Divide and Check: Logical Relations, No Algorithms Attached

Josselin Poiret, Kenji Maillard, Nicolas Tabareau

Divide and Check: Logical Relations, No Algorithms Attached

Josselin Poiret
Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, Nantes, France

Kenji Maillard
Inria, France

Nicolas Tabareau
Inria, France

Abstract

The correctness of type-checking implementations for proof assistants based on dependent type theory relies on metatheoretical properties that ensure the decidability of typing, some of which require substantial logical strength. Recent mechanizations of such algorithms have highlighted the importance of separating the algorithmic components of the proof—often intricate but requiring relatively low logical strength—from the logical components, which depend on stronger metatheoretical properties, such as normalization or the injectivity of type constructors. In this work, we revisit the logical relations technique and show how it can be used to derive these metatheoretical properties in a direct and uniform way for a core dependent type theory featuring Π-types, $\mathbb{N}$, $\bot$ and a universe $\mathbb{U}$. Our presentation yields a compact and conceptually simplified argument that isolates the logically strong reasoning from the algorithmic core. We argue that this approach scales smoothly to richer type theories, and demonstrate this by extending our construction to Exceptional Type Theory (ExcTT), obtaining the first mechanized canonicity proof for this theory.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Type Theory, Proof Assistants

Acknowledgements The authors want to thank Thiago Felicissimo for their many suggestions, and Rafaël Bocquet and Yann Leray for valuable discussions.

1 Introduction

One of the central advantages of proof assistants based on intensional dependent type theory—such as Agda, Lean, or the Rocq Prover—is the existence of decidable type-checking algorithms that mechanically certify the correctness of proofs. Ensuring the correctness and completeness of these algorithms, however, relies on subtle invariants that can easily be invalidated by seemingly benign optimizations or extensions of the theory.

This fragility has motivated substantial efforts toward certifying type checkers themselves, most notably in the Rocq ecosystem [45, 46] and in the on-going Lean4Lean [11] or Agda-core [32] initiatives. These projects focus primarily on the algorithmic core of type checking, while assuming strong metatheoretical properties—such as normalization—that lie outside the scope of the certified implementation.

Logical relations and related realizability techniques play a foundational role in establishing these metatheoretical properties. Originating in Tait's computability argument for the simply typed lambda calculus [51], and extended through Girard's reducibility candidates for System F [20] and early work on Martin-Löf Type Theory (MLTT) [36], logical relations have been the primary tool for proving normalization and canonicity of dependent type theories.

More recently, the proliferation of new dependent type theories has renewed interest in these techniques, both theoretically and practically. Several large formalization efforts have mechanized logical relations arguments in proof assistants, including LogRelMLTT in

Agda [3], LogRelCoq in Rocq [4], and McTT [25]. These developments typically rely on multiple layers of logical relation constructions: either by parametrizing a single construction over increasingly precise data, or by composing distinct relations to establish different aspects of correctness. As a result, logical relations are used not only to justify strong metatheoretical properties, but also to establish fine-grained, algorithmically meaningful invariants.

In parallel, high-level categorical and topos-theoretic frameworks have been proposed to generalize logical relations arguments, including categorical Normalisation by Evaluation (NbE) [5, 18, 14], synthetic Tait computability [49], and internal sconing [8]. While conceptually powerful, these approaches remain challenging to reconcile with concrete mechanized developments, leaving a gap between abstract proofs and implementable techniques.

From an engineering perspective, the complexity of logical relations strongly suggests that they should be deployed only where they are strictly necessary. Yet identifying the minimal metatheoretical assumptions required for decidable type checking is itself nontrivial: many properties are intertwined, and their precise dependencies are not always clear. Recent work by Lennon-Bertrand [30] pushes in this direction by identifying three key properties sufficient to establish decidability of conversion in dependent type theory: injectivity of normal forms, deep normalization, and a characterization of neutral conversion. This line has been partially pursued by Liu et al. [33], who work in an untyped setting and avoid the complexity of full-fledged logical relations by assuming injectivity of Π-types rather than deriving it.

Such assumptions, however, are far from innocuous. In particular, injectivity of normal forms is inconsistent in extensional settings, illustrating that bypassing metatheoretical proofs can compromise soundness when moving to less well-understood extensions of type theory. Avoiding strong, unverified assumptions on the source theory is therefore essential to maintain robust guarantees, especially in the presence of richer language features.

Taken together, these results suggest that a more disciplined and frugal use of logical relations—targeted precisely at the properties required for decidable type checking—may suffice to justify both decidability and the correctness of implemented type-checking algorithms, while keeping the metatheoretical burden under control. In this work, we explain how we successfully pursued these insights to bring pen-and-paper and mechanized proofs significantly closer, simplifying existing mechanized arguments for these core properties.

Plan of the Paper

In Section 2, we present the considered type theory together with a formal definition of the three key metatheoretical properties advocated for in [30]. In Section 3, we introduce the core of the formalization: a mechanized logical relation establishing normalization in a single pass for a core Martin-Löf style type theory with a universe, Π-types, natural numbers $\mathbb{N}$, and the empty type. We extend the theory with exceptions in Section 4, showing that the logical relation scales to this setting and yielding the first mechanized canonicity proof for exceptional type theory. In Section 5, we position our construction within the landscape of closely related techniques, both pen-and-paper and mechanized, clarifying the connections in the literature. In Section 6, we conclude on future work.

Formalization

Our results have been mechanized in the Rocq Prover (version 9.0). Definitions and theorems are referred to in the text using the Rocq icon which marks a clickable link that points to an online version of the formalization.

2 Context

We consider a core dependent type theory à la Martin-Löf. Its judgement forms consist of well-formed contexts $\vdash \Gamma$, types $\Gamma \vdash A$ and terms $\Gamma \vdash t : A$ together with their corresponding equational forms called *conversions* ($\Gamma \vdash A \equiv B$, $\Gamma \vdash t \equiv u : A$). The type theory features dependent products $\Pi(x : A)B$ introduced with λ-abstraction and eliminated using application, a type of natural numbers $\mathbb{N}$ with zero `0`, successor `S` and a dependent eliminator $\texttt{ind}_{\mathbb{N}}$, an empty type $\bot$ with eliminator $\texttt{ind}_{\bot}$ and a universe of small types $\mathbb{U}$ closed under all other type formers. This set of type formers represents a core type theory containing most salient features appearing in [30]. For conciseness, we only present here the inference rules for Lam and App and defer the full system to Section A or the formalization (). Taking an algebraic presentation of syntax, abstraction and application are annotated with the types of the domain and codomain.

$$\textsc{Lam}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \Pi(x : A)B} \qquad \textsc{App}\ \frac{\Gamma \vdash f : \Pi(x : A)B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ @_{(x:A)B}\ a : B[a/x]}$$

Well-formed renamings $\rho : \Delta \geq \Gamma$ () and substitutions $\Delta \vdash \sigma : \Gamma$ () are defined inductively with respect to the target context Γ. Substitution conversion $\Delta \vdash \sigma \equiv \sigma' : \Gamma$ is defined similarly by pointwise conversions of terms.

$$\textsc{RenNil}\ \frac{}{\varepsilon : \Delta \geq \varepsilon} \qquad \textsc{RenCons}\ \frac{\rho : \Delta \geq \Gamma \qquad (y : A[\rho]) \in \Delta}{(\rho, y/x) : \Delta \geq (\Gamma, x : A)}$$

$$\textsc{SubstNil}\ \frac{}{\Delta \vdash \varepsilon : \varepsilon} \qquad \textsc{SubstCons}\ \frac{\Delta \vdash \sigma : \Gamma \qquad \Delta \vdash t : A[\sigma]}{\Delta \vdash (\sigma, t/x) : (\Gamma, x : A)}$$

Formalization: Technical Choices and Automation

In the formalization, the syntax of the type theory is represented using preterms together with a single inductive predicate indexed by the judgement for derivations (). Variables are handled using De Bruijn indices and all management of raw renamings and substitutions is derived automatically using Sulfur [9], a recent Rocq plugin implementing a variant of Autosubst [48].

2.1 Immediate Metatheory

A few basic properties are immediately provable on typing derivations, and we only state these for well-formed terms even though they are proved mutually for all judgement forms.

▶ **Lemma 1** (Basic properties of typing derivations).
Boundary *if* $\Gamma \vdash t : A$ *then* $\vdash \Gamma$ *and* $\Gamma \vdash A$
Renaming *if* $\Gamma \vdash t : A$, $\vdash \Delta$ *and* $\rho : \Delta \geq \Gamma$ *then* $\Delta \vdash t\langle\rho\rangle : A\langle\rho\rangle$
Substitution *if* $\Gamma \vdash t : A$ *and* $\Delta \vdash \sigma : \Gamma$ *then* $\Delta \vdash t[\sigma] : A[\sigma]$
Convertible Substitution *if* $\Gamma \vdash t : A$ *and* $\Delta \vdash \sigma \equiv \sigma' : \Gamma$ *then* $\Delta \vdash t[\sigma] \equiv t[\sigma'] : A[\sigma]$

By inspection on the typing judgements, inversion lemmas for well-formed types and terms are easily derived, for instance:

▶ **Lemma 2** (Typing inversions).
- *If* $\Gamma \vdash \Pi(x : A)B$ *then* $\Gamma \vdash A$ *and* $\Gamma, x : A \vdash B$;
- *If* $\Gamma \vdash f\ @_{(x:A)B}\ a : C$ *then* $\Gamma \vdash f : \Pi(x : A)B$, $\Gamma \vdash a : A$ *and* $\Gamma \vdash \Pi(x : A)B \equiv C$.

$$\frac{f \rightsquigarrow g}{f \mathbin{@_{(x:A)B}} a \rightsquigarrow g \mathbin{@_{(x:A)B}} a} \qquad \frac{n \rightsquigarrow n'}{\mathtt{ind}_\bot\ (x.P)\ n \rightsquigarrow \mathtt{ind}_\bot\ (x.P)\ n'}$$

$$\frac{n \rightsquigarrow n'}{\mathtt{ind}_\mathbb{N}\ (x.P)\ h_0\ (x.i.h_\mathtt{S})\ n \rightsquigarrow \mathtt{ind}_\mathbb{N}\ (x.P)\ h_0\ (x.i.h_\mathtt{S})\ n'}$$

$$\begin{array}{lcl}
(\lambda x : A.t) \mathbin{@_{(x:A)B}} a & \rightsquigarrow & t[a/x] \\
\mathtt{ind}_\mathbb{N}\ (x.P)\ h_0\ (x.i.h_\mathtt{S})\ \mathtt{0} & \rightsquigarrow & h_0 \\
\mathtt{ind}_\mathbb{N}\ (x.P)\ h_0\ (x.i.h_\mathtt{S})\ (\mathtt{S}\ n) & \rightsquigarrow & h_\mathtt{S}[\mathtt{ind}_\mathbb{N}\ (x.P)\ h_0\ (x.i.h_\mathtt{S})\ n/i, n/x]
\end{array}$$

Figure 1 Reduction relation on MLTT terms and types

By contrast, no such inversion lemma are readily available for conversion. In particular we cannot yet prove the injectivity of Π types: without a precise characterization of conversion, a derivation of $\Gamma \vdash \Pi(x : A)B \equiv \Pi(x : A')B'$ may involve arbitrary uses of transitivity.

