From Dependent Type Theory to Dependently Typed Higher Order Logic (Technical Report)

LUCA MAIO, Ludwig-Maximilians-Universität München, Germany
ALEXANDER BENTKAMP, Cryspen, France
JASMIN BLANCHETTE, Ludwig-Maximilians-Universität München, Germany
SOPHIE TOURRET, Inria, LORIA, CNRS, Université de Lorraine, France and Max-Planck-Institut für Informatik, Saarland Informatics Campus, Germany

There is an expressivity gap between proof assistants based on dependent type theory (DTT), such as Lean and Rocq, and automated theorem provers based on higher order logic (HOL). This gap has been reduced with the introduction of dependently typed HOL (DHOL) along with automated theorem proving for DHOL. To reduce the gap even further, we introduce a translation from a fragment of DTT to polymorphic DHOL. The translation is sound and lightweight, preserving constructs such as polymorphism and dependent types. An empirical evaluation finds that the translatable fragment is large in practice, suggesting that the translation could be used in a hammer system.

CCS Concepts: • **Theory of computation → Type theory**.

ACM Reference Format:
Luca Maio, Alexander Bentkamp, Jasmin Blanchette, and Sophie Tourret. 2018. From Dependent Type Theory to Dependently Typed Higher Order Logic (Technical Report). 1, 1 (July 2018), 28 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Proof assistants such as Lean [13], Matita [3] and Rocq [5] are based on rich dependent type theories (DTTs) [21], whereas most state-of-the-art automated theorem provers are based on first order logic (FOL) or higher order logic (HOL) [16]. This makes it difficult to develop hammer systems for these proof assistants. For example, the CoqHammer [12] and the LeanHammer [27, 38] use an unsound translation, which is not entirely satisfactory.

The expressivity gap between dependently typed proof assistants and automated provers has shrunk with the emergence of dependently typed HOL (DHOL) [31, 33], along with DHOL specific automated theorem proving techniques. In particular, the Lash [22] prover is an extension of the HOL based Satallax [7] tableau prover to DHOL. Moreover, there exists a sound and complete translation from DHOL to HOL [31, 33], which can be used in a hammer.

Nevertheless, there is still a substantial gap between DTT (Section 2) and DHOL (Section 3). Both are based on λ-calculi and support types that depend on terms. Examples of dependent types include the integers modulo n and the vectors of length n. A distinguishing feature of traditional

Authors' Contact Information: Luca Maio, luca.maio@ifi.lmu.de, Ludwig-Maximilians-Universität München, Munich, Germany; Alexander Bentkamp, alex@cryspen.com, Cryspen, Paris, France; Jasmin Blanchette, jasmin.blanchette@ifi.lmu.de, Ludwig-Maximilians-Universität München, Munich, Germany; Sophie Tourret, sophie.tourret@inria.fr, sophie.tourret@mpi-inf.mpg.de, Inria, LORIA, CNRS, Université de Lorraine, Nancy, France and Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany.

ACM XXXX-XXXX/2018/7-ART
https://doi.org/XXXXXXX.XXXXXXX

DTT is its reliance on the Curry-Howard correspondence: propositions are represented by types, and proofs are represented by terms. Types themselves have types, which are called universes. Even universes can have types, forming a possibly infinite universe hierarchy. DTT is fundamentally intuitionistic but can be extended with classical axioms. The DTTs implemented by Lean, Matita and Rocq are intensional: they use a weak, computational notion of equality for checking types, ensuring that type checking is decidable, or, in Lean's case, almost decidable in practice.

By contrast, DHOL is an extension of classical HOL, which itself is a polymorphic generalisation of Church's simple theory of types [9]. DHOL has a distinguished type of booleans, and propositions are terms of that type. Types cannot have types, and proofs are not represented within the logic. DHOL is extensional: it uses standard equality for type checking, at the expense of decidability. Due to the lack of higher universes, DHOL is theoretically weaker than DTT with classical axioms, but its closeness to HOL leads us to believe that it is more amenable to automated reasoning.

To bridge the expressivity gap between the two logics and give proof assistant users access to automatic provers, we introduce a translation from a fragment of DTT to DHOL (Section 4). The fragment is restricted to three universes and supports only rank 1 (i.e. top level) polymorphism. The translation is designed to preserve the structure of DTT expressions. In particular, it maps polymorphism to polymorphism and dependent types to dependent types. Based on experience with other translations between logics [6, 12, 20, 27], we hypothesise that such a lightweight translation will lead to more efficient automated proving.

The translation is sound, in the sense that if a translated DTT3 proposition is provable in DHOL, the original proposition is provable in DTT3 (Section 5). We believe that the translation is also complete, but proving this is left for future work.

Although the translation does not natively support advanced features such as inductive definitions [25] and quotient types [10], it still constitutes a large part of DTT as used in practice. Many DTT formalisations require no more than three universes, higher rank polymorphism is uncommon, and inductive definitions can be axiomatised as a pre-processing step [27]. Indeed, in an empirical evaluation, we find that nearly half of the theorem statements in Lean's Mathlib library [19], where we instantiate all universe variables with 0, fall within the translatable fragment (Section 6). For this evaluation, we rely on a tool, the 'fragment checker', which we developed in Lean.

Our main contributions are the following:

- We designed a translation from an expressive fragment of DTT to DHOL.
- We proved this translation sound.
- We implemented a checker for the fragment.
- We evaluated the fragment's practical relevance using the checker on Lean's Mathlib library.

2 Dependent Type Theory

We describe a fragment of full DTT as presented by Carneiro [8], which is a version of the calculus of inductive constructions. We call it DTT3, or 'DTT with three universes'. We will later identify a fragment of DTT3 as the domain of our translation.

As a basis, we use a pure type system [4] with

$$\begin{aligned} \text{sorts } \mathcal{S} &= \{\mathsf{U}_0, \mathsf{U}_1, \mathsf{U}_2\} \\ \text{axioms } \mathcal{A} &= \{(\mathsf{U}_0 : \mathsf{U}_1), (\mathsf{U}_1 : \mathsf{U}_2)\} \\ \text{rules } \mathcal{R} &= \{(\mathsf{U}_i, \mathsf{U}_j, \mathsf{U}_{\mathrm{imax}(i,j)}) \mid 0 \le i, j \le 2\} \end{aligned}$$

Table 1. Abbreviated notations

Abbreviation	Expansion
$\overrightarrow{e}$	$e_1, \ldots, e_n$
$\overrightarrow{e} : \overrightarrow{\alpha}$	$e_1 : \alpha_1, \ldots, e_n : \alpha_n$
$\overrightarrow{e} : \alpha$	$e_1 : \alpha, \ldots, e_n : \alpha$
$\forall_{\overrightarrow{x}:\overrightarrow{\alpha}}$	$\forall_{x_1:\alpha_1} \cdots \forall_{x_n:\alpha_n}$
$\lambda_{\overrightarrow{x}:\overrightarrow{\alpha}}$	$\lambda_{x_1:\alpha_1} \cdots \lambda_{x_n:\alpha_n}$

where imax is defined as follows:

$$\mathrm{imax}(i, j) = \begin{cases} \max(i, j) & \text{if } j > 0 \\ 0 & \text{if } j = 0 \end{cases}$$

We then add signatures, which are a device to enable global definitions for constants, and an axiomatisation of equality.

Table 1 summarizes abbreviations that we use throughout. We use x, y, z to refer to variables, a, b for expressions without variables and c to refer to constants. Sometimes we write $\overrightarrow{x}^n$ instead of $\overrightarrow{x}$ to indicate the number of components. The abbreviations regarding $\forall$ or λ can include the cases where $n = 0$; a notation such as $\forall_{\overrightarrow{x}:\overrightarrow{\alpha}}\beta$ then denotes β. Moreover, we write $f\ \overrightarrow{e}$ to denote an iterated curried application.

The syntax of expressions and contexts is given by the following grammar:

$$\begin{aligned} \ell &::= 0 \mid 1 \mid 2 \\ e &::= x \mid c \mid U_\ell \mid \forall_{x:e} e \mid e\ e \mid \lambda_{x:e} e \\ \Gamma &::= \circ \mid \Gamma, x : e \end{aligned}$$

The syntax of signatures $\mathcal{T}$ is given by the following grammar:

$$\mathcal{T} ::= . \mid \mathcal{T}, (c, e) \mid \mathcal{T}, (c, e, e)$$

Signatures formalise global definitions of constants c. Constants can be declared either only with a type α, which we write as (c, α), or additionally with a value e, which we write as (c, α, e).

The typing rules of DTT3 follow:

$$\boxed{\Gamma \vdash_{\mathcal{T}} e : \alpha}$$

$$\frac{\Gamma \vdash_{\mathcal{T}} \alpha : U_\ell \quad \Gamma \vdash_{\mathcal{T}} e : \beta}{\Gamma, x : \alpha \vdash_{\mathcal{T}} e : \beta}\mathsf{Weak} \qquad \frac{\Gamma \vdash_{\mathcal{T}} \alpha : U_\ell}{\Gamma, x : \alpha \vdash_{\mathcal{T}} x : \alpha}\mathsf{Assume}$$

$$\frac{}{\circ \vdash_{\mathcal{T}} U_0 : U_1}\mathsf{Univ0} \qquad \frac{}{\circ \vdash_{\mathcal{T}} U_1 : U_2}\mathsf{Univ1}$$

$$\frac{\Gamma \vdash_{\mathcal{T}} e_1 : \forall_{x:\alpha}\beta \quad \Gamma \vdash_{\mathcal{T}} e_2 : \alpha}{\Gamma \vdash_{\mathcal{T}} e_1\ e_2 : \beta[e_2/x]}\mathsf{Apply} \qquad \frac{\Gamma, x : \alpha \vdash_{\mathcal{T}} e : \beta}{\Gamma \vdash_{\mathcal{T}} \lambda_{x:\alpha} e : \forall_{x:\alpha}\beta}\lambda\mathsf{Form}$$

$$\frac{\Gamma \vdash_{\mathcal{T}} \alpha : U_{\ell_1} \quad \Gamma, x : \alpha \vdash_{\mathcal{T}} \beta : U_{\ell_2}}{\Gamma \vdash_{\mathcal{T}} \forall_{x:\alpha}\beta : U_{\mathrm{imax}(\ell_1,\ell_2)}}\forall\mathsf{Form} \qquad \frac{\Gamma \vdash_{\mathcal{T}} e : \alpha \quad \Gamma \vdash_{\mathcal{T}} \alpha \equiv \beta}{\Gamma \vdash_{\mathcal{T}} e : \beta}\mathsf{Conv}$$

$$\frac{}{\circ \vdash_{\mathcal{T}} c : \mathcal{T}^{\mathsf{ty}}(c)}\mathsf{ConstTy}$$

where $\mathcal{T}^{\mathsf{ty}}(c)$ looks for an entry in $\mathcal{T}$ with c as the first component and returns the second component of the entry if it exists. Later we will also use $\mathcal{T}^{\mathsf{val}}(c)$ to refer to the same lookup but returning the third component if it exists.

We sometimes write $\mathbb{P}$ for U_0. We also sometimes write $\alpha \to \beta$ for function types instead of $\forall_{x:\alpha}\beta$ when β does not depend on the argument x.

We use $\equiv$ to denote definitional equality [8], including β-, δ- and η-conversion. We also assume proof irrelevance (PI) regarding $\mathbb{P}$, meaning that the following holds:

$$\frac{\Gamma \vdash_{\mathcal{T}} p : \mathbb{P} \quad \Gamma \vdash_{\mathcal{T}} h : p \quad \Gamma \vdash_{\mathcal{T}} h' : p}{\Gamma \vdash_{\mathcal{T}} h \equiv h'}\mathsf{PI}$$

By $e_1[e_2/x]$ we denote the capture avoiding substitution of e_2 for x in e_1. If it is clear which variable is replaced, we simply write $e_1[e_2]$.

We add the following rules for the axiomatisation of propositional, or provable, equality following Angiuli and Gratzer [2]:

$$\frac{\Gamma \vdash_{\mathcal{T}} \alpha \text{ type} \qquad \Gamma \vdash_{\mathcal{T}} a : \alpha \qquad \Gamma \vdash_{\mathcal{T}} b : \alpha}{\Gamma \vdash_{\mathcal{T}} a =_\alpha b : \mathbb{P}}\mathsf{EqForm} \qquad \frac{\Gamma \vdash_{\mathcal{T}} \alpha \text{ type} \qquad \Gamma \vdash_{\mathcal{T}} a : \alpha}{\Gamma \vdash_{\mathcal{T}} \mathsf{refl}_a : a =_\alpha a}\mathsf{EqIntro}$$

$$\frac{\Gamma \vdash_{\mathcal{T}} p : a =_\alpha b \qquad \begin{array}{c}\Gamma \vdash_{\mathcal{T}} \alpha \text{ type} \qquad \Gamma \vdash_{\mathcal{T}} a : \alpha \qquad \Gamma \vdash_{\mathcal{T}} b : \alpha \\ \Gamma, x : \alpha, y : \alpha, \pi : x =_\alpha y \vdash_{\mathcal{T}} Q : U_\ell \qquad \Gamma, x : \alpha \vdash_{\mathcal{T}} R : Q[x, x, \mathsf{refl}_x]\end{array}}{\Gamma \vdash_{\mathcal{T}} \mathcal{J}(R, p) : Q[a, b, p]}\mathsf{EqInd}$$

where $\mathcal{J}(R, \mathsf{refl}_a) \equiv R[a]$ for any $a : \alpha$. Using EqInd, we can then define

$$\mathsf{cast} : \forall_{\alpha,\beta:\mathsf{U}_\ell}\forall_{\pi:\alpha=_{\mathsf{U}_\ell}\beta}\alpha \to \beta.$$

We omit the type arguments of cast whenever they can be inferred from the context.

We define the following judgements for types, contexts and signatures:

$$\boxed{\Gamma \vdash \alpha \text{ type}} \qquad \frac{\Gamma \vdash_{\mathcal{T}} \alpha : U_\ell}{\Gamma \vdash_{\mathcal{T}} \alpha \text{ type}}$$

$$\boxed{\vdash_{\mathcal{T}} \Gamma \text{ ok}} \qquad \frac{\vdash \mathcal{T} \text{ sig}}{\vdash_{\mathcal{T}} \circ \text{ ok}} \qquad \frac{\vdash_{\mathcal{T}} \Gamma \text{ ok} \qquad \Gamma \vdash_{\mathcal{T}} \alpha \text{ type}}{\vdash_{\mathcal{T}} \Gamma, x : \alpha \text{ ok}}$$

$$\boxed{\vdash \mathcal{T} \text{ sig}} \qquad \frac{}{\vdash . \text{ sig}} \qquad \frac{\vdash \mathcal{T} \text{ sig} \quad \vdash_{\mathcal{T}} \alpha \text{ type}}{\vdash \mathcal{T}, (c, \alpha) \text{ sig}} \qquad \frac{\vdash \mathcal{T} \text{ sig} \quad \vdash_{\mathcal{T}} \alpha \text{ type} \quad \vdash_{\mathcal{T}} e : \alpha}{\vdash \mathcal{T}, (c, \alpha, e) \text{ sig}}$$

where c must be a fresh name and for technical reasons may not have a name of the form c_{val} for any name c.

3 Dependently Typed Higher Order Logic

We now present our translation's target language, polymorphic DHOL [31], a generalisation of DHOL [33]. Unless otherwise specified, by DHOL we mean the polymorphic variant. DHOL consists of a HOL basis, except that the function type $A \to B$ is replaced by a dependent type $\Pi_{x:A}B$, where x may occur freely in B. Furthermore, there exists a sound and complete translation to HOL [31, 33], which enables automated theorem proving using existing efficient HOL provers.

We present the syntax and rules mainly following Ranalter et al. [31] and add some rules omitted there following Rothgang et al. [32, 33]. We also adapt theories slightly following Rabe [28]. We further add names to axioms and local assumptions following Rothgang et al. [33]. Finally, we index the ✓ construct used in the substitutions for formulae by the formula they replace. These changes are needed because we want the translation function to be injective in the soundness proof.

The syntax of DHOL theories, contexts, substitutions, types and terms is given by the following grammar:

$$\begin{aligned}
\text{theories } T &::= . \mid T, a : \Pi_\Delta : \text{type} \mid T, c : \Pi_\Delta A \mid T, c \rhd \Pi_\Delta F \\
\text{contexts } \Gamma, \Delta &::= \circ \mid \Gamma, \alpha : \text{type} \mid \Gamma, x : A \mid \Gamma, x \rhd F \\
\text{substitutions } \gamma &::= \bullet \mid \gamma, A \mid \gamma, t \mid \gamma, \checkmark_F \\
\text{types } A, B &::= a\ \overrightarrow{A}\ \overrightarrow{t} \mid \alpha \mid \Pi_{x:A} B \mid o \\
\text{terms } t, u, F, G &::= c\ \overrightarrow{A} \mid x \mid \lambda_{x:A} t \mid t\ u \mid F \Rightarrow G \mid t =_A u
\end{aligned}$$

The type o denotes the booleans, and terms $F : o$ are also called formulae (or propositions). We call (polymorphic) formulae asserted in theories, written $c \rhd \Pi_\Delta F$, global assumptions, and we call (monomorphic) formulae asserted in contexts, written $x \rhd F$, local assumptions. We use the shorthand notation $\Pi_{\overrightarrow{x:A}} B$ to mean $\Pi_{x_1:A_1} \ldots \Pi_{x_n:A_n} B$, and, as in DTT3, $n = 0$ is allowed, in which case the notation degenerates to B. Theories denote global declarations similar to signatures in DTT3. For substitutions, $\checkmark_F$ is the 'term' that may be substituted for a local assumption $x \rhd F$ if and only if F is provable. Note that type is not itself a type. We write $t_1[t_2/x]$ and $t_1[t_2]$ to denote capture avoiding substitution. Again, we sometimes write $A \to B$ instead of $\Pi_{x:A} B$ when B does not depend on x.

Finally, $F \Rightarrow G$ is a dependent implication. This means that to type-check $\Gamma \vdash_T F \Rightarrow G : o$, we need to type-check $\Gamma \vdash_T F : o$ as usual but may assume that F is true when type-checking G, so we need to prove $\Gamma, x \rhd F \vdash_T G : o$. This makes it possible to state formulae such as $\Gamma \vdash a =_A b \Rightarrow f\ a =_{B[a]} f\ b$ for some type A, a type family B and a dependent function $f : \Pi_{x:A} B$, since the truth of $a =_A b$ may be assumed during type checking. DHOL being an extensional logic, it has only one kind of equality instead of distinguishing between definitional and propositional equality as is done in DTT3. As a result, under the assumption that $a =_A b$, the types $B[a]$ and $B[b]$ can be unified directly, without needing to add constructs such as DTT3's cast. On the other hand, this means that type checking in DHOL is undecidable. This is considered tolerable because the primary use case of DHOL, automated theorem proving, is undecidable regardless.

We define the remaining logical connectives and quantifiers following Andrews [1] and Rothgang et al. [33]:

$$\begin{aligned}
\top &:= (\lambda_{x:o} x) =_{o\to o} (\lambda_{x:o} x) \\
\bot &:= (\lambda_{x:o} x) =_{o\to o} (\lambda_{x:o} \top) \\
\neg F &:= F =_o \bot \\
\forall_{x:A} F &:= \lambda_{x:A} F =_{A\to o} \lambda_{x:A} \top \\
\exists_{x:A} F &:= \neg \forall_{x:A} \neg F \\
F \wedge G &:= \neg(F \Rightarrow \neg G) \\
F \vee G &:= \neg F \Rightarrow G
\end{aligned}$$

where F and G are formulae.

