$, for P a closed type in $\mathcal{U}_\Omega$, and that can be introduced by the rule Ax. Although Σ can be freely extended by users, we will assume that it contains at least the axiom Π_{ext} for *function extensionality*, used in the proof of Section 4, stating that, for all f_1, f_2 of type $\Pi(x : A).\, B$, pointwise equality $\Pi(x : A).\, \mathrm{Eq}(B, f_1\ x, f_2\ x)$ implies function equality $\mathrm{Eq}(\Pi(x : A).\, B, f_1, f_2)$.

As we will see later, the main restriction on the declared axioms is that they should be validated in the set-theoretic model of Section 5. This is true for function extensionality but also for *proposition extensionality*, which states that logically equivalent propositions are propositionally equal. Therefore, although none of the results of our paper rely on propositional extensionality, they all support the declaration of this axiom in the signature. Additionally, we assume that the type of any axiom declared in Σ is well formed in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ iff it is well-formed in $\mathcal{T}^{=}_{\mathrm{Acc}}$ (), an assumption that is validated by the common axioms one uses in practice, such as function extensionality, proposition extensionality and excluded middle.

Observational equality. In order to be compatible with definitional proof irrelevance, our theories use *observational equality* [10, 40]. The gist of this flavor of equality is that it is eliminated *via* a type coercion operator (cast) that computes by case analysis on types, instead of the usual J eliminator that computes by discriminating the equality proof – which is unreasonable in the presence of definitional proof irrelevance.

Type coercions between two identical non-parametrized types (between $\mathbb{N}$ and itself, or $\mathcal{U}_\ell$ and itself) can be simply erased, as a consequence of rule Cast-Refl. When applied between two function types, cast returns a function which first casts its argument to the domain of the original function, invokes it, and then casts the result back to the expected type (as specified in Cast-Π). Here, we draw the reader's attention to an important subtlety: since cast computes by making recursive calls on subtypes, we must impose that type formers in proof-relevant universes ($\mathcal{U}_\mathfrak{n}$ with $\mathfrak{n} = 0, 1, \dots$) are injective for propositional equality. In the case of function types, this is achieved by the symbols Π_{inj1} and Π_{inj2}, used in rule Cast-Π.

Aside from cast, observational equality supports a transp operator for eliminating into proof irrelevant types. From cast and transp, it is possible to derive the usual J eliminator, which computes as expected thanks to rule Cast-Refl [40, 39].

▶ Remark 1. Unlike Altenkirch *et al.*'s presentation of observational equality [10], our equality type itself is not equipped with computation rules. For instance, the type of equalities between two functions is not *definitionally equal* to the type of pointwise equalities, but only logically equivalent to it (as in the presentation of [39, 11]). We argue that the resulting equality type still deserves the name of observational equality, as it keeps its defining characteristics which are the cast operator, along with its computation rules and the injectivity of type formers.

Accessibility. To support definitions by well-founded recursion, our theory features an *accessibility* predicate. Given a term a of type $A : \mathcal{U}_\ell$ and a binary relation R on this type, the type $\mathrm{Acc}^\ell(A, R, a)$ asserts that a is *accessible* – a constructive form of well-foundedness – for the relation R. Importantly, its eliminator acc-el may target proof-relevant types P in arbitrary universes, as can be seen in the typing rule Acc-el (there and in the other rules, we write $b \prec_R a$ instead of $R[x := b, y := a]$ to avoid clutter).

Both theories $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ and $\mathcal{T}^{=}_{\mathrm{Acc}}$ feature the rule Acc-el-prop, stating the computation rule for acc-el propositionally. The only difference between the two theories is that $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ additionally features the following rule Acc-el-def, including the computation rule as part of the conversion.[3]

[3] The symbol acc-el-comp is therefore redundant in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$, yet we chose to have it so that $\mathcal{T}^{=}_{\mathrm{Acc}}$ is included in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$, easing the statement of conservativity.

$$\dfrac{\textsc{Acc-el-def} \qquad \text{(same premises as } \textsc{Acc-el-prop}\text{, replacing } \vdash \text{ with } \vdash_{\equiv}\text{)}}{\Gamma \vdash_{\equiv}^{\mathfrak{n}'} \text{acc-el}^{\mathfrak{n},\mathfrak{n}'}(A, xy.R, a, x.P, az.p, q) \equiv p[z := f, a := a] : P[x := a]}$$

Aside from the introduction and elimination rules for Acc, both theories also include a symbol acc-inv which, given a proof of that a is accessible, asserts that all b smaller than a are also accessible. This fact is derivable from acc-el, so acc-inv is actually redundant. Yet, because acc-inv is used to state rules Acc-el-def and Acc-el-prop, its addition simplifies writing them, in particular in the formalization.

Metatheoretic properties. We formally prove in Rocq that both theories satisfy the basic properties expected of all type theories: weakening (), substitution (), as well as *validity* (), asserting that $\Gamma \vdash^{\ell} t : A$ implies $\vdash \Gamma$ and $\Gamma \vdash A : \mathcal{U}_{\ell}$, and that $\Gamma \vdash^{\ell} t_1 \equiv t_2 : A$ implies $\Gamma \vdash^{\ell} t_i : A$ for $i = 1, 2$.

Moreover, aside from minor syntactic differences, $\mathcal{T}_{\mathrm{Acc}}^{=}$ is an extension of $\mathrm{CIC}_{\mathrm{obs}}$ with new symbols, but, crucially, no equations (aside from congruence rules). Therefore, *decidability of typing* and *injectivity of function types* of $\mathcal{T}_{\mathrm{Acc}}^{=}$ directly follow from the fact that these properties hold for $\mathrm{CIC}_{\mathrm{obs}}$ [39]: indeed, the extra symbols of $\mathcal{T}_{\mathrm{Acc}}^{=}$ can be understood in $\mathrm{CIC}_{\mathrm{obs}}$ simply as extra variables in the context. Importantly, these properties also ensure that, in the setting of a practical implementation, a user-friendly unannotated syntax can be correctly elaborated into the core verbose syntax, using *bidirectional typing* [18, 28].

3 Canonicity of $\mathcal{T}_{\mathrm{Acc}}^{\equiv}$

In this section we prove that the theory $\mathcal{T}_{\mathrm{Acc}}^{\equiv}$ enjoys canonicity, despite its conversion being undecidable. For this, we adapt the logical relation for decidability of typing of Abel *et al.* [4], but strip out all the machinery for handling open terms and neutrals, which are not needed for canonicity.

▶ Convention 3.1. In this section, all typing judgments refer to the theory $\mathcal{T}_{\mathrm{Acc}}^{\equiv}$, so we just write $\vdash^{\ell}$ instead of $\vdash_{\equiv}^{\ell}$.

Metatheory and assumptions. This proof takes place in CIC, as witnessed by the Rocq formalization, however it relies on two assumptions which are provable in set theory, as shown in Section 5. As a result, the metatheory required for this proof is really set theory, and we are implicitly using CIC as an internal language for set theory.

In order to state the aforementioned assumptions, let us say that a term A is a *canonical relevant type* if $A = \mathbb{N}$ or $A = \mathcal{U}_{\ell}$ or $A = \Pi_{\ell,\mathfrak{n}}(x : T).\, U$ (note that we ask the codomain level to be different from Ω, otherwise the whole type would live in $\mathcal{U}_{\Omega}$). We then say that two canonical relevant types A and B have the same head when $A = B = \mathbb{N}$ or $A = B = \mathcal{U}_{\ell}$ or $A = \Pi_{\ell,\mathfrak{n}}(x : T).\, U$ and $B = \Pi_{\ell',\mathfrak{n}'}(x : T').\, U'$ with $\ell = \ell'$ and $\mathfrak{n} = \mathfrak{n}'$.

Assumption A. () If A and B are canonical proof-relevant types that do not have the same head, then for no e we have $\vdash e : \mathrm{Eq}(\mathcal{U}_{\ell}, A, B)$.

Assumption B. () If $\vdash e : \mathrm{Acc}^{\mathfrak{n}}(A, xy.R, a)$ for some e, then the relation $\widetilde{R}$ that is defined on closed terms of type A by $\widetilde{R}(t, u)$ iff $\exists p. \vdash p : t \prec_R u$ is well founded for a.

In Section 5 we will show that these assumptions can be justified by defining a model of $\mathcal{T}^{\equiv}_{\text{Acc}}$ in an intuitionistic fragment of set theory ($\mathsf{IZF_R} + \omega$ Grothendieck universes), assuming that the user-specified axioms in Σ are validated in the set theoretic model. Viewing CIC as an internal language for sets therefore yields an assumption-free proof.[4]

Reduction. Following Abel *et al.* [4], to define the logical relation we first need to consider a deterministic weak-head reduction judgment, written $\Gamma \vdash t \longrightarrow u : A$. We omit the definition here for brevity, but most of its rules are adapted from the main conversion rules for $\mathcal{T}^{\equiv}_{\text{Acc}}$. Let us however mention that, because of the use of fully annotated syntax, the previous rule for β-equality needs also to be further adapted to allow type annotations to differ, as long as they remain convertible. We therefore have the following reduction rule:

$$\beta\text{-}\textsc{conv}\quad \frac{\Gamma \vdash A \equiv A' : \mathcal{U}_\ell \qquad \Gamma, x : A \vdash B \equiv B' : \mathcal{U}_{\ell'} \qquad \Gamma, x : A \vdash^{\ell'} t : B \qquad \Gamma \vdash^{\ell} u : A}{\Gamma \vdash^{\ell'} (\lambda^{x:A'.B'}_{\ell,\ell'} x.\, t)@^{x:A.B}_{\ell,\ell'} u \longrightarrow t[x := u] : B[x := u]}$$

We then write $\Gamma \vdash t \longrightarrow\!\!\!\!\!\rightarrow u : A$ for the reflexive-transitive closure of reduction, and $\Gamma \vdash t \searrow u : A$ when $\Gamma \vdash t \longrightarrow\!\!\!\!\!\rightarrow u : A$ and u is irreducible.