Some types might be terms of the universe: we call those types *small*, while general types are called *large*. To help factorize statements about types, we define Level ≔ Small | Large.

Reduction

Efficient implementations of type checkers and conversion checkers typically rely on additional structure in the underlying type theory to obtain fine-grained representations of conversion equivalence classes, most often by enabling the computation of normal forms. Following [30], we use an untyped reduction relation $\rightsquigarrow$ on types and terms described in Figure 1. Reduction is made deterministic by choosing a call-by-name evaluation strategy (leftmost-innermost). An irreducible term for $\rightsquigarrow$ is called a weak-head normal form (whnf), and can either present an introduction form as its head ($\Pi, \mathbb{N}, \bot, \mathbb{U}, \lambda, \mathtt{0}, \mathtt{S}$) or be a neutral term (whne), which can be characterized as a stack of elimination forms stuck on a variable. Additionally, we define a partial function `whnf` that computes the whnf of a term using Rocq-Equations [47] ( ).

Note that, in the absence of injectivity for the Π-type constructor, we cannot yet establish preservation of typing under reduction, a.k.a. subject reduction. We bypass this issue by defining a relation for well-typed reduction, combining reduction and conversion.

▶ **Definition 3** (Well-typed reductions). *Define*

- $\Gamma \vdash A \rightsquigarrow^* B \coloneqq (\mathtt{whnf}\ A = B) \times (\Gamma \vdash A \equiv B)$*;*
- $\Gamma \vdash t \rightsquigarrow^* u : A \coloneqq (\mathtt{whnf}\ t = u) \times (\Gamma \vdash t \equiv u : A)$*;*
- $\Gamma \vdash_l A \rightsquigarrow^* B \coloneqq \begin{cases} \Gamma \vdash A \rightsquigarrow^* B : \mathbb{U} & \text{if } l \text{ is Small} \\ \Gamma \vdash A \rightsquigarrow^* B & \text{if } l \text{ is Large} \end{cases}$

The role of $\Gamma \vdash_l A \rightsquigarrow^* B$ is to give a statement adapted to both small and large types.

2.2 Metatheoretical Properties

Lennon-Bertrand argues in [30] that in order to completely certify a reduction-based conversion checker, the three following properties characterizing normal forms and conversion are enough. These properties are not immediately derivable from the presentation of the type theory and the main goal of this paper is to establish these results using a logical relation argument depicted in Sections 3 and 4.

The first property provides the main characterization of type conversion.

▶ **Property 4** (Injectivity and no-confusion of type formers). *Assume two convertible well-formed types $\Gamma \vdash A \equiv B$ are in whnf. Then, either they share the same type constructor and their respective subterms are also convertible, or they are convertible neutrals at the universe.*

Property 4 implies in particular injectivity of Π types, a key lemma for establishing subject reduction that propagates typing assumptions across reduction. The property also implies that two types with distinct head normal forms will never be convertible, an important point for establishing completeness of conversion checkers.

The second property aims to characterize conversion between neutral terms. To state the property, we must first introduce a distinction between types equipped with an η-law, an extensionality rule, that we will call *negative*, and types without such an η law, called *positive*. Concretely, we consider $\mathbb{U}$, $\mathbb{N}$, $\bot$ and neutral types are positive, while Π is negative.[1]

▶ **Property 5** (Inversion of neutral conversions at positive types). *Assume two well-formed neutral terms $\Gamma \vdash a \equiv b : A$ are convertible at a positive type. Then, they must be the same eliminations of the same variable with convertible arguments (a full definition is given in Section A.1).*

This restriction to positive types is sufficient to show decidability of conversion, as conversion in negative types does not rely on it. Additionally, neutral terms at definitionally irrelevant types, such as a negative unit type, do not satisfy this property.

Finally, the third property ensures the existence of a path to a normal form.

▶ **Property 6** (Deep normalization). *A term t deeply normalizes at type A, written $\Downarrow^A t$, if t and A normalize to weak-head normal forms t' and A' and either*

- *A' is negative and the η-expansion of t deeply normalizes or*
- *A' is positive and the subterms of t deeply normalize at their respective types.*

A type A is said to deeply normalize, written $\Downarrow A$ if, it normalizes to a weak-head normal form whose subterms also deeply normalize (a full definition is given in Section A.2).

2.3 Partial Equivalence Relations (PERs) & Dependent PERs

Since we do not know a priori that well formed types and terms normalize, or even that they always have a whnf, we use PERs to manage this partiality. A PER on a type A is a symmetric and transitive binary relation $R : A \to A \to \text{Type}$. In particular, it restricts to an equivalence relation on reflexive points, those $x : A$ such that $R\, x\, x$ holds.

The notion of PER also extends to a dependent notion.

▶ **Definition 7** (Dependent PERs). *Let (A, R) be a PER, $B : \text{Type}$ a type, and $S : \{a_1\, a_2 : A\} \to R\, a_1\, a_2 \to B \to B \to \text{Type}$ a dependent relation. S is dependent PER when it satisfies:*
Irrelevance *for all $r, r' : R\, a_1\, a_2$, $S\, r\, b_1\, b_2$ implies $S\, r'\, b_1\, b_2$,*
Transport *for all $r : R\, a_1\, a_2$ and $r' : R\, a_1\, a_3$, $S\, r\, b_1\, b_2$ implies $S\, r'\, b_1\, b_2$,*
Symmetry *for all $r : R\, a_1\, a_2$, $S\, r\, b_1\, b_2$ implies $S\, (\text{sym}\, r)\, b_2\, b_1$,*
Transitivity *for all $r : R\, a_1\, a_2, r' : R\, a_2\, a_3$, $S\, r\, b_1\, b_2$ and $S\, r'\, b_2\, b_3$ imply $S\, (\text{trans}\, r\, r')\, b_1\, b_3$.*

Note how the irrelevance property stems from the fact that we consider our PERs to be proof-irrelevant, thus the family of relations is not allowed to extensionally distinguish between different witnesses of the base relation.

[1] When the unit satisfies an η-law (as it is the case in an extension considered in Section 4), then it becomes negative ( ).

3 A One-Pass Logical Relation for Normalization

In this section, we use a logical relations argument to establish the properties listed in Section 2.2. Our logical relation is formulated using PERs, whose inhabitants we call *reducible conversion witnesses*, or simply *reducibility witnesses*. We write $\Gamma \Vdash_l A \equiv B$ for the type of witnesses of reducible conversion between types A and B at level l in context Γ, and $\Gamma \Vdash_l a \equiv b : A \equiv B / R_{AB}$ for the type of witnesses of reducible conversion between terms a and b over $R_{AB} : \Gamma \Vdash_l A \equiv B$. As in other formalizations, these relations are defined mutually using small induction-recursion [17, 23].

We first highlight the salient aspects of the logical relation on neutral terms in Section 3.1, and then illustrate the construction in the case of function types (Section 3.2). Ultimately, we show that this logical relation forms a suitable model of the base type theory, allowing us to interpret the syntax into it and derive the desired metatheoretical properties (Section 3.6).

In practice, we proceed in several steps. We first establish basic properties of the relation, including irrelevance, stability under renaming, and the reify-reflect principle (Section 3.3). We then prove the core PER properties: symmetry, transport (Section 3.4), and finally transitivity. Next, we show that each introduction, elimination, and conversion rule of the type theory is satisfied by the reducibility model. This is done first at the level of a higher-order model, and then transferred systematically to a first-order model via contextualization, as explained in Section 3.5. This allows us to conclude on the so-called fundamental lemma of the logical relation, which establishes, by induction on the typing derivation, that all well-formed types and terms are reducible.

3.1 The Key Case of Neutral Terms

One degree of freedom that arises in most definitions of logical relations concerns the treatment of reducible conversion for neutral terms. For non-neutral terms, reducible conversion is largely determined by the requirements of eliminators, which must be able to decompose non-neutral terms into their constituents. By contrast, neutral terms are simply propagated unchanged; as a result, no property is strictly required on subterms for the logical relations argument to go through. The definition of reducible conversion of neutral terms could include *no more information* than conversion of these terms. Such a choice, however, would yield no informative result about neutral terms once the fundamental lemma has been established.

One major obstacle, compared to the case of normal forms, when defining reducible conversion for neutral terms is the absence of reducibility information for their subterms. Indeed, neutral terms may contain subterms at arbitrary types that do not appear in the type of the neutral term itself, and the induction-recursion principle prevents using the inductive definition of reducible conversion on those types inside the recursive definition.

For instance, in the case of a neutral application at the type of natural numbers, one might wish for a definition along the following lines

$$\frac{R_{AA'} : \Gamma \Vdash_l A \equiv A' \qquad \Gamma \Vdash_l a \equiv a' : A \equiv A' / R_{AA'} \qquad \Gamma, x : A \vdash B \equiv \mathbb{N} \qquad \cdots}{\Gamma \Vdash_l f \mathbin{@}_{(x:A)B} a \equiv f' \mathbin{@}_{(x:A')B'} a' : \mathbb{N} \equiv \mathbb{N} / R_{\mathbb{N}}}$$

but we cannot use $\Gamma \Vdash_l A \equiv A'$ as an argument of this definition (shaded in red), since this would introduce a non–strictly positive occurrence of the logical relation in the inductive–recursive definition.

Additionally, A might be an arbitrarily large type even while defining reducibility at small types, and for which we'd need a definition of reducibility that simply doesn't exist yet.

