A Short and Mechanized Consistency Proof for Dependent Type Theory

YIYUN LIU, University of Pennsylvania, USA
STEPHANIE WEIRICH, University of Pennsylvania, USA

Proof by logical relation is a powerful technique that has been used to derive metatheoretic properties of type systems, such as consistency and parametricity. While there exists a plethora of introductory materials about logical relation in the context of simply typed or polymorphic lambda calculus, a streamlined presentation of proof by logical relation for a dependently language is lacking. In this paper, I present a short consistency proof for a dependently typed language that contains a rich set of features, including a full cumulative universe hierarchy, booleans, and an intensional identity type. We have fully mechanized the consistency proof using the Coq proof assistant in under 1000 lines of code.

Additional Key Words and Phrases: Logical Relation, Dependent Types, Logical Consistency, Coq

ACM Reference Format:
Yiyun Liu and Stephanie Weirich. 2024. A Short and Mechanized Consistency Proof for Dependent Type Theory. 1, 1 (February 2024), 23 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

When a dependently type system is used as a program logic where terms encode proofs, we want our type system to be logically consistent, meaning that the empty type is not inhabited.

The consistency proofs of various dependently typed systems, including Martin-Lof's type theory and the Calculus of Constructions, have long been available in the literature. In particular, recent works such as Wieczorek and Biernacki [2018], Abel et al. [2017], and Adjedj et al. [2024] mechanize stronger results related to the decidability of type conversion or type checking of dependently typed systems, from which consistency can be derived as a corollary.

The underlying technique of the forementioned works is proof by logical relation, which involves interpreting types as reducibility predicates, representing sets of terms satisfying certain properties with respect to the reduction relation. While the proof technique and the consistency result for dependent types are both well-established, there is a severe lack of rigorous and accessible material that shows how proof by logical relation can be applied to dependently typed systems.

Introductory materials about logical relations such as Skorstengaard [2019], Harper [2016] start from logical relations for simply typed languages, and eventually extend the technique to build a relational model for System F in order to derive parametricity. Harper [2022b] gives a gentle introduction on how logical relations for closed terms can be extended to talk about open terms so we can derive properties such as normalization for well-typed open terms. Overall, the introductory texts about logical relations cover systems and properties with varying degrees of complexity,

Authors' addresses: Yiyun Liu, University of Pennsylvania, Philadelphia, USA, liuyiyun@seas.upenn.edu; Stephanie Weirich, University of Pennsylvania, Philadelphia, USA, sweirich@seas.upenn.edu.

ACM XXXX-XXXX/2024/2-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

from simply typed to polymorphicly typed, logical predicate to logical equivalence, closed terms to open terms.

The glaring gap here is the lack of fully dependently typed systems where computations may appear at the type-level. It is far from obvious why proof by logical relation is even applicable to dependent types, since the type may very well-be a computation that is yet to be evaluated. While it is assuring that proof by logical relations for dependent types is available in mechanized forms, Abel et al. [2017]; Adjedj et al. [2024]; Wieczorek and Biernacki [2018] all involve relational, Kripke-style models that obscure the technique for addressing type-level computation. The added complexivity is evident from the size of their code base. All three developments involve 20 thousand to 30 thousand lines of Coq or Agda code.

The goal of this paper is to give a tutorial on proof by logical relation for dependent types in a simple and digestable format. The system we present is small but covers important features that are only available or require special treatments in dependently typed systems. We choose boolean types over natural numbers as our base type for simplicity, but include an intensional identity type in our type system to show how indexed types are treated in a logical relational proof. Unlike Abel et al. [2017]; Adjedj et al. [2024]; Wieczorek and Biernacki [2018] but similar to Anand and Rahli [2014], we include an infinite hierarchy of universes to not only support type-level computations, but also avoid the unnecessary code duplication pointed out by Wieczorek and Biernacki [2018] when big and small types are treated non-uniformly.

Our consistency proof is short and fully mechanized. The proof scripts involve less than 1000 lines of manually written Coq code. In fact, what we find encouraging is that among the 1000 lines of Coq code, 400 lines are related to the specification of the type system, semantics, and properties related to untyped lambda terms. The semantic type soundness proof through logical relation takes almost the same amount of code as our syntactic type soundness proof! Thanks to the conciseness of the proof, we are able to present it in detail in Sections 3 and 4. Moreover, the structure of our mechanization closely corresponds to the proof we present in the text, enabling us to label at the footnote each lemma directly to their counterpart in the proof script for the readers to reference and validate.

The technique we use is most similar to the one from Wieczorek and Biernacki [2018], which leverages impredicativity to define the logical relation as a partial function. Rather than framing impredicativity as a mechanism for encoding induction recursion [**?**], a scheme in which semantic models (including logical relations) for dependent types can be defined, we opt for a direct explanation through the informal language of sets, where impredicativity manifests in the form of second-order logical formulas and thus more intuitive to grasp for readers with a general mathematical background.

In Section 5, we extend our logical relation to prove the stronger property that every well-typed term has a normal form. Most interestingly, we find that extending the language with η rule for functions and the metatheoretic result to normalization for open terms are mostly orthogonal to the fact that the language is dependently typed. In our mechanized proof, the overall structure of the logical relation remains unchanged, and the additional proof obligations are mostly related to properties about the untyped lambda calculus, all of which can be derived through syntactic means. In other words, once we have established the base technique for proof by logical relation for dependent types, we can factor out the complexities of an extension the are not specifically related to dependent types; such an extension, if desired, may be studied in the context of a simply typed language and later ported into a dependently typed setting.

In Section 6, we discuss the details about our mechanization, including our use of existing libraries such as Autosubst 2 [Stark et al. 2019] for handling bindings and CoqHammer [Czajka and

Kaliszyk 2018] for general-purpose automation. In Section 7, we discuss how our proof relates to existing proofs by logical relations for dependent types in the literature.

We hope our success at creating a short and mechanized proof for a relatively feature-complete language will encourage future researchers to leverage the tool of logical relation more often in mechanized proofs for dependent types.

2 SPECIFICATION OF A DEPENDENT TYPE THEORY

Natural numbers			
i, j, n	$\in$	$\mathbb{N}$	
Contexts			
Γ	$::=$	$\cdot \mid \Gamma, A$	
Terms			
a, b, c, t, p, A, B	$::=$	$\mathbf{Set}\ i \mid \mathbf{v}_i \mid \mathbf{Void}$	universes, variables, empty type
	$\mid$	$\Pi\ A\ B \mid \lambda A.\ a \mid a\ b$	function types, abstractions, applications
	$\mid$	$a \sim b \in A \mid \mathbf{refl} \mid \mathbf{J}\ t\ a\ b\ p$	equality types, reflexivity proof, J eliminator
	$\mid$	$\mathbf{Bool} \mid \mathbf{true} \mid \mathbf{false}$	boolean type, true, false
	$\mid$	$\mathbf{if}\ a\ \mathbf{then}\ b_0\ \mathbf{else}\ b_1$	if
Renaming			
ξ	$\in$	$\mathbb{N} \to \mathbb{N}$	
Substitution			
ρ	$\in$	$\mathbb{N} \to \mathit{Term}$	

Fig. 1. Syntax of λ^H

In this section, we present the dynamics and statics of the dependent type theory whose logical consistency will be proven in Section 4. For concision, we refer to this system as λ^H.

The syntax of λ^H can be found in Figure 1. We use the unscoped de Bruijn representation for both our Coq development and the informal presentation in our paper. We find de Bruijn representation advantageous for specification since it leaves very little ambiguity about the variable freeness side conditions, making our proof more easily reproducible. Furthermore, as we discuss in Section 6.2, de Bruijn representation is much more amenable to automated reasoning in proof assistants. We adopt from **?** the notations for simultaneous renaming, substitution, and other auxiliary definitions related to our syntax, summarized in Figure 2. We omit most of the definitions of renaming and substitution and only show the definition of a few representative cases of substitution.

As a dependent type theory, terms and types are collapsed into the same syntactic category. Dependent functions take the form $\Pi\ A\ B$ and we use the notation $A \to B$ when the output type B is not dependent on the input variable. $\mathbf{Set}\ i$ represents the universe type where i ranges over the set of natural numbers. Finally, λ^H also includes an intensional identity type $a \sim b \in A$ whose proofs can be eliminated by the J-eliminator $\mathbf{J}\ t\ a\ b\ p$, where p is an equality proof between a and b, and t is the term whose type is to be casted.

$$
\begin{array}{lcl}
\textit{IdentityRen} & & \\
\mathbf{id}(i) & := & i \\
\\
\textit{IdentityTm} & & \\
\mathbf{id}_{tm}(i) & := & \mathbf{v}_i \\
\\
\textit{ConsRen}(\xi : n) & & \\
(\xi : n)(0) & := & n \\
(\xi : n)(1+i) & := & \xi(i) \\
\\
\textit{ConsSubst}(\rho : a) & & \\
(\rho : a)(0) & := & a \\
(\rho : a)(1+i) & := & \rho(i) \\
\\
\textit{Curried Add} & & \\
\uparrow^n (i) & := & n + i
\end{array}
\qquad
\begin{array}{lcl}
\textit{UpRen} & & \\
(\Uparrow \xi)(0) & := & 0 \\
(\Uparrow \xi)(1+i) & := & 1 + \xi(i) \\
\\
\textit{Renaming}(a\langle\xi\rangle) & & \\
(\Pi\, A\, B)\langle\xi\rangle & := & \Pi\, A\langle\xi\rangle\, B\langle\Uparrow \xi\rangle \\
(a\ b)\langle\xi\rangle & := & a\langle\xi\rangle\ b\langle\xi\rangle \\
\ldots & & \\
\\
\textit{UpTm} & & \\
(\Uparrow \rho)(0) & := & \mathbf{v}_0 \\
(\Uparrow \rho)(1+i) & := & \rho(i)\langle\uparrow^1\rangle \\
\\
\textit{Substitution}(a[\rho]) & & \\
\mathbf{v}_i[\rho] & := & \rho(i) \\
(\Pi\, A\, B)[\rho] & := & \Pi\, A[\rho]\, B[\Uparrow \rho] \\
(a\ b)[\rho] & := & a[\rho]\ b[\rho] \\
\ldots & &
\end{array}
$$

Fig. 2. Auxiliary Functions over Syntax

λ^H is expressive enough to support large elimination, the ability to compute a type using a term as input. For example, the function λ**Bool. if** $\mathbf{v}_0$ **then Bool else Bool** $\rightarrow$ **Bool** returns either **Bool** or **Bool** $\rightarrow$ **Bool** depending on whether the input is **true** or **false**.