The logical relation. We define the logical relation as a predicate $\Vdash^{\ell} A \equiv B \downarrow R$, asserting that A and B are *reducibly convertible* at the universe $\mathcal{U}_\ell$, and yielding a reducibility binary relation R for terms of type A or B. Intuitively, this statement should assert that A and B hereditarily reduce to the same canonical type, whose inhabitants are characterized by the term-level reducibility relation R.

To define the logical relation, we first define the following auxiliary relations, which capture the term-level reducibility relations for elements of the function and natural number types. The relation $\hat{\mathbb{N}}$ specifies that two terms t, u are reducibly convertible natural numbers when they iteratively reduce to the same canonical natural number. The relation $\hat{\Pi}_{\ell,\mathfrak{n}}[T_1, T_2, R_T, U_1, U_2, R_U]$ specifies that two terms f_1, f_2 are reducibly convertible functions when they are convertible and, omitting annotations, $f_1\ t_1$ and $f_2\ t_2$ are related by $R_U[t_1, t_2]$ whenever t_1, t_2 are related by R_T. The relation R_U is thus parametrized over pairs of terms.

$$\frac{\vdash t_i \searrow \mathsf{S}(u_i) : \mathbb{N} \qquad \hat{\mathbb{N}}(u_1, u_2)}{\hat{\mathbb{N}}(t_1, t_2)} \qquad\qquad \frac{\vdash t_i \searrow 0 : \mathbb{N}}{\hat{\mathbb{N}}(t_1, t_2)}$$

$$\frac{\vdash f_1 \equiv f_2 : \Pi_{\ell,\mathfrak{n}}(x : T_1).\, U_1 \qquad \forall t_1 t_2.\ R_T(t_1, t_2) \Rightarrow R_U[t_1, t_2](f_1 @^{x:T_1.U_1}_{\ell,\mathfrak{n}} t_1, f_2 @^{x:T_2.U_2}_{\ell,\mathfrak{n}} t_2)}{\hat{\Pi}_{\ell,\mathfrak{n}}[T_1, T_2, R_T, U_1, U_2, R_U](f_1, f_2)}$$

Next, we define $\Vdash^{\ell} A_1 \equiv A_2 \downarrow R$ by the following rules. The definition is actually also by well-founded induction over ℓ (considering Ω as smaller than all the predicative levels), so we assume this judgment to be already defined for all smaller ℓ. In these rules, we write $R \leftrightarrow R'$ when the relations R and R' are equivalent.

[4] We conjecture that this proof could be done fully in CIC by using *e.g.* setoids to simulate extensionality, but we leave this for future research.

$$\frac{\vdash A_1 \equiv A_2 : \mathcal{U}_\Omega \qquad R \leftrightarrow \{(t_1, t_2) \mid \vdash^\Omega t_1 \equiv t_2 : A_1\}}{\Vdash^\Omega A_1 \equiv A_2 \downarrow R}$$

$$\frac{\vdash A_i \searrow \mathcal{U}_\ell : \mathcal{U}_{\ell+} \qquad R \leftrightarrow \{(B_1, B_2) \mid \exists R'.\, \Vdash^\ell B_1 \equiv B_2 \downarrow R'\}}{\Vdash^{\ell^+} A_1 \equiv A_2 \downarrow R} \qquad \frac{\vdash A_i \searrow \mathbb{N} : \mathcal{U}_0 \qquad R \leftrightarrow \hat{\mathbb{N}}}{\Vdash^0 A_1 \equiv A_2 \downarrow R}$$

$$\frac{\begin{array}{c}\vdash A_i \searrow \Pi_{\ell,\mathfrak{n}}(x : T_i).\, U_i : \mathcal{U}_{\ell \ltimes \mathfrak{n}} \qquad x : T_1 \vdash U_1 \equiv U_2 : \mathcal{U}_{\mathfrak{n}} \\ \Vdash^\ell T_1 \equiv T_2 \downarrow R_T \qquad \forall t_1 t_2.\ R_T(t_1, t_2) \Rightarrow\ \Vdash^{\mathfrak{n}} U_1[x := t_1] \equiv U_2[x := t_2] \downarrow R_U[t_1, t_2] \\ R \leftrightarrow \hat{\Pi}_{\ell,\mathfrak{n}}[T_1, T_2, R_T, U_1, U_2, R_U]\end{array}}{\Vdash^{\ell \ltimes \mathfrak{n}} A_1 \equiv A_2 \downarrow R}$$

Because we are in a setting with definitional proof irrelevance, we do not need to evaluate proofs – and, with proposition extensionality, this would even be impossible, given the result of Abel and Coquand [1] – and so the logical relation at $\ell = \Omega$ only asks for convertibility (which, by proof irrelevance, is equivalent to asking $\vdash^\Omega t_i : A_1$ for $i = 1, 2$). For relevant types, the rules state that A_1 and A_2 are reducibly convertible when they reduce to the same type former, and the associated relation R is equivalent to the expected one. In the case of function types, we additionally ask for T_1 and T_2 to be reducibly convertible with relation R_T, and for $U_1[x := t_1]$ and $U_2[x := t_2]$ to be reducibly convertible with relation $R_U[t_1, t_2]$ for all t_1, t_2 related by R_T. In the case for the universe type, the associated relation R relates B_1, B_2 iff we have $\Vdash^\ell B_1 \equiv B_2 \downarrow R'$ for some R'. This definition therefore mentions $\Vdash$ in a non-strictly positive way, yet this is justified because it is used with ℓ which is strictly smaller than ℓ^+, and so $\Vdash^\ell$ is already fully defined at this stage. Finally, note that, interestingly, there are no rules for handling accessibility and equality, given that such types live in $\mathcal{U}_\Omega$.

▶ Remark 2. Compared with Abel *et al.* [4] and Pujet *et al.* [39], we only define the binary version of the logical relation, and we define it as an inductive predicate instead of relying on *induction-recursion*; two changes which, we believe, make the definition shorter and easier to understand. Moreover, unlike the aforementioned work whose logical relation only handles a finite number of universes, ours relies on impredicativity in order to handle the whole infinite universe hierarchy – in the formalization, the relation is placed in Rocq's Prop. On these specific aspects, our definitions are hence closer to the ones of Gratzer *et al.* [22] and Jang *et al.* [25].

Validity. As usual with canonicity proofs, we use the logical relation to define a model for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ that will allow us to directly derive canonicity. Of course, the model cannot interpret conversion simply by $\exists R.\, \Vdash A \equiv B \downarrow R$. Indeed, a model needs to be defined for all terms, including *open* terms. The correct interpretation is instead captured by the notion of *validity*. For its definition, we first need to extend the logical relation to substitutions, in the following manner ().

$$\frac{}{\Vdash \cdot \equiv \cdot : (\cdot)} \qquad \frac{\Vdash \sigma_1 \equiv \sigma_2 : \Gamma \qquad \Vdash^\ell A[\sigma_1] \equiv A[\sigma_2] \downarrow R \qquad R(t_1, t_2)}{\Vdash (\sigma_1, x := t_1) \equiv (\sigma_2, x := t_2) : \Gamma, x :^\ell A}$$

Now, the validity judgments () can be defined as follows.

- $\Gamma \vDash^\ell t \equiv u : A$ holds if, for all σ_1, σ_2, $\Vdash \sigma_1 \equiv \sigma_2 : \Gamma$ implies $\Vdash^\ell A[\sigma_1] \equiv A[\sigma_2] \downarrow R$ and $R(t[\sigma_1], u[\sigma_2])$ for some R.
- $\Gamma \vDash^\ell t : A$ holds if $\Gamma \vDash^\ell t \equiv t : A$ holds.

Our main goal is now to show that the relations $\Gamma \vDash^{\ell} t : A$ and $\Gamma \vDash^{\ell} t \equiv u : A$ are indeed models of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$– this is exactly the *fundamental theorem* of the logical relation.

Basic properties of the logical relation. Before proving the fundamental theorem, we first establish some usual basic properties [4].

▶ **Lemma 3** (Escape). *Suppose* $\Vdash^{\ell} A \equiv B \downarrow R$. *Then we have* $\vdash A \equiv B : \mathcal{U}_{\ell}$. *Moreover,* $R(t, u)$ *implies* $\vdash t \equiv u : A$ *for all* t, u.

The above lemma is the reason we add conversion premises between f_1, f_2 and U_1, U_2 in the rules for functions in the logical relation. Without them, any two terms f_1, f_2 would for instance be related by $\hat{\Pi}$ as soon as the relation R_T is empty (eg., if $T_1 = T_2 = \bot$ with $\bot := \Pi P : \mathcal{U}_{\Omega}.P$, in which case $R_T(t_1, t_2)$ is equivalent to $\vdash t_1 \equiv t_2 : \bot$).

▶ **Lemma 4** (Closure under reduction and anti-reduction /). *Assume* $\Vdash^{\ell} A_1 \equiv A_2 \downarrow R$.

- *If* $\vdash A_i \longrightarrow\!\!\!\!\rightarrow A'_i : \mathcal{U}_{\ell}$ *(for* $i = 1, 2$*) or* $\vdash A'_i \longrightarrow\!\!\!\!\rightarrow A_i : \mathcal{U}_{\ell}$ *(for* $i = 1, 2$*) then* $\Vdash^{\ell} A'_1 \equiv A'_2 \downarrow R$.
- *If* $R(t_1, t_2)$ *and either* $\vdash t_i \longrightarrow\!\!\!\!\rightarrow t'_i : A_1$ *(for* $i = 1, 2$*) or* $\vdash t'_i \longrightarrow\!\!\!\!\rightarrow t_i : A_1$ *(for* $i = 1, 2$*) then* $R(t'_1, t'_2)$.