Our definition of reducible conversion for neutral terms can therefore only rely on simpler properties; however, these properties must still entail the behavior expected from the logical relation. Accordingly, we choose a property that precisely subsumes Property 5 and Property 6. To that end, we first define conversion of deeply normalizing terms.

▶ **Definition 8** (Conversion of deeply normalizing terms). *We define*

$$\Gamma \vdash A \Downarrow B = (\Gamma \vdash A \equiv B) \times \Downarrow A \times \Downarrow B$$
$$\Gamma \vdash t \ {}^{A}\Downarrow^{B} \ u = (\Gamma \vdash t \equiv u : A) \times (\Gamma \vdash A \equiv B) \times \Downarrow^{A} t \times \Downarrow^{B} u$$

Notice that this notion of conversion is heterogeneous on terms: the reason is that we cannot a priori prove that $\Downarrow^B t$ knowing solely that $\Downarrow^A t$ and $\Gamma \vdash A \equiv B$.

We then define neutral conversion $\Gamma \vdash a \sim_{ne} b : A$ () (a full definition is given in Section A.3) as the inductive relation containing conversion of variables and closed under all congruence rules for eliminators, where we additionally require subterms that are not annotations to also normalize. Here's a prototypical example for applications:

$$\frac{\Gamma \vdash f \sim_{ne} f' : (\Pi(x : A)B) \qquad \Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash B \equiv B' \qquad \Gamma \vdash a \ {}^{A}\Downarrow^{A'} \ a' \qquad \Gamma \vdash R \equiv B[a/x] \qquad \Gamma \vdash R \equiv B'[a'/x]}{\Gamma \vdash (f \ @_{(x:A)B} \ a) \sim_{ne} (f' \ @_{(x:A')B'} \ a') : R}$$

The type annotations A, B and A', B' are merely required to be pairwise convertible, but the applied arguments a, a' need to normalize at their respective types, following our definition of deep normalization for neutral terms.

An important technical point is that, thanks to the presence of annotations on applications, the types at which arguments normalize are determined directly by the terms themselves. In a syntax without such annotations, one would instead need to infer the type of neutral terms out of the information provided by the context for variables, whereas the logical relation only requires checking neutrals against a given reducible type. These two modes do not compose well: a neutral term checked against a positive type cannot serve as the scrutinee of an elimination form that expects an inferring neutral, an obstruction to establish the reify-reflect property (Theorem 11). In this light, the fact that annotations can be inferred should be understood as a corollary of the metatheoretical properties of the annotated system.

3.2 Reducibility at Π

While we will not reproduce the full inductive-recursive definition of the logical relation (), the case of functions is quite informative.

▶ **Definition 9** (Reducible conversion as function types). *Two terms T and T' in context Γ are said to be reducibly convertible as function types at level l if they satisfy the following conditions:*

- $\Gamma \vdash_l T \rightsquigarrow^* (\Pi(x : A)B)$
- $\Gamma \vdash_l T' \rightsquigarrow^* (\Pi(x : A')B')$
- $R_A : \forall(w_f : \vdash \Delta)(\rho : \Delta \geq \Gamma), \Delta \Vdash_l A[\rho] \equiv A'[\rho]$
- $R_B : \forall(w_f : \vdash \Delta)(\rho : \Delta \geq \Gamma)\ a\ a',$
  $\Delta \Vdash_l a \equiv a' : A[\rho] \equiv A'[\rho]/R_A\ w_f\ \rho \ \rightarrow \Delta \Vdash_l B[\rho][a/x] \equiv B'[\rho][a'/x]$

▶ **Definition 10** (Reducible conversion of terms at function types). *We say that terms f and g are reducibly convertible at the above reducibility witness if*

- $\Gamma \vdash f \rightsquigarrow^* f' : \Pi(x : A)B$

- $\Gamma \vdash g \rightsquigarrow^* g' : \Pi(x : A')B'$
- $\forall (w_f : \vdash \Delta)(\rho : \Delta \geq \Gamma)\ a\ a'\ (r_{aa'} : \Delta \Vdash_l a \equiv a' : A \equiv A'/R_A\ w_f\ \rho)$,
  $\Delta \Vdash_l f'\ @_{(x:A[\rho])B[\uparrow\rho]}\ a \equiv g'\ @_{(x:A'[\rho])B'[\uparrow\rho]}\ a' : B[\rho][a/x] \equiv B'[\rho][a'/x]/R_B\ w_f\ \rho\ r_{aa'}$

where $\uparrow \rho : (\Delta, x : A[\rho]) \geq (\Gamma, x : A)$ *is the lifting of* ρ.

Here, notice how we are introducing annotations on applications coming from the given types T and T' in a deterministic way, and those annotations will have to be swapped to prove transport. We will come back to this issue in Section 3.4. Also, because our functions satisfy an η-law, we do not need reducibility witnesses to distinguish neutral terms from other terms, and there is no neutral case for reducible conversion of terms at function types.

More generally, for terms at negative types, the reducibility witness only needs to know when its eliminations are reducible, but nothing about the shape of the term itself. This can be contrasted with positive types, whose reducibility witnesses of terms have cases for each constructor as well as neutral terms.

3.3 Reify-Reflect

Early on in the proof, we want to recover the fact that reducible conversion implies regular conversion, both for types and terms, as well as normalization of types and terms at their respective types. This will also let us derive Property 6 directly once we have reducibility witnesses of all well-typed terms. This statement is called *reify*.

However, to reify a function, we need to apply it to a variable first, and that variable needs to be reducible. In order to prove that variables are reducible, we need to consider the more general property that any two convertible neutral terms are reducibly convertible, as the case of functions introduces neutral applications. This statement is called *reflect*.

This proof is mutual (because we need to reify the reducible argument of a function in order to reflect it), and the corresponding lemma is therefore called *reify-reflect*.

▶ **Theorem 11** (Reify-reflect). *$R_{AB} : \Gamma \Vdash_l A \equiv B$ implies that*

Reify type *if l is small, then $\Gamma \vdash A\ {}^{\mathbb{U}}\Downarrow^{\mathbb{U}} B$ else $\Gamma \vdash A \Downarrow B$;*

Reify *for all $\Gamma \Vdash_l a \equiv b : A \equiv B/R_{AB}$, we have $\Gamma \vdash a\ {}^{A}\Downarrow^{B} b$;*

Reflect *for all $\Gamma \vdash a \sim_{ne} b : A$, we have $\Gamma \Vdash_l a \equiv b : A \equiv B/R_{AB}$.*

Our choice of terminology here is inspired by the NbE literature, where reify and reflect are functions respectively from the evaluation domain to normal forms and from neutral forms to the evaluation domain.

This theorem comes very early in the proof, requiring nothing but the definition of the logical relation. In other formalizations such as LogRelMLTT and LogRelCoq, reify (called escape in those projects) is proven before reflect, non-mutually. The reason is that the corresponding logical relations additionally contain the conclusion of reify at *e.g.,* function types and terms, so that we do not need reflect to prove reify. The tradeoff comes from having to prove additional invariants when inhabiting the logical relation.

3.4 Transport

Transport expresses the fact that reducibly convertible types should have the exact same reducibly convertible terms, in a form suited to PERs.

▶ **Lemma 12** (Right transport). *If we have $R_{AB} : \Gamma \Vdash_l A \equiv B$ and $R_{AC} : \Gamma \Vdash_l A \equiv C$, then $\Gamma \Vdash_l a \equiv b : A \equiv B/R_{AB}$ is equivalent to $\Gamma \Vdash_l a \equiv b : A \equiv C/R_{AC}$.*

This lemma is central to the proof of transitivity, as for Π types, it is necessary to decompose a reducible conversion $\Gamma \Vdash_l a \equiv c : A \equiv C/R_{AC}$ into reducible conversions $\Gamma \Vdash_l a \equiv c : A \equiv B/R_{AB}$ and $\Gamma \Vdash_l c \equiv c : B \equiv C/R_{BC}$. One way to see this requirement is as a PER analogue of the fibrancy conditions on exponentiable functors, where being Conduché [21, 12] is critically used for the composition operation of the resulting category.

The proof of the transport lemma is mostly straightforward on positive types, as two reducible conversion R_{AB} and R_{AC} necessarily reach the same cases by determinism of the reduction, giving the same reducibility conversion relation for terms.

The only difficulty appears for eliminators of negative types, as in that case the reducibility conversion relation for terms injects new annotations that did not exist in the terms being compared. These annotations come from the types at which the terms are compared, and may syntactically differ for arbitrary types B and C. We first need to be able to change these annotations by suitably convertible ones, captured by the following predicate relating two terms that differ only by convertible annotations, meaning on applications in our simple setting.

▶ **Definition 13** (Equality of terms up to annotation). $\Gamma \vdash a \sim_{annot} b$ *is defined inductively as*

$$\frac{}{\Gamma \vdash a \sim_{annot} a} \qquad \frac{\Gamma \vdash A \Downarrow A' \qquad (\Gamma, x : A) \vdash B \Downarrow B' \qquad \Gamma \vdash f \sim_{annot} g \qquad \Gamma \vdash a \ {}^{A}\Downarrow^{A'} \ a}{\Gamma \vdash (f \ @_{(x:A)B} \ a) \sim_{annot} (g \ @_{(x:A')B'} \ a)}$$

With this definition, we are now able to state our desired lemma on changing annotations of reducibly convertible terms.

▶ **Lemma 14** (Change of annotations). *If* $\Gamma \Vdash_l a \equiv b : A \equiv B/R_{AB}$ *and* $\Gamma \vdash b \sim_{annot} b'$ *then* $\Gamma \Vdash_l a \equiv b' : A \equiv B/R_{AB}$.

Consider the case for the lemma above of a neutral application $f \ @_{(x:A)B} \ a$. Our definition of neutral conversion requires a proof of normalization of a at a type A' convertible to A, but because we do not know a priori that terms normalizing at one type also normalize at a convertible type, we need this information to be present in the equality of terms up to annotation. Once this lemma is proven, transport follows easily.

3.5 Higher Order Models and Contextualization

Once the structural rules are proven, we would like to show that the logical relation we've built constitutes a kind of model of our type theory. For our purposes, we will use the definitions of models of Bocquet, Kaposi and Sattler [8], more specifically, the concepts of higher order and first order models internal to an arbitrary presheaf category.