Figure 3 shows the definition of the parallel reduction relation, which takes the form $a \Rightarrow b$. We use $a \Rightarrow^* b$ to represent its transitive and reflexive closure, which in turn allows us to define the coherence relation $a \Leftrightarrow b$. We say that two terms a and b are coherent if they can eventually reduce to some common term c through parallel reduction. The symmetric notation of coherence suggests that it is an equivalence relation.

We sketch out some key properties about parallel reduction and coherence without giving their proofs. Our technique for establishing those results is based on Takahashi [1995] and a modern exposition of the same technique can be found in Wadler et al. [2022].

Lemma 2.1 (Par Refl[1]). *For all terms a, $a \Rightarrow a$.*

Lemma 2.2 (Par cong[2]). *If $a_0 \Rightarrow a_1$ and $b_0 \Rightarrow b_1$, then $a_0[b_0] \Rightarrow a_1[b_1]$.*

Corollary 2.3 (Par subst[3]). *If $a_0 \Rightarrow a_1$, then $a_0[b] \Rightarrow a_1[b]$ for arbitrary b.*

Lemma 2.4 (Par diamond[4]). *If $a \Rightarrow b_0$ and $a \Rightarrow b_1$, then there exists some term c such that $b_0 \Rightarrow c$ and $b_1 \Rightarrow c$.*

Lemma 2.5 (Coherence refl[5]). *For all terms a, $a \Leftrightarrow a$.*

Lemma 2.6 (Coherence sym[6]). *If $a \Leftrightarrow b$, then $b \Leftrightarrow a$.*

Lemma 2.7 (Coherence trans[7]). *If $a_0 \Leftrightarrow a_1$ and $a_1 \Leftrightarrow a_2$, then $a_0 \Leftrightarrow a_2$.*

From Lemma 2.5, 2.6, and 2.7, we conclude that coherence is indeed an equivalence relation. It is then trivial to derive that coherence is exactly the same as the untyped β-equivalence.

[1] `join.v:Par_refl` [2] `join.v:par_cong` [3] `join.v:par_subst` [4] `join.v:par_confluent`
[5] `join.v:Coherent_reflexive` [6] `join.v:Coherent_symmetric` [7] `join.v:Coherent_transitive`

$\boxed{a \Rightarrow b}$ *(Parallel Reduction)*

$$\text{P-Var}\ \frac{}{\mathbf{v}_i \Rightarrow \mathbf{v}_i} \qquad \text{P-Set}\ \frac{}{\mathbf{Set}\ i \Rightarrow \mathbf{Set}\ i} \qquad \text{P-Void}\ \frac{}{\mathbf{Void} \Rightarrow \mathbf{Void}} \qquad \text{P-Pi}\ \frac{A_0 \Rightarrow A_1 \qquad B_0 \Rightarrow B_1}{\Pi\ A_0\ B_0 \Rightarrow \Pi\ A_1\ B_1}$$

$$\text{P-Abs}\ \frac{A_0 \Rightarrow A_1 \qquad a_0 \Rightarrow a_1}{\lambda A_0.\ a_0 \Rightarrow \lambda A_1.\ a_1} \qquad \text{P-App}\ \frac{a_0 \Rightarrow a_1 \qquad b_0 \Rightarrow b_1}{a_0\ b_0 \Rightarrow a_1\ b_1} \qquad \text{P-AppAbs}\ \frac{a_0 \Rightarrow a_1 \qquad b_0 \Rightarrow b_1}{(\lambda A.\ a_0)\ b_0 \Rightarrow a_1[b_1]}$$

$$\text{P-True}\ \frac{}{\mathbf{true} \Rightarrow \mathbf{true}} \qquad \text{P-False}\ \frac{}{\mathbf{false} \Rightarrow \mathbf{false}} \qquad \text{P-If}\ \frac{a_0 \Rightarrow a_1 \qquad b_0 \Rightarrow b_1 \qquad c_0 \Rightarrow c_1}{\mathbf{if}\ a_0\ \mathbf{then}\ b_0\ \mathbf{else}\ c_0 \Rightarrow \mathbf{if}\ a_1\ \mathbf{then}\ b_1\ \mathbf{else}\ c_1}$$

$$\text{P-IfTrue}\ \frac{b_0 \Rightarrow b_1}{\mathbf{if\ true\ then}\ b_0\ \mathbf{else}\ c_0 \Rightarrow b_1} \qquad \text{P-IfFalse}\ \frac{c_0 \Rightarrow c_1}{\mathbf{if\ false\ then}\ b_0\ \mathbf{else}\ c_0 \Rightarrow c_1} \qquad \text{P-Bool}\ \frac{}{\mathbf{Bool} \Rightarrow \mathbf{Bool}}$$

$$\text{P-Eq}\ \frac{a_0 \Rightarrow a_1 \qquad b_0 \Rightarrow b_1 \qquad A_0 \Rightarrow A_1}{a_0 \sim b_0 \in A_0 \Rightarrow a_1 \sim b_1 \in A_1} \qquad \text{P-Refl}\ \frac{}{\mathbf{refl} \Rightarrow \mathbf{refl}} \qquad \text{P-J}\ \frac{t_0 \Rightarrow t_1 \qquad a_0 \Rightarrow a_1 \qquad b_0 \Rightarrow b_1 \qquad p_0 \Rightarrow p_1}{\mathbf{J}\ t_0\ a_0\ b_0\ p_0 \Rightarrow \mathbf{J}\ t_1\ a_1\ b_1\ p_1}$$

$$\text{P-JRefl}\ \frac{t_0 \Rightarrow t_1}{\mathbf{J}\ t_0\ a_0\ b_0\ \mathbf{refl} \Rightarrow t_1}$$

$\boxed{a \Rightarrow^* b}$ *(Transitive Closure of Parallel Reduction)*

$$\text{PS-Refl}\ \frac{a \Rightarrow a}{a \Rightarrow^* a} \qquad \text{PS-Step}\ \frac{a \Rightarrow b \qquad b \Rightarrow^* c}{a \Rightarrow^* c}$$

$\boxed{a \Leftrightarrow b}$ *(Coherence)*

$$\text{C-Intro}\ \frac{a \Rightarrow^* c \qquad b \Rightarrow^* c}{a \Leftrightarrow b}$$

Fig. 3. Parallel reduction and coherence

Figure 4 gives the full typing rules for λ^H. The premises wrapped in gray boxes can be shown to be admissible syntactically, though some of them are required to strengthen the inductive hypothesis of the fundamental theorem. In rule T-Var, $\Gamma(i)$ is the partial function defined with the equations $(\Gamma, A)(0) := A$ and $(\Gamma, A)(1 + i) := \Gamma(i)$. The precondition $i < |\Gamma|$, where $|\Gamma|$ represents the length of the context Γ, ensures that $\Gamma(i)$ has a defined output. Rule T-Conv uses the definition of coherence from earlier as our equality judgment for type conversion. The use of an untyped relation for type conversion makes our formulation slightly different from languages such

$\boxed{\vdash \Gamma}$ *(Context Well-Formedness)*

$$\textsc{Ctx-Empty}\ \frac{}{\vdash \cdot} \qquad \textsc{Ctx-Cons}\ \frac{\vdash \Gamma \qquad \Gamma \vdash A : \mathbf{Set}\ i}{\vdash \Gamma, A}$$

$\boxed{\Gamma \vdash a : A}$ *(Typing)*

$$\textsc{T-Var}\ \frac{\vdash \Gamma \qquad i < |\Gamma|}{\Gamma \vdash \mathbf{v}_i : \Gamma(i)\langle\uparrow^{1+i}\rangle} \qquad \textsc{T-Set}\ \frac{\vdash \Gamma \qquad i < j}{\Gamma \vdash \mathbf{Set}\ i : \mathbf{Set}\ j}$$

$$\textsc{T-Pi}\ \frac{\Gamma \vdash A : \mathbf{Set}\ i \qquad \Gamma, A \vdash B : \mathbf{Set}\ i}{\Gamma \vdash \Pi\, A\, B : \mathbf{Set}\ i} \qquad \textsc{T-Abs}\ \frac{\Gamma \vdash \Pi\, A\, B : \mathbf{Set}\ i \qquad \Gamma, A \vdash b : B}{\Gamma \vdash \lambda A.\ b : \Pi\, A\, B}$$

$$\textsc{T-App}\ \frac{\Gamma \vdash b : \Pi\, A\, B \qquad \Gamma \vdash a : A}{\Gamma \vdash b\ a : B[a]} \qquad \textsc{T-Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash B : \mathbf{Set}\ i \qquad A \Leftrightarrow B}{\Gamma \vdash a : B} \qquad \textsc{T-Refl}\ \frac{\vdash \Gamma \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathbf{refl} : a \sim a \in A}$$

$$\textsc{T-J}\ \frac{\begin{array}{c}\Gamma \vdash a : A \\ \Gamma \vdash b : A \\ \Gamma \vdash A : \mathbf{Set}\ j \\ \Gamma \vdash p : a \sim b \in A \\ \Gamma, A, \mathbf{v}_0 \sim a\langle\uparrow^1\rangle \in A\langle\uparrow^1\rangle \vdash C : \mathbf{Set}\ i \\ \Gamma \vdash t : C[a, \mathbf{refl}]\end{array}}{\Gamma \vdash \mathbf{J}\ t\ a\ b\ p : C[b, p]} \qquad \textsc{T-If}\ \frac{\begin{array}{c}\Gamma, \mathbf{Bool} \vdash A : \mathbf{Set}\ i \\ \Gamma \vdash a : \mathbf{Bool} \\ \Gamma \vdash b_0 : A[\mathbf{true}] \\ \Gamma \vdash b_1 : A[\mathbf{false}]\end{array}}{\Gamma \vdash \mathbf{if}\ a\ \mathbf{then}\ b_0\ \mathbf{else}\ b_1 : A[a]}$$

$$\textsc{T-Bool}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathbf{Bool} : \mathbf{Set}\ i} \qquad \textsc{T-True}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathbf{true} : \mathbf{Bool}} \qquad \textsc{T-False}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathbf{false} : \mathbf{Bool}} \qquad \textsc{T-Void}\ \frac{\vdash \Gamma}{\Gamma \vdash \mathbf{Void} : \mathbf{Set}\ i}$$

Fig. 4. Syntactic typing for λ^H

as MLTT [Martin-Löf 1975], where the judgmental equality takes the form $\Gamma \vdash a \equiv b : A$, which implies $\Gamma \vdash a : A$ and $\Gamma \vdash b : A$, the well-typedness of a and b. Siles and Herbelin [2012] shows the equivalence of Barendregt's Pure Type System [Barendregt 1991], which employs untyped equality, and its variant that uses typed judgmental equality. It is trivial to embed a system with judgmental equality to a system with untyped equality by erasing typing information. As a result, it is easy to port our consistency result from our type system to a variant with typed judgmental equality as long as the typed system does not include η laws that would require type annotations (e.g. the η-law for unit types).