The inference rules for well formed theories T are as follows:

$$\boxed{\vdash T\ \text{Thy}} \qquad \frac{}{\vdash .\ \text{Thy}}\ \text{ThyEmpty} \qquad \frac{\vdash T\ \text{Thy} \qquad a \notin T \qquad \vdash_T \Delta\ \text{Ctx}}{\vdash T, a : \Pi_\Delta : \text{type Thy}}\ \text{ThyTp}$$

$$\frac{\vdash T\ \text{Thy} \qquad c \notin T \qquad \Delta \vdash_T A : \text{type}}{\vdash T, c : \Pi_\Delta A\ \text{Thy}}\ \text{ThyCon} \qquad \frac{\vdash T\ \text{Thy} \qquad c \notin T \qquad \Delta \vdash_T F : o}{\vdash T, c \rhd \Pi_\Delta F\ \text{Thy}}\ \text{ThyAsn}$$

The rules for contexts are given on the left hand side below, and those for substitutions for them are given on the right hand side:

$$\boxed{\vdash_T \Delta\ \text{Ctx}} \qquad \boxed{\Gamma \vdash_T \Delta \leftarrow \delta}$$

$$\frac{\vdash T\ \mathsf{Thy}}{\vdash_T \circ\ \mathsf{Ctx}}\mathsf{CtxEmpty} \qquad \frac{\vdash_T \Gamma\ \mathsf{Ctx}}{\Gamma \vdash_T \circ \leftarrow \bullet}\mathsf{SubEmpty}$$

$$\frac{\vdash_T \Delta\ \mathsf{Ctx}}{\vdash_T \Delta, \alpha : \mathsf{type}\ \mathsf{Ctx}}\mathsf{CtxTp} \qquad \frac{\Gamma \vdash_T \Delta \leftarrow \delta \qquad \Gamma \vdash_T A : \mathsf{type}}{\Gamma \vdash_T \Delta, \alpha : \mathsf{type} \leftarrow \delta, A}\mathsf{SubTp}$$

$$\frac{\vdash_T \Delta\ \mathsf{Ctx} \qquad \Delta \vdash_T A : \mathsf{type}}{\vdash_T \Delta, x : A\ \mathsf{Ctx}}\mathsf{CtxVar} \qquad \frac{\Gamma \vdash_T \Delta \leftarrow \delta \qquad \Delta \vdash_T A : \mathsf{type} \qquad \Gamma \vdash_T t : A[\delta]}{\Gamma \vdash_T \Delta, x : A \leftarrow \delta, t}\mathsf{SubVar}$$

$$\frac{\vdash_T \Delta\ \mathsf{Ctx} \qquad \Delta \vdash_T F : o}{\vdash_T \Delta, x \rhd F\ \mathsf{Ctx}}\mathsf{CtxAsn} \qquad \frac{\Gamma \vdash_T \Delta \leftarrow \delta \qquad \Delta \vdash_T F : o \qquad \Gamma \vdash_T F[\delta]}{\Gamma \vdash_T \Delta, x \rhd F \leftarrow \delta, \checkmark_F}\mathsf{SubAsn}$$

The rules for looking up a type, term or assumption in a theory are given on the left hand side, and the corresponding rules for lookup in a context are given on the right hand side:

$\boxed{\Gamma \vdash_T A : \mathsf{type}}$

$$\frac{\vdash_T \Gamma\ \mathsf{Ctx} \qquad \Gamma \vdash_T \Delta \leftarrow \delta \qquad a : \Pi_\Delta : \mathsf{type}\ \mathsf{in}\ T}{\Gamma \vdash_T a\ \delta : \mathsf{type}}\mathsf{TpSym} \qquad \frac{\vdash_T \Gamma\ \mathsf{Ctx} \qquad \alpha : \mathsf{type}\ \mathsf{in}\ \Gamma}{\Gamma \vdash_T \alpha : \mathsf{type}}\mathsf{TpVar}$$

$\boxed{\Gamma \vdash_T t : A}$

$$\frac{\vdash_T \Gamma\ \mathsf{Ctx} \qquad \Gamma \vdash_T \Delta \leftarrow \delta \qquad c : \Pi_\Delta B\ \mathsf{in}\ T}{\Gamma \vdash_T c\ \delta : B[\delta]}\mathsf{TermSym} \qquad \frac{\vdash_T \Gamma\ \mathsf{Ctx} \qquad x : A\ \mathsf{in}\ \Gamma}{\Gamma \vdash_T x : A}\mathsf{TermVar}$$

$\boxed{\Gamma \vdash_T F}$

$$\frac{\vdash_T \Gamma\ \mathsf{Ctx} \qquad \Gamma \vdash_T \Delta \leftarrow \delta \qquad c \rhd \Pi_\Delta F\ \mathsf{in}\ T}{\Gamma \vdash_T F[\delta]}\mathsf{ValidSym} \qquad \frac{\vdash_T \Gamma\ \mathsf{Ctx} \qquad x \rhd F\ \mathsf{in}\ \Gamma}{\Gamma \vdash_T F}\mathsf{ValidVar}$$

The rules for the formation and the terms of booleans are as follows:

$$\frac{\vdash_T \Gamma\ \mathsf{Ctx}}{\Gamma \vdash_T o : \mathsf{type}}\mathsf{TpBool} \qquad \frac{\Gamma \vdash_T t : A \qquad \Gamma \vdash_T u : A}{\Gamma \vdash_T t =_A u : o}\mathsf{TermEq}$$

$$\frac{\Gamma \vdash_T F : o \qquad \Gamma, x \rhd F \vdash_T G : o}{\Gamma \vdash_T F \Rightarrow G : o}\mathsf{TermImpl}$$

Type formation, λ-abstraction and application are defined as follows:

$$\frac{\Gamma \vdash_T A : \mathsf{type} \qquad \Gamma, x : A \vdash_T B : \mathsf{type}}{\Gamma \vdash_T \Pi_{x:A} B : \mathsf{type}}\mathsf{TpPi} \qquad \frac{\Gamma \vdash_T A : \mathsf{type} \qquad \Gamma, x : A \vdash_T t : B}{\Gamma \vdash_T \lambda_{x:A} t : \Pi_{x:A} B}\mathsf{TermLambda}$$

$$\frac{\Gamma \vdash_T t : \Pi_{x:A} B \qquad \Gamma \vdash_T u : A}{\Gamma \vdash_T t\ u : B[u]}\mathsf{TermApply}$$

The type equality judgement is defined by the following rules:

$\boxed{\Gamma \vdash_T A \equiv A'}$

$$\frac{\vdash_T \Gamma\ \mathsf{Ctx} \qquad \alpha : \mathsf{type}\ \mathsf{in}\ \Gamma}{\Gamma \vdash_T \alpha \equiv \alpha}\mathsf{TpEqVar}$$

$$\frac{\vdash_T \Gamma\ \mathsf{Ctx} \qquad \Gamma \vdash_T \delta \equiv_\Delta \delta' \qquad a : \Pi_\Delta \mathsf{type}\ \mathsf{in}\ T}{\Gamma \vdash_T a\ \delta \equiv a\ \delta'}\mathsf{TpEqSym} \qquad \frac{\vdash_T \Gamma\ \mathsf{Ctx}}{\Gamma \vdash_T o \equiv o}\mathsf{TpEqBool}$$

$$\frac{\Gamma \vdash_T A \equiv A' \qquad \Gamma, x : A \vdash_T B \equiv B'}{\Gamma \vdash_T \Pi_{x:A} B \equiv \Pi_{x:A'} B'}\,\mathsf{TpEqPi} \qquad \frac{\Gamma \vdash_T t : A \qquad \Gamma \vdash_T A \equiv A'}{\Gamma \vdash_T t : A'}\,\mathsf{TermConvert}$$

where $\Gamma \vdash_T \delta \equiv_\Delta \delta'$ abbreviates the component-wise provable equality of two substitutions for Δ.

The following rules were omitted by Ranalter et al. [31], so we adapt them from Rothgang et al. [32, 33]:

$$\frac{\Gamma \vdash_T \lambda_{x:A} t : \Pi_{x:A} B \qquad \Gamma \vdash_T u : A}{\Gamma \vdash_T (\lambda_{x:A} t)\ u =_{B[u]} t[u]}\,\mathsf{TermBeta} \qquad \frac{\Gamma \vdash_T t : \Pi_{x:A} B \qquad x \text{ not in } \Gamma}{\Gamma \vdash_T t =_{\Pi_{x:A} B} \lambda_{x:A} t\ x}\,\mathsf{TermEta}$$

$$\frac{\Gamma \vdash_T A \equiv A' \qquad \Gamma, x : A \vdash_T t =_B t'}{\Gamma \vdash_T \lambda_{x:A} t =_{\Pi_{x:A} B} \lambda_{x:A'} t'}\,\mathsf{CongLambda} \qquad \frac{\Gamma \vdash_T t =_A t' \qquad \Gamma \vdash_T f =_{\Pi_{x:A} B} f'}{\Gamma \vdash_T f\ t =_{B[t]} f'\ t'}\,\mathsf{CongApply}$$

$$\frac{\Gamma \vdash_T t : A}{\Gamma \vdash_T t =_A t}\,\mathsf{Refl} \qquad \frac{\Gamma \vdash_T t =_A s}{\Gamma \vdash_T s =_A t}\,\mathsf{Sym} \qquad \frac{\Gamma \vdash_T F =_o F' \qquad \Gamma \vdash_T F'}{\Gamma \vdash_T F}\,\mathsf{CongValid}$$

$$\frac{\Gamma, x \rhd F \vdash_T G}{\Gamma \vdash_T F \Rightarrow G}\,\mathsf{ImplIntro} \qquad \frac{\Gamma \vdash_T F \Rightarrow G \qquad \Gamma \vdash_T F}{\Gamma \vdash_T G}\,\mathsf{MP}$$

Finally, it is possible to add the following rule for boolean extensionality:

$$\frac{\Gamma \vdash_T t : \Pi_{x:o} o \qquad \Gamma \vdash_T t\ \bot \qquad \Gamma \vdash_T t\ \top}{\Gamma, x : o \vdash_T t\ x}\,\mathsf{BoolExt}$$

If we add this rule, then a suitable rule for the law of the excluded middle, propositional extensionality, function extensionality, as well as suitable axiomatisations of the empty type, the singleton type, the product and the co-product, are needed in DTT3 to guarantee the translation's soundness.

The main differences between DHOL and DTT3 can be seen even more clearly when looking at an alternative presentation of DHOL, also as an extension of a pure type system. DTT3 features nine possible combinations for the rule ∀Form. HOL can be viewed as the pure type system with sorts $\{o, \text{type}, \text{kind}\}$, axioms $\{(o : \text{type}), (\text{type} : \text{kind})\}$ and rules $\{(o, o, o), (\text{type}, o, o), (\text{type}, \text{type}, \text{type})\}$ [15]. DHOL then allows for formations (type, kind, kind) in theories and adds the three rules for polymorphism, restricted to rank 1 and also to theories. This then leaves the rules $(o, \text{type}, \text{type})$ and $(o, \text{kind}, \text{kind})$, which DHOL does not support. These two rules would correspond to abstraction over proof obligations and enable functions with pre-conditions [28].

Full DTT is further strictly stronger than DHOL. This is due to the former's infinite universe hierarchy, which allows for the same strength as classical set theory, specifically Zermelo-Fränkel set theory with choice, with inaccessible cardinals [36]. By contrast, DHOL has the same strength as HOL by the existence of a sound and complete translation. HOL, in turn, is subsumed by the calculus of constructions, which can be expressed in set theory without additional cardinals [36].

We believe that DHOL lends itself well to automated theorem proving, due to its extensionality and because it resembles HOL, for which reasonably strong proof automation has become available recently [34]. This might enable a term rewriting based proof strategy such as superposition, which is highly successful in FOL and HOL, to be implemented in DHOL and achieve high success rates.

4 The Translation

Here we present our translation from DTT3 to DHOL. The main difficulty in defining it is to map the expressions of DTT3 to either DHOL types or DHOL terms depending on the situation. We achieve this through the definition of several mutually recursive translation functions.

There are dedicated functions for translating signatures to theories and DTT3 contexts to DHOL contexts. For DTT3 expressions, however, there are four functions: one for expressions that are 'type like', meaning expressions that, once supplied with all arguments, are of type U_1; one for expressions that are 'term like', meaning expressions that, once supplied with all arguments, are of some type, that in turn is of type U_1; one to translate applications; and one to translate arguments of constants, which is nearly identical to the previous function but also translates proof terms, yielding a DHOL term, type or $\checkmark$, where the latter are the stand in for proof terms in DHOL.

Another important aspect is the transition from intensional to extensional equality. We remove all casts from expressions, since they are not needed in the extensional setting. However, since type checking DTT3 expressions is necessary in some cases of our translation, we cannot simply erase all of them immediately, since they might be necessary at some point. Therefore we need to integrate cast erasure with the translation.

Definition 4.1. Let Γ be a DTT3 context and $\mathcal{T}$ be a DTT3 signature. Then:

- $[\![\cdot]\!]^{\mathcal{T};\Gamma}_{\text{app}}$ maps DTT3 expressions to DHOL types or terms,
- $[\![\cdot]\!]^{\mathcal{T};\Gamma}_{\text{exp}}$ maps DTT3 expressions to DHOL substitution components,
- $[\![\cdot]\!]^{\mathcal{T};\Gamma}_{\text{ty}}$ maps DTT3 expressions α where $\Gamma \vdash_{\mathcal{T}} \alpha : \forall_{\overrightarrow{x}:\overrightarrow{\alpha}} \mathsf{U}_1$ to DHOL types,
- $[\![\cdot]\!]^{\mathcal{T};\Gamma}_{\text{ter}}$ maps DTT3 expressions e where $\Gamma \vdash_{\mathcal{T}} e : \forall_{\overrightarrow{x}:\overrightarrow{\beta}} \alpha$ and $\Gamma \vdash_{\mathcal{T}} \alpha : \mathsf{U}_1$ to DHOL terms,
- $[\![\cdot]\!]_{\text{sig}}$ maps DTT3 signatures to DHOL theories and
- $[\![\cdot]\!]^{\mathcal{T}}_{\text{ctx}}$ maps DTT3 contexts to DHOL contexts.

Moreover, we introduce the function rmc, which takes a DTT3 expression and if it is of the form cast $p\ e$ it replaces it by e. Finally, we let $[\![\cdot]\!]^{\mathcal{T};\Gamma}_{\text{ty0}} = [\![\cdot]\!]^{\mathcal{T};\Gamma}_{\text{ty}} \circ \text{rmc}$ and $[\![\cdot]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} = [\![\cdot]\!]^{\mathcal{T};\Gamma}_{\text{ter}} \circ \text{rmc}$.

$$[\![e]\!]^{\mathcal{T};\Gamma}_{\text{app}} = \begin{cases} c\ \overrightarrow{[\![t]\!]^{\mathcal{T};\Gamma}_{\text{exp}}} & \text{if } e = c\ \overrightarrow{t} \text{ and } c \neq \text{cast is a constant} \\ [\![e_1]\!]^{\mathcal{T};\Gamma}_{\text{app}}\ [\![e_2]\!]^{\mathcal{T};\Gamma}_{\text{app}} & \text{if } e = e_1\ e_2 \text{ and } e_1 \neq c\ \overrightarrow{t} \text{ for some constant } c \\ [\![e]\!]^{\mathcal{T};\Gamma}_{\text{ty0}} & \text{if } \Gamma \vdash_{\mathcal{T}} e : \forall_{\overrightarrow{x}:\overrightarrow{\alpha}} \mathsf{U}_1 \\ & \text{and } e = \text{cast}\ p\ e_1 \text{ or } e \text{ is not an application} \\ [\![e]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} & \text{if } \Gamma \vdash_{\mathcal{T}} e : \forall_{\overrightarrow{x}:\overrightarrow{\beta}} \alpha \text{ with } \Gamma \vdash_{\mathcal{T}} \alpha : \mathsf{U}_1 \\ & \text{and } e = \text{cast}\ p\ e_1 \text{ or } e \text{ is not an application} \end{cases}$$

$$[\![e]\!]^{\mathcal{T};\Gamma}_{\text{exp}} = \begin{cases} c\ \overrightarrow{[\![t]\!]^{\mathcal{T};\Gamma}_{\text{exp}}} & \text{if } e = c\ \overrightarrow{t} \text{ and } c \neq \text{cast is a constant} \\ [\![e_1]\!]^{\mathcal{T};\Gamma}_{\text{app}}\ [\![e_2]\!]^{\mathcal{T};\Gamma}_{\text{app}} & \text{if } e = e_1\ e_2 \text{ and } e_1 \neq c\ \overrightarrow{t} \text{ for some constant } c \\ & \text{and not } \Gamma \vdash_{\mathcal{T}} e : \alpha \text{ with } \Gamma \vdash_{\mathcal{T}} \alpha : \mathbb{P} \\ [\![e]\!]^{\mathcal{T};\Gamma}_{\text{ty0}} & \text{if } \Gamma \vdash_{\mathcal{T}} e : \forall_{\overrightarrow{x}:\overrightarrow{\alpha}} \mathsf{U}_1 \\ & \text{and } e = \text{cast}\ p\ e_1 \text{ or } e \text{ is not an application} \\ [\![e]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} & \text{if } \Gamma \vdash_{\mathcal{T}} e : \forall_{\overrightarrow{x}:\overrightarrow{\beta}} \alpha \text{ with } \Gamma \vdash_{\mathcal{T}} \alpha : \mathsf{U}_1 \\ & \text{and } e = \text{cast}\ p\ e_1 \text{ or } e \text{ is not an application} \\ \checkmark_{[\![\alpha]\!]^{\mathcal{T};\Gamma}_{\text{ter0}}} & \text{if } \Gamma \vdash_{\mathcal{T}} e : \alpha \text{ with } \Gamma \vdash_{\mathcal{T}} \alpha : \mathbb{P} \end{cases}$$

$$[\![\alpha]\!]^{\mathcal{T};\Gamma}_{\text{ty}} = \begin{cases} \alpha & \text{if } \alpha \text{ is a variable or a constant} \\ o & \text{if } \alpha = \mathbb{P} \\ [\![\alpha]\!]^{\mathcal{T};\Gamma}_{\text{app}} & \text{if } \alpha = \alpha_1\ \alpha_2 \\ \Pi_{x:[\![\alpha_1]\!]^{\mathcal{T};\Gamma}_{\text{ty0}}} [\![\alpha_2]\!]^{\mathcal{T};\Gamma,x:\alpha_1}_{\text{ty0}} & \text{if } \alpha = \forall_{x:\alpha_1}\alpha_2 \end{cases}$$