The former relies on the exponential of the presheaf category to express dependencies of judgements in a context-free way, similar to Higher Order Abstract Syntax (HOAS), while the latter makes the contexts explicit in the model, eschewing the dependency on any higher order feature of the host category. While we cannot use those definitions directly as our model will be given by PERs, we can still apply the same ideas *mutatis mutandis*.

In the case of logical relations for open terms, we use the presheaf category over the category of syntactic contexts and renamings. By definition, presheaves over that category are indexed by a syntactic context and are stable by renamings. In particular, the explicit context object Ctx of a first order model is also indexed by a syntactic context Γ and correspond externally to telescopes over Γ, $\mathrm{Ctx}(\Gamma) = \{\Theta \mid \Gamma \vdash \Theta\}$.

Notice how the proof that reducibility is stable by renaming shows that $\Gamma \mapsto \Gamma \Vdash_l A \equiv B$ and $\Gamma \mapsto \Gamma \Vdash_l a \equiv b : A \equiv B/R_{AB}$ are PER presheaves. The so-called Kripke quantifications $\forall (w_f : \vdash \Delta)(\rho : \Delta \geq \Gamma), \dots$ appearing in the reducibility of Π types in Definitions 9 and 10 correspond to the exponentials of the presheaf category, barring naturality conditions that are unnecessary in our case as the logical relation is proof-irrelevant.

▶ Remark 15 (Higher order PER model). $\Gamma \mapsto \Gamma \Vdash_l A \equiv B$ and $\Gamma \mapsto \Gamma \Vdash_l a \equiv b : A \equiv B/R_{AB}$ are the presheaves of types and terms of a (displayed) higher order PER model internal to presheaves over renamings.[2]

However, once this is proven, we do not yet have a result on the syntactic presentation of the type theory itself: we can only interpret syntax by induction in first order models (the syntactic model is only initial with respect to first order models). Thankfully, Bocquet et al. also give a general construction turning any higher order model into a first order model, called contextualization. This construction amounts to building explicit contexts as lists of semantics types internally to the presheaf category, then extending the notion of term of a type to closing substitutions of a context.

More formally, suppose we have types $\vdash \mathrm{Ty}$ and $A : \mathrm{Ty} \vdash \mathrm{Tm}\, A$ in the internal language of the presheaf category, define $\vdash \mathrm{Ctx}$ inductively mutually with the function $[\![-]\!]$ decoding contexts as types:

$$\frac{}{\varepsilon : \mathrm{Ctx}} \qquad [\![\varepsilon]\!] \coloneqq \top \qquad \frac{\Gamma : \mathrm{Ctx} \qquad A : \forall (\gamma : [\![\Gamma]\!]), \mathrm{Ty}}{\Gamma, x : A : \mathrm{Ctx}} \qquad [\![\Gamma, x : A]\!] \coloneqq (\gamma : [\![\Gamma]\!]) \times \mathrm{Tm}(A\ \gamma)$$

With this new definition, a type $\overline{\mathrm{Ty}}\ \Gamma$ in context Γ is then defined as $\forall (\gamma : [\![\Gamma]\!]), \mathrm{Ty}$ and a term $\overline{\mathrm{Tm}}\ \Gamma\ A$ of type A as $\forall (\gamma : [\![\Gamma]\!]), \mathrm{Tm}(A\ \gamma)$.

This specific construction is also present in other formalizations, but is not linked back to this theoretical work: in LogRelMLTT and LogRelCoq, this step is called validity, while in McTT and [34], the resulting object is given the name of semantic typing judgement.

We'd like however to highlight a simplification that is present in all formalizations including our work when compared to the formal definition above: as we've mentioned in the definition of first order models, the explicit context object should itself depend on another context, and would therefore represent reducible telescopes over arbitrary syntactic contexts. However, because the judgements of our type theory only mention closed telescopes, we are able to define a more restricted object that only characterizes this notion. Analytically this time, define RedCtx as the following inductive recursive type (🐉):

$$\frac{}{\varepsilon : \mathrm{RedCtx}} \qquad \frac{\Gamma : \mathrm{RedCtx} \qquad A : \forall (w_f : \vdash \Delta)(\gamma : [\![\Gamma]\!](\Delta, w_f)), \mathrm{Ty}(\Delta)}{(\Gamma, x : A) : \mathrm{RedCtx}}$$

$$[\![\varepsilon]\!](\Delta, w_f) \coloneqq \top \qquad [\![\Gamma, x : A]\!](\Delta, w_f) \coloneqq (\gamma : [\![\Gamma]\!](\Delta, w_f)) \times \mathrm{Tm}(\Delta)(A\ w_f\ \gamma)$$

This helps avoiding tedious manipulations of substitutions in telescopes, at no cost.

With the definition of reducibility of contexts and substitutions, we can prove that it gives rise to a first order model (🐉), finally concluding with the "fundamental lemma" by induction (🐉).

[2] While we do not have a formalized notion of higher order PER model and a corresponding single statement for this remark, all the separate lemmas for each rule are in the `LogicalRelation/Introductions/` (🐉) folder of the formalization.

$$\textsc{Raise}\quad \frac{\Gamma \vdash A}{\Gamma \vdash \mathtt{raise}\ A : A}$$

$$\begin{array}{lcl} (\mathtt{raise}\ C)\ @_{(x:A)B}\ a & \rightsquigarrow & \mathtt{raise}\ B[a/x] \\ \mathtt{ind}_\bot\ (x.P)\ h_r\ (\mathtt{raise}\ C) & \rightsquigarrow & h_r \\ \mathtt{ind}_\mathbb{N}\ (n.P)\ h_0\ (n.x.h_\mathtt{S})\ h_r\ (\mathtt{raise}\ C) & \rightsquigarrow & h_r \end{array}$$

$$\textsc{EmptyElim}\quad \frac{\Gamma \vdash t : \bot \qquad \Gamma, x : \bot \vdash P \qquad \Gamma \vdash h_r : P[\mathtt{raise}\ \bot/x]}{\Gamma \vdash \mathtt{ind}_\bot\ (x.P)\ h_r\ t : P[t/x]}$$

$$\textsc{NatElim}\quad \frac{\Gamma \vdash t : \mathbb{N} \qquad \Gamma, n : \mathbb{N} \vdash P \qquad \Gamma \vdash h_0 : P[\mathtt{0}/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_\mathtt{S} : P[\mathtt{S}\ n/n] \qquad \Gamma \vdash h_r : P[\mathtt{raise}\ \mathbb{N}/n]}{\Gamma \vdash \mathtt{ind}_\mathbb{N}\ (n.P)\ h_0\ (n.x.h_\mathtt{S})\ h_r\ t : P[t/n]}$$

Figure 2 Extension of MLTT to ExcTT (excerpt, see Section B for other rules)

3.6 Corollaries of the Fundamental Lemma

We can now return to the properties of Section 2.2 with the fundamental lemma in hand. Deep normalization (Property 6, in) is obtained easily by inspecting the content of the logical relation associated to each judgement through the fundamental lemma. For no-confusion of type formers (Property 4, in) and the characterization of neutral conversion (Property 5 in), we proceed by a further induction on the witness of reducible conversion induced by a derivation of conversion of the types at hand.

Additionally, we easily obtain the coherence of the type theory (), meaning that the empty type $\bot$ is not inhabited, as a form of canonicity, and subject reduction (), that typing is stable by reduction of the judgement's subject.

Overall, the whole development represents 6.9k lines of Rocq according to `coqwc`, with around 4.2k lines of proof script. The part dedicated to the definition of the type system and its immediate metatheory correspond to around 1.8k lines.

4 Case Study: Exceptional Type Theory

To assess whether our implementation of the logical relations proof technique keeps the metatheoretical burden under control, we experimented with an extension of MLTT featuring a unit type with a definitional η-law, together with definitional proof irrelevance for $\bot$. The corresponding formalization is available in a dedicated branch () and is sufficiently straightforward that we do not expand on it further here.

As a more involved experiment, we extend the core type theory with a primitive `raise` operation for raising exceptions, following the design of ExcTT [38, 39]. This extension is modest in size but affects all existing type formers, since it introduces new inhabitants at every type (). Moreover, exceptions are annotated and must propagate the type at which they are raised, thereby stressing the challenging aspects of type annotations in our approach. One benefit of considering this extension is that it yields the first mechanized canonicity result for ExcTT as a direct corollary.

4.1 A Primer on Exceptional Type Theory

ExcTT presented in Figure 2 extends MLTT with a single operation $\mathtt{raise}\,A$ parametrized by a type A. The term $\mathtt{raise}\,A$ is an element of A standing for a (call-by-name) exception at that type. When an exception occurs on the left-hand side of an application, it is propagated at

the adequate type, the codomain of the application substituted with the argument. Inductive eliminators ($\mathtt{ind}_{\mathbb{N}}, \mathtt{ind}_{\bot}$) are adapted to catch exceptions by adding an extra premise (shaded in green) for the exceptional case, together with a corresponding reduction rule.

Adding exceptions makes the theory immediately inconsistent since $\bot$ is inhabited by $\mathtt{raise}\,\bot$. Using syntactic models, Pédrot and Tabareau [38] show that the theory enjoys a weaker consistency property: the translation of a closed inhabitant of $\bot$ is convertible to the translation of $\mathtt{raise}\,\bot$. However, as the authors of [39, p.21] state: *"[...]we do not know if canonocity holds for the standard conversion [...] because this result amounts to showing the completeness of computational laws with respect to the new constants introduced"*[*sic*]. The following section answers this question in the affirmative.

4.2 Challenges of the Formalization of ExcTT

Extending the syntactic metatheory of MLTT to ExcTT presents few challenges, except for the propagation of the typing annotation C of $\mathtt{raise}\,C$ along the reduction rules presented in Figure 2. In these rules, the annotation on the left-hand side is never inspected; instead, the information contained in the annotations of applications or in the arguments of the eliminators is used to determine the right-hand side. As a consequence, there is no need for a congruence rule on exception annotations with respect to $\rightsquigarrow$. This implies that $\mathtt{raise}\,C$ is a whnf that is not stuck on a variable, and in particular, is never neutral.

The definition of the logical relation must be adapted for type formers whose inhabitants are characterized inductively, namely $\mathbb{N}$, $\bot$, $\mathbb{U}$, and neutral types. By contrast, the logical relation for type formers characterized by their elimination rules, such as Π-types, remains unchanged.