Working with a system with untyped equality has the huge benefit that the confluence result for untyped parallel reduction (Lemma 2.4) is easily derivable without having to resort to the complex syntatic (resp. semantic) technique from Siles and Herbelin [2012] (resp. Abel et al. [2017]) to resolve the circularity of subject reduction and Π-injectivity. Section 5 explains how we generalize our technique to include η-law for functions and show the existence of normal form for well-typed (open and closed) terms, achieving a similar level of expressiveness of the type system and strength of metatheoretic property as Abel et al. [2017].

Finally, since our system has an infinite universe hierarchy, we can present the system à la Russell by using the same judgment form $\Gamma \vdash a : A$ regardless of whether a is a term or a type. There is no need to distinguish between big types and small types and duplicate our typing specification.

3 LOGICAL RELATION

$$\boxed{[A]_i^I \searrow S} \qquad \textit{(Logical Relation)}$$

$$\text{I-Void}\ \frac{}{[\mathbf{Void}]_i^I \searrow \emptyset} \qquad \text{I-Bool}\ \frac{}{[\mathbf{Bool}]_i^I \searrow \{a \mid a \Rightarrow^* \mathbf{true} \vee a \Rightarrow^* \mathbf{false}\}}$$

$$\text{I-Eq}\ \frac{}{[a \sim b \in A]_i^I \searrow \{p \mid p \Rightarrow^* \mathbf{refl}, a \Leftrightarrow b\}} \qquad \text{I-Pi}\ \frac{\begin{array}{c}[A]_i^I \searrow S \\ F \in S \to \mathcal{P}(\mathit{Term}) \\ \forall a, \text{if } a \in S, \text{then } [B[a]]_i^I \searrow F(a)\end{array}}{[\Pi\, A\, B]_i^I \searrow \{b \mid \forall a, \text{if } a \in S, \text{then } b\ a \in F(a)\}}$$

$$\text{I-Set}\ \frac{j < i}{[\mathbf{Set}\, j]_i^I \searrow I(j)} \qquad \text{I-Red}\ \frac{A \Rightarrow B \qquad [B]_i^I \searrow S}{[A]_i^I \searrow S}$$

Fig. 5. Logical relation for λ^H

In this section, we define the logical relation for λ^H in the form of an inductively defined relation (Figure 5). Throughout the text, we use the notation $\mathcal{P}(S)$ denotes the powerset of the set S. The logical relation takes the form $[A]_i^I \searrow S$. The metavariables A and i stand for terms and natural numbers respectively, as introduced earlier in Figure 1. The metavariables I and S are sets with the following signatures.

$$I \in \{j \mid j < i\} \to \mathcal{P}(\mathit{Term})$$
$$S \in \mathcal{P}(\mathit{Term})$$

The function I is a family of sets of terms indexed by natural numbers strictly less than the parameter i, which represents the current universe level. In rule I-Set, function I is used to define the meaning of universes that are strictly smaller than the current level i. The restriction $j < i$ in rule I-Set is crucial for our system to be predicative. Removing the ordering constraint would result in a system where one can encode Girard's paradox [Girard 1972].

$$[A]_i \searrow S := [A]_i^I \searrow S, \text{where } I(i) := \{A \mid \exists S, [A]_i \searrow S\}$$

Fig. 6. Logical relation for all universe levels

To tie the knot and obtain an interpretation of all universe levels, we define in Figure 6 the final version of our interpretation judgment recursively using the well-foundedness of the strict less than relation on natural numbers (recall that the parameter I of $[A]_i^I \searrow S$ takes only natural numbers strictly less than i as its input). The judgment $[A]_i \searrow S$ now reads that the type A is a level-i type *semantically* inhabited by terms from the set S.

With the definition from Figure 6, we can show that the same introduction rules for $[A]_i^I \searrow S$ are admissible for $[A]_i \searrow S$, by instantiating I with $I(i) := \{A \mid \exists S, [A]_i \searrow S\}$, the same function I used in the definition of $[A]_i \searrow S$. We give as example the following rules.

$$\frac{}{[\mathbf{Void}]_i \searrow \emptyset}\ \text{IR-Void} \qquad \frac{j < i}{[\mathbf{Set}\, j]_i \searrow \{A \mid \exists S, [A]_j \searrow S\}}\ \text{IR-Set}$$

In most informal presentations, instead of defining the logical relation in two steps as we have shown above, the rules for $[A]_i \searrow S$ are given directly, with the implicit understanding that the relation is an inductive definition nested inside a recursive function over the universe level i. We choose the more explicit definition not only because it is directly definable in existing proof assistants where inductive definitions must appear at the top level, but also because it makes clear the induction principle we are allowed to use when reasoning about $[A]_i \searrow S$.

For the majority of the properties we are about to prove in this section, we do not need any information about the parameterized function I. Each property about $[A]_i \searrow S$ follows as a corollary of a property about $[A]_i^I \searrow S$ with no or few assumptions imposed on I. As a result, we usually state our lemmas in terms of $[A]_i^I \searrow S$ without duplicating them in terms of $[A]_i \searrow S$.

To derive consistency, it suffices to restrict S to the set of terms to reduce to closed terms. Rules I-Void and I-Bool are unsurprising; when considering only closed terms, the empty type should not be inhabited and therefore corresponds to the empty set, whereas the boolean type is semantically inhabited by terms that evaluate to the boolean values **true** or **false**. Rule I-Eq says that an equality type $a \sim b \in A$ corresonds to the set of terms that evaluate to **refl** when $a \Leftrightarrow b$ holds and otherwise corresponds to the empty set. Side conditions like $a \Leftrightarrow b$ are typically required for indexed types, of which equality types are an instance.

Rule I-Pi is the most interesting. The precondition consists of a mysterious function F that takes an element from the set S, the interpretation of the type A, and returns a subset of terms. F is further subject to the constraint that $\forall a$, if $a \in S$, then $[B[a]]_i^I \searrow F(a)$. Ignoring the content below the horizontal bar of the derivation, the statements from the precondition are equivalent to the conjunction of the following statements:

- $[A]_i^I \searrow S$
- $\forall a$, if $a \in S$, then $\exists S_0$, $[B[a]]_i^I \searrow S_0$

In general, let R be a binary relation over the sets A and B, it easy to verify the following equivalence, which is used to justify the skolemization process [**?**].

$$\forall a \in A, \exists b \in B \text{ such that } (a, b) \in R \iff \exists F \in A \to B, \forall a \in A, (a, F(a)) \in R$$

The left-hand side is equivalent to the right-hand side, but has no mentioning of any functions. In the context of rule I-Pi, through the equivalence, we can more easily tell what's happening: the function type $\Pi\, A\, B$ has a semantic interpretation if its input type A can be semantically interpreted as some set S, and for all terms $a \in S$, the type $B[a]$, obtained by substituing a into the output type B, has a semantic interpretation.

If we look at $[\Pi\, A\, B]_i^I \searrow \{b \mid \forall a, \text{if } a \in S, \text{then } b\ a \in F(a)\}$, the conclusion of rule I-Pi, we see why the presentation with an explicit $F \in S \to \mathcal{P}(Term)$ is useful. We want to state the fact that $B[a]$ has a semantic interpretation of some set S_0, but we also want to refer to the set S_0 when we talk about the set that $\Pi\, A\, B$ corresponds to. With the function F, we can use $F(a)$ to retrive the witness in the precondition for $a \in S$. With the alternative representation that is free of the function F, we might want to reformulate our logical relation in the following way.

I-PiAlt
$$\frac{[A]_i^I \searrow S \quad \forall a, \text{if } a \in S, \text{ then } \exists S_0, [B[a]]_i^I \searrow S_0}{[\Pi\, A\, B]_i^I \searrow \{b \mid \forall a, a \in S, \forall S_0, [B[a]]_i^I \searrow S_0, b\ a \in F(a)\}}$$

Unfortunately, rule I-PiAlt not only violates the syntactic strict positivity constraint required in proof assistants, but is genuinely non-monotone when written as an endofunction over the domain of relations. Intuitively, the failure of monotonicity stems from the fact that the witness picked in the precondition is not necessarily the same witness being referred to in the post condition. While it might be possible to restrict the domain with additional constraints such as functionality and inversion properties, we opt for our current skolemized formulation of rule I-Pi so we immediately obtain a well-defined inductive relation and a usable induction principle. The slight disadvantage of rule I-Pi is that we need to construct the function F each time we apply it. Therefore, in Lemma 3.5, we show the admissibility of rule I-PiAlt a posteriori and use that as a more convenient introduction rule for the rest of our proofs.

In the rest of the section, we establish some important facts about the logical relation that will be useful for proving fundamental theorem in Section 4.

The relation $[A]_i^I \searrow S$ satisfies the following inversion principles.

Lemma 3.1 (Inversion of the logical relation).

(1) *If* $[\mathbf{Void}]_i^I \searrow S$, *then* $S = \emptyset$.[8]
(2) *If* $[\mathbf{Bool}]_i^I \searrow S$, *then* $S = \{a \mid a \Rightarrow^* \mathbf{true} \vee a \Rightarrow^* \mathbf{false}\}$.[9]
(3) *If* $[a \sim b \in A]_i^I \searrow S$, *then* $S = \{p \mid p \Rightarrow^* \mathbf{refl}, a \Leftrightarrow b\}$.[10]
(4) *If* $[\Pi\, A\, B]_i^I \searrow S_0$, *then there exists* S, F *such that:*[11]
   - $[A]_i^I \searrow S$
   - $F \in S \to \mathcal{P}(Term)$
   - $\forall a$, *if* $a \in S$, *then* $[B[a]]_i^I \searrow F(a)$
   - $S_0 = \{b \mid \forall a, \text{if } a \in S, \text{ then } b\ a \in F(a)\}$
(5) *If* $[\mathbf{Set}\, j]_i^I \searrow S$, *then* $j < i$ *and* $S = I(j)$.[12]

Proof. We only show the most involved function case; the rest follows a similar but simpler pattern. We start by inducting over the derivation of $[\Pi\, A\, B]_i^I \searrow S$. There are only two possible cases we need to consider.

Rule I-Pi**:** Immediate.