$$[\![e]\!]^{\mathcal{T};\Gamma}_{\text{ter}} = \begin{cases} \top & \text{if } e = \top \\ \bot & \text{if } e = \bot \\ e & \text{if } e \text{ is a variable or a constant different from the above} \\ [\![e_1]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} \diamond [\![e_2]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} & \text{if } e = e_1 \diamond e_2 \text{ with } \diamond \in \{\wedge, \vee\} \\ \neg [\![e_1]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} & \text{if } e = \neg e_1 \\ [\![e_1]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} =_{[\![\alpha]\!]^{\mathcal{T};\Gamma}_{\text{ty0}}} [\![e_2]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} & \text{if } e = (e_1 =_\alpha e_2) \\ [\![e]\!]^{\mathcal{T};\Gamma}_{\text{app}} & \text{if } e \text{ is an application different from the above} \\ \lambda_{x:[\![\alpha]\!]^{\mathcal{T};\Gamma}_{\text{ty0}}}.[\![e_1]\!]^{\mathcal{T};\Gamma,x:\alpha}_{\text{ter0}} & \text{if } e = \lambda_{x:\alpha}.e_1 \\ \exists_{x:[\![\alpha]\!]^{\mathcal{T};\Gamma}_{\text{ty0}}} [\![e_1]\!]^{\mathcal{T};\Gamma,x:\alpha}_{\text{ter0}} & \text{if } e = \exists_{x:\alpha} e_1 \\ \forall_{x:[\![\alpha]\!]^{\mathcal{T};\Gamma}_{\text{ty0}}} [\![e_1]\!]^{\mathcal{T};\Gamma,x:\alpha}_{\text{ter0}} & \text{if } e = \forall_{x:\alpha} e_1 \text{ with } \Gamma \vdash_{\mathcal{T}} \alpha : \beta \text{ and } \beta \neq \mathbb{P} \\ [\![\alpha]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} \Rightarrow [\![e_1]\!]^{\mathcal{T};\Gamma}_{\text{ter0}} & \text{if } e = \forall_{x:\alpha} e_1 \text{ with } \Gamma \vdash_{\mathcal{T}} \alpha : \mathbb{P} \end{cases}$$

$$[\![\Gamma]\!]^{\mathcal{T}}_{\text{ctx}} = \begin{cases} \circ & \text{if } \Gamma = \circ \\ [\![\Gamma']\!]^{\mathcal{T}}_{\text{ctx}}, x : \text{type} & \text{if } \Gamma = \Gamma', x : \mathsf{U}_1 \\ [\![\Gamma']\!]^{\mathcal{T}}_{\text{ctx}}, x : o & \text{if } \Gamma = \Gamma', x : \mathbb{P} \\ [\![\Gamma']\!]^{\mathcal{T}}_{\text{ctx}}, x \rhd [\![e]\!]^{\mathcal{T};\Gamma'}_{\text{ter}} & \text{if } \Gamma = \Gamma', x : e \text{ with } e : \mathbb{P} \\ [\![\Gamma']\!]^{\mathcal{T}}_{\text{ctx}}, x : [\![e]\!]^{\mathcal{T};\Gamma'}_{\text{ty}} & \text{if } \Gamma = \Gamma', x : e \text{ with } e \neq \mathbb{P} \text{ and } e : \mathsf{U}_1 \end{cases}$$

$$[\![\mathcal{T}]\!]_{\text{sig}} = \begin{cases} . & \text{if } \mathcal{T} = . \\ [\![\mathcal{T}', (c, \alpha)]\!]_{\text{sig}}, c_{\text{val}} \rhd c =_{[\![\alpha]\!]^{\mathcal{T};\Gamma}_{\text{ty}}} [\![e]\!]^{\mathcal{T};\Gamma}_{\text{ter}} & \text{if } \mathcal{T} = \mathcal{T}', (c, \alpha, e) \\ [\![\mathcal{T}']\!]_{\text{sig}}, c \rhd \Pi_{[\![\Delta]\!]_{\text{ctx}}} [\![e]\!]^{\mathcal{T};\Gamma,\Delta}_{\text{ter}} & \text{if } \mathcal{T} = \mathcal{T}', (c, \forall_\Delta e) \text{ with } e : \mathbb{P} \\ [\![\mathcal{T}']\!]_{\text{sig}}, c : \Pi_{[\![\Delta]\!]_{\text{ctx}}} : \text{type} & \text{if } \mathcal{T} = \mathcal{T}', (c, \forall_\Delta \mathsf{U}_1) \\ [\![\mathcal{T}']\!]_{\text{sig}}, c : \Pi_{[\![\Delta]\!]_{\text{ctx}}} [\![\alpha]\!]^{\mathcal{T};\Gamma,\Delta}_{\text{ty}} & \text{if } \mathcal{T} = \mathcal{T}', (c, \forall_\Delta \alpha) \text{ with } \alpha : \mathsf{U}_1 \end{cases}$$

The annotations on some of the translation functions with a context and signature are necessary for the definition, but we omit them when they can easily be inferred.

The translation functions for DTT3 expressions inspect them without β-, δ- or η-normalisation. The definition of $[\![\cdot]\!]_{\text{ter}}$ includes cases for $\top$, $\bot$, $\wedge$, $\vee$, $\neg$ and $\exists$, which are not built into DTT3, and maps these to the corresponding DHOL constructs. We expect that preserving the DTT3 expressions' original structure will facilitate more efficient theorem proving on the resulting DHOL formulae.

The use of multiple translation functions is necessary to ensure that certain expressions get translated to DHOL terms and others to DHOL types. This is especially relevant for dependent function types, which may be viewed as either (dependent) implications, meaning terms, or as dependent function types in DHOL, depending on the context.

Due to the differences between DTT3 and DHOL we have pointed out at the end of Section 3, we also restrict the translation's domain. We define the *translatable* fragment by excluding higher rank polymorphism, enforcing all constants to be fully applied and disallowing λ-abstractions over propositions, as well as the corresponding function types. The last restriction is necessary because

DHOL supports abstraction over proof obligations and polymorphism only in theories, so we need to restrict them accordingly and make sure proof terms only occur as arguments to constants.

Definition 4.2 (Translatability). Given a DTT3 signature $\mathcal{T}$ and a DTT3 context Γ, we define $(\mathcal{T},\Gamma)$*-pre-translatability* of a DTT3 expression e inductively:

(1) $\mathbb{P}$ is $(\mathcal{T},\Gamma)$-pre-translatable.
(2) variables, constants c are $(\mathcal{T},\Gamma)$-pre-translatable, if $\Gamma \vdash_{\mathcal{T}} c : \alpha$, where $\Gamma \vdash_{\mathcal{T}} \alpha \not: \mathbb{P}$ and $\alpha \neq \mathsf{U}_2$.
(3) $e_1\ e_2$ is $(\mathcal{T},\Gamma)$-pre-translatable, if
  (a) $e_1 = c\ \overrightarrow{t}$, where $c \neq \text{cast}$ is a $(\mathcal{T},\Gamma)$-pre-translatable constant and each t_i as well as e_2 is either $(\mathcal{T},\Gamma)$-pre-translatable or of type α with $\Gamma \vdash_{\mathcal{T}} \alpha : \mathbb{P}$, or
  (b) e_1 and e_2 are $(\mathcal{T},\Gamma)$-pre-translatable, or
  (c) $e_1 = \text{cast}\ p$ and e_2 is $(\mathcal{T},\Gamma)$-pre-translatable.
(4) $e_1 =_\alpha e_2$ is $(\mathcal{T},\Gamma)$-pre-translatable, if e_1, e_2 and α are and if $\alpha \not: \mathbb{P}$.
(5) $\lambda_{x:\alpha} e_1$ is $(\mathcal{T},\Gamma)$-pre-translatable, if
  (a) α is $(\mathcal{T},\Gamma)$-pre-translatable,
  (b) e_1 is $(\mathcal{T},\Gamma,x:\alpha)$-pre-translatable,
  (c) and $\Gamma \vdash_{\mathcal{T}} \lambda_{x:\alpha} e_1 : \forall_{x:\alpha}\beta$ with $\Gamma \vdash_{\mathcal{T}} \alpha : \mathsf{U}_1$ and $\Gamma, x:\alpha \vdash_{\mathcal{T}} \beta : \mathsf{U}_1$,
(6) $\forall_{x:\alpha}\beta$ is $(\mathcal{T},\Gamma)$-pre-translatable, if
  (a) α is $(\mathcal{T},\Gamma)$-pre-translatable,
  (b) β is $(\mathcal{T},\Gamma,x:\alpha)$-pre-translatable,
  (c) $\Gamma \vdash_{\mathcal{T}} \forall_{x:\alpha}\beta \not: \mathsf{U}_2$,
  (d) and if $\Gamma \vdash_{\mathcal{T}} \forall_{x:\alpha}\beta \not: \mathbb{P}$, then $\Gamma \vdash_{\mathcal{T}} \alpha \not: \mathbb{P}$.

We call e $(\mathcal{T},\Gamma)$*-translatable* if it is $(\mathcal{T},\Gamma)$-pre-translatable and all occurring constants are fully applied, meaning the type of all sub-expressions $c\ \overrightarrow{e}$, where $\overrightarrow{e}$ are all arguments applied to c, is not a function type.

Given a DTT3 signature $\mathcal{T}$, we define $\mathcal{T}$*-translatability* of a DTT3 context Γ inductively:

(1) $\circ$ is $\mathcal{T}$-translatable,
(2) $\Gamma, x:\alpha$ is $\mathcal{T}$-translatable, if Γ is $\mathcal{T}$-translatable and either α is $(\mathcal{T},\Gamma)$-translatable or $\alpha = \mathsf{U}_1$.

We define *translatability* of a DTT3 signature $\mathcal{T}$ inductively:

(1) . is translatable,
(2) $\mathcal{T}, (c,\alpha)$ is translatable, if $\mathcal{T}$ is translatable and α is $(\mathcal{T},\circ)$-translatable or if α is of the form $\forall_\Delta \beta$, where Δ is $\mathcal{T}$-translatable as a context and β is $(\mathcal{T},\Delta)$-translatable or $\beta = \mathsf{U}_1$,
(3) $\mathcal{T}, (c,\alpha,e)$ is translatable, if $\mathcal{T}, (c,\alpha)$ is translatable and e is $(\mathcal{T},\circ)$-translatable.

We omit $\mathcal{T}$ and Γ if they are clear from the context.

We next show that our translation functions are well defined on the translatable fragment and that all translated constructs are actually translatable. This means that the domain of our functions is exactly the translatable DTT3 constructs.

Theorem 4.3 (Well definedness). *The translation functions from Def. 4.1 are well defined on the translatable fragment of DTT3, meaning they are defined for all well typed translatable arguments.*

Proof. The proof is by mutual induction on the definition of the translation functions.

The cases for $[\![\cdot]\!]_{\text{ctx}}$ and $[\![\cdot]\!]_{\text{sig}}$ are straightforward, since all side conditions directly match the definition of translatability.

For $[\![\cdot]\!]_{\text{app}}$, $[\![\cdot]\!]_{\text{exp}}$, $[\![\cdot]\!]_{\text{ty}}$ and $[\![\cdot]\!]_{\text{ter}}$, we will perform a case distinction on the expression e being translated:

If e is an application, then for $[\![\cdot]\!]_{\text{ty}}$ and $[\![\cdot]\!]_{\text{ter}}$ we are again done by induction. For $[\![\cdot]\!]_{\text{app}}$ and $[\![\cdot]\!]_{\text{exp}}$, if $e = c\ \overrightarrow{t}$, we have two cases. If $c = \text{cast}$, then $e = \text{cast}\ p\ e_1$, due to e being translatable and

therefore cast being fully applied. Then we perform a case distinction on e_1 as above and are done by induction. If $c \neq$ cast, then we apply the induction hypothesis regarding $[\![t_i]\!]_{\text{exp}}$, where in the translatability of the t_i, they are allowed to be proof terms, in which case the last case of $[\![\cdot]\!]_{\text{exp}}$ applies.

If e is a variable or constant, $[\![e]\!]_{\text{ter}}$ and $[\![e]\!]_{\text{ty}}$ both output e and $[\![e]\!]_{\text{app}}$ and $[\![e]\!]_{\text{exp}}$ translate to either $[\![e]\!]_{\text{ter}}$ or $[\![e]\!]_{\text{ty}}$ so we are done in all cases.

If $e = e_1 =_\alpha e_2$, we know that $\Gamma \vdash_{\mathcal{T}} e : \mathbb{P}$, which means the fourth case of $[\![\cdot]\!]_{\text{app}}$ and $[\![\cdot]\!]_{\text{exp}}$ apply respectively. For $[\![\cdot]\!]_{\text{ter}}$, we are done by induction right away, since by the translatability of e, we know that $\alpha \not: \mathbb{P}$ but is still translatable, so the induction hypothesis regarding $[\![\alpha]\!]_{\text{ty}}$ applies. For $[\![\cdot]\!]_{\text{ty}}$, e is not in the domain and we are done.

If $e = \lambda_{x:\alpha} e_1$, since by e being translatable we know that $\Gamma \vdash_{\mathcal{T}} e : \forall_{x:\alpha} \beta$ and $\Gamma \vdash_{\mathcal{T}} \forall_{x:\alpha} \beta \not: \mathsf{U}_2$. From this we further know that $\beta \not: \mathsf{U}_2$. We also get from the translatability that $\Gamma \vdash_{\mathcal{T}} \alpha \not: \mathbb{P}$ and $\Gamma \vdash_{\mathcal{T}} \alpha \not: \mathbb{P}$. This means either $\Gamma \vdash_{\mathcal{T}} \beta : \mathsf{U}_1$, meaning for $[\![\cdot]\!]_{\text{app}}$ and $[\![\cdot]\!]_{\text{exp}}$ case three applies. For $[\![\cdot]\!]_{\text{ter}}^{\mathcal{T};\Gamma}$, we are again done by induction right away. For $[\![\cdot]\!]_{\text{ty}}$, e is not in the domain and we are done.

If $e = \forall_{x:\alpha} \beta$, since by e being translatable we know that $\Gamma \vdash_{\mathcal{T}} \forall_{x:\alpha} \beta \not: \mathsf{U}_2$, either $\Gamma \vdash_{\mathcal{T}} e : \mathsf{U}_1$ or $\Gamma \vdash_{\mathcal{T}} e : \mathbb{P}$, meaning for $[\![\cdot]\!]_{\text{app}}$ and $[\![\cdot]\!]_{\text{exp}}$ either case two or three apply. For $[\![\cdot]\!]_{\text{ter}}$, we know that $\Gamma \vdash_{\mathcal{T}} e : \mathbb{P}$, meaning $\Gamma \vdash_{\mathcal{T}} \beta : \mathbb{P}$. Then depending on the type of α, either the $\forall$- or the $\Rightarrow$-case applies. For $[\![\cdot]\!]_{\text{ty}}$, note that $\Gamma \vdash_{\mathcal{T}} \alpha \not: \mathbb{P}$, because e is translatable and $\Gamma \vdash_{\mathcal{T}} e : \mathsf{U}_1$ is in the domain of $[\![\cdot]\!]_{\text{ty}}$. Therefore we are immediately done by induction. □

For the reverse direction, we need to slightly weaken the conclusion for expressions from 'translatable' to 'pre-translatable', since during the translation process, when considering an application, the left sub-expression can then have constants which are no longer fully applied. We also exclude $[\![\cdot]\!]_{\text{exp}}$ from the theorem statement, as it is also defined for proof terms, which according to (3)(a) in Def. 4.2 are only ever translatable as arguments to constants. The other functions make sure that $[\![\cdot]\!]_{\text{exp}}$ is only ever called in such cases, so the theorem still holds for the other functions using $[\![\cdot]\!]_{\text{exp}}$.

Theorem 4.4. *All well typed DTT3 contexts and signatures for which the translation functions from Def. 4.1 are defined are translatable. All well typed DTT3 expressions for which the translation functions* $[\![\cdot]\!]_{\text{app}}$, $[\![\cdot]\!]_{\text{ter}}$ *and* $[\![\cdot]\!]_{\text{ty}}$ *are defined are pre-translatable.*

Proof. The proof is again by mutual induction on the definition of the translation functions.

Again, the cases for $[\![\cdot]\!]_{\text{ctx}}$ and $[\![\cdot]\!]_{\text{sig}}$ are straightforward, since all side conditions directly match the definition of translatability.

For $[\![\cdot]\!]_{\text{app}}$, $[\![\cdot]\!]_{\text{ty}}$ and $[\![\cdot]\!]_{\text{ter}}$, we will again perform a case distinction on the expression e being translated:

If e is an application then for $[\![e]\!]_{\text{ty}}$ and $[\![e]\!]_{\text{ter}}$ we are immediately done by induction. If $e = c\ \overrightarrow{t}$, we have two cases. If $c =$ cast, then $[\![e]\!]_{\text{app}}$ is only defined if $e =$ cast $p\ e_1$, and only if either $[\![e]\!]_{\text{ty}}$ or $[\![e]\!]_{\text{ter}}$ are defined, so we are done by induction. If $c \neq$ cast, then $[\![e]\!]_{\text{app}}$ is defined exactly if $[\![t_i]\!]_{\text{exp}}$ is defined for all t_i, where we can unfold $[\![\cdot]\!]_{\text{exp}}$ and find that all cases but the last one are identical to $[\![\cdot]\!]_{\text{app}}$, so in these we are done by respective induction steps, and the last case of $[\![\cdot]\!]_{\text{exp}}$ applies for proof terms, which are translatable only in this argument position. Note that this is the only case in which $[\![\cdot]\!]_{\text{exp}}$ is ever called, so the translation functions are not defined on proof terms in any other position within another expression. We therefore conclude that e is translatable.

Constants and variables are always translatable when they are in the domain, which coincides with the requirements on them in Def. 4.2 so there is nothing to be done.

If $e = e_1 =_\alpha e_2$, then $[\![e]\!]_{\text{app}}$ is defined as $[\![e]\!]_{\text{ter}}$, which is in turn defined for e, since $[\![e_1]\!]_{\text{ter}} =_{[\![\alpha]\!]_{\text{ty}}} [\![e_2]\!]_{\text{ter}}$. We are then done by induction, since $[\![\cdot]\!]_{\text{ty}}$ is not defined on expressions of type $\mathbb{P}$ and

therefore $\alpha \not: \mathbb{P}$ in the cases we need to prove. Finally $[\![e]\!]_{\text{ty}}$ is not defined, since $e : \mathbb{P}$, so there is nothing to be done.

If $e = \lambda_{x:\alpha} e_1$, if $[\![e]\!]_{\text{app}}$ is defined, we are done by induction regarding either $[\![e]\!]_{\text{ter}}$ or $[\![e]\!]_{\text{ty}}$, depending on which case applied. If $[\![e]\!]_{\text{ter}}$ is defined, then $[\![e_1]\!]_{\text{ter}}$ and $[\![\alpha]\!]_{\text{ty}}$ are defined, which implies that e_1 and α are translatable and further, from the function's domains, that $e_1 : \forall_{\overrightarrow{x}:\overrightarrow{\beta}} \beta_0$, where $\beta_0 : \mathsf{U}_1$, and $\alpha : \forall_{\overrightarrow{x}:\overrightarrow{\alpha}} \mathsf{U}_1$. This implies that $e : \forall_{x:\alpha} \beta_0$, and then $\forall_{x:\alpha} \beta_0 \not: \mathsf{U}_2$, since α and β_0 are both not of type U_2. From the types of α and β_0, we also get that $\alpha \not: \mathbb{P}$ and $\beta \not: \mathbb{P}$. So we get that both $\alpha : \mathsf{U}_1$ and $\beta : \mathsf{U}_1$. Therefore e is translatable. Finally $[\![e]\!]_{\text{ty}}$ is never defined on λ's, so there is nothing to be done.