Compared with prior work, the use of a fully annotated syntax introduces the need for the following annotation conversion result for applications, used to show irrelevance of the logical relation (Lemma 6). The proof of Lemma 5 goes by showing a stronger statement () with a more general annotation conversion relation (), but we omit it here for brevity.

▶ **Lemma 5** (Annotation conversion /). *Assume we have* $\Vdash^{\ell} A \equiv B \downarrow R$ *and* $\vdash T_1 \equiv T_2 : \mathcal{U}_{\ell}$ *and* $x : T_1 \vdash U_1 \equiv U_2 : \mathcal{U}_{\ell'}$. *If* $R(t @^{x:T_1.U_1}_{\ell,\ell'} u, v)$ *then* $R(t @^{x:T_2.U_2}_{\ell,\ell'} u, v)$*, and if* $R(v, t @^{x:T_1.U_1}_{\ell,\ell'} u)$ *then* $R(v, t @^{x:T_2.U_2}_{\ell,\ell'} u)$.

▶ **Lemma 6** (Irrelevance). *If* $\Vdash^{\ell} A \equiv B \downarrow R$ *and* $\Vdash^{\ell} A \equiv B' \downarrow R'$ *then* $R \leftrightarrow R'$.

▶ **Lemma 7** (Symmetry /). *If* $\Vdash^{\ell} A \equiv B \downarrow R$ *then* $\Vdash^{\ell} B \equiv A \downarrow R$ *and* R *is a symmetric relation.*

▶ **Lemma 8** (Transitivity /). *If* $\Vdash^{\ell} A \equiv B \downarrow R$ *and* $\Vdash^{\ell} B \equiv C \downarrow R$ *then* $\Vdash^{\ell} A \equiv C \downarrow R$ *and* R *is a transitive relation.*

Recall that a relation is a *partial equivalent relation (PER)* when it is symmetric and transitive.

▶ **Lemma 9** (Substitution reducibility is a PER /). *The relation* $\Vdash \_ \equiv \_ : \Gamma$ *is a PER for all* Γ.

▶ **Lemma 10** (Validity is a PER /). *The relation* $\Gamma \Vdash^{\ell} \_ \equiv \_ : A$ *is a PER for all* Γ, ℓ, A.

Fundamental theorem. The hardest step of the proof is showing the fundamental theorem, which establishes that the validity judgments correctly interpret the theory $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$.

▶ **Theorem 11** (Fundamental theorem /). *If* $\Gamma \vdash^{\ell} t : A$ *then* $\Gamma \vDash^{\ell} t : A$*, and if* $\Gamma \vdash^{\ell} t \equiv u : A$ *then* $\Gamma \vDash^{\ell} t \equiv u : A$.

The proof of the fundamental theorem consists in showing that each of the typing rules holds when interpreting the judgments using validity. Most rules, such as the ones pertaining to functions and natural numbers, are handled similarly to prior work [4, 39, 6]. The only interesting cases are cast and acc-el. The rules for cast are handled similarly to Pujet *et al.* [39], so we not detail the proof here, but simply mention that it relies on Assumption A.[5]

Handling acessibility. Let us now discuss the main novel aspect of this section: handling of the eliminator of accessibility in the proof of the fundamental theorem (Theorem 11). To illustrate the difficulty of handling acc-el, let us first explain the usual strategy for handling eliminators for positive types, focusing on the eliminator of naturals, $\mathbb{N}$-el.

To prove that $\mathbb{N}$-el is handled by the model, we must prove the following implication, represented here as an inference rule for readability purposes:

$$\frac{\begin{array}{c}\Gamma, m : \mathbb{N} \vDash P : \mathcal{U}_\ell \qquad \Gamma \vDash^\ell t_0 : P[m := 0] \\ \Gamma, n : \mathbb{N}, x : P[m := n] \vDash^\ell t_\mathrm{S} : P[m := \mathrm{S}(n)] \qquad \Gamma \vDash^0 n : \mathbb{N}\end{array}}{\Gamma \vDash^\ell \mathbb{N}\text{-el}^\ell(m.P, t_0, nx.t_\mathrm{S}, n) : P[m := n]}$$

Unfolding the definition of validity, we must show that, for all σ_1, σ_2 satisfying $\Vdash \sigma_1 \equiv \sigma_2 : \Gamma$, the terms $\mathbb{N}\text{-el}(\dots)[\sigma_1]$ and $\mathbb{N}\text{-el}(\dots)[\sigma_2]$ are reducibly convertible. Instantiating $\Gamma \vDash^0 n : \mathbb{N}$ with $\Vdash \sigma_1 \equiv \sigma_2 : \Gamma$, we easily obtain that $\hat{\mathbb{N}}(n[\sigma_1], n[\sigma_2])$.

With $\hat{\mathbb{N}}(n[\sigma_1], n[\sigma_2])$ in hand, we can show the result by induction on its derivation.[6] Indeed, we either have that $\vdash n[\sigma_i] \searrow 0 : \mathbb{N}$ or $\vdash n[\sigma_i] \searrow \mathrm{S}(u_i) : \mathbb{N}$ and $\hat{\mathbb{N}}(u_1, u_2)$. In both cases, we then conclude using the validity premises for t_0 and t_S and closure under anti-reduction, with the successor case also using the induction hypothesis.

Let us see now why the same approach breaks for acc-el.

$$\frac{\begin{array}{c}\Gamma \vDash A : \mathcal{U}_\mathfrak{n} \qquad \Gamma \vDash^\mathfrak{n} a : A \qquad \Gamma, x : A, y : A \vDash R : \mathcal{U}_\Omega \qquad \Gamma \vDash^\Omega q : \mathrm{Acc}(A, xy.R, a) \\ \Gamma, x : A \vDash P : \mathcal{U}_{\ell'} \qquad \Gamma, a : A, z : \Pi(b : A).\, b \prec_R a \to P[x := b] \vDash^{\ell'} p : P[x := a]\end{array}}{\Gamma \vDash^{\ell'} \text{acc-el}^{\mathfrak{n},\ell'}(A, xy.R, a, x.P, az.p, q) : P[x := a]}$$

When instantiating $\Gamma \vDash q : \mathrm{Acc}^\mathfrak{n}(A, xy.R, a)$ with $\Vdash \sigma_1 \equiv \sigma_2 : \Gamma$ we do not get any useful information for doing an induction anymore. Indeed, recall that the logical relation at level $\ell = \Omega$ is defined simply as conversion, so we only get $\vdash q[\sigma_1] \equiv q[\sigma_2] : \mathrm{Acc}^\mathfrak{n}(A, xy.R, a)[\sigma_1]$.

This is where Assumption B plays a critical role, as it allows us to deduce from $\vdash q[\sigma_1] : \mathrm{Acc}^\mathfrak{n}(A, xy.R, a)[\sigma_1]$ that the relation $\widetilde{R[\sigma_1]}$, defined on closed terms of type $A[\sigma_1]$ by $\widetilde{R[\sigma_1]}(t, u)$ iff $\exists p. \vdash p : t \prec_{R[\sigma_1]} u$, is well founded for $a[\sigma_1]$, allowing us to proceed by well-founded induction over it.

With the induction set up, the proof goes essentially in the same spirit as the one for $\mathbb{N}$-el, using the validity premise for p to construct reducibly convertible terms which are reducts of $\text{acc-el}^{\mathfrak{n},\ell'}(A, xy.R, a, x.P, az.p, q)[\sigma_i]$, and then concluding by closure under anti-reduction and the induction hypothesis. As usual with such proofs, the complete argument is full of administrative details – we refer the reader to the formalization for a detailed account ( ).

[5] Assumption A is actually not required by Pujet *et al.* [39] for proving the fundamental theorem, as their logical relation includes neutral terms. On the other hand, for that same reason, their logical relation does not directly imply canonicity. To conclude it, they additionally need to rule out the existence of neutrals in the empty context, a fact that implies our Assumption A, and that they also establish using a set-theoretic model.

[6] Actually, we first need to massage the validity premises for t_0 and t_S using $\Vdash \sigma_1 \equiv \sigma_2 : \Gamma$, but we deliberately simplify the description for clarity.

Canonicity. We obtain canonicity as a direct consequence of the definition of the logical relation and Theorem 11.

▶ **Corollary 12** (Canonicity for $\mathcal{T}^{\equiv}_{\text{Acc}}$). *If* $\vdash t : \mathbb{N}$ *then we have* $\vdash t \equiv \mathrm{S}^n(0) : \mathbb{N}$ *for some* n*.*

Canonicity asserts the existence of a concrete numeral n such that t is convertible to $\mathrm{S}^n(0)$, without saying how to compute it. Of course, in an actual implementation, one would like to employ the usual untyped reduction algorithm operating on unannotated terms, as in Rocq. We thus show an effective version of canonicity justifying this strategy.

For this, we first consider a grammar of unannotated terms in which application is written as $t\ u$, abstraction as $\lambda x.t$, etc (), and define an erasure function $|-|$ () mapping a standard term to its unannotated version. Then, we redefine the previous notion of weak-head reduction for unannotated terms in a totally untyped way as the relation $t \longrightarrow u$ (), from which we define $t \longrightarrow\!\!\!\!\!\rightarrow u$ () as its reflexive-transitive closure. Finally, we define $t \Downarrow n$ () by $t \Downarrow 0$ when we have $t \longrightarrow\!\!\!\!\!\rightarrow 0$, and $t \Downarrow n+1$ when we have both $t \longrightarrow\!\!\!\!\!\rightarrow \mathrm{S}(u)$ and $u \Downarrow n$.