Rule I-Red**:** We are given that $[\Pi\, A\, B]_i^I \searrow S_0$. We know that there exists some A_0 and B_0 such that $\Pi\, A\, B \Rightarrow \Pi\, A_0\, B_0$ and $[\Pi\, A_0\, B_0]_i^I \searrow S_0$. From the induction hypothesis, there exists S and F such that :
   - $[A_0]_i^I \searrow S$
   - $F \in S \to \mathcal{P}(Term)$
   - $\forall a, \text{if } a \in S, \text{ then } [B_0[a]]_i^I \searrow F(a)$
   - $S_0 = \{b \mid \forall a, \text{if } a \in S, \text{ then } b\ a \in F(a)\}$

   By inverting $\Pi\, A\, B \Rightarrow \Pi\, A_0\, B_0$, we derive $A \Rightarrow A_0$ and $B \Rightarrow B_0$. By Lemma 2.3, we have $B[a] \Rightarrow B_0[a]$ for all a. As a result, by rule I-Red, the same S and F satisfies the following properties:
   - $[A]_i^I \searrow S$
   - $F \in S \to \mathcal{P}(Term)$
   - $\forall a, \text{if } a \in S, \text{ then } [B[a]]_i^I \searrow F(a)$

[8] `semtyping.v:InterpExt_Void_inv` [9] `semtyping.v:InterpExt_Bool_inv` [10] `semtyping.v:InterpExt_Eq_inv`
[11] `semtyping.v:InterpExt_Fun_inv` [12] `semtyping.v:InterpExt_Univ_inv`

These properties are exactly what's required in the conclusion.

□

Rule I-Red bakes into the logical relation the backward preservation property. The following property shows that preservation holds in the usual forward direction too.

Lemma 3.2 (Preservation of the logical relation[13]). *If* $[A]_i^I \searrow S$ *and* $A \Rightarrow B$, *then* $[B]_i^I \searrow S$.

Proof. We carry out the proof by induction over the derivation of $[A]_i^I \searrow S$.

Rule I-Void**:** There exists some B such that $\mathbf{Void} \Rightarrow B$. By inverting the derivation of $\mathbf{Void} \Rightarrow B$, B must be **Void** and the result trivially follows.

Rules I-Bool **and** I-Set**:** Similar to the case for rule I-Void.

Rule I-Eq**:** We know that $[a \sim b \in A]_i^I \searrow \{p \mid p \Rightarrow^* \mathbf{refl}, a \Leftrightarrow b\}$ and, by inverting the derivation of parallel reduction, $a \Rightarrow a_0$, $b \Rightarrow b_0$, $A \Rightarrow A_0$ for some a_0, b_0, and A_0. Our goal is to show that $[a_0 \sim b_0 \in A]_i^I \searrow \{p \mid p \Rightarrow^* \mathbf{refl}, a \Leftrightarrow b\}$. By rule I-Eq, we already know that $[a_0 \sim b_0 \in A]_i^I \searrow \{p \mid p \Rightarrow^* \mathbf{refl}, a_0 \Leftrightarrow b_0\}$ and therefore it suffices to show that the sets $\{p \mid p \Rightarrow^* \mathbf{refl}, a \Leftrightarrow b\}$ and $\{p \mid p \Rightarrow^* \mathbf{refl}, a_0 \Leftrightarrow b_0\}$ are equal. Equivalently, it suffices to show that $a \Leftrightarrow b$ if and only if $a_0 \Leftrightarrow b_0$. By definition, from $a \Rightarrow a_0$ and $b \Rightarrow b_0$, we derive $a \Leftrightarrow a_0$ and $b \Leftrightarrow b_0$. The result then immediately follows from the fact that coherence is an equivalence relation.

Rule I-Pi**:** There exists S and F such that:

- $[\Pi\, A\, B]_i^I \searrow \{b \mid \forall a, \text{if } a \in S, \text{then } b\ a \in F(a)\}$
- $[A]_i^I \searrow S$
- $F \in S \to \mathcal{P}(Term)$
- $\forall a, \text{if } a \in S, \text{then } [B[a]]_i^I \searrow F(a)$

There exists some A_0 and B_0 such that $A \Rightarrow A_0$ and $B \Rightarrow B_0$. Our goal is to show that $[\Pi\, A_0\, B_0]_i^I \searrow \{b \mid \forall a, \text{if } a \in S, \text{then } b\ a \in F(a)\}$. By rule I-Pi, it suffices to show:

- $[A_0]_i^I \searrow S$
- $F \in S \to \mathcal{P}(Term)$
- $\forall a, \text{if } a \in S, \text{then } [B_0[a]]_i^I \searrow F(a)$

Since $A \Rightarrow A_0$ and $B[a] \Rightarrow B_0[a]$ forall a, the latter of which follows from Lemma 2.3, the above conditions follow immediately from the induction hypothesis.

Rule I-Red**:** There exists B_0 and S such that $A \Rightarrow B_0$ and $[B_0]_i^I \searrow S$. Given an arbitrary B such that $A \Rightarrow B$, our goal is to show that $[B]_i^I \searrow S$. By the diamond property of parallel reduction (Lemma 2.4), there exists some term C such that $B_0 \Rightarrow C$ and $B \Rightarrow C$. By the induction hypothesis, we derive $[C]_i^I \searrow S$ from $[B_0]_i^I \searrow S$. By rule I-Red and $B \Rightarrow C$, we conclude that $[B]_i^I \searrow S$.

□

From Lemma 3.2 and rule I-Red, we can easily derive the following corollary that two coherent types have the same interpretation.

Corollary 3.3 (Irrelevance of logical relation[14]). *If* $[A]_i^I \searrow S$ *and* $A \Leftrightarrow B$, *then* $[B]_i^I \searrow S$.

Next, we show that $[A]_i^I \searrow S$ is in fact a partial function.

Lemma 3.4 (Logical relation is functional[15]). *If* $[A]_i^I \searrow S_0$ *and* $[A]_i^I \searrow S_1$, *then* $S_0 = S_1$.

Proof. We start by inducting over the derivation of the first premise $[A]_i^I \searrow S_0$.

[13] `semtyping.v:InterpExt_preservation`

[14] `semtyping.v:InterpUnivN_Coherent`

[15] `semtyping.v:InterpExt_deterministic`

Rule I-Void**:** We know that $[\mathbf{Void}]_i^I \searrow \emptyset$. Given $[\mathbf{Void}]_i^I \searrow S_1$, our goal is to show that $\emptyset = S_1$. This is immediate by applying the **Void** case of Lemma 3.1 to $[\mathbf{Void}]_i^I \searrow S_1$.

Rules I-Bool**,** I-Eq**, and** I-Set**:** Similar to the rule I-Void case by applying the matching case of Lemma 3.1 to $[A]_i^I \searrow S_1$.

Rule I-Pi**:** There exists S and F such that:

- $[A]_i^I \searrow S$
- $F \in S \to \mathcal{P}(Term)$
- $\forall a$, if $a \in S$, then $[B[a]]_i^I \searrow F(a)$

Our goal is to show that given $[\Pi\, A\, B]_i^I \searrow S_1$, we have $S_1 = \{b \mid \forall a, \text{if } a \in S, \text{then } b\ a \in F(a)\}$. By the function case of Lemma 3.1, there exists some S_0 and F_0 such that:

- $[A]_i^I \searrow S_0$
- $F_0 \in S_0 \to \mathcal{P}(Term)$
- $\forall a$, if $a \in S_0$, then $[B[a]]_i^I \searrow F_0(a)$
- $S_1 = \{b \mid \forall a, \text{if } a \in S_0, \text{then } b\ a \in F_0(a)\}$

It suffices to show that $S = S_0$ and $F = F_0$. The equality $S = S_0$ immediately follows from the induction hypothesis since $[A]_i^I \searrow S$ and $[A]_i^I \searrow S_0$. Therefore, functions F and F_0 have the same domain and codomain and thus it suffices to show that forall $a \in S$, $F(a) = F_0(a)$. Suppose $a \in S$, we must have $[B[a]]_i^I \searrow F(a)$ and $[B[a]]_i^I \searrow F_0(a)$ from the two $\forall$-quantified statements above. The equality $F(a) = F_0(a)$ then immediately follows from the induction hypothesis.

Rule I-Red**:** There exists some B such that $A \Rightarrow B$ and $[B]_i^I \searrow S_0$. Our goal is to show that given $[A]_i^I \searrow S_1$, we have $S_0 = S_1$. By Lemma 3.2, from $[A]_i^I \searrow S_1$ and $A \Rightarrow B$, we have $[B]_i^I \searrow S_1$. From the induction hypothesis, we can conclude that $S_0 = S_1$ since $[B]_i^I \searrow S_0$ and $[B]_i^I \searrow S_1$.

□

Lemma 3.4 enables us to show the admissibility of rule I-PiAlt and its related inversion principle.

Lemma 3.5 (Pi Intro Alt[16]). *Suppose the following two statements hold:*

- $[A]_i^I \searrow S$
- *if* $\forall a, a \in S$, *then* $\exists S_0, [B[a]]_i^I \searrow S_0$

Then we have $[\Pi\, A\, B]_i^I \searrow \{b \mid \forall a, a \in S, \forall S_0, [B[a]]_i^I \searrow S_0, b\ a \in F(a)\}$

Proof. Let F be the relation defined as follows:

$$(a, S) \in F \iff [B[a]]_i^I \searrow S$$

By the second bullet from the premise and Lemma 3.4, F is a function that is total on the set S. The conclusion then trivially follows from rule I-Pi. □

Lemma 3.6 (Pi Inv Alt[17]). *Suppose* $[\Pi\, A\, B]_i^I \searrow S$, *then there exists some* S_0 *such that the following constraints hold:*

- $[A]_i^I \searrow S_0$
- $\forall a$, *if* $a \in S_0$, *then* $\exists S_1, [B[a]]_i^I \searrow S_1$
- $S = \{b \mid \forall a, a \in S_0, \forall S_1, [B[a]]_i^I \searrow S_1, b\ a \in S_1\}$

Proof. Immedaite from Lemmas 3.1 and 3.4. □

[16] `semtyping.v:InterpExt_Fun_nopf` [17] `semtyping.v:InterpExt_Fun_inv_nopf`

The next lemma shows that our logical relation satisfies cumulativity. That is, if a type has an interpretation at a lower universe level, then we can obtain the same interpretation at a higher universe level.

Lemma 3.7 (Logical relation cumulativity[18]). *If* $[A]^I_{i_0} \searrow S$ *and* $i_0 < i_1$, *then* $[A]^I_{i_1} \searrow S$.

Proof. Trivial by structural induction over the derivation of $[A]^I_{i_0} \searrow S$. □

Note that in the statement of Lemma 3.7, we implicitly assume that I is defined on the set of natural numbers less than i_1.

Corollary 3.8 (Logical relation is functional with different levels[19]). *If* $[A]^I_{i_0} \searrow S_0$ *and* $[A]^I_{i_1} \searrow S_1$, *then* $S_0 = S_1$.

Proof. Immediate from Lemma 3.4 and 3.7. □

We say that a set of terms S is closed under expansion if given $a \in S$, then $b \in S$ for all $b \Rightarrow a$. The final property we want to show is that the output set S from the logical relation is closed under expansion. Unlike the previous lemmas, we need to constrain the function I so its outputs are all closed under expansion.