If $e = \forall_{x:\alpha} \beta$, if $[\![e]\!]_{\text{app}}$ is defined, we are again done by induction as above. If $[\![e]\!]_{\text{ter}}$ is defined, since we know that $e : \mathsf{U}_\ell$, due to e being well typed by assumption and the only possible rule to use would being $\forall$Form, we know that $e : \mathbb{P}$, since from the function's domain we know that $\mathsf{U}_\ell : \mathsf{U}_1$ and for this to hold, $\ell = 0$ must be the case. Then there are two cases to consider. In the first case, $[\![e]\!]_{\text{ter}} = \forall_{x:[\![\alpha]\!]_{\text{ty}}} [\![\beta]\!]_{\text{ter}}$. In the second case, $[\![e]\!]_{\text{ter}} = [\![\alpha]\!]_{\text{ter}} \Rightarrow [\![\beta]\!]_{\text{ter}}$. In either case $[\![\alpha]\!]_{\text{ty}}$ (or respectively $[\![\alpha]\!]_{\text{ter}}$) and $[\![\beta]\!]_{\text{ter}}$ are defined, which implies that α and β are translatable. Since $e : \mathbb{P}$, conditions (6)(c) and (6)(d) from the definition of translatability are both satisfied. Therefore e is translatable. □

For the corollary we assume all constants to be fully applied, to regain translatability from Theorem 4.4.

Corollary 4.5. *For well typed DTT3 constructs Θ, where all occurring constants are fully applied, the following are equivalent:*

(1) *Θ is translatable.*
(2) *One of $[\![\cdot]\!]_{\text{app}}$, $[\![\cdot]\!]_{\text{ter}}$, $[\![\cdot]\!]_{\text{ty}}$, $[\![\cdot]\!]_{\text{sig}}$ or $[\![\cdot]\!]_{\text{ctx}}$ is defined on Θ.*

Some realistic examples of expressions that fall outside of the translatable fragment are the identity function for U_1, $\lambda_{x:\mathsf{U}_1} x$, the function that returns the natural number n if it is equal to 0, $\lambda_{n:\mathbb{N}} \lambda_{p:n=_{\mathbb{N}} 0} n$, as well as its type, $\forall_{n:\mathbb{N}} \forall_{p:n=_{\mathbb{N}} 0} \mathbb{N}$.

This fragment of DTT3 may seem restrictive, but it is sufficient for many applications of DTT. In particular, higher rank polymorphism and higher universe levels are not needed to formalise many concepts of mathematics that do not relate to category theory or higher algebra.

Example 4.6. An axiomatisation of polymorphic length indexed vectors as a signature $\mathcal{T}$ in DTT3 might look like this:

$(\mathbb{N}, \mathsf{U}_1)$, $(0, \mathbb{N})$, $(\mathsf{suc}, \forall_{n:\mathbb{N}} \mathbb{N})$, $(\mathbb{N}\mathsf{Rec}, \forall_{P:\mathbb{N}\to\mathbb{P}} \forall_{z:P\,0} \forall_{s:\forall_{n:\mathbb{N}} P\,n \to P\,(\mathsf{suc}\,n)} \forall_{n:\mathbb{N}} P\,n)$,
$(\mathsf{Vec}, \forall_{\alpha:\mathsf{U}_1} \forall_{n:\mathbb{N}} \mathsf{U}_1)$, $(\mathsf{nil}, \forall_{\alpha:\mathsf{U}_1} \mathsf{Vec}\ \alpha\ 0)$, $(\mathsf{cons}, \forall_{\alpha:\mathsf{U}_1} \forall_{n:\mathbb{N}} \forall_{v:\mathsf{Vec}\,\alpha\,n} \forall_{x:\alpha} \mathsf{Vec}\ \alpha\ (\mathsf{suc}\ n))$,
$(\mathsf{VecRec}, \forall_{\alpha:\mathsf{U}_1} \forall_{n:\mathbb{N}} \forall_{P:\forall_{n:\mathbb{N}} (\mathsf{Vec}\,\alpha\,n) \to \mathbb{P}} \forall_{z:P\,0\,(\mathsf{nil}\,\alpha)} \forall_{c:\forall_{v:\mathsf{Vec}\,\alpha\,n} \forall_{x:\alpha} P\,n\,v \to P\,((\mathsf{suc}\,n))\,(\mathsf{cons}\,\alpha\,n\,v\,x)} \forall_{v:\mathsf{Vec}\,\alpha\,n} P\ n\ v)$

The translation of this signature, $[\![\mathcal{T}]\!]_{\text{sig}}$, is the following DHOL theory:

$\mathbb{N} : \mathsf{type}$, $0 : \mathbb{N}$, $\mathsf{suc} : \Pi_{n:\mathbb{N}} \mathbb{N}$, $\mathbb{N}\mathsf{Rec} \rhd \forall_{P:\mathbb{N}\to o} P\ 0 \Rightarrow (\forall_{n:\mathbb{N}} P\ n \Rightarrow P\ (\mathsf{suc}\ n)) \Rightarrow \forall_{n:\mathbb{N}} P\ n$,
$\mathsf{Vec} : \Pi_{\alpha:\mathsf{type},\, n:\mathbb{N}} : \mathsf{type}$, $\mathsf{nil} : \Pi_{\alpha:\mathsf{type}} \mathsf{Vec}\ \alpha\ 0$, $\mathsf{cons} : \Pi_{\alpha:\mathsf{type},\, n:\mathbb{N},\, v:\mathsf{Vec}\,\alpha\,n,\, x:\alpha} \mathsf{Vec}\ \alpha\ (\mathsf{suc}\ n)$,
$\mathsf{VecRec} \rhd \Pi_{\alpha:\mathsf{type},\, n:\mathbb{N},\, P:\Pi_{n:\mathbb{N}} (\mathsf{Vec}\,\alpha\,n) \to o}$
$P\ 0\ (\mathsf{nil}\ \alpha) \Rightarrow (\forall_{v:\mathsf{Vec}\,\alpha\,n} \forall_{x:\alpha} P\ n\ v \Rightarrow P\,(\mathsf{suc}\,n)\,(\mathsf{cons}\ \alpha\ n\ v\ x)) \Rightarrow \forall_{v:\mathsf{Vec}\,\alpha\,n} P\ n\ v$

We can state, the congruence of cons in DTT3:

$$\alpha : \mathsf{U}_1 \vdash_{\mathcal{T}} \forall_{x:\alpha} \forall_{y:\alpha} \forall_{n:\mathbb{N}} \forall_{v:\mathsf{Vec}\,\alpha\,n} \forall_{p:x=_\alpha y} \mathsf{cons}\,\alpha\,n\,v\,x =_{\mathsf{Vec}\,\alpha\,(\mathsf{suc}\,n)} \mathsf{cons}\,\alpha\,n\,v\,y : \mathbb{P}$$

We can then translate the context, the signature, the proposition and its type, using $[\![\cdot]\!]_{\text{app}}$ for the last two, as follows:

$$\alpha : \mathsf{type} \vdash_{[\![\mathcal{T}]\!]_{\text{sig}}} \forall_{x:\alpha}\forall_{y:\alpha}\forall_{n:\mathbb{N}}\forall_{v:\mathsf{Vec}\,\alpha\,n} x =_{\alpha} y \Rightarrow \mathsf{cons}\;\alpha\;n\;v\;x =_{\mathsf{Vec}\,\alpha\,(\mathsf{suc}\,n)} \mathsf{cons}\;\alpha\;n\;v\;y : o$$

Notably, the translation maps the $\mathbb{P}$-valued dependent function types of DTT3 that quantify over non-propositions to the $\forall$ quantifier in DHOL, which is defined in terms of equality and implication, instead of the DHOL dependent function type. Furthermore, since the last dependent function quantifies over a proposition, DHOL's dependent implication is used.

Example 4.7. This example demonstrates how casts are erased and how this interacts with provability. We reuse the length indexed vectors from above and further assume an axiomatisation of $+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$, which we write infix as usual, as well as of

$$\mathsf{append} : \forall_{\alpha:\mathsf{U}_1}\forall_{m:\mathbb{N}}\forall_{n:\mathbb{N}}\forall_{v:\mathsf{Vec}\,\alpha\,m}\forall_{w:\mathsf{Vec}\,\alpha\,n}\mathsf{Vec}\;\alpha\;(m+n)$$

in $\mathcal{T}$. In DTT3 the following statement is ill-typed:

$$\alpha : \mathsf{U}_1 \vdash_{\mathcal{T}} \forall_{m:\mathbb{N}}\forall_{n:\mathbb{N}}\forall_{v:\mathsf{Vec}\,\alpha\,m}\forall_{w:\mathsf{Vec}\,\alpha\,n}\mathsf{append}\;\alpha\;m\;n\;v\;w =_{\mathsf{Vec}\,\alpha\,(m+n)} \mathsf{append}\;\alpha\;n\;m\;w\;v : \mathbb{P}$$

This is because $m + n$ and $n + m$ are propositionally equal but not definitionally equal. Therefore the types $\mathsf{Vec}\;\alpha\;(m + n)$ and $\mathsf{Vec}\;\alpha\;(n + m)$ are also not definitionally equal, which is required to even express propositional equality of the two vectors. If, however, we replace the right hand side of the equation by $\alpha : \mathsf{U}_1 \vdash_{\mathcal{T}} \mathsf{cast}\;p\;(\mathsf{append}\;\alpha\;n\;m\;w\;v)$, where p is an easy-to-construct proof term of type $\mathsf{Vec}\;\alpha\;(n + m) =_{\mathsf{U}_1} \mathsf{Vec}\;\alpha\;(m + n)$, this type-checks.

By contrast, the translated version of the above statement type-checks without issue, since DHOL is extensional. Specifically, while type checking, the rule TpEqSym would be used, where one of the typing conditions would be that $\Gamma \vdash_{[\![\mathcal{T}]\!]_{\text{sig}}} m + n =_{\mathbb{N}} n + m$, which, given a sufficient axiomatisation of $+$, is provable. For this reason, the translation erases all occurrences of cast.

5 Soundness

A key theoretical property of our translation is that it reflects judgements: if some DHOL terms, types, theories or contexts satisfy a DHOL judgement, then there exist corresponding translatable DTT3 expressions, signatures and contexts of which the DHOL counterparts are translations and for which a corresponding DTT3 judgement holds. This entails soundness in the following sense: if a translated DTT3 proposition is provable in DHOL, the original proposition is provable in DTT3 (i.e. a proof term exists).

To prove reflection of judgements, we first show a certain kind of injectivity of our translation. We need to do this to identify DTT3 constructs that according to the induction hypothesis translate to the same corresponding DHOL construct but originate from different applications of the induction hypothesis.

Lemma 5.1 (Injectivity). *The translation functions $[\![\cdot]\!]_{\text{ty}}$, $[\![\cdot]\!]_{\text{ter}}$, $[\![\cdot]\!]_{\text{app}}$, $[\![\cdot]\!]_{\text{exp}}$, $[\![\cdot]\!]_{\text{sig}}$ and $[\![\cdot]\!]_{\text{ctx}}$ are all injective on the translatable fragment of DTT3 in the following sense:*

- *For translatable signatures $\mathcal{T}$ and $\mathcal{T}'$, if $[\![\mathcal{T}]\!]_{\text{sig}} = [\![\mathcal{T}']\!]_{\text{sig}}$ then $\mathcal{T} \equiv \mathcal{T}'$.*
- *For translatable contexts Γ and Γ', if $[\![\Gamma]\!]_{\text{ctx}} = [\![\Gamma']\!]_{\text{ctx}}$ then $\Gamma \equiv \Gamma'$.*
- *For translatable expressions e and e', that have definitionally equal types, if $[\![e]\!]_{\text{app}} = [\![e']\!]_{\text{app}}$, $[\![e]\!]_{\text{exp}} = [\![e']\!]_{\text{exp}}$, $[\![e]\!]_{\text{ter}} = [\![e']\!]_{\text{ter}}$ or $[\![e]\!]_{\text{ty}} = [\![e']\!]_{\text{ty}}$ then $e \equiv e'$.*

where additionally for all expressions, every sub-expression that is not a cast*, is the argument of a (possibly trivial, i.e. along* refl*)* cast*. For DTT3 we use definitional equality $\equiv$; for DHOL we use syntactic equality $=$, where we do not unfold defined connectives.*

Proof. We prove injectivity by mutual structural induction on the inputs, separating the cases by function. Since the DTT3 expressions have casts in front of every sub-expression, which are stripped by the translation functions, two such DTT3 expressions that translate to the same DHOL type or term are definitionally equal if the remaining non-cast components are, since by assumption the expressions with casts have the same type. For this the proofs which are the first argument of the casts are always definitionally equal, since either are proofs of the same proposition, so PI applies.

$[\![\cdot]\!]_{\text{ty}}$: Let α, β be translatable DTT3 expressions of type $\forall_{\overrightarrow{x:\alpha}} \mathsf{U}_1$. We show that if $[\![\alpha]\!]_{\text{ty}} = [\![\beta]\!]_{\text{ty}}$ then $\alpha \equiv \beta$:

If $\alpha = x$ or $\alpha = c$, the translation function is the identity, therefore the conclusion follows immediately.

If $\alpha = \mathbb{P}$, then $[\![\alpha]\!]_{\text{ty}} = o = [\![\beta]\!]_{\text{ty}}$. This immediately implies that $\beta = \mathbb{P}$, since $\mathbb{P}$ by induction is the only pre-image of o under $[\![\cdot]\!]_{\text{ty}}$.

If $\alpha = \alpha_1\ \alpha_2$, we have $[\![\alpha]\!]_{\text{ty}} = [\![\alpha]\!]_{\text{app}} = [\![\beta]\!]_{\text{ty}}$, which again by induction has only one pre-image and we conclude $\alpha \equiv \beta$ using the induction hypothesis regarding $[\![\cdot]\!]_{\text{app}}$.

If $\alpha = \forall_{x:\alpha_1} \alpha_2$, then $[\![\alpha]\!]_{\text{ty}} = \Pi_{x:[\![\alpha_1]\!]_{\text{ty0}}} [\![\alpha_2]\!]_{\text{ty0}} = [\![\beta]\!]_{\text{ty}}$. For the casts removed in $[\![\cdot]\!]_{\text{ty0}}$ the argument above applies, since by assumption α and β must have the same type. This is the case as well in all following occurrences of $[\![\cdot]\!]_{\text{ty0}}$ and also $[\![\cdot]\!]_{\text{ter0}}$. Again by induction this only has one pre-image, therefore $\beta = \forall_{x:\alpha_1} \alpha_2$.

There are no translatable expressions of the form $\lambda_{x:\alpha} e$ in the domain of $[\![\cdot]\!]_{\text{ty}}$, since they would need to be of type $\forall_{x:\alpha} \mathsf{U}_1$, which again is of type U_2, which violates condition (6)(c) in Def 4.2.

Expressions of the form $\alpha_1 =_{\alpha_2} \alpha_3$ are by definition of type $\mathbb{P}$ and can therefore not occur here.

$[\![\cdot]\!]_{\text{ter}}$: Let e, d be translatable DTT3 expressions of type $\forall_{\overrightarrow{x:\beta}} \alpha$, where $\alpha : \mathsf{U}_1$. We show that if $[\![e]\!]_{\text{ter}} = [\![d]\!]_{\text{ter}}$ then $e \equiv d$:

Cases $e = x, e = c, e = e_1\ e_2$ are completely analogous to $[\![\cdot]\!]_{\text{ty}}$.

If $e = \exists_{x:\alpha} e_1$, then $[\![e]\!]_{\text{ter}} = \exists_{x:[\![\alpha]\!]_{\text{ty0}}} [\![e_1]\!]_{\text{ter0}}$. We use the induction hypothesis of the mutual induction regarding $[\![\cdot]\!]_{\text{ty}}$, which gets us that $[\![d]\!]_{\text{ter}} = \exists_{x:[\![\alpha]\!]_{\text{ty0}}} [\![e_1]\!]_{\text{ter0}}$. We conclude $e \equiv d$ from the definition of $[\![\cdot]\!]_{\text{ter}}$.

If $e = \forall_{x:\alpha} e_1$, we need to perform a case distinction on whether $\alpha : \mathbb{P}$ or not. If $\alpha : \mathbb{P}$ we have $[\![e]\!]_{\text{ter}} = ([\![\alpha]\!]_{\text{ter0}} \Rightarrow [\![e_1]\!]_{\text{ter0}})$. We use the induction hypothesis, which gets us that $[\![d]\!]_{\text{ter0}} = [\![\alpha]\!]_{\text{ter}} \Rightarrow [\![e_1]\!]_{\text{ter0}}$. We conclude $e \equiv d$ from the definition of $[\![\cdot]\!]_{\text{ter}}$. If not, we have $[\![e]\!]_{\text{ter}} = \forall_{x:[\![\alpha]\!]_{\text{ty0}}} [\![e_1]\!]_{\text{ter0}}$ and the rest is analogous to the $\exists$ case.

For $e = (e_1 =_\alpha e_2)$ and $e = \lambda_{x:\alpha} e_1$ we employ the induction hypothesis of the mutual induction similarly.

Finally, $e = e_1 \wedge e_2, e = e_1 \vee e_2, e = \neg e_1$ all follow analogously to $e = (e_1 =_\alpha e_2)$.

$[\![\cdot]\!]_{\text{app}}$: In the case of e being of the form cast $p\ e_1$, or not being an application, this immediately follows from the injectivity of $[\![\cdot]\!]_{\text{ty}}$ and $[\![\cdot]\!]_{\text{ter}}$ and the argument laid out at the beginning of the proof. Otherwise, if e is an application without top-level cast, we invoke the induction hypothesis directly.

$[\![\cdot]\!]_{\text{exp}}$: Analogous to $[\![\cdot]\!]_{\text{app}}$, but for the $\checkmark$ case we use the induction hypotheses again, as well as PI.

$[\![\cdot]\!]_{\text{sig}}$: Let $\mathcal{T}_1, \mathcal{T}_2$ be well formed DTT3 signatures. We show that if $[\![\mathcal{T}_1]\!]_{\text{sig}} = [\![\mathcal{T}_2]\!]_{\text{sig}}$ then $\mathcal{T}_1 \equiv \mathcal{T}_2$:

If $\mathcal{T}_1 = .$, then $[\![\mathcal{T}_1]\!]_{\text{sig}} = . = [\![\mathcal{T}_2]\!]_{\text{sig}}$ and trivially $\mathcal{T}_1 = \circ$.

If $\mathcal{T}_1 = \mathcal{T}_1', (c, \forall_{\overrightarrow{x:\mathsf{U}_1}^n} e)$ we need to do a case distinction on e. First if $e : \mathbb{P}$, then $[\![\mathcal{T}_1]\!]_{\text{sig}} = [\![\mathcal{T}_1']\!]_{\text{sig}}, c \rhd \Pi_{\overrightarrow{x}^n} [\![e]\!]_{\text{ter}}$. Using the induction hypothesis regarding $[\![\cdot]\!]_{\text{ter}}$, we get that this also equals

$[\![\mathcal{T}_2]\!]_{\text{sig}}$. From the definition of $[\![\cdot]\!]_{\text{sig}}$, since c does not have the val subscript, by induction there is only one possible pre-image, so we conclude that $\mathcal{T}_1 \equiv \mathcal{T}_2$.

Next, if $e : \mathsf{U}_1$, then $[\![\mathcal{T}_1]\!]_{\text{sig}} = [\![\mathcal{T}_1']\!]_{\text{sig}}, c : \Pi_{\overrightarrow{x:n}}[\![e]\!]_{\text{ty}}$. Analogously to the previous case we use the induction hypothesis, regarding $[\![\cdot]\!]_{\text{ty}}$ this time, and then by induction there is only one pre-image, meaning $\mathcal{T}_1 \equiv \mathcal{T}_2$.