▶ **Corollary 13** (Effective Canonicity for $\mathcal{T}^{\equiv}_{\text{Acc}}$). *If* $\vdash t : \mathbb{N}$ *then* $\vdash t \equiv \mathrm{S}^n(0) : \mathbb{N}$ *for the unique* n *such that* $|t| \Downarrow n$*.*

4 Conservativity of $\mathcal{T}^{\equiv}_{\text{Acc}}$ over $\mathcal{T}^{=}_{\text{Acc}}$

We establish conservativity of $\mathcal{T}^{\equiv}_{\text{Acc}}$ over $\mathcal{T}^{=}_{\text{Acc}}$ using a translation technique inspired by Winterhalter *et al.* [45]. We present the key technical ingredients of the argument, while omitting some details that are fully formalized in Rocq.

Let us first give the high-level plan for the proof. The main idea is to define a translation from $\mathcal{T}^{\equiv}_{\text{Acc}}$ to $\mathcal{T}^{=}_{\text{Acc}}$ in which conversion is translated as propositional equality and the conversion rule is simulated by casting along the obtained equality proof. Translated terms then become decorated with occurrences of cast, and two occurrences of the same term might be decorated differently. We address this by showing the *fundamental lemma* of the translation, ensuring that different decorations of the same term can be shown equal in $\mathcal{T}^{=}_{\text{Acc}}$. With the fundamental lemma and the translation from $\mathcal{T}^{\equiv}_{\text{Acc}}$ to $\mathcal{T}^{=}_{\text{Acc}}$ proven, conservativity follows as a corollary.

Heterogeneous equality. As for Winterhalter *et al.* [45], a key ingredient of our proof is the use of McBride's *heterogeneous equality*, which allows to equate terms at different types. We define $\mathrm{HEq}^{\mathsf{n}}(A, B, a, b)$ in $\mathcal{T}^{=}_{\text{Acc}}$ as[7]

$$\Sigma(e : \mathrm{Eq}(\mathcal{U}_{\mathsf{n}}, A, B)).\ \mathrm{Eq}(B, \mathrm{cast}(A, B, e, a), b)$$

For uniformity reasons, it will be more convenient to have HEq at all levels, so we extend the above definition with $\mathrm{HEq}^{\Omega}(A, B, a, b) := \top$, where $\top := \Pi(P : \mathcal{U}_\Omega).\, P \to P$.

Decorations. As mentioned, our proof employs a *decoration* relation on terms, that is essentially the identity up to cast. Formally, $t \sqsubset u$ () is the least reflexive-transitive relation containing $a \sqsubset \mathrm{cast}^{\ell}(A, B, a, e)$ for all ℓ, A, B, a, e and compatible with all constructors of the syntax. We extend $\sqsubset$ pointwise to contexts (), and write $\sim$ () for the reflexive, symmetric, transitive closure of $\sqsubset$.

[7] Recall that $\mathcal{T}^{=}_{\text{Acc}}$ does not feature dependent sums natively, yet the above one can be constructed with the *impredicative encoding*.

The fundamental lemma. To state the fundamental lemma, let us say that two contexts Γ_1 and Γ_2 are compatible () if they have the same variables (modulo α-renaming) in the same order and annotated with the same levels. In this case, we define $\Gamma_1 * \Gamma_2$ () as the context containing entries $x_1 :^\ell A_1, x_2 :^\ell A_2, \hat{x} : \mathrm{HEq}^\ell(A_1, A_2, x_1, x_2)$ for each $x :^\ell A_i \in \Gamma_i$. We now come to the fundamental lemma:

▶ **Lemma 14** (Fundamental lemma). *Assume $t_1 \sim t_2$ and $\Gamma_i \vdash^{\mathtt{n}}_{\equiv} t_i : A_i$ for $i = 1, 2$ and that Γ_1 and Γ_2 are compatible. Then we have $\Gamma_1 * \Gamma_2 \vdash_{=} e : \mathrm{HEq}^{\mathtt{n}}(A_1, A_2, t_1, t_2)$ for some e.*

The proof is done by induction on $t_1 \sim t_2$, and crucially relies on various important properties of HEq – in particular its congruence laws, such as the following one – whose proofs in turn employ function extensionality and proof irrelevance.

$$\begin{aligned}&\mathrm{HEq\text{-}app\text{-}cong} : \mathrm{HEq}(\Pi(x : A_1).\, B_1, \Pi(x : A_2).\, B_2, t_1, t_2) \to \\ &\quad \mathrm{HEq}(A_1, A_2, u_1, u_2) \to \mathrm{HEq}(B_1[x := u_1], B_2[x := u_2], t_1\ u_1, t_2\ u_2)\end{aligned}$$

Constructing all these proof-terms and building their typing derivations explicitly would be simply unfeasible, but thankfully we can leverage all the support provided by Rocq by *working internally to* $\mathcal{T}^{=}_{\mathrm{Acc}}$. Concretely, we postulate sufficiently many primitives of our theory to turn Rocq into a proof assistant for the theory $\mathcal{T}^{=}_{\mathrm{Acc}}$. With the help of user-level features such as implicit arguments and proof mode, showing the required HEq laws becomes a simple exercise (), although we also need to make sure to avoid using features of Rocq that are not covered by $\mathcal{T}^{=}_{\mathrm{Acc}}$. These proofs then justify postulating their external statements ().

Decorating translation. With the fundamental lemma in hand, we now come to the main intermediate result leading to conservativity: the decorating translation. Its proof in turn requires multiple other intermediate lemmas, which are already discussed by Winterhalter *et al.* [45] and can be found in our formalization.

▶ **Lemma 15** (Decorating translation).

- *If $\Gamma \vdash^{\ell}_{\equiv} t : A$, then for all Γ' such that $\Gamma \sqsubset \Gamma'$ and $\vdash_{=} \Gamma'$, we have $\Gamma' \vdash^{\ell}_{=} t' : A'$ for some t', A' satisfying $t \sqsubset t'$, $A \sqsubset A'$.*
- *If $\Gamma \vdash^{\mathtt{n}}_{\equiv} t \equiv u : A$, then for all Γ' such that $\Gamma \sqsubset \Gamma'$ and $\vdash_{=} \Gamma'$, we have $\Gamma' \vdash_{=} e : \mathrm{Eq}^{\mathtt{n}}(A', t', u')$ for some e, t', u', A' satisfying $t \sqsubset t'$, $u \sqsubset u'$, $A \sqsubset A'$.*

Compared with the development of Winterhalter *et al.* [45], the presence of definitional proof irrelevance in the theory simplifies a bit the proof, as the above statement does not need to consider conversions between irrelevant terms ($\Gamma \vdash^{\Omega}_{\equiv} t \equiv u : A$).

Conservativity and canonicity. Together, Lemmas 14 and 15 directly entail conservativity, stating that all $\mathcal{T}^{=}_{\mathrm{Acc}}$ types inhabited in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ are also inhabited in $\mathcal{T}^{=}_{\mathrm{Acc}}$. In particular, all $\mathcal{T}^{=}_{\mathrm{Acc}}$ propositions provable in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ are hence also provable in $\mathcal{T}^{=}_{\mathrm{Acc}}$.

▶ **Theorem 16** (Conservativity of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ over $\mathcal{T}^{=}_{\mathrm{Acc}}$). *If $\Gamma \vdash_{=} A : \mathcal{U}_\ell$ and $\Gamma \vdash^{\ell}_{\equiv} t : A$ then we have $\Gamma \vdash^{\ell}_{=} u : A$ for some u.*

Combined with the (effective) canonicity result for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ (Corollary 13), we now derive its propositional version for $\mathcal{T}^{=}_{\mathrm{Acc}}$– we skip here the non-effective statement (Corollary 12), which follows from the following stronger result.

▶ **Corollary 17** (Propositional canonicity for $\mathcal{T}^{=}_{\mathrm{Acc}}$). *If $\vdash_{=} t : \mathbb{N}$ then, for the unique n such that $|t| \Downarrow n$, we have $\vdash_{=} e : \mathrm{Eq}(\mathbb{N}, t, \mathrm{S}^n(0))$ for some e.*

5 Set-Theoretic Model

In this section, we build a model for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ in set theory, in order to justify not only its consistency, but also Assumptions A and B from Section 3, thereby completing the proof of canonicity. As $\mathcal{T}^{=}_{\mathrm{Acc}}$ is a subset of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$, this also yields a model for $\mathcal{T}^{=}_{\mathrm{Acc}}$. Our construction is mostly adapted from Pujet *et al.* [39]; the main novelty here is that we formalized the main part of the argument.

Metatheory. Our model construction takes place in intuitionistic ZF set theory[8] with a countable hierarchy of Grothendieck universes, embedded in the type theory of Rocq (). Even though ZF is usually presented as a first-order theory, we allow ourselves to use Rocq's higher-order logic in order to simplify the formalization. This is entirely justified, given that the use of higher-order logic is known to be consistent with set theory (assuming one additional inaccessible cardinal) [35]. However, we do not use Rocq's dependent types, meaning that we are effectively working in HOL with ZF axioms.

Concretely, we start by postulating a type `ZFset` and a relation $\in$: `ZFset` → `ZFset` → `Prop`. Then, following Obua [35], we postulate the ZF axioms in Skolemized form, so that for each axiom asserting the existence of a set, we add a new constant and an axiom describing its behavior. For instance, the empty set axiom becomes:

```
Parameter ∅ : ZFset.
Axiom ZFempty : ∀ x, x ∈ ∅ → False.
```

We postulate the Grothendieck universes as an infinite hierarchy of transitive sets $\mathbb{V}_0 \in \mathbb{V}_1 \in \mathbb{V}_2 \in \ldots$ which are closed under the axioms of ZF. We also postulate Russell's definite description operator to convert back and forth between higher-order functions and functional relations.