Lemma 3.9 (Logical Relation Elements back preservation[20]). *If* $[A]^I_i \searrow S$ *and* I *satisfies the property that for all* i, $I(i)$ *is closed under expansion, then the set* S *is closed under expansion.*

Proof. Trivial by structural induction over the derivation of $[A]^I_i \searrow S$. The function case requires the following simple fact about parallel reduction: If $b_0 \Rightarrow b_1$ then $b_0\ a \Rightarrow b_1\ a$ for all a. This fact is not immediate from the definition of parallel reduction but follows from Lemma 2.1 and rule P-App. □

Corollary 3.10 (Logical Relation Elements back preservation (rec)[21]). *If* $[A]_i \searrow S$, *then* S *is closed under expansion.*

Proof. Immediate from Lemma 3.9, the definition of $[A]_i \searrow S$, and rule I-Red. □

4 SEMANTIC TYPING AND CONSISTENCY

$$
\begin{array}{lcl}
\rho \vDash \Gamma & := & \forall i\ j\ S,\ \text{if } i < |\Gamma| \text{ and } [(\Gamma(i)\langle\uparrow^{1+i}\rangle)[\rho]]_j \searrow S, \text{then } \rho(i) \in S \\
\Gamma \vDash a : A & := & \forall \rho,\ \text{if } \rho \vDash \Gamma \text{ then there exists some } j \text{ and } S \text{ such that } [A[\rho]]_j \searrow S \text{ and } a[\rho] \in S \\
\vDash \Gamma & := & \forall i < |\Gamma|, \exists j, \Gamma \downarrow_{1+i} \vDash \Gamma(i) : \mathbf{Set}\, j
\end{array}
$$

Fig. 7. Semantic Typing for λ^H

The logical relation we define in Figure 5 does not include cases for variables. Likewise, for the base types such as boolean and equality, the output set S contains only terms that evaluate to closed terms. To generalize our logical relation to open terms, we define the semantic typing judgment by closing the open terms with a substitution whose codomain consists of terms that respect the interpretation of the types from the context. The full definitions of well-formed substitution ($\rho \vDash \Gamma$), semantic typing ($\Gamma \vDash a : A$), and semantic context well-formedness ($\vDash \Gamma$) are presented in Figure 7. In the definition of $\vDash \Gamma$, we use the notation $\Gamma \downarrow_i$ to denote the typing context obtained by dropping the last i elements of Γ. When Γ has less than i elements, $\Gamma \downarrow_i$ returns the empty list.

The following lemma makes the statement $\Gamma \vDash A : \mathbf{Set}\ i$ easier to work with.

[18] `semtyping.v:InterpExt_cumulative` [19] `semtyping.v:InterpExt_deterministic'`
[20] `semtyping.v:InterpExt_back_clos` [21] `semtyping.v:InterpUnivN_back_clos`

Lemma 4.1 (Set Inv[22]). *The following two statements are equivalent:*

- $\Gamma \vDash A : \mathbf{Set}\ i$
- $\forall\ \rho$, *if* $\rho \vDash \Gamma$, *then there exists* S *such that* $[(A[\rho])]_i \searrow S$

Proof. The forward direction is immediate by Lemma 3.1. We now consider the backward direction and show that $\Gamma \vDash A : \mathbf{Set}\ i$ given the second bullet.

Suppose $\rho \vDash \Gamma$, then we know that there exists some S such that $[(A[\rho])]_i \searrow S$. By the definition of semantic typing, it suffices to show that there exists some j and S_0 such that $[\mathbf{Set}\ i]_j \searrow S_0$ and $A[\rho] \in S_0$. Pick $1 + i$ for j and $\{A \mid \exists S, [A]_i \searrow S\}$ for S_0 and it's trivial to verify the conditions hold. □

The semantic context well-formedness judgment ($\vDash \Gamma$), unlike its syntactic counterpart $\vdash \Gamma$, is defined through a for all quantified statement rather than inductively over the context. It is easy to recover the same structural rules:

Lemma 4.2 (Semantic context well-formedness cons). *If* $\vDash \Gamma$ *and* $\Gamma \vDash A : \mathbf{Set}\ i$, *then* $\vDash \Gamma, A$.

Proof. By the definition of semantic context well-formedness, the goal is to show that given $i < |\Gamma, A| = 1+|\Gamma|$, $(\Gamma, A) \downarrow_{1+i} \vDash (\Gamma, A)(i) : \mathbf{Set}\ j$. The statement can be easily proven by case analysis on whether i is zero. □

Likewise, the semantic well-formendness judgment for substitutions satisfies similar structural rules.

Lemma 4.3 (Well-formed ρ cons[23]). *If* $[A]_i \searrow S$, $a \in S$, *and* $\rho \vDash \Gamma$, *then* $\rho : a \vDash \Gamma, A$.

Proof. Start by unfolding the definition of $\rho : a \vDash \Gamma, A$ and performing a case analysis similar to the proof of Lemma 4.2. The case where the number is 0 requires Lemma 3.8 to finish the proof. □

Next, we show some non-trivial cases of the fundamental theorem as top-level lemmas and leave the remaining cases as exercises for the reader.

First, we formulate the definition of valid renamings and prove that semantic typing satisfies renaming so we can weaken the context when reasoning about the variable case of the fundamental lemma (Lemma 4.7). Intuitively, given a valuation $\rho \vDash \Gamma, \Delta$, it is easy to show that we can extract some valuation ρ_0 such that $\rho_0 \vDash \Gamma$, where ρ_0 is obtained by "truncating" ρ. As a result, if we know that $\Gamma \vDash a : A$, then we can conclude that $\Gamma, \Delta \vDash a_0 : A_0$, where a_0 and A_0 are obtained by shifting a and A after weakening the context; this implication holds because $\rho \vDash \Gamma, \Delta$ induces a context ρ_0 such that $\rho_0 \vDash \Gamma$ so we can make use of the premise $\Gamma \vDash a : A$ to derive what we need for the conclusion. We recommend the readers to skip the proofs of Lemmas 4.4 through 4.6 during the first read as long as they have an intuitive understanding of what the renaming property is meant to capture.

We say that ξ is valid from the context Γ to the context Δ if the following condition holds.

$$\forall i, \text{ if } i < |\Gamma|, \text{ then } \xi(i) < |\Delta| \text{ and } \Delta(\xi(i))\langle\uparrow^{1+\xi(i)}\rangle = \Gamma(i)\langle\uparrow^{1+i}\rangle\langle\xi\rangle$$

Lemma 4.4 (Truncate is valid). *If* $i < |\Gamma|$, *then* $\uparrow^i$ *is a valid renaming from* $\Gamma \downarrow_i$ *to* Γ.

Proof. By the definition of a valid renaming, we must show that given $j < |\Gamma \downarrow_i| = |\Gamma| - i$, then the following conditions hold:

- $\uparrow^i (j) = i + j < |\Gamma|$
- $\Gamma(\uparrow^i (j))\langle\uparrow^{1+\uparrow^i(j)}\rangle = \Gamma \downarrow_i (j)\langle\uparrow^{1+j}\rangle\langle\uparrow^i\rangle$

[22] `soundness.v:SemWt_Univ` [23] `soundness.v:γ_ok_cons`

The first bullet point is immediate from the fact that $j < |\Gamma| - i$. The second bullet follows from unfolding the definitions, and the fact that $\Gamma \downarrow_i (j) = \Gamma(i + j)$ when $i + j < |\Gamma|$. □

Lemma 4.5 (Renaming for $\rho \vDash \Gamma$). *If $\rho \vDash \Delta$ and ξ is a valid renaming from Γ to Δ, then we have $\rho_0 \vDash \Gamma$ where $\rho_0(i) = \rho(\xi(i))$.*

Proof. Unfolding the definition of $\rho_0 \vDash \Gamma$, the goal is to show that for all i, j, and S, if $i < |\Gamma|$ and $[(\Gamma(i)\langle\uparrow^{1+i}\rangle)[\rho_0]]_j \searrow S$, then $\rho_0(i) = \rho(\xi(i)) \in S$.

By the definition of ρ_0 and the validity of ξ, we have $\Gamma(i)\langle\uparrow^{1+i}\rangle[\rho_0] = \Gamma(i)\langle\uparrow^{1+i}\rangle\langle\xi\rangle[\rho] = \Delta(\xi(i))\langle\uparrow^{1+\xi(i)}\rangle[\rho]$. From $\rho \vDash \Delta$ and $\xi(i) < |\Delta|$, we have $[\Delta(\xi(i))\langle\uparrow^{1+\xi(i)}\rangle[\rho]]_j \searrow S$ and $\rho(\xi(i)) \in S$. Done. □

Lemma 4.6 (Renaming for $\Gamma \vDash a : A$). *If $\Gamma \vDash a : A$ and ξ is a valid renaming from Γ to Δ, then $\Delta \vDash a\langle\xi\rangle : A\langle\xi\rangle$.*

Proof. Immediate from the definition of semantic typing and Lemma 4.5. □

Lemma 4.7 (ST-Var). *If $\vDash \Gamma$ and $i < |\Gamma|$, then $\Gamma \vDash \mathbf{v}_i : \Gamma(i)\langle\uparrow^{1+i}\rangle$.*

Proof. Suppose $\rho \vDash \Gamma$. By the definition of semantic typing, we need to show that there exists some j and S such that

- $[\Gamma(i)\langle\uparrow^{1+i}\rangle[\rho]]_j \searrow S$
- $\rho(i) \in S$

From the definition of $\vDash \Gamma$, we know that there exists some j such that $\Gamma \downarrow_{1+i} \vDash \Gamma(i) : \mathbf{Set}\, j$. By Lemma 4.4, we know that $\uparrow^{1+i}$ is a valid renaming from $\Gamma \downarrow_{1+i}$ to Γ. By Lemma 4.6, we deduce $\Gamma \vDash \Gamma(i)\langle\uparrow^{1+i}\rangle : \mathbf{Set}\, j$. By the identity from Lemma 4.1, we know that $[\Gamma(i)\langle\uparrow^{1+i}\rangle[\rho]]_j \searrow S$. It now suffices to show $\rho(i) \in S$, but that is immediate from the definition of $\rho \vDash \Gamma$. □

Lemma 4.8 (ST-Set). *If $i < j$, then $\Gamma \vDash \mathbf{Set}\, i : \mathbf{Set}\, j$.*

Proof. Immedaite by Lemma 4.1 and rule I-Set. □

Lemma 4.9 (ST-Pi). *If $\Gamma \vDash A : \mathbf{Set}\, i$ and $\Gamma, A \vDash B : \mathbf{Set}\, i$, then $\Gamma \vDash \Pi\, A\, B : \mathbf{Set}\, i$.*

Proof. Applying Lemma 4.1 to the conclusion, it now suffices to show that given $\rho \vDash \Gamma$, there exists some S such that $[\Pi\, A[\rho]\, B[\Uparrow \rho]]_i \searrow S$. From Lemma 4.1 and $\Gamma \vDash A : \mathbf{Set}\, i$, we know that there exists some set S_0 such that $[A[\rho]]_i \searrow S_0$. From $\Gamma, A \vDash B : \mathbf{Set}\, i$, we know that there must exists S such that $[B[\rho : a]]_i \searrow S$ for every $a \in S_0$. The conclusion immediately follows from Lemma 3.5. □

Lemma 4.10 (ST-Abs). *If $\Gamma \vDash \Pi\, A\, B : \mathbf{Set}\, i$ and $\Gamma, A \vDash b : B$, then $\Gamma \vDash \lambda A.\, b : \Pi\, A\, B$.*

Proof. By unfolding the definition of $\Gamma \vDash \lambda A.\, b : \Pi\, A\, B$, we need to show that given some $\rho \vDash \Gamma$, there exists some i and S such that $[\Pi\, A[\rho]\, B[\Uparrow \rho]]_i \searrow S$ and $\lambda A[\rho].\, b[\Uparrow \rho] \in S$.