Finally, if e had neither type U_1 nor type $\mathbb{P}$, in our fragment it must have the form $\forall_{\overrightarrow{x:\alpha}^m}\mathsf{U}_1$, in which case $[\![\mathcal{T}_1]\!]_{\text{sig}} = [\![\mathcal{T}_1']\!]_{\text{sig}}, c : \Pi_{\overrightarrow{x:n}}\Pi_{\overrightarrow{x:[\![\alpha]\!]_{\text{ty}}}^m} : \text{type}$. Using the induction hypothesis regarding $[\![\cdot]\!]_{\text{ty}}$ again and the definition of $[\![\cdot]\!]_{\text{sig}}$, we again conclude $\mathcal{T}_1 \equiv \mathcal{T}_2$.

If $\mathcal{T}_1 = \mathcal{T}_1', (c, \alpha, e)$ then $[\![\mathcal{T}_1]\!]_{\text{sig}} = [\![\mathcal{T}_1', (c, \alpha)]\!]_{\text{sig}}, c_{\text{val}} \rhd c =_{[\![\alpha]\!]_{\text{ty}}} [\![e]\!]_{\text{ter}}$. Using the induction hypotheses regarding $[\![\cdot]\!]_{\text{ty}}$ and $[\![\cdot]\!]_{\text{ter}}$, we get that this also equals $[\![\mathcal{T}_2]\!]_{\text{sig}}$. From the definition of $[\![\cdot]\!]_{\text{sig}}$, since c does have the val subscript, by induction there is only one possible pre-image, so we conclude that $\mathcal{T}_1 \equiv \mathcal{T}_2$.

$[\![\cdot]\!]_{\text{ctx}}$: Let Γ, Δ be well formed DTT3 contexts. We show that if $[\![\Gamma]\!]_{\text{ctx}} = [\![\Delta]\!]_{\text{ctx}}$ then $\Gamma \equiv \Delta$:

If $\Gamma = \circ$ we are again done analogously to before.

If $\Gamma = \Gamma', x : \mathsf{U}_1$, then $[\![\Gamma]\!]_{\text{ctx}} = ([\![\Gamma']\!]_{\text{ctx}}, x : \text{type}) = [\![\Delta]\!]_{\text{ctx}}$. There is exactly one pre-image by induction and we conclude $\Gamma \equiv \Delta$.

If $\Gamma = \Gamma', x : \mathbb{P}$, then $[\![\Gamma]\!]_{\text{ctx}} = ([\![\Gamma']\!]_{\text{ctx}}, x : o) = [\![\Delta]\!]_{\text{ctx}}$. Again from the definition of $[\![\cdot]\!]_{\text{ctx}}$ we conclude $\Gamma \equiv \Delta$.

If $\Gamma = \Gamma', x : e$, where e is neither $\mathbb{P}$ nor U_1, we need to do a case distinction on the type of e. It must either be $\mathbb{P}$ or U_1.

If $e : \mathbb{P}$, then $[\![\Gamma]\!]_{\text{ctx}} = [\![\Gamma']\!]_{\text{ctx}}, x \rhd [\![e]\!]_{\text{ter}}$. Using the mutual induction hypothesis regarding $[\![\cdot]\!]_{\text{ter}}$ we get that it is also equal to $[\![\Delta]\!]_{\text{ctx}}$. With the definition of $[\![\cdot]\!]_{\text{ctx}}$ we again conclude $\Gamma \equiv \Delta$.

If finally $e : \mathsf{U}_1$, since $e \neq \mathbb{P}$, $[\![\Gamma]\!]_{\text{ctx}} = [\![\Gamma']\!]_{\text{ctx}}, x : [\![e]\!]_{\text{ty}}$. Analogous to the previous case, using the induction hypothesis regarding $[\![\cdot]\!]_{\text{ty}}$ and the definition of $[\![\cdot]\!]_{\text{ctx}}$ we conclude $\Gamma \equiv \Delta$. □

We also need to prove that $[\![\cdot]\!]_{\text{exp}}$ is compatible with substitution, meaning that substituting in the DTT3 expression and then translating leads to the same result as translating the expression that is substituted into as well as the expressions used in the substitution individually first, and performing the DHOL substitution after.

Lemma 5.2 (Substitution). *Let t be a translatable DTT3 expression, and let $\overrightarrow{e}$ be a list of translatable DTT3 expressions. Then $[\![t[\overrightarrow{e}]]\!]_{\text{app}} = [\![t]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]$.*

Proof. Since casts are erased and do not matter for substitution, we will disregard them here and always write $[\![\cdot]\!]_{\text{ter}}$ and $[\![\cdot]\!]_{\text{ty}}$ instead of $[\![\cdot]\!]_{\text{ter0}}$ and $[\![\cdot]\!]_{\text{ty0}}$.

The proof is by induction on t:

$t = x_i$: Since x_i is translatable, x_i is not a proof term and therefore the translation of x_i is $[\![x_i]\!]_{\text{app}} = x_i = [\![x_i]\!]_{\text{exp}}$. Similarly we also know that $[\![e_i]\!]_{\text{app}} = [\![e_i]\!]_{\text{exp}}$, since e_i can also not be a proof term in this case. Therefore $[\![x_i[\overrightarrow{e}]]\!]_{\text{app}} = [\![e_i]\!]_{\text{app}} = [\![e_i]\!]_{\text{exp}} = x_i[\overrightarrow{[\![e]\!]_{\text{exp}}}]\!]_{\text{exp}}] = [\![x_i]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]_{\text{exp}}] = [\![x_i]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]$.

$t = \mathbb{P}$: Then $[\![t]\!]_{\text{app}} = o = [\![t]\!]_{\text{exp}}$ and $t[\overrightarrow{e}] = t$ for any e, meaning $[\![t[\overrightarrow{e}]]\!]_{\text{app}} = [\![t]\!]_{\text{app}} = o = o[\overrightarrow{[\![e]\!]_{\text{exp}}}] = [\![t]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]$.

$t = c$: Analogous to previous case.

$t = u_1\, u_2$: If $t = c\,\overrightarrow{u}$, then we assume $c \neq$ cast as mentioned above. Then $[\![t]\!]_{\text{app}} = c\,\overrightarrow{[\![u]\!]_{\text{exp}}} = [\![t]\!]_{\text{exp}}$. The induction hypothesis does not immediately apply for the $[\![u_i]\!]_{\text{exp}}$, so we do a case analysis on u_i,

that will proceed exactly like the current one if u_i is not a proof term. If u_i however is a proof term of proposition α, $[\![u_i]\!]_{\text{exp}} = \checkmark_{[\![\alpha]\!]_{\text{ter}}}$. Regarding the substitution on these proof terms then, if u_i is a variable x_j, then $x_j[\overrightarrow{e}] = e_j$, where e_j is also a proof term of the proposition α. Therefore we also have that $[\![e_j]\!]_{\text{exp}} = \checkmark_{[\![\alpha]\!]_{\text{ter}}}$, so in every case we get $[\![u_i[\overrightarrow{e}]]\!]_{\text{exp}} = [\![u_i]\!]_{\text{exp}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]$, which then for t gives us finally $[\![t[\overrightarrow{e}]]\!]_{\text{app}} = [\![c[\overrightarrow{e}]\ \overrightarrow{u[\overrightarrow{e}]}]\!]_{\text{app}} = [\![c]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]\ \overrightarrow{[\![u]\!]_{\text{exp}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]} = [\![c\ \overrightarrow{[\![u]\!]_{\text{exp}}}]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}] = [\![t]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]$. If $t = u_1\ u_2$, where $u_1 \neq c\ \overrightarrow{u}$, then $[\![t]\!]_{\text{app}} = [\![u_1]\!]_{\text{app}}\ [\![u_2]\!]_{\text{app}} = [\![t]\!]_{\text{exp}}$ and also $t[\overrightarrow{e}] = u_1[\overrightarrow{e}]\ u_2[\overrightarrow{e}]$. So we get $[\![t[\overrightarrow{e}]]\!]_{\text{app}} = [\![u_1[\overrightarrow{e}]]\!]_{\text{app}}\ [\![u_2[\overrightarrow{e}]]\!]_{\text{app}} \overset{I.H.}{=} [\![u_1]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]\ [\![u_2]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}] = [\![t]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]$.

$t = \lambda_{x:\alpha} u$: From the translatability of t, we can infer that $[\![t]\!]_{\text{app}} = [\![t]\!]_{\text{ter}} = [\![t]\!]_{\text{exp}}$ and analogously for α and u. Further $t[\overrightarrow{e}] = \lambda_{x:\alpha[\overrightarrow{e}]} u[\overrightarrow{e}]$. So we get $[\![t[\overrightarrow{e}]]\!]_{\text{ter}} = \lambda_{x:[\![\alpha[\overrightarrow{e}]]\!]_{\text{ty}}} [\![u[\overrightarrow{e}]]\!]_{\text{ter}} = \lambda_{x:[\![\alpha[\overrightarrow{e}]]\!]_{\text{app}}} [\![u[\overrightarrow{e}]]\!]_{\text{app}} \overset{I.H.}{=} \lambda_{x:[\![\alpha]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]} [\![u]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}] = \lambda_{x:[\![\alpha]\!]_{\text{ty}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]} [\![u]\!]_{\text{ter}}[\overrightarrow{[\![e]\!]_{\text{exp}}}] = [\![t]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]$. Since α must be of type U_1, due to otherwise either condition (5)(c) from Def. 4.2 being violated or $\forall_{x:\alpha}\beta : \mathsf{U}_2$, where $u : \beta$, which would violate condition (5)(c), it is equivalent to use $[\![\cdot]\!]_{\text{ty}}$ instead of $[\![\cdot]\!]_{\text{app}}$.

$t = \forall_{x:\alpha} u$: There are two cases, $t : \mathbb{P}$ and $t : \mathsf{U}_1$. If $t : \mathbb{P}$, then if $\alpha : \mathsf{U}_1$, we have $[\![t]\!]_{\text{app}} = [\![t]\!]_{\text{ter}} = \forall_{x:[\![\alpha]\!]_{\text{ty}}} [\![u]\!]_{\text{ter}} = [\![t]\!]_{\text{exp}}$ where $u : \mathbb{P}$. If $\alpha : \mathbb{P}$, then $[\![t]\!]_{\text{app}} = [\![t]\!]_{\text{ter}} = [\![\alpha]\!]_{\text{ter}} \Rightarrow [\![u]\!]_{\text{ter}} = [\![t]\!]_{\text{exp}}$ where again $u : \mathbb{P}$. Also $t[\overrightarrow{e}] = \forall_{x:\alpha[\overrightarrow{e}]} u[\overrightarrow{e}]$. So we get $[\![t[\overrightarrow{e}]]\!]_{\text{app}} = [\![\forall_{x:\alpha[\overrightarrow{e}]} u[\overrightarrow{e}]]\!]_{\text{app}} = \forall_{x:[\![\alpha[\overrightarrow{e}]]\!]_{\text{ty}}} [\![u[\overrightarrow{e}]]\!]_{\text{ter}} \overset{I.H.}{=} \forall_{x:[\![\alpha]\!]_{\text{ty}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]} [\![u]\!]_{\text{ter}}[\overrightarrow{[\![e]\!]_{\text{exp}}}] = [\![t]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]$. If $t : \mathsf{U}_1$, then $[\![t]\!]_{\text{app}} = [\![t]\!]_{\text{ty}} = \Pi_{x:[\![\alpha]\!]_{\text{ty}}} [\![u]\!]_{\text{ty}} = [\![t]\!]_{\text{exp}}$ with $\alpha : \mathsf{U}_1$ and $u : \mathsf{U}_1$. Also $t[\overrightarrow{e}] = \Pi_{x:\alpha[\overrightarrow{e}]} u[\overrightarrow{e}]$. So we get $[\![t[\overrightarrow{e}]]\!]_{\text{app}} = [\![\Pi_{x:\alpha[\overrightarrow{e}]} u[\overrightarrow{e}]]\!]_{\text{app}} = \Pi_{x:[\![\alpha[\overrightarrow{e}]]\!]_{\text{ty}}} [\![u[\overrightarrow{e}]]\!]_{\text{ty}} \overset{I.H.}{=} \Pi_{x:[\![\alpha]\!]_{\text{ty}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]} [\![u]\!]_{\text{ty}}[\overrightarrow{[\![e]\!]_{\text{exp}}}] = [\![t]\!]_{\text{app}}[\overrightarrow{[\![e]\!]_{\text{exp}}}]$

The cases for $e_1 =_\alpha e_2$ and the other connectives follow analogously. □

Next we state the reflection of judgements theorem. In general, we try to match the judgements of DHOL with those of DTT3. Notable differences concern the cases for provability of a formula F and for the well-formedness of a substitution δ. For the former, we require the existence of a term of the corresponding proposition $[\![F]\!]_{\text{ter}}$ in DTT3. For the latter, since DTT3 does not have explicit substitutions, we spell out the condition each of the components of δ needs to fulfil individually. This works since δ can simply be viewed as the list $\overrightarrow{\delta}^n$ of terms, types and $\checkmark_F$'s for some DHOL context Δ with corresponding types, type variables and local assumptions $\overrightarrow{\Delta}^n$ for some n.

Theorem 5.3 (Reflection of Judgements). *Let Γ, Δ be a DHOL context, A, B be DHOL types, t be a DHOL term, F be a DHOL formula, δ be a DHOL substitution and T be a DHOL theory. In Table 2, if the left hand side holds, then there exist translatable DTT3 contexts Γ', Δ', translatable DTT3 expressions $e, \alpha, \beta, \delta'_i$ and a translatable DTT3 signature $\mathcal{T}$ such that the right hand side also holds.*

We actually prove a slightly stronger version, where all DTT3 expressions have casts in front of every sub-expression. This is necessary to fulfil the requirements of Lemma 5.1. The stronger condition can easily be fulfilled by adding casts that use a proof of reflexivity in front of sub-expressions that do not have casts. We omit trivial casts for readability.

Another observation that is necessary to invoke Lemma 5.1 is that, whenever we translate an expression that gets translated to a DHOL term, we also translate the type of that expression to a DHOL type. For these type like expressions, since they are translatable, they can only have the type U_ℓ, and ℓ must always be the same across different uses of the induction hypothesis regarding

Table 2. Correspondence between DHOL and DTT3 judgements

DHOL	*DTT3*
$\Gamma \vdash^{\mathrm{DHOL}}_{T} F$	*There exists* p *such that* $\Gamma' \vdash^{\mathrm{DTT3}}_{\mathcal{T}} p : e$ *and* $[\![\Gamma']\!]^{\mathcal{T}}_{\mathrm{ctx}} = \Gamma, [\![e]\!]^{\mathcal{T};\Gamma'}_{\mathrm{ter}} = F, [\![\mathcal{T}]\!]_{\mathrm{sig}} = T$
$\Gamma \vdash^{\mathrm{DHOL}}_{T} t : A$	$\Gamma' \vdash^{\mathrm{DTT3}}_{\mathcal{T}} e : \alpha$ *and* $[\![\Gamma']\!]^{\mathcal{T}}_{\mathrm{ctx}} = \Gamma, [\![e]\!]^{\mathcal{T};\Gamma'}_{\mathrm{ter}} = t, [\![\alpha]\!]^{\mathcal{T};\Gamma'}_{\mathrm{ty}} = A, [\![\mathcal{T}]\!]_{\mathrm{sig}} = T$
$\vdash^{\mathrm{DHOL}} T\ \mathsf{Thy}$	$\vdash^{\mathrm{DTT3}} \mathcal{T}\ \mathsf{sig}$ *and* $[\![\mathcal{T}]\!]_{\mathrm{sig}} = T$
$\vdash^{\mathrm{DHOL}}_{T} \Gamma\ \mathsf{Ctx}$	$\vdash^{\mathrm{DTT3}}_{\mathcal{T}} \Gamma'\ \mathsf{ok}$ *and* $[\![\Gamma']\!]^{\mathcal{T}}_{\mathrm{ctx}} = \Gamma, [\![\mathcal{T}]\!]_{\mathrm{sig}} = T$
$\Gamma \vdash^{\mathrm{DHOL}}_{T} \Delta \leftarrow \delta$	$\Gamma' \vdash^{\mathrm{DTT3}}_{\mathcal{T}} \delta'_i : \alpha[\overrightarrow{\delta'}^{i-1}]$ *when* $\Delta'_i = x : \alpha[\overrightarrow{\delta'}^{i-1}]$ *for* $1 \le i \le \lvert\delta\rvert$ *and* $[\![\Gamma']\!]^{\mathcal{T}}_{\mathrm{ctx}} = \Gamma, [\![\Delta']\!]^{\mathcal{T}}_{\mathrm{ctx}} = \Delta, [\![\delta'_i]\!]^{\mathcal{T};\Gamma'}_{\mathrm{exp}} = \delta_i, [\![\mathcal{T}]\!]_{\mathrm{sig}} = T$
$\Gamma \vdash^{\mathrm{DHOL}}_{T} A : \mathsf{type}$	$\Gamma' \vdash^{\mathrm{DTT3}}_{\mathcal{T}} \alpha : \mathsf{U}_1$ *and* $[\![\Gamma']\!]^{\mathcal{T}}_{\mathrm{ctx}} = \Gamma, [\![\alpha]\!]^{\mathcal{T};\Gamma'}_{\mathrm{ty}} = A, [\![\mathcal{T}]\!]_{\mathrm{sig}} = T$
$\Gamma \vdash^{\mathrm{DHOL}}_{T} A \equiv B$	*There exists* p *such that* $\Gamma' \vdash^{\mathrm{DTT3}}_{\mathcal{T}} p : \alpha =_{\mathsf{U}_1} \beta$ *and* $[\![\Gamma']\!]^{\mathcal{T}}_{\mathrm{ctx}} = \Gamma, [\![\alpha]\!]^{\mathcal{T};\Gamma'}_{\mathrm{ty}} = A, [\![\beta]\!]^{\mathcal{T};\Gamma'}_{\mathrm{ty}} = B, [\![\mathcal{T}]\!]_{\mathrm{sig}} = T$

the same DHOL type. Therefore for the expressions that translate to DHOL types, we fulfil the condition of Lemma 5.1, which then gives us this same condition for the expressions that are of this type and get translated to DHOL terms.

Since Lemma 5.1 only provides definitional equality of DTT3 constructs, we need to further invoke the compatibility of DTT3 judgements with definitional equality, i.e. subject reduction, meaning if some DTT3 judgement holds and there exist definitionally equal constructs, the same judgement also holds for these. This holds in the generalisation of DTT3 given by Carneiro [8], so it must hold for DTT3 as well.

For example, regarding the typing judgement we have

$$\frac{\Gamma \vdash_{\mathcal{T}} e \equiv e' \qquad \Gamma \vdash_{\mathcal{T}} \alpha \equiv \alpha' \qquad \vdash \mathcal{T} \equiv \mathcal{T}' \qquad \vdash_{\mathcal{T}} \Gamma \equiv \Gamma' \qquad \vdash \mathcal{T}\ \mathsf{sig} \qquad \vdash_{\mathcal{T}} \Gamma\ \mathsf{ok} \qquad \Gamma \vdash_{\mathcal{T}} \alpha\ \mathsf{type} \qquad \Gamma \vdash_{\mathcal{T}} e : \alpha}{\Gamma' \vdash_{\mathcal{T}'} e' : \alpha'}$$

Similar statements for all other judgements hold.

Proof. The proof is by mutual induction over the DHOL proof of our assumption. We will first present some illustrative cases with further comments and then give the remaining cases.

We begin with a very detailed case that also illustrates the correspondence between DHOL theories and DTT3 signatures.