For $A = \mathbb{N}, \bot, \mathtt{raise}\,\mathbb{U}$ or neutral, the additional case to be added is uniform: Two terms t and u are reducible at type A when they both reduce to exceptions whose annotations are convertible with A:

$$(\Gamma \vdash t \rightsquigarrow^* \mathtt{raise}\,A_t : A_t) \times (\Gamma \vdash u \rightsquigarrow^* \mathtt{raise}\,A_u : A_u) \times (\Gamma \vdash A_t \equiv A) \times (\Gamma \vdash A_u \equiv A)$$

With this adapted logical relation, the additional cases introduced in the proof are straightforward, with the exception of the transport case. Following the discussion of Section 3.4, the relation $\Gamma \vdash t \sim_{annot} t'$ needs to take care of the type annotation carried by an exception. This means adding $\Gamma \vdash \mathtt{raise}\,A \sim_{annot} \mathtt{raise}\,A'$ whenever $\Gamma \vdash A \equiv A'$ so that $\sim_{annot}$ remains a simulation with respect to $\rightsquigarrow$.

With these properties in hand, we prove by induction on $R_{AB} : \Gamma \Vdash_l A \equiv B$ that $\Gamma \Vdash_l \mathtt{raise}\,A \equiv \mathtt{raise}\,B : A \equiv B / R_{AB}$. In each case, A reduces either to a type whose elements are characterized inductively and for which we added a case for $\mathtt{raise}\,A$, or to a type characterized by its elimination forms (only Π here) where the appropriate reduction together with the induction hypothesis allows to conclude.

Completing the fundamental lemma, we can then obtain the following canonicity property for ExcTT.

▶ **Theorem 16** (Rocq). *If* $\vdash t : \bot$ *then* $\vdash t \equiv \mathtt{raise}\,\bot : \bot$.

Overall, the formalization of ExcTT represents an additional 1.5kloc.

5 Comparing Logical Relations for Normalization

Our logical relation was designed to characterize conversion abstractly and derive only the normalization and inversion properties required for further proofs on implementations of

the type theory. Other design choices, that we discuss now, appear in the literature: on the representation of normalization, on algorithmic aspects, proof-relevance and arity of the relation.

5.1 Different Approaches to Normalization

The statement of normalization usually refers to the existence of normal forms representing the equivalence classes induced by conversion. Concretely, [14, 15, 49, 22, 50, 8] obtain a normalization function $\mathtt{norm}$ that assesses the unique existence of a normal form $\mathtt{norm}^A\ t$ for a term $t : A$. This normalization function must be *sound* ($t \equiv \mathtt{norm}^A\ t$) and *complete* ($t \equiv t' : A \iff \mathtt{norm}^A\ t = \mathtt{norm}^A\ t'$). In mechanized work [3, 4] however, this statement can be strengthened with an intensional characterization of the conversion relation between a term and its normal form: Property 6 not only requires the existence of a normal form $\mathtt{norm}^A t$ but also a witness of conversion between t and $\mathtt{norm}^A\ t$ using a standard derivation, namely one obtained by an alternation of reduction to weak-head normal form and extensionality principles. This further characterization is then employed to prove the correctness of different implementation strategies for conversion checkers, for instance dropping some of the typing information as in Coquand's algorithm [13, 30], or delaying the propagation of substitutions to reduce terms' traversals.

5.2 Algorithmic Aspects of the Logical Relation

Logical relations can also be employed to verify directly the correctness of an implementation of a conversion checker, designing relations with an algorithmic flavor directly within the logical relation. NbE, devised by [7] and pioneered for dependent types in [1], provides an effective technique to directly prove the soundness and completeness of an evaluator, with mechanizations both in Agda [6, 24] and Rocq [52, 25]. As shown in [25], the resulting evaluator can be directly extracted and employed as a basis for a verified checker for type theory. Technically, NbE relies on the definition of a domain of values to which terms evaluate together with a reflection of values as terms. NbE employs two logical relations, one to show that the evaluator maps conversions of terms to a designed PER on values (completeness) and another to show that the reflection of the evaluation of a term is convertible to the original term (soundness). This methodology requires however to design and fix the evaluator from the very beginning. Changes to the evaluation or conversion checking strategy, for instance to allow for fine grained conversion checking and early termination in case of failure, likely require modifications to the logical relations themselves.

Abel et al. [3] propose an alternative approach to NbE suitable for mechanization where the logical relation is parametrized by an interface abstracting over the presentation of the conversion, further extended to all typing judgements in [4]. This interface allows for two presentations of conversion: an algebraic presentation called declarative as we use in this work, and an algorithmic presentation with stronger inversion principles. In [3, 4], the logical relation is instantiated multiple times to obtain more properties of the type theory: injectivity of normal forms with the declarative instance, normalization and characterization of neutral conversion with the algorithmic instance.[3] This modular design of the logical relation is versatile enough to adapt to non-trivial extensions to a proof-irrelevant sort of propositions [19],

[3] [4] further instantiate the logical relation a third time with a bidirectional typing judgement to obtain completeness of bidirectional typing, but this can also be obtained purely syntactically from the other properties [29].

with graded modalities [2], to Observational Type Theory (ObsTT) [41, 43, 42], with a `quote`-operator [37], to first-class universe levels [16] or to sheaf models [35]. The strong inversion principles of the algorithmic presentation make it possible to prove the correction of Coquand's algorithm with purely syntactical manipulations [30] (no additional logical relation) but Lennon-Bertrand also observes that only some of the properties derived from the algorithmic presentation of conversion are really necessary for his proof. These properties, recalled in Section 2.2, can be equipped directly on the declarative presentation of conversion as explained in Section 3.1, bypassing entirely any algorithmic design prior to establishing the metatheory of the type theory and allowing our mechanization to use a single logical relation argument.

5.3 Proof-Relevant and Proof-Irrelevant Reducibility Witnesses

Most mechanized logical relations for dependent types, such as LogRelMLTT, McTT or the present work, enforce and leverage an essential property of type reducibility witnesses: that they are proof-irrelevant. Proof-irrelevance means that the formalization never depends on the actual computational content of reducibility proofs, that can then be taken as opaque black-boxes, as term reducibility only depends on the shape of its type index.

This is in sharp contrast with the definitions of type reducibility that can be found in pen-and-paper proofs [49, 14], where instead of prescribing which reducibility witnesses exist by induction recursion, reducibility witnesses are described by the structure they must be equipped with—a.k.a. *normalization structure*. These do not satisfy proof-irrelevance, and thus the reducibility predicate of typed terms depends on the reducibility proof for the type.

This has some advantages, primarily avoiding the use of a global reduction relation external to the definition of the type theory: in our logical relation, reduction is crucial to ensure that the different reducibility witnesses are non-overlapping, which is key to proof-irrelevance. Indeed, it is quite remarkable that metatheoretical results can be derived without referring to any specific operational semantics, although one could argue that in the proof-relevant case, the proof itself *is* the operational semantics.

However, the main drawback for mechanization is that the computational behavior of reducibility proofs is often highly intricate: equational reasoning tends to pollute proof terms with numerous rewriting steps, leading to what may be described as a form of "path-algebra hell". Kaposi [26] observes that a complete canonicity proof for the simply typed lambda-calculus requires 190 lines of Agda code, mostly consisting of equational algebra, which can be compared with a 40 lines argument obtained by turning some equations into conversions using (a priori unsafe) rewriting. Further strictification techniques [27] are then needed with a corresponding operational cost on the mechanization.

An alternative for mechanization is to embrace strong extensionality principles, such as Li et al. [31] who rely on the features of the Istari proof assistant [28], but these features also come with the cost of an undecidable typechecking problem that must then be guided by the user.

5.4 Unary vs Binary Logical Relations

Both unary logical relations (a.k.a. logical predicate or reducibility candidates) and binary logical relations are employed to prove normalization of dependent type theories with a clear divide depending on the formalization medium: most pen-and-paper formalization use the unary version while mechanized formalizations often rely on the binary version.

The binary version can be seen as a workaround to the limitations of the identity type in off-the-shelf intentional type theories: equality of reducibility witnesses would not be extensional enough to represent reducible conversion. One example symptom of this issue is that without quotients in the metatheory, one cannot build an initial model where conversion *is* equality. While [14, p.2][4] argues that a PER model should only be necessary for interpreting an extensional type theory, the existence of such an initial model is central to recent paper proofs of normalization.

A mechanized construction of the initial model using a quotient of the syntax of terms by conversion has been achieved in [10] using (axiomatized) effective definitionally proof-irrelevant quotients, propositional and function extensionality. Such a model could be also built in any metatheory supporting quotients, for example in Lean, or using ObsTT.

Working without such quotients, mechanizations in Rocq and Agda need to show explicitly that the logical relation preserves conversion in the form of a PER model.

Liu and Weirich's recent work [33] stands as an exception amongst mechanized results and only uses a proof-irrelevant unary logical predicate for normalization, characterizing conversion by mere untyped joinability. This characterization is sufficient because, by the definition of their type theory, Π-types are already assumed to be injective, meaning subject reduction holds. However, this non-standard type theory can only be shown to be equivalent to ours by first showing Property 4, which requires such a PER logical relation, as the authors point out [33, p. 22].

6 Conclusion and Future Work

We have shown that logical relations provide a direct and uniform way to derive the meta-theoretical properties required for the correctness of type-checking algorithms in dependent type theory, while cleanly separating algorithmic reasoning from logically strong arguments. For a core type theory with Π-types, natural numbers, the empty type, and a universe, this yields a compact and conceptually streamlined development. We further demonstrated that this approach scales beyond the core theory by extending it to ExcTT, obtaining the first mechanized canonicity proof for a dependent type theory with exceptions. This case study shows that logical relations can accommodate non-standard computational features without obscuring the metatheoretical structure.

Future Work. A natural next step is to increase the level of tactic automation in the development. While the current mechanization closely follows the mathematical structure of the logical relations argument, many proof patterns are repetitive and could be discharged more systematically. Improving automation would not only reduce proof size and maintenance costs, but also make the framework more accessible for extending the metatheory to additional type formers and effects. Another important direction concerns *sort polymorphism*, as introduced in the Rocq Prover and studied in recent work [40, 44]. Supporting sort-polymorphic universes would significantly increase the expressiveness of the framework and bring it closer to the type theory implemented in modern proof assistants. We believe that the modular separation between algorithmic and logical components advocated in this work will facilitate such an integration.