By Lemma 4.1 and the premise $\Gamma \vDash \Pi\, A\, B : \mathbf{Set}\, i$, there exists some set S such that $[\Pi\, A[\rho]\, B[\Uparrow \rho]]_i \searrow S$. It now suffices to show that $\lambda A[\rho].\, b[\Uparrow \rho] \in S$. By Lemma 3.6, there exists some S_0 such that all following conditions hold:

- $[A[\rho]]_i \searrow S_0$
- $\forall a$, if $a \in S_0$, then $\exists S_1, [B[\rho : a]]_i \searrow S_1$
- $S = \{b \mid \forall a, a \in S_0, \forall S_1, [B[\rho : a]]_i \searrow S_1, b\ a \in S_1\}$

To show that $\lambda A[\rho].\, b[\Uparrow \rho] \in S$, we need to prove that given $a \in S_0$, $[B[\rho : a]]_i^I \searrow S_1$, we have $(\lambda A[\rho].\, b[\Uparrow \rho])\ a \in S_1$. By Lemma 3.9, the set S_1 is closed under expansion. By Lemma **??**, since $(\lambda A[\rho].\, b[\Uparrow \rho])\ a \Rightarrow b[\Uparrow \rho][a] = b[\rho : a]$, it suffices to show that $b[\rho : a] \in S_1$, which is immediate from $\Gamma, A \vDash b : B$ and the fact that the logical relation is deterministic and cumulative (Lemma 3.8). □

Lemma 4.11 (ST-App). *If* $\Gamma \vDash b : \Pi\, A\, B$ *and* $\Gamma \vDash a : A$, *then* $\Gamma \vDash b\ a : B[a]$.

Proof. Suppose $\rho \vDash \Gamma$. The goal is to show that there exists some i and S_1 such that $b[\rho]\ a[\rho] \in S_1$ and $[B[a][\rho]]_i \searrow S_1$, or equivalently, $[B[\rho : a[\rho]]]_i \searrow S_1$ since $B[a][\rho] = B[\rho : a[\rho]]$. By the premise $\Gamma \vDash b : \Pi\, A\, B$, Lemma 4.1, and Lemma 3.6, there exists some i and S_0 such that:

- $[A[\rho]]_i \searrow S_0$
- $\forall a_0$, if $a_0 \in S_0$, then $\exists S_1, [B[\rho : a_0]]_i \searrow S_1$
- $\forall a_0, a_0 \in S_0, \forall S_1, [B[\rho : a_0]]_i \searrow S_1, b[\rho]\ a_0 \in S_1$

Instantiating the variable a_0 from the last two bullets with the term $a[\rho]$, the conclusion immediately follows. □

Theorem 4.12 (The Fundamental Theorem[24]).

- *If* $\Gamma \vdash a : A$, *then* $\Gamma \vDash a : A$.
- *If* $\vdash \Gamma$, *then* $\vDash \Gamma$.

Proof. Proof by mutual induction over the derivation of $\Gamma \vdash a : A$ and $\vdash \Gamma$. The cases related to context well-formedness immediately follows from Lemma 4.2. Lemmas 4.7, 4.8, 4.9, 4.10, 4.11 can be used to discharge their syntactic counterpart (e.g. Lemma 4.10 for case rule T-Abs). The remaining cases not covered by the lemmas are similar to the ones already shown or simpler and therefore omitted from the text.

□

Corollary 4.13 (Logical Consistency). *The judgment* $\cdot \vdash a : \mathbf{Void}$ *is not derivable.*

Proof. Immediate from Theorem 4.12 and the **Void** case of Lemma 3.1. □

5 TOWARD DECIDABILITY OF TYPE CONVERSION AND η LAWS

$$
\begin{array}{rcl}
e & ::= & i \mid e\ f \mid \mathbf{J}\ e\ f\ f\ f \mid \mathbf{if}\ e\ \mathbf{then}\ f\ \mathbf{else}\ f \\
\\
f & ::= & e \mid \mathbf{Set}\ i \mid \mathbf{Void} \mid \Pi\, f\, f \mid f \sim f \in f \\
 & \mid & \lambda f \mid \mathbf{refl} \mid \mathbf{Bool} \mid \mathbf{true} \mid \mathbf{false}
\end{array}
$$

Fig. 8. β-neutral and normal forms

In this section, we sketch out how the logical relation from Section 3 can be extended show the existence of $\beta\eta$ normal forms for (open and closed) well-typed terms. We first extend parallel reduction to include η reduction for functions.

$$\textsc{P-AbsEta}\ \frac{a \Rightarrow a_0}{\lambda(a\langle\uparrow^1\rangle\ \mathbf{v}_0) \Rightarrow a_0}$$

[24] `soundness.v:soundness`

$\boxed{[A]_i^I \searrow S}$ *(Logical Relation)*

$$\text{I-Ne}\ \frac{\mathbf{ne}\ A}{[A]_i^I \searrow \{a \mid \mathbf{wne}\ a\}} \qquad \text{I-VoidNew}\ \frac{}{[\mathbf{Void}]_i^I \searrow \{a \mid \mathbf{wne}\ a\}}$$

$$\text{I-BoolNew}\ \frac{}{[\mathbf{Bool}]_i^I \searrow \{a \mid a \Rightarrow^* \mathbf{true} \vee a \Rightarrow^* \mathbf{false} \vee \mathbf{wne}\ a\}}$$

$$\text{I-EqNew}\ \frac{}{[a \sim b \in A]_i^I \searrow \{p \mid (p \Rightarrow^* \mathbf{refl}, a \Leftrightarrow b) \vee \mathbf{wne}\ p\}}$$

Fig. 9. Extended logical relation

Rule P-AbsEta effectively adds η laws for functions to our equational theory since coherence, the untyped relation used for type conversion, is built on top of parallel reduction. We can recover the same confluence result about parallel reduction using the standard techniques from Barendregt [1993]; Takahashi [1995], though specifically for the de Bruijn representation, we need the following anti-renaming lemma about parallel reduction.

Lemma 5.1 (Par anti-renaming). *If $a\langle\xi\rangle \Rightarrow b_0$, then there exists some b such that $b\langle\xi\rangle = b_0$ and $a \Rightarrow b$.*

The syntactic forms e and f (Figure 8) capture the neutral terms and normal forms with respect to β-reduction, but can still take η reduction steps. We sometimes use the judgment forms **ne** a and **nf** a to indicate that there exists e or f such that $a = e$ or $a = f$.

We can show that parallel reduction preserves β-normal and neutral forms.

Lemma 5.2 (Par preserves β-neutral and normal forms). *If $a \Rightarrow b$, then*

- **ne** *a implies* **ne** *b*
- **nf** *a implies* **nf** *b*

This lemma is mainly to show that rule P-AbsEta does not introduce any new β redex because η reduction is the only real step we can take.

The predicates **wne** a and **wn** a describe terms that can evaluate into β-neutral or β-normal form through parallel reduction and are defined as follows.

$$\mathbf{wne}\ a \iff \exists e, a \Rightarrow^* e$$
$$\mathbf{wn}\ a \iff \exists f, a \Rightarrow^* f$$

The updated logical relation and its auxiliary definitions are shown in Figure 9.

In the definition of the logical relation, we omit the rules for the function and universe cases since they remain identical to the original version in Figure 5. The changes to rule I-Bool and rule I-Void follow the exact same pattern. An open term of type **Bool** does not necessarily reduce to **true** or **false**, but may reduce to a variable, or more generally, a neutral term. Likewise, the **Void** type, while remains uninhabited under an empty context, may be inherited by the set of neutral terms when there is a variable in the context that allows us to inhabit **Void**.

The rule for equality type $a \sim b \in A$ is augmented with the preconditions that a, b, and A are all in their normal forms since otherwise our model would include equality types that are themselves

not normalizing. Furthermore, the side condition $a \Leftrightarrow b$ is only available when the equality proof reduces to **refl**. If the proof term reduces to a neutral term, then there is nothing we can learn about the relationship between a and b.

Finally, in a non-empty context, a type itself may evaluate to a neutral term and in turn can only inhabited by neutral terms, thus the addition of rule I-Ne.

Interestingly, all the properties we have shown in Section 3 and 4 before the fundamental lemma can be proven in the exact same order, where the new cases due to rule I-Ne and the augmentation of neutral terms to rules I-Void, I-Eq, and I-Bool can be immediately discharged by Lemma 5.2.

Before we can prove the fundamental theorem and derive the normalization property as its corollary, we need to additionally formulate and prove the adequacy property.

Let us first start with some auxiliary lemmas and definitions.

Lemma 5.3 (Ext Wn). *If* $\mathbf{wn}\ a\ \mathbf{v}_i$, *then* $\mathbf{wn}\ a$.

Proof. By induction over the length of the reduction sequence in $\mathbf{wn}\ a\ \mathbf{v}_i$. The proof relies on Lemma 5.1 and 5.2. □

Lemma 5.4 (wne wn). *If* $\mathbf{wne}\ a$ *and* $\mathbf{wn}\ b$, *then* $\mathbf{wne}\ a\ b$.

Proof. Immediate by lexicographical induction over the length of the reduction sequences in $\mathbf{wne}\ a$ and $\mathbf{wn}\ b$. □

Definition 5.5 (CR). Let S be a set of lambda terms. We say that

- $S \in CR_1 \iff \forall a$, if $\mathbf{wne}\ a$, then $a \in S$
- $S \in CR_2 \iff \forall a$, if $a \in S$, then $\mathbf{wn}\ a$
- $S \in CR \iff S \in CR_1$ and $S \in CR_2$

Lemma 5.6 (CR_1 for type). *If* $\mathbf{wne}\ A$, *then we have* $[A]^I_i \searrow \{a \mid \mathbf{wne}\ a\}$.