Case ThyTp: The assumptions of the rule are

$$(1) \vdash^{\mathrm{DHOL}} T\ \mathsf{Thy}, (2)\ a \notin T, (3) \vdash^{\mathrm{DHOL}}_{T} \Delta\ \mathsf{Ctx}$$

The conclusion is (4) $\vdash^{\mathrm{DHOL}} T, a : \Pi_\Delta : \mathsf{type}\ \mathsf{Thy}$. The induction hypothesis allows us to use the assumptions (1) and (3) to derive that there exists a DTT3 signature $\mathcal{T}$ and a DTT3 context Δ', which have casts in front of every sub-expression, such that (1′) $\vdash^{\mathrm{DTT3}} \mathcal{T}\ \mathsf{sig}$ where $[\![\mathcal{T}]\!]_{\mathrm{sig}} = T$ and (3′) $\vdash^{\mathrm{DTT3}}_{\mathcal{T}} \Delta'\ \mathsf{ok}$ where $[\![\mathcal{T}]\!]_{\mathrm{sig}} = T$, $[\![\Delta']\!]_{\mathrm{ctx}} = \Delta$. Note that we implicitly use the injectivity of the translation function stated in Lemma 5.1 to equate the $\mathcal{T}$ in (1′) and (3′). To prove this case of the theorem, we now need to show given (1′), (2), (3′), (4) that there exist a DTT3 signature $\mathcal{T}'$ such that if (4) holds, it holds that (4′) $\vdash^{\mathrm{DTT3}} \mathcal{T}'\ \mathsf{sig}$ with $[\![\mathcal{T}']\!]_{\mathrm{sig}} = T, a : \Pi_\Delta : \mathsf{type}$. We

therefore assume (4) and show (4′). We choose $\mathcal{T}' = \mathcal{T}, (a, \forall_{\Delta'} \mathsf{U}_1)$, which translates to the theory $[\![\mathcal{T}]\!]_{\text{sig}}, a : \Pi_{[\![\Delta']\!]_{\text{ctx}}} : \text{type}$, which is equal to $T, a : \Pi_\Delta : \text{type}$ due to the equalities associated with (1′) and (3′). It remains to prove $\vdash^{\text{DTT3}} \mathcal{T}, (a, \forall_{\Delta'} \mathsf{U}_1)$ sig. For this it suffices to show that $\vdash^{\text{DTT3}}_{\mathcal{T}} \forall_{\Delta'} \mathsf{U}_1$ type. To show our goal we use ∀Form. It therefore suffices, for $\Delta' = \overrightarrow{x} : \overrightarrow{\alpha}$, that $\overrightarrow{x}^{i-1} : \overrightarrow{\alpha}^{i-1} \vdash^{\text{DTT3}}_{\mathcal{T}} \alpha_i$ type for all α_i. This then follows from (3′), which concludes the case.

In the following cases we will only state the results of applying the induction hypothesis to the DHOL pre-conditions (here, (1′), (3′)), possible leftover assumptions (here, (2)), the collection of equalities associated with these and state directly what we must prove (here, (4′)), without necessarily mentioning the DHOL conclusion directly.

The next case demonstrates our handling of substitutions in DTT3, where we lack explicit substitutions. It also shows how proof terms of propositions of DTT3 are translated to the ✓ substitution component.

Case SubAsn: From the induction hypothesis, we get (1) $\Gamma' \vdash_{\mathcal{T}} \delta'_i : \alpha[\overrightarrow{\delta'}^{i-1}]$ when $\Delta'_i = x : \alpha[\overrightarrow{\delta'}^{i-1}]$ and (2) $\Delta' \vdash_{\mathcal{T}} e : \beta$ and there exists p such that (3) $\Gamma' \vdash_{\mathcal{T}} p : e'$ with $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\Delta']\!]_{\text{ctx}} = \Delta, [\![\delta'_i]\!]_{\text{exp}} = \delta_i, [\![\beta]\!]_{\text{ty}} = o, [\![e]\!]_{\text{ter}} = F, [\![e']\!]_{\text{ter}} = F[\delta]$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We need to show that $\Gamma' \vdash_{\mathcal{T}} \delta''_i : \beta[\overrightarrow{\delta''}^{i-1}]$ when $\Delta''_i = x : \beta[\overrightarrow{\delta''}^{i-1}]$ where $[\![\Delta'']\!]_{\text{ctx}} = \Delta, \alpha : \text{type}$ and $[\![\delta''_i]\!]_{\text{exp}} = \delta_i$ for all $1 \leq i \leq |\delta|$ and $[\![\delta''_{|\delta|+1}]\!]_{\text{exp}} = p$. We choose $\delta''_i = \delta'_i$ for $1 \leq i \leq |\delta|$ and $\delta''_{|\delta|+1} = p$ and $\Delta'' = \Delta', x : e$. $[\![\delta''_i]\!]_{\text{exp}} = \delta_i$, for $1 \leq i \leq |\delta|$, and $[\![\delta_{|\delta|+1}]\!]_{\text{exp}} = \checkmark_{[\![e']\!]_{\text{ter}}} = \checkmark_e$. Further $[\![\Delta'']\!]_{\text{ctx}} = [\![\Delta']\!]_{\text{ctx}}, x \rhd e = \Delta, x \rhd e$. For all $1 \leq i \leq |\delta|$, our goal is equal to (1). For $i = |\delta| + 1$ we need to show $\Gamma' \vdash_{\mathcal{T}} p : e[\overrightarrow{\delta''}^{(|\delta|+1)-1}]$. This is equivalent to $\Gamma' \vdash_{\mathcal{T}} p : e[\delta']$. Using the induction hypotheses and Lemma 5.2 this then follows immediately.

The next case concerns DHOL's dependent implication. It illustrates why it is important that we chose dependent functions $\forall_{x:\alpha}\beta$ where both α and β are of type $\mathbb{P}$ to be translated to dependent implications, instead of translating the non-dependent function type $\alpha \to \beta$ to dependent implication, which is usually seen as the corresponding DTT construct to logical implication.

Case TermImpl: From the induction hypothesis, we have (1) $\Gamma' \vdash_{\mathcal{T}} e : \alpha$ and (2) $\Gamma'' \vdash_{\mathcal{T}} u : \alpha$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\Gamma'']\!]_{\text{ctx}} = \Gamma, x \rhd F, [\![e]\!]_{\text{ter}} = F, [\![u]\!]_{\text{ter}} = G, [\![\alpha]\!]_{\text{ty}} = o$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $\Gamma'' = \Gamma', x : e$ and $\alpha = \mathbb{P}$. We need to show that $\Gamma' \vdash_{\mathcal{T}} t : \beta$ where $[\![t]\!]_{\text{ter}} = F \Rightarrow G$ and $[\![\beta]\!]_{\text{ty}} = o$. We choose $t = \forall_{x:e} u$ and $\beta = \alpha$. $[\![t]\!]_{\text{ter}}$ and $[\![\beta]\!]_{\text{ty}}$ compute as desired. We prove $\Gamma' \vdash_{\mathcal{T}} \forall_{x:e} u : \mathbb{P}$. We use ∀Form, meaning it suffices to show $\Gamma' \vdash_{\mathcal{T}} e$ type and $\Gamma', x : e \vdash_{\mathcal{T}} u : \mathbb{P}$ due to the definition of imax. Using (1) and (2) respectively with the fact that $\alpha = \mathbb{P}$ these follow immediately.

In the final case shown here, the addition of casts is needed. This is because even though in DHOL, all equal terms and types are convertible to one another, in DTT3 only definitionally equal expressions are. The corresponding construct in DTT3 to DHOL's equality, however, is propositional equality and only definitionally equal expressions are convertible. So some equalities might not type-check in DTT3 even though their translation does in DHOL. However, if we amend the casts at the corresponding sub-expressions, we can create a definitionally equal expression that does type-check in DTT3. This is another reason why we require all DTT3 expressions to have a cast at each sub-expression, even if they do not always change the type.

Case CongApply: From the induction hypothesis, we get that there exists p_1 such that $\Gamma' \vdash_{\mathcal{T}} p_1 : e$ and that there exists p such that $\Gamma \vdash_{\mathcal{T}} p_2 : e'$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![e]\!]_{\text{ter}} = t =_A t', [\![e']\!]_{\text{ter}} = f =_{\Pi_{x:A} B} f'$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $e = (u =_\alpha u')$

and $e' = (g =_{\forall_{x:\alpha}\beta} g')$ where $[\![u]\!]_{\text{ter}} = t, [\![u']\!]_{\text{ter}} = t', [\![g]\!]_{\text{ter}} = f, [\![g']\!]_{\text{ter}} = f', [\![\alpha]\!]_{\text{ty}} = A$ and $[\![\beta]\!]_{\text{ty}} = B$. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : e''$ and $[\![e'']\!]_{\text{ter}} = (f\ t =_{B[t]} f'\ t')$. We choose $e'' = (g\ u =_{\beta[u]} \text{cast}\ p_1'\ (g'\ u'))$. The cast added around the right hand side of the equation is needed to make this statement type-check. The proof term $p_1' : \beta[u] = \beta[u']$ can easily be derived using p_1 and the rules for equality. Then $[\![e'']\!]_{\text{ter}}$ computes to $f\ t =_{B[t]} f'\ t'$, as desired, using Lemma 5.2. Then using induction twice yields the desired p.

The remaining cases follow similar patterns.

Case ThyEmpty: Follows trivially from the first case of the sig judgement.

Case ThyCon: From the induction hypothesis, we get that (1) $\vdash \mathcal{T}$ sig and (2)$\Delta' \vdash_{\mathcal{T}} \beta : \mathsf{U}_1$ with $[\![\mathcal{T}]\!]_{\text{sig}} = T, [\![\Delta']\!]_{\text{ctx}} = \Delta$ and $[\![\beta]\!]_{\text{ty}} = A$. We further get the assumption that $c \notin T$. We need to show that $\vdash \mathcal{T}'$ sig, where $[\![\mathcal{T}']\!]_{\text{sig}} = T, c : \Pi_\Delta A$. We choose $\mathcal{T}' = \mathcal{T}, (c, \forall_{\Delta'}\beta)$. $[\![\mathcal{T}']\!]_{\text{sig}}$ computes as desired. It suffices to show $\vdash_{\mathcal{T}} \forall_{\Delta'}\beta$ type. We use $\forall$Form, so we need to show $\Delta' \vdash_{\mathcal{T}} \beta$ type, which follows from (2), and $\overrightarrow{x}^{i-1} : \overrightarrow{\alpha}^{i-1} \vdash_{\mathcal{T}} \alpha_i$ type for $\Delta' = \overrightarrow{x} : \overrightarrow{\alpha}$. From (2), we can infer that $\vdash_{\mathcal{T}} \Delta'$ ok and with this the desired result follows immediately.

Case ThyAsn: From the induction hypothesis, we get that (1) $\vdash \mathcal{T}$ sig and (2)$\Delta' \vdash_{\mathcal{T}} e : \beta$ with $[\![\mathcal{T}]\!]_{\text{sig}} = T, [\![\Delta']\!]_{\text{ctx}} = \Delta, [\![e]\!]_{\text{ter}} = F$ and $[\![\beta]\!]_{\text{ty}} = o$. We further get the assumption that $c \notin T$. We need to show that $\vdash \mathcal{T}'$ sig, where $[\![\mathcal{T}']\!]_{\text{sig}} = T, c \rhd \Pi_\Delta F$. We choose $\mathcal{T}' = \mathcal{T}, (c, \forall_{\Delta'} e)$. $[\![\mathcal{T}']\!]_{\text{sig}}$ computes as desired. The rest of the proof is analogous to the previous case, apart from showing $\Delta \vdash_{\mathcal{T}} e$ type, which follows from (2) and from $\beta = \mathbb{P}$, due to $[\![\beta]\!]_{\text{ty}} = o$ and the definition of the translation function.

Case CtxEmpty: Follows trivially from the first case of the ok judgement.

Case CtxTp: From the induction hypothesis, we get $\vdash_{\mathcal{T}} \Delta'$ ok with $[\![\Delta']\!]_{\text{ctx}} = \Delta$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We need to show that $\vdash_{\mathcal{T}} \Delta''$ ok with $[\![\Delta'']\!]_{\text{ctx}} = \Delta, \alpha : \text{type}$. We choose $\Delta'' = \Delta', \alpha : \mathsf{U}_1$. $[\![\Delta'']\!]_{\text{ctx}}$ computes as desired. We then only need to show that $\vdash_{\mathcal{T}} \mathsf{U}_1$ type. This is then immediate.

Case CtxVar: From the induction hypothesis, we get $\vdash_{\mathcal{T}} \Delta'$ ok and $\Delta' \vdash_{\mathcal{T}} \alpha : \mathsf{U}_1$ with $[\![\Delta']\!]_{\text{ctx}} = \Delta, [\![\alpha]\!]_{\text{ty}} = A$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We need to show that $\vdash_{\mathcal{T}} \Delta''$ ok with $[\![\Delta'']\!]_{\text{ctx}} = \Delta, x : A$. We choose $\Delta'' = \Delta', x : \alpha$. $[\![\Delta'']\!]_{\text{ctx}}$ computes as desired. Thus we need to show $\vdash_{\mathcal{T}} \Delta'$ ok and $\Delta' \vdash_{\mathcal{T}} \alpha$ type, which follows immediately from our assumptions.

Case CtxAsn: From the induction hypothesis, we get $\vdash_{\mathcal{T}} \Delta'$ ok and $\Delta' \vdash_{\mathcal{T}} e : \alpha$ with $[\![\Delta']\!]_{\text{ctx}} = \Delta, [\![e]\!]_{\text{ter}} = F, [\![\alpha]\!]_{\text{ty}} = o$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We need to show that $\vdash_{\mathcal{T}} \Delta''$ ok with $[\![\Delta'']\!]_{\text{ctx}} = \Delta', x \rhd F$. We choose $\Delta'' = \Delta', x : e$. $[\![\Delta'']\!]_{\text{ctx}}$ computes as desired. Thus we need to show $\vdash_{\mathcal{T}} \Delta'$ ok and $\Delta' \vdash_{\mathcal{T}} e$ type. The former we already have as an assumption, the latter holds due to $[\![\alpha]\!]_{\text{ty}} = o$, which means that $\alpha = \mathbb{P}$.

Case SubEmpty: Trivial.

Case SubTp: From the induction hypothesis, we get (1)$\Gamma' \vdash_{\mathcal{T}} \delta_i' : \beta[\overrightarrow{\delta'}^{i-1}]$ when $\Delta_i' = x : \beta[\overrightarrow{\delta'}^{i-1}]$ and (2)$\Gamma' \vdash_{\mathcal{T}} \gamma : \mathsf{U}_1$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\Delta']\!]_{\text{ctx}} = \Delta, [\![\delta_i']\!]_{\text{exp}} = \delta_i, [\![\gamma]\!]_{\text{ty}} = A$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We need to show that $\Gamma' \vdash_{\mathcal{T}} \delta_i'' : \beta[\overrightarrow{\delta''}^{i-1}]$ when $\Delta_i'' = x : \beta[\overrightarrow{\delta''}^{i-1}]$ where $[\![\Delta'']\!]_{\text{ctx}} = \Delta, \alpha : \text{type}$ and $[\![\delta_i'']\!]_{\text{exp}} = \delta_i$ for all $1 \le i \le |\delta|$ and $[\![\delta''_{|\delta|+1}]\!]_{\text{exp}} = A$. We choose $\delta_i'' = \delta_i'$ for $1 \le i \le |\delta|$ and $\delta''_{|\delta|+1} = \gamma$ and $\Delta'' = \Delta', \alpha : \mathsf{U}_1$. $[\![\delta_i'']\!]_{\text{exp}}$ and $[\![\Delta'']\!]_{\text{ctx}}$ compute as desired. For all $1 \le i \le |\delta|$, our goal is equal to (1). For $i = |\delta| + 1$, we need to show $\Gamma' \vdash_{\mathcal{T}} \gamma : \mathsf{U}_1$, which is exactly (2).

Case SubVar: From the induction hypothesis, we get (1)$\Gamma' \vdash_{\mathcal{T}} \delta'_i : \alpha[\overrightarrow{\delta'}^{i-1}]$ when $\Delta'_i = x : \alpha[\overrightarrow{\delta'}^{i-1}]$ and (2)$\Delta' \vdash_{\mathcal{T}} \beta : \mathsf{U}_1$ and (3)$\Gamma' \vdash_{\mathcal{T}} e : \gamma$ with $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![\Delta']\!]_{\text{ctx}} = \Delta$, $[\![\delta'_i]\!]_{\text{exp}} = \delta_i$, $[\![\beta]\!]_{\text{ty}} = A$, $[\![\gamma]\!]_{\text{ty}} = A[\delta]$, $[\![e]\!]_{\text{ter}} = t$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. The goal and choice is analogous to the previous case, with the exception of $\delta''_{|\delta|+1} = e$ and $\Delta'' = \Delta', x : \beta$. $[\![\delta''_i]\!]_{\text{exp}}$ and $[\![\Delta'']\!]_{\text{ctx}}$ compute as desired. For all $1 \leq i \leq |\delta|$, our goal is equal to (1). For $i = |\delta| + 1$, we need to show that $\Gamma' \vdash_{\mathcal{T}} e : \beta[\overrightarrow{\delta''}^{(|\delta|+1)-1}]$, which is equal to $\Gamma' \vdash_{\mathcal{T}} e : \beta[\delta']$. Using the induction hypotheses and Lemma 5.2 this then follows immediately.

Case TpSym: From the induction hypothesis, we get $\vdash_{\mathcal{T}} \Gamma'$ ok and (2)$\Gamma' \vdash_{\mathcal{T}} \delta'_i : \alpha[\overrightarrow{\delta'}^{i-1}]$ when $\Delta'_i = x : \alpha[\overrightarrow{\delta'}^{i-1}]$ with $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![\Delta']\!]_{\text{ctx}} = \Delta$, $[\![\delta'_i]\!]_{\text{exp}} = \delta_i$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We also get the assumption that (3) $a : \Pi_\Delta : \text{type}$ is in T. We need to show that $\Gamma' \vdash_{\mathcal{T}} \beta : \mathsf{U}_1$ where $[\![\beta]\!]_{\text{ty}} = a\ \delta$. We choose $\beta = a\ \delta'$. $[\![\beta]\!]_{\text{ty}}$ computes as desired. From the definition of the translation function and (3), we know that $(a, \forall_\Delta \mathsf{U}_1) \in \mathcal{T}$. This, in combination with (2), then yields the desired result using Apply and ConstTy.

Case TermSym: From the induction hypothesis, we get $\vdash_{\mathcal{T}} \Gamma'$ ok and (2)$\Gamma' \vdash_{\mathcal{T}} \delta'_i : \alpha[\overrightarrow{\delta'}^{i-1}]$ when $\Delta'_i = x : \alpha[\overrightarrow{\delta'}^{i-1}]$ with $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![\Delta']\!]_{\text{ctx}} = \Delta$, $[\![\delta'_i]\!]_{\text{exp}} = \delta_i$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We also get the assumption that (3) $c : \Pi_\Delta B$ is in T. We need to show that $\Gamma' \vdash_{\mathcal{T}} e : \beta$ where $[\![e]\!]_{\text{ter}} = c\ \delta$ and $[\![\beta]\!]_{\text{ty}} = B[\delta]$. From the definition of the translation function and (3), we know that $(c, \forall_\Delta \beta') \in \mathcal{T}$ where $[\![\beta']\!]_{\text{ty}} = B$. We therefore choose $e = c\ \delta'$ and $\beta = \beta'[\delta']$. $[\![\beta]\!]_{\text{ty}}$ and $[\![e]\!]_{\text{ter}}$ compute as desired using Lemma 5.2. The desired result follows with (2) using Apply and ConstTy.