[4] *"Yet another aspect that was not satisfactory in previous attempts [1,8] is that it involved essentially a partial equivalence relation model. While one expects that this would be needed for a type theory with an extensional equality, this should not be necessary for the present, intensional, version of type theory. This issue disappears here: we only consider predicates (that are proof-relevant)."*

References

1 Andreas Abel. Normalization by evaluation: Dependent types and impredicativity.

2 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):920–954, 2023. doi:10.1145/3607862.

3 Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.*, 2(POPL):23:1–23:29, 2018. doi:10.1145/3158111.

4 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.

5 Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In David H. Pitt, David E. Rydeheard, and Peter T. Johnstone, editors, *Category Theory and Computer Science, 6th International Conference, CTCS '95, Cambridge, UK, August 7-11, 1995, Proceedings*, volume 953 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 1995. doi:10.1007/3-540-60164-3_27.

6 Thorsten Altenkirch and Ambrus Kaposi. Normalisation by evaluation for type theory, in type theory. *Log. Methods Comput. Sci.*, 13(4), 2017. doi:10.23638/LMCS-13(4:1)2017.

7 Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed lambda-calculus. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 203–211. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151645.

8 Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. For the metatheory of type theory, internal sconing is enough. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, Rome, Italy, July 3-6, 2023*, volume 260 of *LIPIcs*, pages 18:1–18:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.FSCD.2023.18, doi:10.4230/LIPICS.FSCD.2023.18.

9 Mathis Bouverot-Dupuis, Théo Winterhalter, Kenji Maillard, and Kathrin Stark. Sulfur: Substitution generation using a logical framework. 15m talk in RocqPL session; extended abstract available, January 2026.

10 Guillaume Brunerie and Menno Gilbers. initiality. https://github.com/guillaumebrunerie/initiality/tree/master, 2020.

11 Mario Carneiro. Lean4lean: Verifying a typechecker for lean, in lean, 2025. URL: https://arxiv.org/abs/2403.14064, arXiv:2403.14064.

12 François Conduché. Algèbre des catégories. - au sujet de l'existence d'adjoints à droite aux foncteurs "image réciproque" dans la catégorie des catégories. *Comptes rendus hebdomadaires des séances de l'Académie des sciences. Séries A et B, Sciences mathématiques et Sciences physiques, Académie des sciences (France)*, 1972. Public domain. 1972/11/06 (SERA-B,T275,N19). Ark identifier: ark:/12148/bpt6k56191352. Source: Archives de l'Académie des sciences. Provenance: Bibliothèque nationale de France. Online since 2010-12-01. URL: http://catalogue.bnf.fr/ark:/12148/cb34416987n.

13 Thierry Coquand. An algorithm for type-checking dependent types. *Sci. Comput. Program.*, 26(1-3):167–177, 1996. doi:10.1016/0167-6423(95)00021-6.

14 Thierry Coquand. Canonicity and normalization for dependent type theory. *Theor. Comput. Sci.*, 777:184–191, 2019. URL: https://doi.org/10.1016/j.tcs.2019.01.015, doi:10.1016/J.TCS.2019.01.015.

15 Thierry Coquand. Reduction free normalisation for a proof irrelevant type of propositions. *Log. Methods Comput. Sci.*, 19(3), 2023. URL: https://doi.org/10.46298/lmcs-19(3:5)2023, doi:10.46298/LMCS-19(3:5)2023.

16 Nils Anders Danielsson, Naïm Camille Favier, and Ondřej Kubánek. Normalisation for first-class universe levels. *Proc. ACM Program. Lang.*, 10(POPL), January 2026. doi:10.1145/3776645.

17 Peter Dybjer and Anton Setzer. Indexed induction-recursion. *J. Log. Algebraic Methods Program.*, 66(1):1–49, 2006. URL: https://doi.org/10.1016/j.jlap.2005.07.001, doi:10.1016/J.JLAP.2005.07.001.

18 Marcelo Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. *Math. Struct. Comput. Sci.*, 32(8):1028–1065, 2022. doi:10.1017/S0960129522000263.

19 Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without K. *Proc. ACM Program. Lang.*, 3(POPL):3:1–3:28, 2019. doi:10.1145/3290316.

20 Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, jun 1972.

21 Jean Giraud. Méthode de la descente. *Bulletin de la Société mathématique de France. Mémoire*, (2):156, 1964. doi:10.24033/msmf.2.

22 Daniel Gratzer. Normalization for multimodal type theory. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 2:1–2:13. ACM, 2022. doi:10.1145/3531130.3532398.

23 Peter G. Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. Small induction recursion. In Masahito Hasegawa, editor, *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*, volume 7941 of *Lecture Notes in Computer Science*, pages 156–172. Springer, 2013. doi:10.1007/978-3-642-38946-7_13.

24 Jason Z. S. Hu, Junyoung Jang, and Brigitte Pientka. Normalization by evaluation for modal dependent type theory. *J. Funct. Program.*, 33, 2023. URL: https://doi.org/10.1017/s0956796823000060, doi:10.1017/S0956796823000060.

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), August 2025. doi:10.1145/3747511.

26 Ambrus Kaposi. Towards quotient inductive-inductive-recursive types. *TYPES, 2023*, 2023. URL: https://types2023.webs.upv.es/TYPES2023.pdf.

27 Ambrus Kaposi and Loïc Pujet. Type theory in type theory using a strictified syntax. *Proc. ACM Program. Lang.*, 9(ICFP), August 2025. doi:10.1145/3747535.

28 Carl Krary. The istari proof assistant. URL: https://istarilogic.org/.

29 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, Rome, Italy (Virtual Conference), June 29 - July 1, 2021*, volume 193 of *LIPIcs*, pages 24:1–24:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.ITP.2021.24, doi:10.4230/LIPICS.ITP.2021.24.

30 Meven Lennon-Bertrand. What does it take to certify a conversion checker? In Maribel Fernández, editor, *10th International Conference on Formal Structures for Computation and Deduction, FSCD 2025, July 14-20, 2025, Birmingham, UK*, volume 337 of *LIPIcs*, pages 27:1–27:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. URL: https://doi.org/10.4230/LIPIcs.FSCD.2025.27, doi:10.4230/LIPICS.FSCD.2025.27.

31 Runming Li, Yue Yao, and Robert Harper. Mechanizing synthetic tait computability in istari. In Kathrin Stark, Yannick Zakowski, Nikhil Swamy, and Nicolas Tabareau, editors, *Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2026, Rennes, France, January 12-13, 2026*, pages 231–247. ACM, 2026. doi:10.1145/3779031.3779085.

32 Bohdan Liesnikov and Jesper Cockx. Building a correct-by-construction type checker for a dependently typed core language. In *Programming Languages and Systems: 22nd Asian*

Symposium, APLAS 2024, Kyoto, Japan, October 22-24, 2024, Proceedings, page 63–83, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-981-97-8943-6_4.

33 Yiyun Liu and Stephanie Weirich. Algorithmic conversion with surjective pairing: A syntactic and untyped approach. *Proc. ACM Program. Lang.*, 10(POPL), January 2026. doi:10.1145/3776672.

34 Yiyun Liu and Stephanie Weirich. Algorithmic conversion with surjective pairing: A syntactic and untyped approach. *Proc. ACM Program. Lang.*, 10(POPL), January 2026. doi:10.1145/3776672.

35 Pierre-Marie Pédrot Martin Baillon, Assia Mahboubi. In cantor space no one can hear you stream. *ESOP*, 2026. URL: https://www.pÃ¯drot.fr/articles/esop2026.pdf.

36 Per Martin-Löf. An intuitionistic theory of types. In Giovanni Sambin and Jan M. Smith, editors, *Twenty Five Years of Constructive Type Theory*, pages 127–172. Clarendon Press, 1998. Also known as MLTT72, published version of an unpublished 1972 note.

37 Pierre-Marie Pédrot. "upon this quote I will build my church thesis". In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 65:1–65:12. ACM, 2024. doi:10.1145/3661814.3662070.

38 Pierre-Marie Pédrot and Nicolas Tabareau. Failure is not an option - an exceptional type theory. In Amal Ahmed, editor, *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10801 of *Lecture Notes in Computer Science*, pages 245–271. Springer, 2018. doi:10.1007/978-3-319-89884-1_9.

39 Pierre-Marie Pédrot, Nicolas Tabareau, Hans Jacob Fehrmann, and Éric Tanter. A reasonably exceptional type theory. *Proc. ACM Program. Lang.*, 3(ICFP):108:1–108:29, 2019. doi:10.1145/3341712.

40 Josselin Poiret, Gaëtan Gilbert, Kenji Maillard, Pierre-Marie Pédrot, Matthieu Sozeau, Nicolas Tabareau, and Éric Tanter. All your base are belong to $\mathcal{U}_s$: Sort polymorphism for proof assistants. *Proceedings of the ACM on Programming Languages*, 9(POPL):76:1–76:29, 2025. doi:10.1145/3704912.

41 Loïc Pujet, Yann Leray, and Nicolas Tabareau. Observational equality meets CIC. *ACM Trans. Program. Lang. Syst.*, 47(2):6:1–6:35, 2025. doi:10.1145/3719342.

42 Loïc Pujet and Nicolas Tabareau. Observational equality: now for good. *Proc. ACM Program. Lang.*, 6(POPL):1–27, 2022. doi:10.1145/3498693.

43 Loïc Pujet and Nicolas Tabareau. Observational equality meets CIC. In Stephanie Weirich, editor, *Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I*, volume 14576 of *Lecture Notes in Computer Science*, pages 275–301. Springer, 2024. doi:10.1007/978-3-031-57262-3_12.

44 Johann Rosain, Tomás Díaz, Kenji Maillard, Matthieu Sozeau, Nicolas Tabareau, Éric Tanter, and Théo Winterhalter. Bounded sort polymorphism with elimination constraints. In *Proceedings of the 2026 ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, 2026. doi:10.1145/3776732.

45 Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq Project. *Journal of Automated Reasoning*, February 2020. URL: https://hal.inria.fr/hal-02167423, doi:10.1007/s10817-019-09540-0.

46 Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq coq correct! Verification of type checking and erasure for Coq, in Coq. *Proceedings of the ACM on Programming Languages*, 4(POPL), December 2019. doi:10.1145/3371076.

47 Matthieu Sozeau and Cyprien Mangin. Equations reloaded: high-level dependently-typed functional programming and proving in coq. *Proc. ACM Program. Lang.*, 3(ICFP), July 2019. doi:10.1145/3341690.