Constructions in set theory. () From this setup, we perform a series of constructions which will let us use type theory as an internal language for our set theory. We start by reproducing the standard construction of cartesian products and function sets, along with their equations. Next, given `A` : `ZFset` and `B` : `ZFset` → `ZFset`, we define their set-theoretic dependent sum $\Sigma\, A\, B$ and their set-theoretic dependent product $\Pi\, A\, B$, and we use the replacement axiom to prove that $\Pi\, A\, B$ and $\Sigma\, A\, B$ are elements of $\mathbb{V}_n$ whenever $A \in \mathbb{V}_n$ and $\forall x \in A, B\; x \in \mathbb{V}_n$. We define ω as the smallest set containing $\varnothing$ and closed under successor, using the axioms of infinity and comprehension.

The higher-order model. () Since we have access to well-behaved dependent products in our set theory, and since we know that the set-theoretic equality is fully extensional, we can use ZF set theory as a model of a logical framework [23]. We now use this logical framework to construct a *higher-order model* for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$, meaning that we define a family of sets $\Omega, \mathbb{U}_0, \mathbb{U}_1, \mathbb{U}_2 \ldots$ and show that they support all the rules of our theory (seen as a second-order generalized algebraic theory (SOGAT) [26]).

Since $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ features an observational equality, it relies on the injectivity of type formers: from $A \to B = C \to D$, we must be able to deduce $A = C$ and $B = D$. Unfortunately, this is false if we interpret types as plain sets, as $\varnothing \to A$ and $\varnothing \to B$ have the same elements

[8] More precisely, we use $\mathsf{IZF_R}$, that is, intuitionistic ZF with the scheme of replacement instead of the scheme of collection.

regardless of A and B. To sidestep this issue, we interpret types as *triplets* which include their set of elements, an integer encoding their head type former, and a list of subtypes on which we want injectivity. For instance, the type $\mathbb{N}$ will be encoded as $\langle \omega \ ; \ 3 \ ; \ \varnothing \rangle$, where ω is the set of inhabitants of $\mathbb{N}$, the number 3 is a code for the type of natural numbers and $\varnothing$ means that $\mathbb{N}$ does not have any subtypes on which we need injectivity. We interpret the universes and their decoding functions as follows:

$$\Omega := \langle \mathcal{P} \ \{\varnothing\} \ ; \ 0 \ ; \ \varnothing \rangle \qquad \mathtt{El}_\Omega \ A := A$$
$$\mathbb{U}_\mathfrak{n} := \langle \mathbb{V}_n \times \omega \times \mathbb{V}_\mathfrak{n} \ ; \ 1 \ ; \ \varnothing \rangle \qquad \mathtt{El}_\mathfrak{n} \ A := \mathtt{fst} \ A$$

Given $A \in \mathtt{El}_{\mathfrak{n}+1} \ \mathbb{U}_\mathfrak{n}$, we denote its second and third components by $\mathtt{hd} \ A$ and $\mathtt{lbl} \ A$ respectively (the first component having already been defined as $\mathtt{El}_\mathfrak{n} \ A$). From these definitions, it should be clear that we have $\Omega \in \mathtt{El}_0 \ \mathbb{U}_0$ and $\mathbb{U}_\mathfrak{n} \in \mathtt{El}_{\mathfrak{n}+1} \ \mathbb{U}_{\mathfrak{n}+1}$. Furthermore, the inhabitants of Ω are proof-irrelevant: given $A \in \Omega$, all of its inhabitants are equal.

Dependent products. Our higher-order model supports dependent products, which have two separate definitions – one for the predicative case and one for the impredicative case:

$$\widehat{\Pi}_{\ell,\mathfrak{n}} \ A \ B := \langle \Pi \ (\mathsf{El}_\ell \ A) \ (\lambda \, a \, . \, \mathsf{El}_\mathfrak{n} \ (B \ a)) \ ; \ 2 \ ; \ \langle \ell ; \mathfrak{n} ; A ; B \rangle \rangle$$
$$\widehat{\Pi}_{\ell,\Omega} \ A \ B := \mathtt{sub} \ (\forall \, a \in \mathsf{El}_\ell \ A, \mathtt{isTrue} \ (B \ a))$$

where $\mathtt{sub} \ P$ is defined as $\{x \in \{\varnothing\} \mid P\}$, and $\mathtt{isTrue} \ A$ is defined as $\varnothing \in A$. Given two levels ℓ and ℓ' (which may be either an integer $\mathfrak{n}$ or Ω), one may easily see that if $A \in \mathbb{U}_\ell$ and $\forall a \in \mathsf{El}_\ell \ A, B \ a \in \mathbb{U}_{\ell'}$, then we have $\widehat{\Pi}_{\ell,\ell'} \ A \ B \in \mathbb{U}_{\ell \ltimes \ell'}$. The definitions of λ-abstraction, application, and the proofs of β and η are straightforward and found in the formalization.

Remark that the label of $\widehat{\Pi}_{\ell,\mathfrak{n}} \ A \ B$ is defined as a tuple containing ℓ, $\mathfrak{n}$, A and the graph of B. This ensures that whenever two proof-relevant dependent products are equal, their levels, domains and codomains are equal, which is important for interpreting rules Eq-Π_1 and Eq-Π_2 of our theories.

Set-theoretic accessibility. In order to define an accessibility predicate in set theory, we can simply reproduce the standard impredicative encoding of inductive propositions:

```
Definition acc (A : ZFset) (R : Rel A) (a : ZFset) : Prop :=
∀ X ∈ 𝒫 A, (∀ b ∈ A, (∀ c ∈ A, R c b → c ∈ X) → b ∈ X) → a ∈ X.
```

As a consequence of this definition, one can easily show that an element of A is accessible if and only if all of its predecessors under R are accessible. If we want to use this predicate to interpret type-theoretic accessibility, we also need an eliminator. For this purpose, assume that we have a set P, indexed over A, and that we have a set-theoretic function `rec` that outputs an element of $P \ a$ given an element of $P \ b$ for all b such that $R \ b \ a$. Then, we define E to be the smallest subset of $\Sigma \, A \, P$ closed under `rec`, and we use impredicative reasoning to show that E is the graph of a function. Finally, we use definite description to turn E into a higher-order function, and we have our set-theoretic eliminator `acc_el`. We conclude by proving its computational rule. Using the same methods, we can also construct an eliminator for the set of natural numbers ω.

Observational equality and cast. The propositional equality of our theory is interpreted as the set-theoretic equality by defining $\mathtt{eq} \ \mathtt{A} \ \mathtt{t} \ \mathtt{u} := \mathtt{sub} \ (t = u)$. This lets us interpret typecasting as the identity function, which satisfies all the required equations. Additionally, we obtain function extensionality from the extensional encoding of functions in set theory. This concludes the definition of our model.

$$
\begin{aligned}
[\![\ \cdot\ ]\!] &:= \{\varnothing\} \\
[\![\ \Gamma, x :^{\mathtt{n}} A\ ]\!] &:= \Sigma\ [\![\ \Gamma\ ]\!]\ (\gamma \mapsto \mathtt{El}_{\mathtt{n}}\ [\![\ \Gamma \vdash A\ ]\!]_\gamma) \\
[\![\ \Gamma, x :^{\Omega} A\ ]\!] &:= \Sigma\ [\![\ \Gamma\ ]\!]\ (\gamma \mapsto \mathtt{El}_{\Omega}\ [\![\ \Gamma \vdash A\ ]\!]_\gamma)
\end{aligned}
$$

Figure 4 Interpretation of the contexts.

$$
\begin{aligned}
[\![\ \Gamma \vdash x\ ]\!]_\gamma &:= \gamma(x) \\
[\![\ \Gamma \vdash \mathcal{U}_{\mathtt{n}}\ ]\!]_\gamma &:= \mathbb{U}_{\mathtt{n}} \\
[\![\ \Gamma \vdash \mathcal{U}_{\Omega}\ ]\!]_\gamma &:= \Omega \\
[\![\ \Gamma \vdash \Pi_{\ell,\ell'}(x : A).\, B\ ]\!]_\gamma &:= \widehat{\Pi}_{\ell,\ell'}\ [\![\ \Gamma \vdash A\ ]\!]_\gamma\ (a \mapsto [\![\ \Gamma, A \vdash B\ ]\!]_{\gamma,a}) \\
[\![\ \Gamma \vdash \lambda^{x:A.B}_{\ell,\mathtt{n}} x.\, t\ ]\!]_\gamma &:= (a \in \mathtt{El}_\ell\ [\![\ \Gamma \vdash A\ ]\!]_\gamma) \mapsto ([\![\ \Gamma, A \vdash t\ ]\!]_{\gamma,a}) \\
[\![\ \Gamma \vdash t @^{x:A.B}_{\ell,\mathtt{n}} u\ ]\!]_\gamma &:= [\![\ \Gamma \vdash t\ ]\!]_\gamma([\![\ \Gamma \vdash u\ ]\!]_\gamma) \\
[\![\ \Gamma \vdash \mathbb{N}\ ]\!]_\gamma &:= \langle \omega\ ;\ 3\ ;\ \varnothing \rangle \\
[\![\ \Gamma \vdash 0\ ]\!]_\gamma &:= 0 \\
[\![\ \Gamma \vdash \mathrm{S}(n)\ ]\!]_\gamma &:= 1 + [\![\ \Gamma \vdash n\ ]\!]_\gamma \\
[\![\ \Gamma \vdash \mathbb{N}\text{-el}^{\ell}(P, t_0, t_{\mathrm{S}}, n)\ ]\!]_\gamma &:= \mathtt{\omega\_el}\ [\![\ \Gamma \vdash P\ ]\!]_\gamma\ [\![\ \Gamma \vdash t_0\ ]\!]_\gamma\ [\![\ \Gamma \vdash t_{\mathrm{S}}\ ]\!]_\gamma\ [\![\ \Gamma \vdash n\ ]\!]_\gamma \\
[\![\ \Gamma \vdash \mathrm{Eq}(A, t, u)\ ]\!]_\gamma &:= \mathtt{sub}\ ([\![\ \Gamma \vdash t\ ]\!]_\gamma = [\![\ \Gamma \vdash u\ ]\!]_\gamma) \\
[\![\ \Gamma \vdash \mathrm{cast}^{\mathtt{n}}(A, B, e, t)\ ]\!]_\gamma &:= [\![\ \Gamma \vdash t\ ]\!]_\gamma \\
[\![\ \Gamma \vdash \mathrm{Acc}^{\mathtt{n}}(A, R, a)\ ]\!]_\gamma &:= \mathtt{sub}\ (\mathtt{acc}\ [\![\ \Gamma \vdash A\ ]\!]_\gamma\ [\![\ \Gamma \vdash R\ ]\!]_\gamma\ [\![\ \Gamma \vdash a\ ]\!]_\gamma)
\end{aligned}
$$