Proof. Immediately from rule I-Ne and rule I-Par. □

Lemma 5.7 (CR for interpreted sets). *If* $I(j) \in CR$ *for all* $j < i$ *and* $[A]^I_i \searrow S$, *then* $S \in CR$.

Proof. By induction over the derivation of $[A]^I_i \searrow S$. The base cases are all immediate.

In the function case ($\Pi\ A\ B$), we use the induction hypothesis to show that variables, which are special cases of neutral terms, semantically inhabit the input type A. We apply the function to an arbitrary variable and we can then use Lemma 5.3 to conclude the CR_2 property.

To show the CR_1 property, we need to prove that given a neutral term b such that $\mathbf{wne}\ b$, and a term a that semantically inhabits A, $b\ a$ gives us a term that semantically inhabits $B[a]$. From the induction hypothesis, A also satisfies CR_2 and therefore $\mathbf{wn}\ a$ holds. By Lemma 5.4, we have $\mathbf{wne}\ b\ a$. By CR_2 from the induction hypothesis, since every term that evaluates to some neutral form semantically inhabits $B[a]$. Done. □

Lemma 5.8 (CR_2 for type). *If* $I(j) \in CR$ *for all* $j < i$ *and* $[A]^I_i \searrow S$, *then* $\mathbf{wn}\ A$ *holds.*

Proof. By induction over the derivation of $[A]^I_i \searrow S$. The function case uses Lemma 5.7 and Lemma 5.3. □

Lemma 5.9 (CR_2 for types (Rec)). *If* $[A]_i \searrow S$, *then* $\mathbf{wn}\ A$.

Proof. By strong induction over i and Lemma 5.8. □

Lemma 5.10 (CR for terms (Rec)). *If* $[A]_i \searrow S$, *then* $S \in CR$.

Proof. After unfolding the definition of $[A]_i \searrow S$, trivial by Lemmas 5.6, 5.7 and 5.9. □

The formulation of the valuation and semantic well-typedness from Figure 7 and the fundamental lemma remains unchanged. The proof of the fundamental lemma is still carried out by induction over the typing derivation, where the additional neutral term related cases are handled by Lemmas 5.9 and 5.10.

The normalization property then follows as a corollary of the fundamental theorem.

Corollary 5.11 (Existence of β-normal form). *If* $\Gamma \vdash a : A$, *then* $\mathbf{wn}\ a$ *and* $\mathbf{wn}\ A$.

Proof. By the fundamental lemma, we know that $\Gamma \vDash a : A$. That is, for all $\rho \vDash \Gamma$, there exists some i and S such that $[A[\rho]]_i \searrow S$ and $a[\rho] \in S$. We pick the valuation $\rho = \mathbf{id}_{tm}$ (defined in Figure 2), which injects natural numbers as term variables. The side condition $\mathbf{id}_{tm} \vDash \Gamma$ is satisfied since Lemma 5.10 says neutral terms, including variables, semantically inhabit any S_0 where S_0 is the interpretation of some type. With our choice of ρ, we have $A[\rho] = A[\mathbf{id}_{tm}] = A$ and $a[\rho] = a[\mathbf{id}_{tm}] = a$. Then we know that $[A]_i \searrow S$ and $a \in S$ for some i and S. By Lemmas 5.10 and 5.9, we conclude that $\mathbf{wn}\ a$ and $\mathbf{wn}\ A$ respectively. □

From Corollary 5.11, we can show that there exists $\beta\eta$-normal forms for well-typed terms, since η reduction preserves β-normal form (implied by Lemma 5.2) and also strictly decreases the size of the term.

Due to the non-deterministic nature of parallel reduction, we need to take a few more steps to convert the existence of $\beta\eta$-normal form into a decision procedure for type conversion. More specifically, we can show that a deterministic evaluation strategy such as leftmost-outermost reduction can always find the $\beta\eta$-normal form if there exists one. However, we omit such proofs since they can can be formulated on untyped lambda terms and thus orthogonal to the specifics of dependently typed systems. Instead, we redirect readers to Accattoli et al. [2019]; Takahashi [1995] for the details.

Finally, we want to point out that Lemmas 5.6, 5.9, and 5.10, often referred to as adequacy of the logical relation, are in fact required in the normalization proof for simply typed languages [Abel et al. 2019]. Similar to the simply typed scenario, adequacy needs to be proven before the fundamental theorem so we can handle rules such as rule T-If where the scrutinee is a neutral term. In Abel et al. [2019], a variant of Lemma 5.3 is used in the exact same way to show that lambda terms are themselves normalizing as we have done in Lemma 5.7.

Dependent types make the proof slightly more complicated as we also need to know that every type has a normal form. Whereas the CR_1 property for types (Lemma 5.6) is directly derivable, the CR_2 property for types (Lemma 5.8) requires the CR_1 property about the interpreted sets (Lemma 5.7).

Overall, despite the dependently typed setting, the extension of our logical relation to prove normalization of open *and* closed terms closely mirrors the progression from normalization of closed terms [Harper 2022a] to normalization of open terms [Harper 2022b] in the simply typed lambda calculus. It is in fact reassuring that once we have laid the foundational technique for handling dependent types in our logical relation, the further extensions more or less boil down to properties that can be derived through syntactic means, which tend to not get in the way of Gödel's second incompleteness theorem.

6 MECHANIZATION

Figure 10 shows the statics of our development, including the base consistency proof from Section 3 and 4 and the extension to normalization for open and closed terms from Section 5.

	Consistency	Normalization
Library (Autosubst 2)	491	-
Syntactic typing (specification)	69	-
Renaming (common)	46	-
Syntactic soundness	660	-
Untyped reduction	350	834
Logical relation	342	515
Semantic typing and soundness	169	197

Fig. 10. Statistics of the Coq Development

Since the normalization and the consistency development share the same syntactic and typing specification, they differ in only the categories that are related to the logical relation. We use - as a marker that the line count is the same.

Autosubst 2 takes our syntax specification in higher-order abstract syntax and generates Coq syntax specification, renaming and substitution functions, and lemmas and tactics that allow reasoning about those functions. The auto-generated files and the library header files are counted toward the *Library (Autosubst 2)* category.

Orthogonal to the development of our consistency and normalization proof, we prove the syntactic soundness of our system through subject reduction and progress. The syntactic soundness proof shares the definition of renaming with our semantic soundness proof, which is factored out as a separate file under the Renaming (common) category.

6.1 Artifacts Specific to Coq

In this section, we discuss the Coq encoding of the definitions and proofs presented in Section 4 and the artifacts that are specific to Coq and may not appear in other proof assistants.

The powerset $\mathcal{P}(\mathit{Term})$ is encoded as the type `tm -> Prop`, a predicate over λ^H terms. The inductive definition of the logical relation in Figure 5 requires the impredicativity of Coq's `Prop` sort since in rule I-Pɪ, the function F can be later instantiated into the logical relation itself (e.g. in the proof of Lemma 3.5).

In Coq, there is a distinction between computable functions and relations that can later be proven to be functional. The former can be viewed as a strict subset of the latter in axiom-free Coq. To be more precise, given a relation `R : A -> B -> Prop` subject to the totality and functionality constraints (∀ `a` ∈ `A`, there exists a unique `b` ∈ `B` such that `R a b` is inhabited), we do not immediately obtain a function `F : A -> B` such that ∀ `a` ∈ `A`, `R a (F a)` is inhabited. However, the functional side conditions of a relation is clunky to express and tend to block automation. A simple workaround is to assume the axiom of unique choice, which is known to be consistent with Coq and allows us to induce a function `F : A -> B` once we have shown the relation `R : A -> B -> Prop` is functional. This approach would make our Coq development match the text version of our proof from Section 4 more closely.

However, we choose instead an axiom-free workaround and define rule I-Pɪ as follows in our Coq mechanization.

I-PiCoq

$$\frac{[A]_i^I \searrow S \quad R \in S \times \mathcal{P}(Term) \quad \forall a, \exists S_0, (a, S_0) \in R \quad \forall a, \forall S_0, \text{if } (a, S_0) \in R, \text{then } [B[a]]_i^I \searrow S_0}{[\Pi\, A\, B]_i^I \searrow \{b \mid \forall a, \forall S_0, \text{if } (a, S_0) \in R, \text{then } b\ a \in S_0\}}$$

It should be easy to verify that the preconditions of rule I-Pi, rule I-PiAlt, and rule I-PiCoq are all equivalent. After establishing Lemma 3.4, it is possible to further show that the conclusions of the rules are equivalent, too.

We choose to keep this discrepancy between the Coq development and the logical relation presented in Section 3 since the skolemization process is more intuitively expressed in terms of function symbols rather than relation symbols. Otherwise, we do not see a clear advantage of rule I-PiCoq over rule I-Pi in set theory, where there is no distinction between computable functions and functions in general.

6.2 Automation

Our Coq mechanization heavily uses automation, though instead of defining custom tactics, we rely mostly on off-the-shelf tools such as Autosubst 2 [Stark et al. 2019] and CoqHammer [Czajka and Kaliszyk 2018].

We use the Autosubst 2 framework to specify our syntax in HOAS and produce Coq syntax files in de Bruijn representation. Additionallly, Autosubst 2 provides a powerful tactic `asimpl` which can be used to prove the equivalence of two terms constructed using the primitive operators provided by the framework. This greatly simplifies the reasoning about substitution in our development as almost all substitution related properties are immediately discharged by `asimpl` without having to manually prove or even directly invoke any substitution related lemmas.

For other automation tasks that are not specific to binding, we use the powerful `sauto` tactic provided by CoqHammer to write short and declarative proofs. For example, here is a one-line proof of the triangle property about parallel reduction, from which the diamond property (Lemma 2.4) follows as a corollary.

```
Lemma par_triangle a : forall b, Par a b ->  Par b (tstar a).
Proof.
  apply tstar_ind; hauto lq:on inv:Par use:Par_refl,par_cong ctrs:Par.
Qed.
```

In prose, the triangle property can be proven by induction over the graph of the `tstar` function, which stands for the Takahashi translation [Takahashi 1995]. Options `inv:Par` and `ctrs:Par` say that the proof involves inverting and constructing of the derivations of parallel reduction. The option `use:Par_refl,par_cong` allows the automation tactic to use the reflexivity and congruence properties of parallel reduction as lemmas.

The automation here not only gives us a proof that is shorter and more resiliant to changes in definition, but also gives useful documentation for readers who wish to understand how the underlying proof works since the automation tactic is guaranteed to not use lemmas or invert derivations that are not specified in the `use` or `inv` flag.