48 Kathrin Stark, Steven Schäfer, and Jonas Kaiser. Autosubst 2: reasoning with multi-sorted de bruijn terms and vector substitutions. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 166–180. ACM, 2019. doi:10.1145/3293880.3294101.

49 Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory*. PhD thesis, Carnegie Mellon University, 2021. Version 1.1, revised May 2022. doi:10.5281/zenodo.6990769.

50 Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–15. IEEE, 2021. doi:10.1109/LICS52264.2021.9470719.

51 William W. Tait. Intensional interpretations of functionals of finite type I. *J. Symb. Log.*, 32(2):198–212, 1967. doi:10.2307/2271658.

52 Paweł Wieczorek and Dariusz Biernacki. A Coq formalization of normalization by evaluation for Martin-Löf type theory. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, page 266–279, New York, NY, USA. Association for Computing Machinery. doi:10.1145/3167091.

A Core Type Theory

$$\textsc{InCtxHere}\ \frac{}{(x : A) \in \Gamma, x : A} \qquad \textsc{InCtxThere}\ \frac{(x : A) \in \Gamma}{(x : A) \in \Gamma, x : B}$$

$$\textsc{CtxNil}\ \frac{}{\vdash \cdot} \qquad \textsc{CtxCons}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A}{\vdash \Gamma, x : A}$$

$$\textsc{Var}\ \frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{Pi}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi(x : A)B}$$

$$\textsc{ConvPi}\ \frac{\Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash B \equiv B'}{\Gamma \vdash \Pi(x : A)B \equiv \Pi(x : A')B'} \qquad \textsc{U}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{U}} \qquad \textsc{PiU}\ \frac{\Gamma \vdash A : \mathbb{U} \qquad \Gamma, x : A \vdash B : \mathbb{U}}{\Gamma \vdash \Pi(x : A)B : \mathbb{U}}$$

$$\textsc{ConvPiU}\ \frac{\Gamma \vdash A \equiv A' : \mathbb{U} \qquad \Gamma, x : A \vdash B \equiv B' : \mathbb{U}}{\Gamma \vdash \Pi(x : A)B \equiv \Pi(x : A')B' : \mathbb{U}} \qquad \textsc{NatU}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathbb{N} : \mathbb{U}} \qquad \textsc{EmptyU}\ \frac{\vdash \Gamma}{\Gamma \vdash \bot : \mathbb{U}}$$

$$\textsc{El}\ \frac{\Gamma \vdash A : \mathbb{U}}{\Gamma \vdash A} \qquad \textsc{ConvEl}\ \frac{\Gamma \vdash A \equiv A' : \mathbb{U}}{\Gamma \vdash A \equiv A'} \qquad \textsc{Lam}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : \Pi(x : A)B}$$

$$\textsc{ConvLam}\ \frac{\Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash t \equiv t' : B}{\Gamma \vdash \lambda x : A.t \equiv \lambda x : A'.t' : \Pi(x : A)B} \qquad \textsc{App}\ \frac{\Gamma \vdash f : \Pi(x : A)B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ @_{(x:A)B}\ a : B[a/x]}$$

$$\textsc{ConvBeta}\ \frac{\Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash t : B \qquad \Gamma \vdash a : A}{\Gamma \vdash \lambda x : A'.B't\ @_{(x:A)B}\ a \equiv t[a/x] : B[a/x]}$$

$$\textsc{ConvEta}\ \frac{\Gamma \vdash f : \Pi(x : A)B}{\Gamma \vdash f \equiv \lambda x : A.f\ @_{(x:A)B}\ x : \Pi(x : A)B}$$

$$\textsc{ConvApp}\ \frac{\Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash B \equiv B' \qquad \Gamma \vdash f \equiv f' : \Pi(x : A)B \qquad \Gamma \vdash a \equiv a' : A}{\Gamma \vdash f\ @_{(x:A)B}\ a \equiv f'\ @_{(x:A')B'}\ a' : B[a/x]}$$

$$\textsc{Zero}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathtt{0} : \mathbb{N}} \qquad \textsc{Succ}\ \frac{\Gamma \vdash t : \mathbb{N}}{\Gamma \vdash \mathtt{S}\ t : \mathbb{N}} \qquad \textsc{ConvSucc}\ \frac{\Gamma \vdash t \equiv u : \mathbb{N}}{\Gamma \vdash \mathtt{S}\ t \equiv \mathtt{S}\ u : \mathbb{N}}$$

$$\textsc{NatElim}\ \frac{\Gamma \vdash t : \mathbb{N} \qquad \Gamma, n : \mathbb{N} \vdash P \qquad \Gamma \vdash h_0 : P[\mathtt{0}/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_\mathtt{S} : P[\mathtt{S}\ n/n]}{\Gamma \vdash \mathtt{ind}_\mathbb{N}\ (n.P)\ h_0\ (n.x.h_\mathtt{S})\ t : P[t/n]}$$

$$\textsc{NatElimZero}\ \frac{\Gamma, n : \mathbb{N} \vdash P \qquad \Gamma \vdash h_0 : P[\mathtt{0}/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_\mathtt{S} : P[\mathtt{S}\ n/n]}{\Gamma \vdash \mathtt{ind}_\mathbb{N}\ (n.P)\ h_0\ (n.x.h_\mathtt{S})\ \mathtt{0} \equiv h_0 : P[\mathtt{0}/n]}$$

$$\textsc{NatElimSucc}\ \frac{\Gamma \vdash t : \mathbb{N} \qquad \Gamma, n : \mathbb{N} \vdash P \qquad \Gamma \vdash h_0 : P[\mathtt{0}/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_\mathtt{S} : P[\mathtt{S}\ n/n]}{\Gamma \vdash \mathtt{ind}_\mathbb{N}\ (n.P)\ h_0\ (n.x.h_\mathtt{S})\ (\mathtt{S}\ t) \equiv h_\mathtt{S}[\mathtt{ind}_\mathbb{N}\ (n.P)\ h_0\ (n.x.h_\mathtt{S})\ t/x, t/n] : P[\mathtt{S}\ t/n]}$$

$$\textsc{ConvNatElim}\ \frac{\begin{array}{c}\Gamma \vdash t \equiv t' : \mathbb{N} \\ \Gamma, n : \mathbb{N} \vdash P \equiv P' \qquad \Gamma \vdash h_0 \equiv h_0' : P[\mathtt{0}/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_\mathtt{S} \equiv h_\mathtt{S}' : P[\mathtt{S}\ n/n]\end{array}}{\Gamma \vdash \mathtt{ind}_\mathbb{N}\ (n.P)\ h_0\ (n.x.h_\mathtt{S})\ t \equiv \mathtt{ind}_\mathbb{N}\ (n.P')\ h_0'\ (n.x.h_\mathtt{S}')\ t' : P[t/n]}$$

$$\textsc{EmptyElim}\ \frac{\Gamma \vdash t : \bot \qquad \Gamma, x : \bot \vdash P}{\Gamma \vdash \mathtt{ind}_\bot\ (x.P)\ t : P[t/x]} \qquad \textsc{ConvEmptyElim}\ \frac{\Gamma \vdash t \equiv t' : \bot \qquad \Gamma, x : \bot \vdash P \equiv P'}{\Gamma \vdash \mathtt{ind}_\bot\ (x.P)\ t \equiv \mathtt{ind}_\bot\ (x.P')\ t' : P[t/x]}$$

$$\textsc{ConvTyRefl}\ \frac{\Gamma \vdash A}{\Gamma \vdash A \equiv A} \qquad \textsc{ConvTySym}\ \frac{\Gamma \vdash A \equiv B}{\Gamma \vdash B \equiv A} \qquad \textsc{ConvTyTrans}\ \frac{\Gamma \vdash A \equiv B \qquad \Gamma \vdash B \equiv C}{\Gamma \vdash A \equiv C}$$

$$\textsc{Conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash t : B} \qquad \textsc{ConvConv}\ \frac{\Gamma \vdash t \equiv u : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash t \equiv u : B} \qquad \textsc{ConvTmRefl}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t \equiv t : A}$$

$$\textsc{ConvTmSym}\ \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \textsc{ConvTmTrans}\ \frac{\Gamma \vdash t \equiv u : A \qquad \Gamma \vdash u \equiv v : A}{\Gamma \vdash t \equiv v : A}$$

A.1 Characterization of Neutral Conversion

We define inductively the relation $\Gamma \vdash n \approx_{ne} n' : T$ of conversion at type T in context Γ between weak-head neutral terms n, n'. This relation allows to characterize conversion

between neutrals at positive types: Property 5 states that

$$\text{positive } T \quad \Longrightarrow \quad \forall n\, n', \text{whne}\, n \to \text{whne}\, n' \to (\Gamma \vdash n \equiv n' : T \iff \Gamma \vdash n \approx_{ne} n' : T)$$

$$\textsc{ConvNeVar}\ \frac{\vdash \Gamma \qquad (x : A) \in \Gamma \qquad \Gamma \vdash T \equiv A}{\Gamma \vdash x \approx_{ne} x : T}$$

$$\textsc{ConvNeApp}\ \frac{\Gamma \vdash f \approx_{ne} f' : \Pi(x : A)B \qquad \Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash B \equiv B' \qquad \Gamma \vdash a \equiv a' : A \qquad \Gamma \vdash T \equiv B[a/x] \qquad \Gamma \vdash T \equiv B'[a'/x]}{\Gamma \vdash f\ @_{(x:A)B}\ a \approx_{ne} f'\ @_{(x:A')B'}\ a' : T}$$

$$\textsc{ConvNeNatElim}\ \frac{\Gamma \vdash t \approx_{ne} t' : \mathbb{N} \qquad \Gamma, n : \mathbb{N} \vdash P \equiv P' \qquad \Gamma \vdash h_0 \equiv h_0' : P[0/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_S \equiv h_S' : P[S\ n/n] \qquad \Gamma \vdash T \equiv P[t/n] \qquad \Gamma \vdash T \equiv P'[t'/n]}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}\ (n.P)\ h_0\ (n.x.h_S)\ t \approx_{ne} \mathtt{ind}_{\mathbb{N}}\ (n.P')\ h_0'\ (n.x.h_S')\ t' : T}$$

$$\textsc{ConvNeEmptyElim}\ \frac{\Gamma \vdash t \approx_{ne} t' : \bot \qquad \Gamma, x : \bot \vdash P \equiv P' \qquad \Gamma \vdash T \equiv P[t/x] \qquad \Gamma \vdash T \equiv P'[t'/x]}{\Gamma \vdash \mathtt{ind}_{\bot}\ (x.P)\ t \approx_{ne} \mathtt{ind}_{\bot}\ (x.P')\ t' : T}$$

A.2 Deep Normalization

Deep normalization is defined using three predicates:

- normalization of types $\Downarrow A$;
- normalization of terms t at a type A, $\Downarrow^A t$;
- and normalization of neutral terms $\Downarrow^{\text{ne}} n$.