Case ValidSym: From the induction hypothesis, we get $\vdash_{\mathcal{T}} \Gamma'$ ok and (2)$\Gamma' \vdash_{\mathcal{T}} \delta'_i : \alpha[\overrightarrow{\delta'}^{i-1}]$ when $\Delta'_i = x : \alpha[\overrightarrow{\delta'}^{i-1}]$ with $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![\Delta']\!]_{\text{ctx}} = \Delta$, $[\![\delta'_i]\!]_{\text{exp}} = \delta_i$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We also get the assumption that (3) $c \triangleright \Pi_\Delta F$ is in T. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : e$ and $[\![e]\!]_{\text{ter}} = F[\delta]$. From the definition of the translation function and (3), we know that $(c, \forall_\Delta e') \in \mathcal{T}$ where $[\![e']\!]_{\text{ter}} = F$. We therefore choose $e = e'[\delta']$ and $p = c\ \delta'$. $[\![e]\!]_{\text{ter}}$ computes as desired using Lemma 5.2. The desired result follows with (2) using Apply and ConstTy.

Case TpVar: From the induction hypothesis, we have $\vdash_{\mathcal{T}} \Gamma'$ ok where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We also get the assumption that (2) $\alpha : \text{type} \in \Gamma$. From this and the definition of the translation function, we therefore know that (3) $\alpha : \mathsf{U}_1 \in \Gamma'$. We need to show that $\Gamma' \vdash_{\mathcal{T}} \alpha : \mathsf{U}_1$. This follows from the Assume rule and (3).

Case TermVar: Analogous to TpVar.

Case ValidVar: From the induction hypothesis, we have $\vdash_{\mathcal{T}} \Gamma'$ ok where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We also get the assumption that (2) $x \triangleright F \in \Gamma$. From this and the definition of the translation function we therefore know that (3) $x : e \in \Gamma'$ where $[\![e]\!]_{\text{ter}} = F$. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : e'$ and $[\![e']\!]_{\text{ter}} = F$. We choose $e' = e$ and $p = x$. The result then follows from the Assume rule and (3).

Case TpBool: Follows trivially from the Univ rule.

Case TermEq: From the induction hypothesis, we get that $\Gamma' \vdash_{\mathcal{T}} t' : \alpha$ and $\Gamma' \vdash_{\mathcal{T}} u' : \alpha$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![t']\!]_{\text{ter}} = t$, $[\![u']\!]_{\text{ter}} = u$, $[\![\alpha]\!]_{\text{ty}} = A$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We need to show that $\Gamma' \vdash_{\mathcal{T}} e : \beta$ where $[\![e]\!]_{\text{ter}} = (t =_A u)$ and $[\![\beta]\!]_{\text{ty}} = o$. We choose $e = (t' =_\alpha u')$ and $\beta = \mathbb{P}$. $[\![e]\!]_{\text{ter}}$ and $[\![\beta]\!]_{\text{ty}}$ compute as desired. The result then immediately follows from the induction hypotheses and EqForm.

Case TpPi: From the induction hypothesis, we have $\Gamma' \vdash_{\mathcal{T}} \alpha : \mathsf{U}_1$ and $\Gamma'' \vdash_{\mathcal{T}} \beta : \mathsf{U}_1$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\Gamma'']\!]_{\text{ctx}} = \Gamma, x : A, [\![\alpha]\!]_{\text{ty}} = A, [\![\beta]\!]_{\text{ty}} = B$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $\Gamma'' = \Gamma', x : \alpha$. We need to show that $\Gamma' \vdash_{\mathcal{T}} \gamma : \mathsf{U}_1$ where $[\![\gamma]\!]_{\text{ty}} = \Pi_{x:A} B$. We choose $\gamma = \forall_{x:\alpha} \beta$. $[\![\gamma]\!]_{\text{ty}}$ computes as desired. Using ∀Form, it suffices to show that $\Gamma' \vdash_{\mathcal{T}} \alpha : \mathsf{U}_1$ and $\Gamma', x : \alpha \vdash_{\mathcal{T}} \beta : \mathsf{U}_1$, which is exactly the induction hypotheses.

Case TermLambda: From the induction hypothesis, we have $\Gamma' \vdash_{\mathcal{T}} \alpha : \mathsf{U}_1$ and (2)$\Gamma'' \vdash_{\mathcal{T}} t' : \beta$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\Gamma'']\!]_{\text{ctx}} = \Gamma, x : A, [\![\alpha]\!]_{\text{ty}} = A, [\![\beta]\!]_{\text{ty}} = B, [\![t']\!]_{\text{ter}} = t$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $\Gamma'' = \Gamma', x : \alpha$. We need to show that $\Gamma' \vdash_{\mathcal{T}} e : \gamma$ where $[\![e]\!]_{\text{ter}} = \lambda_{x:A} t$ and $[\![\gamma]\!]_{\text{ty}} = \Pi_{x:A} B$. We choose $e = \lambda_{x:\alpha} t'$ and $\gamma = \forall_{x:\alpha} \beta$. Using λForm we need to show $\Gamma', x : \alpha \vdash_{\mathcal{T}} t' : \beta$, which is exactly (2).

Case TermApply: From the induction hypothesis, we have $\Gamma' \vdash_{\mathcal{T}} t' : \gamma$ and $\Gamma' \vdash_{\mathcal{T}} u' : \alpha$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\alpha]\!]_{\text{ty}} = A, [\![\gamma]\!]_{\text{ty}} = \Pi_{x:A} B, [\![t']\!]_{\text{ter}} = t, [\![u']\!]_{\text{ter}} = u$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we also know that $\gamma = \forall_{x:\alpha} \beta$, where $[\![\beta]\!]_{\text{ty}} = B$. We need to show that $\Gamma' \vdash_{\mathcal{T}} e : \beta'$ where $[\![e]\!]_{\text{ter}} = t\ u, [\![\beta']\!]_{\text{ty}} = B[u]$. We choose $e = t'\ u'$ and $\beta' = \beta[u']$. $[\![\beta']\!]_{\text{ty}}$ and $[\![e]\!]_{\text{ter}}$ compute as desired using Lemma 5.2. Using Apply, it suffices to show $\Gamma' \vdash_{\mathcal{T}} t' : \forall_{x:\alpha} \beta$ and $\Gamma' \vdash_{\mathcal{T}} u' : \alpha$, which are exactly the induction hypotheses.

Case TpEqVar: Follows trivially from EqIntro.

Case TpEqSym: From the induction hypothesis, we have $\vdash_{\mathcal{T}} \Gamma'$ ok and that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : \gamma_i =_{\Delta_i'[\overrightarrow{\gamma}^{i-1}]} \gamma_i'$ since $\equiv_\Delta$ means the component-wise equality of the terms/types, where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\Delta']\!]_{\text{ctx}} = \Delta, [\![\gamma_i]\!]_{\text{exp}} = \delta_i, [\![\gamma_i']\!]_{\text{exp}} = \delta_i'$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We also get the assumption that (3)$a : \Pi_\Delta : \text{type}$ is in T. From the definition of the translation function we then know that $(a, \forall_\Delta' \mathsf{U}_1) \in \mathcal{T}$. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p' : \alpha =_{\mathsf{U}_1} \beta$ and $[\![\alpha]\!]_{\text{ty}} = a\ \delta$ and $[\![\beta]\!]_{\text{ty}} = a\ \delta'$. We choose $\alpha = a\ \gamma, \beta = a\ \gamma'$. $[\![\alpha]\!]_{\text{ty}}$ and $[\![\beta]\!]_{\text{ty}}$ compute as desired. Using the congruence rules about equality derivable in DTT3 and the induction hypotheses we can construct p'.

Case TpEqBool: Follows trivially from the introduction rule of equality.

Case TpEqPi: From the induction hypothesis, we have that there exists p_1 such that $\Gamma' \vdash_{\mathcal{T}} p_1 : \alpha =_{\mathsf{U}_1} \alpha'$ and that there exists p_2 such that (2)$\Gamma'' \vdash_{\mathcal{T}} p_2 : \beta =_{\mathsf{U}_1} \beta'$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\Gamma'']\!]_{\text{ctx}} = \Gamma, x : A, [\![\alpha]\!]_{\text{ty}} = A, [\![\alpha']\!]_{\text{ty}} = A', [\![\beta]\!]_{\text{ty}} = B, [\![\beta]\!]_{\text{ty}} = B'$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $\Gamma'' = \Gamma', x : \alpha$. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : \gamma =_{\mathsf{U}_1} \gamma'$ and $[\![\gamma]\!]_{\text{ty}} = \Pi_{x:A} B$ and $[\![\gamma']\!]_{\text{ty}} = \Pi_{x:A'} B$. We choose $\gamma = \forall_{x:\alpha} \beta$ and $\gamma' = \forall_{x:\alpha'} \beta'$. After this everything type-checks. $[\![\gamma]\!]_{\text{ty}}$ and $[\![\gamma']\!]_{\text{ty}}$ compute as desired. Using the common derivable rules for equality in DTT3 and the induction hypotheses we can construct p.

Case TermConvert: From the induction hypothesis, we have $\Gamma' \vdash_{\mathcal{T}} t' : \alpha$ and that there exists p such that (2)$\Gamma' \vdash_{\mathcal{T}} p : \alpha =_{\mathsf{U}_1} \alpha'$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\Delta']\!]_{\text{ctx}} = \Delta, [\![\alpha]\!]_{\text{ty}} = A, [\![\alpha']\!]_{\text{ty}} = A', [\![t']\!]_{\text{ter}} = t$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. We need to show that $\Gamma' \vdash_{\mathcal{T}} e : \beta$ where $[\![e]\!]_{\text{ter}} = t$ and $[\![\beta]\!]_{\text{ty}} = A'$. We choose $e = t''$ and $\beta = \alpha'$. t'' is t' where the top-level cast is replaced by one that uses p. $[\![e]\!]_{\text{ter}}$ and $[\![\beta]\!]_{\text{ty}}$ compute as desired. Due to the replacement of the cast our goal follows immediately.

Case TermBeta: From the induction hypothesis, we get $\Gamma' \vdash_{\mathcal{T}} e : \gamma$ and $\Gamma' \vdash_{\mathcal{T}} u' : \alpha$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, [\![\gamma]\!]_{\text{ty}} = \Pi_{x:A} B, [\![\alpha]\!]_{\text{ty}} = A, [\![e]\!]_{\text{ter}} = \lambda_{x:A} t, [\![u']\!]_{\text{ter}} = u$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $\gamma = \forall_{x:\alpha} \beta$ where $[\![\beta]\!]_{\text{ty}} = B$ and $e = \lambda_{x:\alpha} t'$ where $[\![t']\!]_{\text{ter}} = t$. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : e'$ and $[\![e']\!]_{\text{ter}} = (\lambda_{x:A} t)\ u =_{B[u]} t[u]$. We choose $e' = (\lambda_{x:\alpha} t')\ u' =_{\beta[u']} t'[u']$ and $p = refl_{t'[u']}$. $[\![e']\!]_{\text{ter}}$ computes as desired using Lemma 5.2. This holds definitionally in DTT3 so it is proved by $refl$.

Case TermEta: From the induction hypothesis, we get $\Gamma' \vdash_{\mathcal{T}} t' : \gamma$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![\gamma]\!]_{\text{ty}} = \Pi_{x:A}B$, $[\![t']\!]_{\text{ter}} = t$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $\gamma = \forall_{x:\alpha}\beta$ where $[\![\alpha]\!]_{\text{ty}} = A$ and $[\![\beta]\!]_{\text{ty}} = B$. We additionally get the assumption that x does not occur in Γ. It therefore can also not occur in Γ' or we can perform α-renaming to achieve this. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : e$ and $[\![e]\!]_{\text{ter}} = t =_{\Pi_{x:A}B} \lambda_{x:A} t\ x$. We choose $e = t' =_{\gamma} \lambda_{x:\alpha} t'\ x$ and $p = refl_{t'}$. $[\![e]\!]_{\text{ter}}$ computes as desired. Again this equation holds definitionally in DTT3 so it is proved by $refl$.

Case CongLambda: From the induction hypothesis, we get that there exists p_1 such that $\Gamma' \vdash_{\mathcal{T}} p_1 : \alpha =_{\mathcal{U}_1} \alpha'$ and that there exists p_2 such that $\Gamma'' \vdash_{\mathcal{T}} p_2 : e$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![\Gamma'']\!]_{\text{ctx}} = \Gamma, x : A$, $[\![\alpha]\!]_{\text{ty}} = A$, $[\![\alpha']\!]_{\text{ty}} = A'$, $[\![e]\!]_{\text{ter}} = t =_B t'$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $e = u =_{\beta} u'$ and $\Gamma'' = \Gamma', x : \alpha$ where $[\![u]\!]_{\text{ter}} = t$, $[\![u']\!]_{\text{ter}} = t'$ and $[\![\beta]\!]_{\text{ty}} = B$. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : e'$ and $[\![e']\!]_{\text{ter}} = \lambda_{x:A} t =_{\Pi_{x:A}B} \lambda_{x:A'} t'$. We choose $e' = \lambda_{x:\alpha} u =_{\forall_{x:\alpha}\beta} \lambda_{x:\alpha'} u'$. $[\![e]\!]_{\text{ter}}$ computes as desired. To make this type-check, all occurrences of x in u' must have their casts amended by p_1. Then using induction twice yields the desired p.

Case Refl: Follows trivially from the introduction rule of equality.

Case Sym: From the induction hypothesis, we get there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : e$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![e]\!]_{\text{ter}} = t =_A s$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $e = t' =_{\alpha} s'$ with $[\![t']\!]_{\text{ter}} = t$, $[\![s']\!]_{\text{ter}} = s$ and $[\![\alpha]\!]_{\text{ty}} = A$ We need to show that there exists p' such that $\Gamma' \vdash_{\mathcal{T}} p' : e'$ and $[\![e']\!]_{\text{ter}} = s =_A t$. We choose $e' = t' =_{\alpha} s'$. $[\![e']\!]_{\text{ter}}$ computes as desired. Using induction on p we can construct the desired p'.

Case CongValid: From the induction hypothesis, we get that there exists p_1 such that $\Gamma' \vdash_{\mathcal{T}} p_1 : e$ and that there exists p_2 such that $\Gamma' \vdash_{\mathcal{T}} p_2 : e'$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![e]\!]_{\text{ter}} = F =_o F'$, $[\![e']\!]_{\text{ter}} = F'$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $e = t =_{\mathbb{P}} e'$ where $[\![t]\!]_{\text{ter}} = F$. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : t$. We can construct p by changing the cast around p_2 to one that casts along p_1.

Case ImplIntro: From the induction hypothesis, we get that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : e$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma, x \triangleright F$, $[\![e]\!]_{\text{ter}} = G$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $\Gamma' = \Gamma'', x : e'$ where $[\![\Gamma'']\!]_{\text{ctx}} = \Gamma$ and $[\![e']\!]_{\text{ter}} = F$. We need to show that there exists p' such that $\Gamma'' \vdash_{\mathcal{T}} p' : u$ and $[\![u]\!]_{\text{ter}} = F \Rightarrow G$. We choose $u = \forall_{x:e'} e$ and $p' = \lambda_{x:e'} p$. $[\![u]\!]_{\text{ter}}$ computes as desired. Using λForm with the induction hypothesis immediately yields the result.

Case MP:. From the induction hypothesis, we get that there exists p_1 such that $\Gamma' \vdash_{\mathcal{T}} p_1 : e$ and that there exists p_2 such that $\Gamma' \vdash_{\mathcal{T}} p_2 : e'$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![e]\!]_{\text{ter}} = F \Rightarrow G$, $[\![e']\!]_{\text{ter}} = F$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $e = \forall_{x:e'} e''$ where $[\![e'']\!]_{\text{ter}} = G$. We need to show that there exists p such that $\Gamma' \vdash_{\mathcal{T}} p : e''$. We choose $p = p_1\ p_2$. Using Apply with the induction hypotheses immediately yields the result.

Case BoolExt: From the induction hypothesis, we get $\Gamma' \vdash_{\mathcal{T}} t' : \alpha$, that there exists p_1 such that $\Gamma' \vdash_{\mathcal{T}} p_1 : e$ and that there exists p_2 such that $\Gamma' \vdash_{\mathcal{T}} p_2 : e'$ where $[\![\Gamma']\!]_{\text{ctx}} = \Gamma$, $[\![t']\!]_{\text{ter}} = t$, $[\![e]\!]_{\text{ter}} = t\ \bot$, $[\![e']\!]_{\text{ter}} = t\ \top$, $[\![\alpha]\!]_{\text{ty}} = \Pi_{x:o} o$ and $[\![\mathcal{T}]\!]_{\text{sig}} = T$. From the definition of the translation function we further know that $\alpha = \forall_{x:\mathbb{P}}\mathbb{P}$, that $e = t'\ \bot$ and that $e' = t'\ \top$. As stated earlier for this case we assume axiomatisations of a constant $\vdash_{\mathcal{T}}$ lem : $\forall_{e:\mathbb{P}} e \vee (\neg e)$, a constant $\vdash_{\mathcal{T}}$ propExt : $\forall_{e_1:\mathbb{P}}\forall_{e_2:\mathbb{P}}((e_1 \to e_2) \wedge (e_2 \to e_1)) \to e_1 =_{\mathbb{P}} e_2$ the empty type $\bot$, the singleton type $\top$, the product $\wedge$, co-product $\vee$ and negation $\neg$. We need to prove that there exists p such that $\Gamma'' \vdash_{\mathcal{T}} p : e''$, where $[\![\Gamma'']\!]_{\text{ctx}} = \Gamma, x : o$, $[\![e'']\!]_{\text{ter}} = t\ x$. We choose $\Gamma'' = \Gamma', x : \mathbb{P}$, $e'' = t'\ x$. Then $[\![\Gamma'']\!]_{\text{ctx}}$ and $[\![e'']\!]_{\text{ter}}$

compute as desired. To acquire p, we use the induction principle associated with $\vee$ on $\mathsf{lem}\ x$ with e and e', which yields the desired result. □

We do not prove the cases regarding $\top$, $\bot$, $\wedge$, $\vee$, $\neg$ and $\exists$ since they depend on some design decisions when axiomatising their counterparts in DTT. For commonly used definitions, however, these easily follow in the same style as the other cases.

From Theorem 5.3, we easily derive soundness:

Theorem 5.4 (Soundness). *Let $\mathcal{T}$ be a translatable DTT3 signature, let Γ be a translatable DTT3 context, let e be a translatable DTT3 expression such that $\Gamma \vdash_{\mathcal{T}} e : \mathbb{P}$. If $[\![\Gamma]\!]_{\mathrm{ctx}} \vdash^{\mathrm{DHOL}}_{[\![\mathcal{T}]\!]_{\mathrm{sig}}} [\![e]\!]_{\mathrm{ter}}$, then there exists p such that $\Gamma \vdash^{\mathrm{DTT3}}_{\mathcal{T}} p : e$.*

Proof. Consequence of the first non-header row of Table 2 and injectivity together with subject reduction. □

6 Evaluation

To gauge the practical relevance of our translatable DTT3 fragment, we have implemented a *fragment checker*, which can be used to determine how many of the theorems in Lean's Mathlib library [19] fall within our fragment. For this purpose, we have adapted the setup used to evaluate Lean's Aesop tactic [14] to check all human written theorems regarding their translatability. Since universe polymorphism is heavily used in Mathlib, but DTT3 does not support it, the checker instantiates all universe variables with 0. This means that it actually checks whether an instance of the theorem is translatable. It additionally unfolds occurrences of `outParam`, a construct that marks the output parameter of type classes, while being definitionally equal to the marked type. If it were to not unfold this, many functions from type classes, such as

$$\begin{aligned}\texttt{HAdd.hAdd} : (\alpha : \texttt{Type}\ u) \to (\beta : \texttt{Type}\ v) \to (\gamma : \texttt{outParam}\ (\texttt{Type}\ w))\\ \to [self : \texttt{HAdd}\ \alpha\ \beta\ \gamma] \to \alpha \to \beta \to \gamma,\end{aligned}$$

where u, v, w are instantiated with 0, would be considered untranslatable, since `outParam (Type 0)` is of type `Type 1`, which corresponds to U_2 of DTT3, and the only translatable such type in signatures is U_1 or `Type 0` in Lean. But since `outParam (Type 0)` is defined to be exactly `Type 0`, if we unfold this definition, it becomes translatable. There are more meta-operations we could use to increase the amount of translatable theorems, which we do not consider here.