$$
\begin{aligned}
&[\![\ \Gamma \vdash \mathrm{acc\text{-}el}^{\mathtt{n},\ell}(A, R, a, P, p, q)\ ]\!]_\gamma := \\
&\mathtt{acc\_el}\ [\![\ \Gamma \vdash A\ ]\!]_\gamma\ [\![\ \Gamma \vdash R\ ]\!]_\gamma\ [\![\ \Gamma \vdash a\ ]\!]_\gamma\ [\![\ \Gamma \vdash P\ ]\!]_\gamma\ [\![\ \Gamma \vdash p\ ]\!]_\gamma\ [\![\ \Gamma \vdash q\ ]\!]_\gamma
\end{aligned}
$$

Figure 5 Interpretation of the proof-relevant terms.

Interpreting the syntax. Our higher-order model in hand, it remains to interpret the syntax of $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ in it. The interpretation of raw syntax and typing judgments of a SOGAT into a higher-order model is a very administrative task, which has already been treated in very broad generality by Uemura [43]. We do not believe that formalizing it would provide new interesting insights, and so this part of the argument is only carried out on paper.

The interpretation is defined in Figure 4 and Figure 5 as partial functions from the syntax to the semantics. We use a function $[\![\_]\!]$ that interprets contexts as sets and a function $[\![\Gamma \vdash \_]\!]_\gamma$ that interprets terms and types in context Γ as sets indexed by $\gamma \in [\![\Gamma]\!]$. Both functions are mutually defined by recursion on the raw syntax, and will eventually be proven to be total functions on well-typed terms.

To prove soundness of our interpretation, we need to extend it to weakenings and substitutions between contexts. Assume Γ and Δ are a syntactical contexts, and A and t are syntactical terms. In case $[\![\Gamma, x : A : s, \Delta]\!]$ and $[\![\Gamma, \Delta]\!]$ are well-defined, let π_A be the projection:

$$
\begin{aligned}
\pi_A : [\![\Gamma, x : A : s, \Delta]\!] &\to [\![\Gamma, \Delta]\!] \\
(\vec{x_\Gamma}, x_A, \vec{x_\Delta}) &\mapsto (\vec{x_\Gamma}, \vec{x_\Delta}).
\end{aligned}
$$

In case $[\![\Gamma, \Delta[x := t]]\!]$ and $[\![\Gamma, x : A : s, \Delta]\!]$ are well-defined, we define the function σ_t by:

$$
\begin{aligned}
\sigma_t : [\![\Gamma, \Delta[x := t]]\!] &\to [\![\Gamma, x : A : s, \Delta]\!] \\
(\vec{x_\Gamma}, \vec{x_\Delta}) &\mapsto (\vec{x_\Gamma}, [\![\ \Gamma \vdash t\ ]\!]_{\vec{x_\Gamma}}, \vec{x_\Delta}).
\end{aligned}
$$

▶ **Lemma 18** (Weakening). *π_A is the semantic counterpart to the weakening of A: for all terms u, when both sides are well defined, we have $[\![\ \Gamma, x : A : s, \Delta \vdash u\ ]\!]_\gamma = [\![\ \Gamma, \Delta \vdash u\ ]\!]_{\pi_A(\gamma)}$.*

▶ **Lemma 19** (Substitution). *σ_t is the semantic counterpart to the substitution by t: for all terms u, when both sides are well defined, we have $[\![\ \Gamma, \Delta[x := t] \vdash u[x := t]\ ]\!]_\gamma = [\![\ \Gamma, x : A : s, \Delta \vdash u\ ]\!]_{\sigma_t(\gamma)}$.*

As a last step before proving soundness of the model, we must ensure that the axioms of Σ indeed hold in set theory. Formally, we assume that $[\![\vdash A]\!]$ is defined and inhabited for all $c : A \in \Sigma$, an hypothesis satisfied by many important axioms, such as function extensionality, excluded middle and various forms of choice.

▶ **Theorem 20** (Soundness of the Standard Model).

1. *If $\vdash \Gamma$ then $[\![\Gamma]\!]$ is defined.*
2. *If $\Gamma \vdash A : \mathcal{U}_\Omega$ then $[\![\ \Gamma \vdash A\ ]\!]_\gamma \in \mathtt{El}_0\ \Omega$ for all $\gamma \in [\![\Gamma]\!]$.*
3. *If $\Gamma \vdash A : \mathcal{U}_\mathtt{n}$ then $[\![\ \Gamma \vdash A\ ]\!]_\gamma \in \mathtt{El}_{\mathtt{n}+1}\ \mathbb{U}_\mathtt{n}$ for all $\gamma \in [\![\Gamma]\!]$.*
4. *If $\Gamma \vdash^\Omega t : A$ then $\varnothing \in \mathtt{El}_\Omega\ [\![\ \Gamma \vdash A\ ]\!]_\gamma$ for all $\gamma \in [\![\Gamma]\!]$.*
5. *If $\Gamma \vdash^\mathtt{n} t : A$ then $[\![\ \Gamma \vdash t\ ]\!]_\gamma \in \mathtt{El}_\mathtt{n}\ ([\![\ \Gamma \vdash A\ ]\!]_\gamma)$ for all $\gamma \in [\![\Gamma]\!]$.*
6. *If $\Gamma \vdash t \equiv u : A$ then $[\![\ \Gamma \vdash t\ ]\!]_\gamma = [\![\ \Gamma \vdash u\ ]\!]_\gamma$ for all $\gamma \in [\![\Gamma]\!]$.*

Proof. By induction on the typing derivations, using Lemmas 18 and 19. ◀

Consequences. As the false proposition is interpreted as the empty set, we get that our theories are consistent.

▶ **Theorem 21** (Consistency). *The type $\bot$, defined as $\Pi(X : \mathcal{U}_\Omega).X$, is not inhabited in $\mathcal{T}^{=}_{\mathrm{Acc}}$ or $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ in the empty context.*

We now focus on proving the assumptions of the canonicity proof. Since we equipped every relevant type former with a unique identifier, we get that one cannot prove an equality between two relevant types with different heads in the empty context (Assumption A).

To prove Assumption B, we need the following easy lemma. Recall that a *relation morphism* from R_1 to R_2 is a function ϕ such that $R_1(x, y)$ implies $R_2(\phi(x), \phi(y))$.

▶ **Lemma 22** (Morphisms reflect accessibility). *If ϕ is a relation morphism from R_1 to R_2 and $\phi(a)$ is well founded for R_2 then a is well founded for R_1.*

Now, assuming $\vdash e : \mathrm{Acc}^\mathtt{n}(A, xy.R, a)$, Theorem 20 yields a proof of $[\![\vdash \mathrm{Acc}^\mathtt{n}(A, xy.R, a)]\!]$, meaning that the element $[\![\vdash a]\!]$ is well founded for the binary relation $[\![\vdash R]\!]$ over $[\![\vdash A]\!]$. Theorem 20 implies also that $x \mapsto [\![\vdash x]\!]$ is a relation morphism from $\widetilde{R}$ to $[\![\vdash R]\!]$, thus we conclude that a is well founded for $\widetilde{R}$ (Assumption B).

6 Accessibility in SProp in Action

We base our implementation on the work of Pujet *et al.* [39], which implements $\mathrm{CIC}_{\mathrm{obs}}$ using the rewrite rule mechanism introduced in Rocq [16, 29]. This makes it possible to postulate the cast operation (where $\sim$ is the notation for observational equality) together with its reduction rules implementing the conversion rules such as Cast-Π (not shown):

```
Symbol cast : ∀ (A B : Type), A ∼ B → A → B.
```

Defining Acc in SProp. The accessibility predicate can be defined in Rocq as the following inductive predicate in SProp:

```
Inductive Acc A (R: A → A → SProp) (x:A) : SProp :=
| acc_in : (∀ y : A, R y x → Acc A R y) → Acc A R x.
```

However, only the eliminator for predicates in SProp is generated by default, so we postulate a new symbol `acc_el` (*i.e.*, a new constant) corresponding to rule Acc-el:

```
Symbol acc_el : ∀ A R a (P : ∀ x : A, Type),
  (∀ a, (∀ b : A, R b a → P b) → P a) → ∀ (q : Acc R a), P a.
```

Supporting $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$. It remains to implement the two computation rules for `acc_el`, the propositional one and the definitional one for $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$. The propositional one is simply an axiom:

```
Axiom Acc_el_comp : ∀ A R a P p q,
  acc_el A R a P p q ~ p a (fun b r => acc_el A R b P p (acc_inv A R a q b r)).
```

For the other, we define a rewrite rule corresponding to an oriented version of Acc-el-def (‘`?`’ is used to introduce variables bound by the left-hand side of the rewrite rule):

```
#[local] Rewrite Rule Acc_el_def :=
| acc_el ?A ?R ?a ?P ?p ?q =>
  ?p ?a (fun b r => acc_el ?A ?R b ?P ?p (acc_inv ?A ?R ?a ?q b r)).
```

The term `acc_inv` used above corresponds to rule Acc-inv.