For arbitrary terms, a type is required to force η-expansions when applicable. In the neutral case, the structure of the term is sufficient to guide normalization and the type can be dropped. Access to the context is never required in these judgements and is left out. In the figure below, we use x to stand for object level variables.

$$\textsc{NormType}\ \frac{\mathtt{whnf}\, T = U \qquad \Downarrow U}{\Downarrow T} \qquad \textsc{NormTerm}\ \frac{\mathtt{whnf}\, T = U \qquad \mathtt{whnf}\, t = u \qquad \Downarrow^U u}{\Downarrow^T t} \qquad \textsc{PiTyNf}\ \frac{\Downarrow A \qquad \Downarrow B}{\Downarrow \Pi(x : A)B}$$

$$\textsc{NatTyNf}\ \frac{}{\Downarrow \mathbb{N}} \qquad \textsc{EmptyTyNf}\ \frac{}{\Downarrow \bot} \qquad \textsc{UnivTyNf}\ \frac{}{\Downarrow \mathbb{U}} \qquad \textsc{NeTyNf}\ \frac{\Downarrow^{\text{ne}} A}{\Downarrow A} \qquad \textsc{PiTmNf}\ \frac{\Downarrow^{\mathbb{U}} A \qquad \Downarrow^{\mathbb{U}} B}{\Downarrow^{\mathbb{U}} \Pi(x : A)B}$$

$$\textsc{NatTmNf}\ \frac{}{\Downarrow^{\mathbb{U}} \mathbb{N}} \qquad \textsc{EmptyTmNf}\ \frac{}{\Downarrow^{\mathbb{U}} \bot} \qquad \textsc{FunTmNf}\ \frac{\Downarrow^B f\ @_{(x:A)B}\ x}{\Downarrow^{\Pi(x:A)B} f} \qquad \textsc{ZeroTmNf}\ \frac{}{\Downarrow^{\mathbb{N}} 0} \qquad \textsc{SuccTmNf}\ \frac{\Downarrow^{\mathbb{N}} n}{\Downarrow^{\mathbb{N}} S\ n}$$

$$\textsc{NeTmNf}\ \frac{\Downarrow^{\text{ne}} n \qquad \text{positive } A}{\Downarrow^A n} \qquad \textsc{VarNe}\ \frac{}{\Downarrow^{\text{ne}} x} \qquad \textsc{AppNe}\ \frac{\Downarrow^{\text{ne}} f}{\Downarrow^{\text{ne}} f\ @_{(x:A)B}\ a}$$

$$\textsc{NatElimNe}\ \frac{\Downarrow^{\text{ne}} n \qquad \Downarrow P \qquad \Downarrow^{P[0/x]} h_0 \qquad \Downarrow^{P[S\ x/x]} h_S}{\Downarrow^{\text{ne}} \mathtt{ind}_{\mathbb{N}}\ (x.P)\ h_0\ (x.i.h_S)\ n} \qquad \textsc{EmptyElimNe}\ \frac{\Downarrow^{\text{ne}} n \qquad \Downarrow P}{\Downarrow^{\text{ne}} \mathtt{ind}_{\bot}\ (x.P)\ n}$$

A.3 Reducibility of Neutral Terms

$$\textsc{NeVar}\ \frac{\vdash \Gamma \qquad (x : A) \in \Gamma \qquad \Gamma \vdash T \equiv A}{\Gamma \vdash x \sim_{ne} x : T}$$

$$\textsc{NeApp}\ \frac{\Gamma \vdash f \sim_{ne} f' : \Pi(x : A)B \qquad \Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash B \equiv B' \qquad \Gamma \vdash a \ {}^{A}\Downarrow^{A'} \ a' \qquad \Gamma \vdash T \equiv B[a/x] \qquad \Gamma \vdash T \equiv B'[a'/x]}{\Gamma \vdash f \ @_{(x:A)B} \ a \sim_{ne} f' \ @_{(x:A')B'} \ a' : T}$$

$$\textsc{NeNatElim}\ \frac{\Gamma \vdash t \sim_{ne} t' : \mathbb{N} \qquad \Gamma, n : \mathbb{N} \vdash P \Downarrow P' \qquad \Gamma \vdash h_0 \ {}^{P[0/n]}\Downarrow^{P'[0/n]} \ h_0' \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_s \ {}^{P[s\ n/n]}\Downarrow^{P'[s\ n/n]} \ h_s' \qquad \Gamma \vdash T \equiv P[t/n] \qquad \Gamma \vdash T \equiv P'[t'/n]}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}\ (n.P)\ h_0\ (n.x.h_s)\ t \sim_{ne} \mathtt{ind}_{\mathbb{N}}\ (n.P')\ h_0'\ (n.x.h_s')\ t' : T}$$

$$\textsc{NeEmptyElim}\ \frac{\Gamma \vdash t \sim_{ne} t' : \bot \qquad \Gamma, x : \bot \vdash P \Downarrow P' \qquad \Gamma \vdash T \equiv P[t/x] \qquad \Gamma \vdash T \equiv P'[t'/x]}{\Gamma \vdash \mathtt{ind}_{\bot}\ (x.P)\ t \sim_{ne} \mathtt{ind}_{\bot}\ (x.P')\ t' : T}$$

B Exceptional Type Theory

$$\textsc{Raise}\ \frac{\Gamma \vdash A}{\Gamma \vdash \mathtt{raise}\ A : A} \qquad \textsc{ConvRaise}\ \frac{\Gamma \vdash A \equiv B}{\Gamma \vdash \mathtt{raise}\ A \equiv \mathtt{raise}\ B : A}$$

$$\textsc{ConvAppRaise}\ \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\mathtt{raise}\ (\Pi(x : A)B))\ @_{(x:A)B}\ a \equiv \mathtt{raise}\ B[a/x] : B[a/x]}$$

$$\textsc{NatElim}\ \frac{\Gamma \vdash t : \mathbb{N} \qquad \Gamma, n : \mathbb{N} \vdash P \qquad \Gamma \vdash h_0 : P[0/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_s : P[s\ n/n] \qquad \Gamma \vdash h_r : P[\mathtt{raise}\ \mathbb{N}/n]}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}\ (n.P)\ h_0\ (n.x.h_s)\ h_r\ t : P[t/n]}$$

$$\textsc{NatElimZero}\ \frac{\Gamma, n : \mathbb{N} \vdash P \qquad \Gamma \vdash h_0 : P[0/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_s : P[s\ n/n] \qquad \Gamma \vdash h_r : P[\mathtt{raise}\ \mathbb{N}/n]}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}\ (n.P)\ h_0\ (n.x.h_s)\ h_r\ 0 \equiv h_0 : P[0/n]}$$

$$\textsc{NatElimSucc}\ \frac{\Gamma \vdash t : \mathbb{N} \qquad \Gamma, n : \mathbb{N} \vdash P \qquad \Gamma \vdash h_0 : P[0/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_s : P[s\ n/n] \qquad \Gamma \vdash h_r : P[\mathtt{raise}\ \mathbb{N}/n]}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}\ (n.P)\ h_0\ (n.x.h_s)\ h_r\ (s\ t) \equiv h_s[\mathtt{ind}_{\mathbb{N}}\ (n.P)\ h_0\ (n.x.h_s)\ h_r\ t/x, t/n] : P[s\ t/n]}$$

$$\textsc{NatElimRaise}\ \frac{\Gamma, n : \mathbb{N} \vdash P \qquad \Gamma \vdash h_0 : P[0/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_s : P[s\ n/n] \qquad \Gamma \vdash h_r : P[\mathtt{raise}\ \mathbb{N}/n]}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}\ (n.P)\ h_0\ (n.x.h_s)\ h_r\ (\mathtt{raise}\ \mathbb{N}) \equiv h_r : P[\mathtt{raise}\ \mathbb{N}/n]}$$

ConvNatElim
$$\frac{\Gamma \vdash t \equiv t' : \mathbb{N} \qquad \Gamma, n : \mathbb{N} \vdash P \equiv P' \qquad \Gamma \vdash h_0 \equiv h'_0 : P[\mathtt{0}/n] \qquad \Gamma, n : \mathbb{N}, x : P \vdash h_{\mathtt{S}} \equiv h'_{\mathtt{S}} : P[\mathtt{S}\ n/n] \qquad \Gamma \vdash h_r \equiv h'_r : P[\mathtt{raise}\ \mathbb{N}/n]}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}\ (n.P)\ h_0\ (n.x.h_{\mathtt{S}})\ h_r\ t \equiv \mathtt{ind}_{\mathbb{N}}\ (n.P')\ h'_0\ (n.x.h'_{\mathtt{S}})\ h'_r\ t' : P[t/n]}$$

EmptyElim
$$\frac{\Gamma \vdash t : \bot \qquad \Gamma, x : \bot \vdash P \qquad \Gamma \vdash h_r : P[\mathtt{raise}\ \bot/x]}{\Gamma \vdash \mathtt{ind}_{\bot}\ (x.P)\ h_r\ t : P[t/x]}$$

EmptyElimRaise
$$\frac{\Gamma, x : \bot \vdash P \qquad \Gamma \vdash h_r : P[\mathtt{raise}\ \bot/x]}{\Gamma \vdash \mathtt{ind}_{\bot}\ (x.P)\ h_r\ (\mathtt{raise}\ \bot) \equiv h_r : P[\mathtt{raise}\ \bot/x]}$$

ConvEmptyElim
$$\frac{\Gamma \vdash t \equiv t' : \bot \qquad \Gamma, x : \bot \vdash P \equiv P' \qquad \Gamma \vdash h_r \equiv h'_r : P[\mathtt{raise}\ \bot/n]}{\Gamma \vdash \mathtt{ind}_{\bot}\ (x.P)\ h_r\ t \equiv \mathtt{ind}_{\bot}\ (x.P')\ h'_r\ t' : P[t/x]}$$