7 RELATED WORK AND DISCUSSION

Martin-Löf [1975], Geuvers [1994], and Barendregt [1993] are some of the earlier works that establish metatheoretic results strong enough to derive consistency for dependently typed systems. Some of these techniques are still in use in more recent works. For example, the technique for

proving strong normalization by Geuvers [1994] is adapted by Moon et al. [2021] to show the same property to a dependently typed system extended with modalities.

Abel et al. [2017] mechanizes in Agda the decidability of type conversion rule for a dependently typed language with one predicative universe level and typed judgmental equality with function η-law. Unlike our logical relation for λ^H, Abel et al. [2017] uses a Kripke-style logical relation parameterized over an type-directed equivalence relation satisfying certain properties. Their logical relation is defined using the induction-recursion scheme, which is available in Agda but not in Coq. Adjedj et al. [2024] manages to encode the logical relation from Abel et al. [2017] in the predicative fragment of Coq using a special way of encoding induction recursion. Their work further extends the decidability of type conversion result from Abel et al. [2017] to the decidability type checking of a bidirectional type system.

Anand and Rahli [2014] mechanizes the metatheory of Nuprl [Constable et al. 1986] in Coq. The metatheory is an extensional type theory with features such as dependent functions, inductive types, and a full universe hierarchy. Wieczorek and Biernacki [2018] mechanizes the normalization-by-evaluation algorithm in Coq for a dependently typed language with one predicative universe, similar to Abel et al. [2017] and Adjedj et al. [2024]. Both Anand and Rahli [2014] and Wieczorek and Biernacki [2018] leverage the impredicative `Prop` sort of Coq to define the interpretation of dependent function types and thus are closely related to our mechanization. However, instead of explaining the purely inductive logical relation as a convoluted workaround to the lack of induction-recursion in Coq, we give a self-contained explanation of our logical relation.

Finally, it is worth pointing out that all systems we have discussed so far builds a relational model for their logical relation. Recall that our logical relation takes the form $[A]_i \searrow S$ where S is a set of terms. In a relational model, each type is interpreted as a partial equivalence relation over terms rather than a set. A relational model is necessary for an extensional type theory such as Nuprl, though Abel et al. [2017]; Adjedj et al. [2024]; Wieczorek and Biernacki [2018] all use a relational model either to justify the function η-law or to derive Π-injectivity. In Section 5, our η law for function is baked into the untyped reduction relation and therefore a relational model is not needed.

The dependently typed systems we have discussed so far also vary greatly in expressiveness. For example, the Calculus of Constructions [Coquand and Huet 1986] and MLTT without universe levels are unable to encode large elimination. Abel et al. [2017] and Wieczorek and Biernacki [2018] lack identity types. Wieczorek and Biernacki [2018] lacks η-law for functions. Among the mechanized systems we have discussed so far, only Anand and Rahli [2014] has a full universe hierarchy. Our object language λ^H, while small, has a decent coverage of features commonly seen in dependently typed languages, including dependent pattern matching, dependent function types, a full universe hierarchy, an intensional identity type. We hope features such as type-directed reduction (necessary for unit η-law) and impredicative sorts can be implemented in a similarly streamlined fashion and we will leave those extensions as part of our future work.

The final question we want to address is: why is our proof, even with the extension from Section 5, so much shorter than the proofs from Abel et al. [2017]; Adjedj et al. [2024]; Wieczorek and Biernacki [2018]? First, unlike developments that mechanize the correctness of a clever type conversion algorithm, we only show the existence of normal form for open and closed terms and state our properties in terms of the untyped small-step reduction relation. This removes a lot of scaffolding in the specification of our type system and our logical relation.

This contrasts our development with Adjedj et al. [2024], where the irrelevance property turns out to be the most difficult property to prove due to the complexity of the equality judgment used to define the logical relation. In our system, the irrelevance property (Corollary 3.3) is a direct consequence of Lemma 3.2, which is straightforward after we prove the classical results about

parallel reduction in Section 2. In other words, we keep the the logical relation itself minimal at the cost of extra lemmas that can be proven through syntactic means independently from the logical relation. For example, as commented near the end of Section 5, our normalization property (Corollary 5.11) does not immediately induce an efficient decision procedure, but it is possible to recover an efficient algorithm by reasoning about untyped lambda terms.

Another key to our proof is the very early establishment of the confluence property of the paralell reduction relation (Lemma 2.4). This property is used in Lemma 3.2 to show that the interpretation of a type is preserved under evaluation. That is, if $[A]_i \searrow S$ and $A \Rightarrow B$, then we also have $[B]_i \searrow S$. In a system with typed directed reduction, the confluence result is harder to establish since confluence requires subject reduction, which circularly depends on Π-injectivity, a consequence of confluence [Siles and Herbelin 2012]. There are different workarounds to this problem. Siles and Herbelin [2012] proposes a syntactic approach by defining a type system where the confluence result is directly provable and later show that the system of interest is equivalent to the system with the confluence property. Abel et al. [2017] uses a relational model and the relational counterpart of Lemma 5.3 to derive Π-injectivity. The relational model is parameterized by a typed indexed equivalence relation and is reused to prove derive different properties in their development. Another perhaps simpler approach is to simply extend the judgmental equality with rules about Π-injectivity [Weirich et al. 2017]. This allows subject reduction to be proven independently from confluence and Π-injectivity can later be shown to be admissible. We find the approach from [Weirich et al. 2017] the most lightweight as it allows us to derive confluence early on even in systems with typed conversion.

In cases where the judgmental equality does not leverage type information heavily (e.g. η-law for the unit type), it might be possible to show that $\Gamma \vdash a \equiv b : A$ is equivalent to the conjunction of $\Gamma \vdash a : A$, $\Gamma \vdash b : A$, and $a \Leftrightarrow b$. If such a property holds (we believe this is likely true for both Abel et al. [2017] and Adjedj et al. [2024]), we can reuse our existing logical relation in terms of untyped parallel reduction to show the decidability of type conversion. Otherwise, to fully model the system, we need to bake typing information into our logical relation and we need to redefine our logical relation in Kripke-style so we can relax the scoping constraint in order to prove adequacy, though a relational model would still be unnecessary as long as the η laws are baked into the typed directed relation.

Since confluence plays such as key role in our proof, we are uncertain how to generalize our technique to systems where confluence does not hold, and we will investigate those scenarios in our future work.

8 CONCLUSION

REFERENCES

Andreas Abel, Guillaume Allais, Aliya Hameer, Brigitte Pientka, Alberto Momigliano, Steven Schäfer, and Kathrin Stark. 2019. POPLMark reloaded: Mechanizing proofs by logical relations. *Journal of Functional Programming* 29 (2019), e19.

Andreas Abel, Joakim Öhman, and Andrea Vezzosi. 2017. Decidability of Conversion for Type Theory in Type Theory. *Proc. ACM Program. Lang.* 2, POPL, Article 23 (dec 2017), 29 pages. https://doi.org/10.1145/3158111

Beniamino Accattoli, Claudia Faggian, and Giulio Guerrieri. 2019. Factorization and Normalization, Essentially. In *Programming Languages and Systems*, Anthony Widjaja Lin (Ed.). Springer International Publishing, Cham, 159–180.

Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. 2024. Martin-Löf à la Coq. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs* (London, UK) *(CPP 2024)*. Association for Computing Machinery, New York, NY, USA, 230–245. https://doi.org/10.1145/3636501.3636951

Abhishek Anand and Vincent Rahli. 2014. Towards a formally verified proof assistant. In *Interactive Theorem Proving: 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings 5*. Springer, 27–44.

Henk Barendregt. 1991. Introduction to generalized type systems. *Journal of Functional Programming* 1, 2 (1991), 462–490. https://doi.org/10.1017/S0956796800020025

Henk P. Barendregt. 1993. *Lambda Calculi with Types.* Oxford University Press, Inc., USA, 117–309.

R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. 1986. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Inc., USA.

Thierry Coquand and Gérard Huet. 1986. *The calculus of constructions.* Technical Report RR-0530. INRIA. https://hal.inria.fr/inria-00076024

Łukasz Czajka and Cezary Kaliszyk. 2018. Hammer for Coq: Automation for dependent type theory. *Journal of automated reasoning* 61 (2018), 423–453.

Herman Geuvers. 1994. A short and flexible proof of strong normalization for the calculus of constructions. In *International Workshop on Types for Proofs and Programs*. Springer, 14–38.

Jean-Yves Girard. 1972. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur.* Ph. D. Dissertation. Université Paris 7.

Robert Harper. 2016. *Practical foundations for programming languages.* Cambridge University Press.

Robert Harper. 2022a. How to (Re)Invent Tait's Method. (2022).

Robert Harper. 2022b. Kripke-Style Logical Relations for Normalization. (2022).

Per Martin-Löf. 1975. An intuitionistic theory of types: predicative part. In *Logic Colloquium '73, Proceedings of the Logic Colloquium*, H.E. Rose and J.C. Shepherdson (Eds.). Studies in Logic and the Foundations of Mathematics, Vol. 80. North-Holland, 73–118. https://doi.org/10.1016/S0049-237X(08)71945-1

Benjamin Moon, Harley Eades III, and Dominic Orchard. 2021. Graded Modal Dependent Type Theory. In *Programming Languages and Systems*, Nobuko Yoshida (Ed.). Springer International Publishing, Cham, 462–490. https://doi.org/10.1007/978-3-030-72019-3_17

Vincent Siles and Hugo Herbelin. 2012. Pure type system conversion is always typable. *Journal of Functional Programming* 22, 2 (2012), 153–180.

Lau Skorstengaard. 2019. An Introduction to Logical Relations. arXiv:1907.11133 [cs.PL]

Kathrin Stark, Steven Schäfer, and Jonas Kaiser. 2019. Autosubst 2: reasoning with multi-sorted de Bruijn terms and vector substitutions. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Cascais, Portugal) *(CPP 2019)*. Association for Computing Machinery, New York, NY, USA, 166–180. https://doi.org/10.1145/3293880.3294101

M. Takahashi. 1995. Parallel Reductions in λ-Calculus. *Information and Computation* 118, 1 (1995), 120–127. https://doi.org/10.1006/inco.1995.1057

Philip Wadler, Wen Kokke, and Jeremy G. Siek. 2022. *Programming Language Foundations in Agda.* https://plfa.inf.ed.ac.uk/22.08/

Stephanie Weirich, Antoine Voizard, Pedro Henrique Avezedo de Amorim, and Richard A. Eisenberg. 2017. A Specification for Dependent Types in Haskell. *Proc. ACM Program. Lang.* 1, ICFP, Article 31 (Aug. 2017), 29 pages. https://doi.org/10.1145/3110275

Paweł Wieczorek and Dariusz Biernacki. 2018. 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* (Los Angeles, CA, USA) *(CPP 2018)*. Association for Computing Machinery, New York, NY, USA, 266–279. https://doi.org/10.1145/3167091