We ran our evaluation on a machine using an Intel® Xeon® Silver 4114 CPU with 40 cores running at 2.20 GHz and with 187 GiB of RAM, which took about five hours. We evaluated Mathlib version 4.27.0-rc1, which contains 222 747 human written theorems. Of these, 69 265 theorems have statements that fall outside the translatable fragment. Another 50 727 theorems have statements that are translatable, but use untranslatable constants, which would need to be present in the signature. The most common dependencies that caused this were `CategoryTheory.Category` and its recursor, appearing in 30.1 % of these theorems. Finally 102 756 theorems are fully translatable, meaning their statement is translatable and all used constants have definitions that are signature-translatable. The complete numbers for all modules can be found in Table 3. The fragment checker's source code and the raw evaluation data are available in the artefact.

We have also calculated the weighted variance (WVAR) and the weighted standard deviation (WSD) for every top level module, where the weights are the number of theorems in the sub-modules. These are always only in respect to the direct sub-modules e.g. for `Algebra`, `Algebra.Group` was considered as a whole, not all sub-modules of `Algebra.Group` individually. If a module has no sub-modules or only exactly one sub-module the standard deviation is 0. The standard deviation is the range around the ratio in which most sub-modules, weighted by number of theorems, perform.

Table 3. Translatability of Mathlib modules

	Theorems				
Module	Total	Translatable	Ratio (%)	WSD (%)	WVAR
Algebra	40 153	21 718	54.1	29.3	0.086
CategoryTheory	25 115	320	1.3	2.6	0.001
Analysis	22 515	12 508	55.6	25.9	0.067
Data	20 721	16 348	78.9	19.9	0.040
Topology	20 334	11 652	57.3	28.2	0.080
Order	14 775	10 434	70.6	26.8	0.072
RingTheory	13 951	5 390	38.6	26.5	0.070
MeasureTheory	10 073	1 986	19.7	16.5	0.027
LinearAlgebra	9 306	2 856	30.7	23.6	0.056
GroupTheory	6 228	3 993	64.1	19.6	0.039
Combinatorics	5 606	4 891	87.2	4.4	0.002
NumberTheory	5 264	3 115	59.2	24.4	0.059
Geometry	4 270	414	9.7	12.5	0.016
AlgebraicGeometry	3 849	876	22.8	35.0	0.123
SetTheory	3 641	236	6.5	8.5	0.007
Probability	3 461	649	18.8	19.0	0.036
AlgebraicTopology	2 245	295	13.1	24.4	0.060
FieldTheory	2 213	411	18.6	21.7	0.047
Logic	2 054	1 098	53.5	21.5	0.046
Tactic	1 364	1 220	89.4	35.2	0.124
Computability	1 178	922	78.3	0	0
ModelTheory	1 132	110	9.7	14.9	0.022
RepresentationTheory	1 053	36	3.4	0	0
Dynamics	731	430	58.8	41.4	0.171
Control	259	8	3.1	0	0
Condensed	192	0	0.0	0	0
InformationTheory	99	23	23.2	0	0
Lean	17	17	100.0	0	0
Deprecated	16	12	75.0	0	0
Testing	7	5	71.4	0	0
Util	3	3	100.0	0	0
All of Mathlib	222 747	102 756	46.1	25.1	0.063

For example, the numbers for all of Mathlib in the last row mean that the modules that perform between 21.0 % and 71.2 % contain a majority of all theorems. The variance is the square of the standard deviation; higher numbers mean more variance.

Unsurprisingly, the vast majority of theorems concerning category theory are not translatable, since they are set up making heavy use of higher universe levels and higher rank polymorphism. This includes not only the `CategoryTheory` module, where 1.3 % of theorems are translatable, but also other sub-modules that concern category theory, such as `Algebra.Category`, containing 2261 where 1.4 % are translatable. This is mainly since, the type class `CategoryTheory.Category` is not

signature-translatable, due to it having type $(\texttt{obj} : \texttt{Type}) \to \texttt{Type}\ 1$ after instantiating all universe variables, so no theorem referencing this definition can be translated.

On the other hand, packages such as `Data`, `Order` and `Combinatorics` are very amenable to our translation, where specifically `Data.Int`, `Data.Nat`, `Data.Finset`, `Data.Vector` and `Data.List` have translatability rates between 85.3 % and 91.3 %. Many of the remaining non-translatable theorems in these sub-modules concern problems with translating λ's that either have a proposition as their domain or have `Type` (i.e. U_1 in DTT3) as their co-domain. Sometimes these λ's are not directly apparent such as in this example from `Data.Nat.Factorization.PrimePow`:

$$\texttt{exists\_ordCompl\_eq\_one\_iff\_isPrimePow}\ (n : \mathbb{N})\ (hn : n \neq 1) :\\ \texttt{IsPrimePow}\ n \leftrightarrow \exists p, \texttt{Nat.Prime}\ p \wedge n\ /\ p\ \hat{}\ n.\texttt{factorization}\ p = 1$$

Here `factorization` is actually not a function, but a `Finsupp`, which are so called 'finitely supported functions', functions that have as co-domain a type with a designated zero element and have only finitely many inputs for which they return this zero element. Lean treats these functions as 'function like' and when applying them, performs a coercion to a standard function, as follows for $n.\texttt{factorization}\ p$, where $\to_0$ is the `Finsupp` type:

$$\texttt{DFunLike.coe}\ (\mathbb{N} \to_0 \mathbb{N})\ \mathbb{N}\ (\texttt{fun}\ x : \mathbb{N} \Rightarrow \mathbb{N})\ \texttt{Finsupp.instFunLike}\ n.\texttt{factorization}\ p$$

The $\texttt{fun}\ x : \mathbb{N} \Rightarrow \mathbb{N}$ sub-term is Lean syntax for $\lambda_{x:\mathbb{N}}\mathbb{N}$, and it returns a type, namely $\mathbb{N}$, which DHOL only supports at theory level. Applying λ-lifting could be a solution for a future implementation to support more statements.

Remarkably, `Algebra`, `Analysis` and `Topology` also perform above average. For these the variance in translatability of sub-modules is high, meaning there are both sub-modules that perform very poorly, e.g. `Algebra.Category` and `Algebra.Homology` with 1.4 % and 3.3 % respectively, and some that perform very well, e.g. `Algebra.Ring` and `Algebra.Order` with 74.1 % and 83.2 % respectively. An example from `Algebra.Order.Field`, a fully translatable sub-module, is the following:

$$\texttt{Monotone.div\_const}\ (\beta : \texttt{Type})\ [\texttt{Preorder}\ \beta]\ (f : \beta \to \alpha)(hf : \texttt{Monotone}\ f)(c : \alpha)(hc : 0 \leq c) :\\ \texttt{Monotone}\ (\texttt{fun}\ x : \beta \Rightarrow f\ x\ /\ c)$$

Here we can also see a translatable λ.

7 Related Work

Our work extends the recent research on DHOL. Rothgang et al. [33] introduced the original monomorphic DHOL, including a sound and complete translation to monomorphic HOL. DHOL is reminiscent of the formalisms underlying the PVS [24] and Nuprl [11] proof assistants. Ranalter et al. [31] generalised the logic and the translation to support rank 1 polymorphism. This logic is the target of our own translation. Rabe [28] designed a semantics for an extension of this logic.

Rothgang and Rabe [32] also extended monomorphic DHOL with subtyping and adapted the translation to HOL accordingly. Moreover, Ranalter et al. [29] described an extension of monomorphic DHOL with Hilbert's choice operator and adapted the translation to HOL. On the theorem proving front, Niederhauser et al. [22] designed a sound and complete tableau style proof calculus for monomorphic DHOL and implemented it in the Lash prover. Finally, Ranalter et al. [30] extended the TPTP (Thousands of Problems for Theorem Provers) syntax and infrastructure with support for polymorphic DHOL.

Apart from the aforementioned translations from variants of DHOL to HOL, translations from DTTs are also relevant to our work. Jacobs and Melham [18] encoded Martin-Löf type theory into polymorphic HOL. In the context of hammers, Czajka and Kaliszyk [12] and Qian et al. [27] developed pragmatic translations from DTT to FOL and HOL, respectively. The logical frameworks

LF [17] and Twelf [26] also explored encodings of DTT to support automated metatheoretic reasoning. Finally, Oury [23], Winterhalter et al. [37] and Vaishnav [35] all translated extensional DTTs to intensional DTTs. They therefore take an inverse direction to our work, which translates from an intensional logic to an extensional logic.

8 Conclusion

We presented a translation from a fragment of DTT to polymorphic DHOL. The translation is sound and lightweight, preserving the structure of DTT expressions. The translatable fragment constitutes a large part of DTT as used in practice. Moreover, constructs such as inductive definitions and quotient types can be axiomatised, thereby further increasing our approach's applicability. The translation could be used to integrate DHOL based provers into proof assistants based on DTT as part of a hammer.

There are several avenues for future work. First, we should try to prove completeness and thereby obtain a precise characterisation of DHOL as a fragment of DTT. Second, we want to extend the translation to support more advanced features of DTT (e.g. inductive definitions and quotient types). Third, we would like to formalise the soundness argument using a proof assistant. Finally, to reach users we should implement the translation in a hammer system.

Acknowledgments

We thank Massin Guerdi for invaluable discussions on the definitions and theorems. We thank Kenji Maillard for valuable feedback regarding definitions and theorems. We thank an anonymous reviewer for pointing out how (D)HOL can be represented as a PTS. We thank Mark Summerfield and the anonymous reviewers for helpful textual suggestions.

Blanchette's research was co-funded by the European Union (ERC, Nekoka, 101083038). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Peter B. Andrews. 2002. *An Introduction to Mathematical Logic and Type Theory.* Applied Logic Series, Vol. 27. Springer Dordrecht.

[2] Carlo Angiuli and Daniel Gratzer. [n. d.]. *Principles of Dependent Type Theory.* Cambridge University Press. To appear.

[3] Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. 2011. The Matita Interactive Theorem Prover. In *CADE-23 (LNCS, Vol. 6803)*, Nikolaj S. Bjørner and Viorica Sofronie-Stokkermans (Eds.). Springer, 64–69. doi:10.1007/978-3-642-22438-6_7

[4] Henk Barendregt, Wil Dekkers, and Richard Statman. 2013. *Lambda Calculus with Types.* Cambridge University Press.

[5] Yves Bertot and Pierre Castéran. 2004. *Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions.* Springer. doi:10.1007/978-3-662-07964-5

[6] Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. 2016. Encoding Monomorphic and Polymorphic Types. *Log. Methods Comput. Sci.* 12, 4 (2016). doi:10.2168/LMCS-12(4:13)2016

[7] Chad E. Brown. 2012. Satallax: An Automatic Higher-Order Prover. In *IJCAR 2012 (LNCS, Vol. 7364)*, Bernhard Gramlich, Dale Miller, and Uli Sattler (Eds.). Springer, 111–117. doi:10.1007/978-3-642-31365-3_11

[8] Mario Carneiro. 2019. *The Type Theory of Lean.* Master's thesis. Carnegie Mellon University. https://github.com/digama0/lean-type-theory/releases/tag/v1.0

[9] Alonzo Church. 1940. A Formulation of the Simple Theory of Types. *J. Symb. Log.* 5, 2 (1940), 56–68. doi:10.2307/2266170

[10] Cyril Cohen. 2013. Pragmatic Quotient Types in Coq. In *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7998)*, Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie (Eds.). Springer, 213–228. doi:10.1007/978-3-642-39634-2_17

[11] Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, Robert Harper, Douglas J. Howe, Todd B. Knoblock, Nax Paul Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. 1986. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall. http://dl.acm.org/citation.cfm?id=10510

[12] Łukasz Czajka and Cezary Kaliszyk. 2018. Hammer for Coq: Automation for Dependent Type Theory. *J. Autom. Reason.* 61, 1-4 (2018), 423–453. doi:10.1007/S10817-018-9458-4

[13] Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 Theorem Prover and Programming Language. In *CADE-28 (LNCS, Vol. 12699)*, André Platzer and Geoff Sutcliffe (Eds.). Springer, 625–635. doi:10.1007/978-3-030-79876-5_37

[14] Xavier Généreux and Jannis Limperg. 2026. Incremental Forward Reasoning for White-Box Proof Search. In *Tools and Algorithms for the Construction and Analysis of Systems*, Sebastian Junges and Guy Katz (Eds.). Springer Nature Switzerland, Cham, 276–294.

[15] J.H. Geuvers. 1994. *The calculus of constructions and higher order logic.* Katholieke Universiteit Leuven, Belgium, 139–191.

[16] Michael J. C. Gordon. 1991. Introduction to the HOL System. In *1991 International Workshop on the HOL Theorem Proving System and Its Applications*, Myla Archer, Jeffrey J. Joyce, Karl N. Levitt, and Phillip J. Windley (Eds.). IEEE Computer Society, 2–3.

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

[18] Bart Jacobs and Thomas F. Melham. 1993. Translating Dependent Type Theory into Higher Order Logic. In *TLCA '93 (LNCS, Vol. 664)*, Marc Bezem and Jan Friso Groote (Eds.). Springer, 209–229. doi:10.1007/BFB0037108

[19] The mathlib Community. 2020. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New Orleans, LA, USA) *(CPP '20)*. ACM. doi:10.1145/3372885.3373824

[20] Jia Meng and Lawrence C. Paulson. 2008. Translating Higher-Order Clauses to First-Order Clauses. *J. Autom. Reason.* 40, 1 (2008), 35–60. doi:10.1007/S10817-007-9085-Y

[21] Rob Nederpelt and Herman Geuvers. 2014. *Type Theory and Formal Proof: An Introduction.* Cambridge University Press.

[22] Johannes Niederhauser, Chad E. Brown, and Cezary Kaliszyk. 2024. Tableaux for Automated Reasoning in Dependently-Typed Higher-Order Logic. In *IJCAR 2024, Part I (LNCS, Vol. 14739)*, Christoph Benzmüller, Marijn J. H. Heule, and Renate A. Schmidt (Eds.). Springer, 86–104. doi:10.1007/978-3-031-63498-7_6

[23] Nicolas Oury. 2005. Extensionality in the Calculus of Constructions. In *TPHOLs 2005 (LNCS, Vol. 3603)*, Joe Hurd and Thomas F. Melham (Eds.). Springer, 278–293. doi:10.1007/11541868_18

[24] Sam Owre, S. Rajan, John M. Rushby, Natarajan Shankar, and Mandayam K. Srivas. 1996. PVS: Combining Specification, Proof Checking, and Model Checking. In *CAV '96 (LNCS, Vol. 1102)*, Rajeev Alur and Thomas A. Henzinger (Eds.). Springer, 411–414. doi:10.1007/3-540-61474-5_91

[25] Frank Pfenning and Christine Paulin-Mohring. 1989. Inductively Defined Types in the Calculus of Constructions. In *MFPS 1989 (LNCS, Vol. 442)*, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt (Eds.). Springer, 209–228. doi:10.1007/BFB0040259

[26] Frank Pfenning and Carsten Schürmann. 1999. System Description: Twelf – a Meta-Logical Framework for Deductive Systems. In *CADE-16 (LNCS, Vol. 1632)*, Harald Ganzinger (Ed.). Springer, 202–206. doi:10.1007/3-540-48660-7_14

[27] Yicheng Qian, Joshua Clune, Clark W. Barrett, and Jeremy Avigad. 2025. Lean-Auto: An Interface Between Lean 4 and Automated Theorem Provers. In *CAV 2025, Part III (LNCS, Vol. 15933)*, Ruzica Piskac and Zvonimir Rakamaric (Eds.). Springer, 175–196. doi:10.1007/978-3-031-98682-6_10

[28] Florian Rabe. 2026. Semantics for Dependently-Typed Higher-Order Logic. In *IJCAR 2026 (LNCS)*, Armin Biere, Carsten Lutz, and Sara Negri (Eds.). Springer.

[29] Daniel Ranalter, Chad Brown, and Cezary Kaliszyk. 2024. Experiments with Choice in Dependently-Typed Higher-Order Logic. In *Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning (EPiC Series in Computing, Vol. 100)*, Nikolaj Bjørner, Marijn Heule, and Andrei Voronkov (Eds.). EasyChair, 311–320. doi:10.29007/2v8h

[30] Daniel Ranalter, Cezary Kaliszyk, Florian Rabe, and Geoff Sutcliffe. 2025. The Dependently Typed Higher-Order Form for the TPTP World. In *FroCoS 2025 (LNCS, Vol. 15979)*, René Thiemann and Christoph Weidenbach (Eds.). Springer, 287–305. doi:10.1007/978-3-032-04167-8_16

[31] Daniel Ranalter, Florian Rabe, and Cezary Kaliszyk. 2026. Polymorphism Meets DHOL. In *FSCD 2026 (LIPIcs)*, F. Pfenning (Ed.). Schloss Dagstuhl–Leibniz Center for Informatics. to appear.

[32] Colin Rothgang and Florian Rabe. 2025. Subtyping in Dependently-Typed Higher-Order Logic. In *FroCoS 2025 (LNCS, Vol. 15979)*, René Thiemann and Christoph Weidenbach (Eds.). Springer, 98–114. doi:10.1007/978-3-032-04167-8_6

[33] Colin Rothgang, Florian Rabe, and Christoph Benzmüller. 2025. Dependently Typed Higher-Order Logic. *ACM Trans. Comput. Logic* 27, 1, Article 6 (Dec. 2025), 54 pages. doi:10.1145/3771725

[34] Geoff Sutcliffe and Martin Desharnais. 2024. The CADE-29 Automated Theorem Proving System Competition - CASC-29. *AI Commun.* 37, 4 (2024), 485–503. doi:10.3233/AIC-230325

[35] Rishikesh Vaishnav. 2026. Lean4Less: Eliminating Definitional Equalities from Lean via an Extensional-to-Intensional Translation. In *Theoretical Aspects of Computing – ICTAC 2025*, Zhiming Liu, Adnane Saoud, and Heike Wehrheim (Eds.). Springer Nature Switzerland, Cham, 202–219.

[36] Benjamin Werner. 1997. Sets in types, types in sets. In *Theoretical Aspects of Computer Software*, Martín Abadi and Takayasu Ito (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 530–546.

[37] Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. 2019. Eliminating reflection from type theory. 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, 91–103. doi:10.1145/3293880.3294095

[38] Thomas Zhu, Joshua Clune, Jeremy Avigad, Albert Qiaochu Jiang, and Sean Welleck. 2025. Premise Selection for a Lean Hammer. *CoRR* abs/2506.07477 (2025). arXiv:2506.07477 doi:10.48550/ARXIV.2506.07477