Note that we have introduced in Rocq (in a modified v9.2) a way to localize the use of a rewrite rule, using the pragma `#[local]`. This flag specifies that the rewrite rule is disabled by default, and must be explicitly enabled *locally* with the `#[rewrite_rules(Acc_el_def)]` pragma when one wants to work in the theory $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$, as exemplified below:

```
#[rewrite_rules(Acc_el_def)]
Theorem example : (...).
Proof.
  (...)
Qed.
```

By Theorem 16, such a proof in $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ could be elaborated to one in the theory $\mathcal{T}^{=}_{\mathrm{Acc}}$. Yet, as mentioned in the introduction, because proofs are irrelevant we never need to perform this costly translation, so the implementation treats the above theorem is $\mathcal{T}^{=}_{\mathrm{Acc}}$ as a justified axiom.

Well-founded recursion done right – and fast. To test our hypothesis that having a definitional computation rule for `acc_el` is crucial in some cases, we define the `gcd` (greatest common divisor) function, following the running example of Leroy [30].

We do not recall the concrete implementation of `gcd`, which can be found in the Rocq formalization. Here, what matters is the ability to prove simple helpful lemmas such as:

```
Lemma gcd_test : (gcd (2 ^ N) 2 <? 5) ~ true.
```

for some given number `N`. Indeed, when doing proofs in computational algebra for instance, it is not rare to have to prove concrete bounds on functions applied to specific values in order to be able to apply a lemma. This is the case for instance when approximating definite integrals, as explained by Mahboubi *et al.* [32].

Table 1 Performance comparison of three approaches to prove the lemma `gcd_test` for some values of `N`. (✗ indicates a 10-min timeout).

N	Proof normalization in Prop	Rewriting in $\mathcal{T}^{=}_{\text{Acc}}$	Conversion in $\mathcal{T}^{\equiv}_{\text{Acc}}$
5	$<$ 0.001 sec	0.085 sec	$<$ 0.001 sec
6	$<$ 0.001 sec	0.38 sec	$<$ 0.001 sec
7	$<$ 0.001 sec	2.638 sec	$<$ 0.001 sec
8	0.001 sec	21.468 sec	0.001 sec
9	0.002 sec	203.198 sec	0.002 sec
10	0.004 sec	✗	0.004 sec

We test the performance of the proofs in three different scenarios, when `N` varies:

1. by evaluating the normalization proof of `gcd` in Prop as it can be done in standard CIC,
2. by (automated) rewriting with the propositional equality `Acc_el_prop` in $\mathcal{T}^{=}_{\text{Acc}}$, similar to using the `simp` tactic in Lean,
3. by evaluating `acc_el` directly in $\mathcal{T}^{\equiv}_{\text{Acc}}$.

Working directly in $\mathcal{T}^{=}_{\text{Acc}}$, the proof is performed as:

```
Lemma gcd_test : (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  auto_Acc_unfold; reflexivity.
Qed.
```

The tactic `auto_Acc_unfold` performs rewrites with the propositional equality `Acc_el_prop` until it is no longer applicable.

Using the conservativity result (Theorem 16), the proof can be performed by locally moving to $\mathcal{T}^{\equiv}_{\text{Acc}}$, and exploiting conversion:

```
#[rewrite_rules(Acc_el_def)]
Lemma gcd_test_def (gcd (2 ^ N) 2 <? 5) ~ true.
Proof.
  reflexivity.
Qed.
```

The results shown in Table 1 illustrate the performance benefits of being able to compute with `acc_el` in SProp or Prop. The results were obtained on a MacBook Pro 14-in (M2 Pro, 32GB RAM), taking the average of 10 runs.

The most salient result of this small experiment is that automatic rewriting with the `Acc_el_prop` lemma in $\mathcal{T}^{=}_{\text{Acc}}$ performs poorly compared to the other two approaches, and degrades very quickly – reaching a 10-min timeout for $N = 10$. Note that for a given N, $2^{N-1}+1$ rewrites are performed. The fact that computation time grows much faster than the number of rewritings to perform is due to the exponential growth in the size of the generated proof terms. The drastic performance difference between automatic rewriting in $\mathcal{T}^{=}_{\text{Acc}}$ and relying on conversion in $\mathcal{T}^{\equiv}_{\text{Acc}}$ comes from the fact that, with conversion, the proof is always just `refl`, independently of the number of reduction steps to be performed.

The apparent efficiency of evaluating the termination proof in Prop – which here is comparable to direct computation in SProp – is in fact misleading, as it depends on the size of the proof of accessibility passed on to the eliminator. Indeed, recall that the computation rule for accessibility in Prop requires the proof term to be first reduced to `acc_in`. For `gcd` this proof is rather small but, as we show next with a System F evaluator, this approach does not scale at all when the termination argument is larger and more intricate.

Perils of reduction. We now explain what happens when trying to prove the above lemma generically for all n, highlighting the interest of considering both $\mathcal{T}^{=}_{\mathrm{Acc}}$ and $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$ in concert. When working inside $\mathcal{T}^{\equiv}_{\mathrm{Acc}}$, fully evaluating the `gcd` (using the `lazy` tactic) does not terminate; indeed, the canonicity theorem (Corollary 12) only holds on closed terms, and thus does not apply when n is a variable.

```
#[rewrite_rules(Acc_el_def)]
Lemma gcd_test_gen : ∀ n, (gcd (2 ^ n) 2 <? 5) ∼ true.
Proof.
  Fail Timeout 2 lazy; reflexivity.
Abort.
```

Note that non-termination can be avoided by only reducing to weak head normal form with tactics such as `simpl` or `hnf`. But controlling reduction is fragile in proof assistants. As mentioned in Section 1, this is the reason why in Lean, when `acc_el` is set to compute, subtle changes in heuristics to control unfolding can unexpectedly break proofs. In that case, the better approach is to show this property in $\mathcal{T}^{=}_{\mathrm{Acc}}$, using rewriting during reasoning.

Normalization of System F. To illustrate the practical benefits of the proposed approach to reconcile definitional proof irrelevance and accessibility in presence of *impredicativity*, we now consider the stereotypical case of an evaluator of System F. We first explain how the evaluator can be obtained from the proof of normalization of System F, and then quickly report on the performance comparison of different approaches to compute with this evaluator.

Note that compared to the `gcd` example above, the semantic termination argument of System F is rather involved. Also, an evaluator for System F cannot be defined in $\mathrm{CIC}_{\mathrm{obs}}$ without accessibility defined in SProp, as normalization of System F fundamentally relies on impredicativity and the presence of an accessibility predicate.

For a term `t` of System F, being normalizing is encoded by the fact that anti-reduction is accessible at `t`:

```
Definition Normalizing (t : Term) : SProp :=
  Acc (fun t' t => t ⇝ t') t.
```

where `t ⇝ t'` denotes that `t` β-reduces to `t'` in exactly one step. Put differently, all β-reduction chains starting from `t` are finite.

The main part of the proof from Girard [21], using reducibility candidates, is to show that any well-typed term of System F is actually normalizing.

```
Lemma termf_norm : ∀ ctx t ty,
  ctx ⊢ t :: ty → Normalizing t.
```

To prove this result in Rocq, in SProp, we adapted the development of Blot [14], which was carried out in Prop. From this proof of normalization, it is possible to derive an evaluator `nf` for System F by first showing that being in normal form is a decidable property.

```
Definition nf : ∀ ctx t ty, ctx ⊢ t :: ty → Term.
```

Having derived the evaluator from the normalization proof, we look again at the three different approaches to compute with it, via the simple example of proving $2^{n+1} = 2^n + 2^n$ with concrete values of n. Recall that in System F, the type of Church numerals can be defined as $\forall X, X \to (X \to X) \to X$, thanks to impredicativity, therefore requiring the use of an impredicative universe.

The results are given in Table 2. They confirm that using automatic rewrite of the propositional computation rule is slow and does not scale. Notably, while the proof normalization in Prop appeared reasonable in the `gcd` example, here we see the impact of an involved

Table 2 Performance comparison of three approaches to prove $2^{n+1} = 2^n + 2^n$ for some values of n, with the System F evaluator. (✗ indicates a 30-min timeout).

n	Proof normalization in **Prop**	Rewriting in $\mathcal{T}^{=}_{\text{Acc}}$	Computation in $\mathcal{T}^{\equiv}_{\text{Acc}}$
2	38 sec	12.2 sec	0.002 sec
3	✗	35.4 sec	0.005 sec
4	✗	147 sec	0.015 sec
9	✗	✗	14.3 sec

termination argument: with $n = 2$ the approach is already 3x slower than rewriting in $\mathcal{T}^{=}_{\text{Acc}}$, and reaches a 30-min timeout with just $n = 3$. On the contrary, direct computation in $\mathcal{T}^{\equiv}_{\text{Acc}}$ performs orders of magnitude better, and scales adequately, achieving practical performance even for $n = 9$ and beyond.

7 Conclusion, Related and Future Work

We have presented a dual approach to reconciling definitional proof irrelevance with accessibility predicates, with an ambient decidable theory $\mathcal{T}^{=}_{\text{Acc}}$ that only features a propositional computation rule for accessibility elimination, and a more flexible theory $\mathcal{T}^{\equiv}_{\text{Acc}}$ that supports a definitional computation rule at the expense of potential divergence. Crucially, we prove the theory $\mathcal{T}^{\equiv}_{\text{Acc}}$ to be conservative over $\mathcal{T}^{=}_{\text{Acc}}$, ensuring that it can be used to simplify the writing of proofs in the latter. The two theories are consistent and satisfy canonicity properties. We have briefly discussed an implementation in the Rocq prover, based on local rewrite rules.

We have already discussed the most salient related work that gives rise to this work in the introduction (Section 1), in particular the tension between definitional proof irrelevance and accessibility [20]. The definition of accessibility dates back to Aczel [5], with the introduction of the accessibility predicate technique later by Nordström [34], which is nowadays the standard approach for well-founded definitions in type theory-based proof assistants.

The metatheory of definitional proof irrelevance was studied in the setting of intentional Martin-Löf Type Theory initially by Werner [44], even if this principle was already used as justification for other systems [7, 13]. However, the work of Werner mentions a wrong conjecture on the behavior of propositional equality in this context, which has recently been clarified by Abel and Coquand [1]. This illustrates the difficulty of mixing definitional proof irrelevance with logical principles on the metatheoretic level. Further metatheoretic study of definitional proof irrelevance came later with the work of Abel *et al.* [2], Gilbert *et al.* [20] and Coquand [17]. Pujet and Tabareau [40] subsequently extended the work of Gilbert *et al.* [20] by also considering observational equality, originally proposed by Altenkirch *et al.* [10]. Their work was extended to account for impredicativity [41] and indexed inductive types [39].

Finally, the work on extraction of Letouzey [31], recently formalized by Forster *et al.* [19], can be seen as an external version of our approach. Indeed, after the erasure of proofs, the eliminator for accessibility computes exactly as specified by rule Acc-el-def, ignoring the accessibility witness. However, our canonicity result is stronger as it allows the use of axioms in SProp that are consistent in the set-theoretic model – in particular, classical principles. This is similar to the work of Abel *et al.* [3], whose results also allow the evaluation of programs that use consistent axioms. One promising line of work is to study whether our proof technique can be used to derive a similar result for Rocq's extraction, therefore drawing a line of which kind of axioms can be used safely in a formalization while ensuring the good computational properties of the extracted code.

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.

2 Andreas Abel, Thierry Coquand, and Miguel Pagano. A modular type-checking algorithm for type theory with singleton types and proof irrelevance. *Logical Methods in Computer Science*, 7, 2011. doi:10.2168/LMCS-7(2:4)2011.

3 Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. A graded modal dependent type theory with a universe and erasure, formalized. *Proc. ACM Program. Lang.*, 7(ICFP), 2023. doi:10.1145/3607862.

4 Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proceedings of the ACM on Programming Languages*, 2(POPL):23:1–23:29, January 2018. doi:10.1145/3158111.

5 Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *HANDBOOK OF MATHEMATICAL LOGIC*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 739–782. Elsevier, 1977. doi:10.1016/S0049-237X(08)71120-0.

6 Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. Martin-löf à la coq. In Amin Timany, Dmitriy Traytel, Brigitte Pientka, and Sandrine Blazy, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, pages 230–245. ACM, 2024. doi:10.1145/3636501.3636951.

7 T. Altenkirch. Extensional equality in intensional type theory. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 412–420, 1999. doi:10.1109/LICS.1999.782636.

8 Thorsten Altenkirch. Extensional equality in intensional type theory. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, LICS '99, page 412, USA, 1999. IEEE Computer Society.

9 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2016.

10 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.

11 Robert Atkey. Simplified observational type theory, 2018. Implementation at https://github.com/bobatkey/sott.

12 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/.

13 Gilles Barthe. The relevance of proof-irrelevance. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming*, pages 755–768, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. doi:10.1007/BFB0055099.

14 Valentin Blot. Normalization of System F, formalized in Coq, 2022. URL: https://gitlab.inria.fr/-/snippets/797.

15 Guillaume Brunerie, Menno de Boer, Peter LeFanu Lumsdaine, and Anders Mörtberg. A formalization of the initiality conjecture in agda. *Slides from a talk about joint work with Menno de Boer, Peter Lumsdaine, and Anders Mörtberg, at HoTT, Pittsburgh, CMU*, 2019.

16 Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. The taming of the rew: A type theory with computational assumptions. *Proc. ACM Program. Lang.*, 5(POPL), 2021. doi:10.1145/3434341.

17 Thierry Coquand. Reduction free normalisation for a proof irrelevant type of propositions. *Logical Methods in Computer Science*, Volume 19, Issue 3, July 2023. doi:10.46298/lmcs-19(3:5)2023.

18 Thiago Felicissimo. Generic bidirectional typing for dependent type theories. *ACM Trans. Program. Lang. Syst.*, 47(1), 2025. doi:10.1145/3715095.

19 Yannick Forster, Matthieu Sozeau, and Nicolas Tabareau. Verified extraction from coq to ocaml. *Proc. ACM Program. Lang.*, PLDI, June 2024. doi:10.1145/3656379.

20 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.

21 Jean-Yves Girard. Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur. Thèse de Doctorat d'État, Université de Paris VII, 1972.

22 Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. Implementing a modal dependent type theory. *Proc. ACM Program. Lang.*, 3(ICFP), 2019. doi:10.1145/3341711.

23 Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, January 1993. doi:10.1145/138027.138060.

24 Martin Hofmann. Conservativity of equality reflection over intensional type theory. In *International Workshop on Types for Proofs and Programs*, pages 153–164. Springer, 1995. doi:10.1007/3-540-61780-9_68.

25 Junyoung Jang, Antoine Gaulin, Jason Z. S. Hu, and Brigitte Pientka. Mctt: A verified kernel for a proof assistant. *Proc. ACM Program. Lang.*, 9(ICFP), 2025. doi:10.1145/3747511.

26 Ambrus Kaposi and Szumi Xie. Second-Order Generalised Algebraic Theories: Signatures and First-Order Semantics. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*, volume 299 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:24, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2024.10.

27 Gyesik Lee and Benjamin Werner. Proof-irrelevant model of cc with predicative induction and judgmental equality. *Logical Methods in Computer Science*, Volume 7, Issue 4, November 2011. doi:10.2168/LMCS-7(4:5)2011.

28 Meven Lennon-Bertrand. Complete Bidirectional Typing for the Calculus of Inductive Constructions. In Liron Cohen and Cezary Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving (ITP 2021)*, volume 193 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITP.2021.24.

29 Yann Leray, Gaëtan Gilbert, Nicolas Tabareau, and Théo Winterhalter. The Rewster: Type Preserving Rewrite Rules for the Coq Proof Assistant. In *International Conference on Interactive Theorem Proving (ITP 2024)*, volume 15th International Conference on Interactive Theorem Proving (ITP 2024), page 18, Tbilisi, Georgia, September 2024. Yves Bertot and Temur Kutsia and Michael Norrish. doi:10.4230/LIPIcs.ITP.2024.26.

30 Xavier Leroy. Well-founded recursion done right. In *CoqPL 2024: The Tenth International Workshop on Coq for Programming Languages*, London, United Kingdom, January 2024. ACM. URL: https://inria.hal.science/hal-04356563.

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

32 Assia Mahboubi, Guillaume Melquiond, and Thomas Sibut-Pinote. Formally Verified Approximations of Definite Integrals. *Journal of Automated Reasoning*, 62(2):281–300, February 2019. doi:10.1007/s10817-018-9463-7.

33 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:10.1016/S0049-237X(08)71945-1.

34 Bengt Nordström. Terminating general recursion. *BIT*, 28(3):605–619, 1988. doi:10.1007/BF01941137.

35 Steven Obua. Partizan games in isabelle/holzf. In Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *Theoretical Aspects of Computing - ICTAC 2006*, pages 272–286, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11921240_19.

36 Nicolas Oury. Extensionality in the calculus of constructions. In *International Conference on Theorem Proving in Higher Order Logics*, pages 278–293. Springer, 2005. doi:10.1007/11541868_18.

37 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. URL: https://hal.inria.fr/hal-01094195.

38 Pierre-Marie Pédrot. Russian constructivism in a prefascist theory. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 782–794, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394740.

39 Loïc Pujet, Yann Leray, and Nicolas Tabareau. Observational Equality Meets CIC. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 47(2):1–35, April 2025. doi:10.1145/3719342.

40 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.

41 Loïc Pujet and Nicolas Tabareau. Impredicative Observational Equality. *Proceedings of the ACM on Programming Languages*, 7(POPL):74, January 2023. doi:10.1145/3571739.

42 Amin Timany and Matthieu Sozeau. Consistency of the Predicative Calculus of Cumulative Inductive Constructions (pCuIC). Research Report RR-9105, KU Leuven, Belgium ; Inria Paris, October 2017. Version 2 fixes some typos from version 1.Version 3 fixes a typo in a typing rule from version 2. URL: https://inria.hal.science/hal-01615123.

43 Taichi Uemura. *Abstract and concrete type theories*. PhD thesis, University of Amsterdam, 2021.

44 Benjamin Werner. On the strength of proof-irrelevant type theories. *Logical Methods in Computer Science*, 4:1–20, September 2008. doi:10.2168/LMCS-4(3:13)2008.

45 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, pages 91–103, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293880.3294095.