







# Theorem Proving in Dependently-Typed Higher-Order Logic

Colin Rothgang<sup>1</sup>  , Florian Rabe<sup>2</sup> , and Christoph Benzmüller<sup>1,3</sup> 

<sup>1</sup> Mathematics and Computer Science, FU, Berlin, Germany  
colin.rothgang@gmx.de

<sup>2</sup> Computer Science, University Erlangen-Nürnberg, Erlangen, Germany

<sup>3</sup> AI Systems Engineering, University Bamberg, Bamberg, Germany

**Abstract.** Higher-order logic HOL offers a very simple syntax and semantics for representing and reasoning about typed data structures. But its type system lacks advanced features where types may depend on terms. Dependent type theory offers such a rich type system, but has rather substantial conceptual differences to HOL, as well as comparatively poor proof automation support.

We introduce a dependently-typed extension DHOL of HOL that retains the style and conceptual framework of HOL. Moreover, we build a translation from DHOL to HOL and implement it as a preprocessor to a HOL theorem prover, thereby obtaining a theorem prover for DHOL.

## 1 Introduction and Related Work

Theorem proving in higher-order logic (HOL) [5, 11] has been a long-running research strand producing multiple mature interactive provers [10, 13, 17] and automated provers [2, 4, 23]. Similarly, many, mostly interactive, theorem provers are available for various versions of dependent type theory (DTT) [7, 9, 15, 18]. However, it is (maybe surprisingly) difficult to develop theorem provers for dependently-typed higher-order logic (DHOL).

In this paper, we use HOL to refer to a version of Church’s *simply*-typed  $\lambda$ -calculus with a base type `bool` for Booleans, simple function types  $\rightarrow$ , and equality  $=_A: A \rightarrow A \rightarrow \text{bool}$ . This already suffices to define the usual logical quantifiers and connectives.<sup>1</sup> Intuitively, it is straightforward to develop DHOL accordingly on top of the *dependently*-typed  $\lambda$ -calculus, which uses a dependent function type  $\Pi x:A. B$  instead of  $\rightarrow$ . However, several subtleties arise that seem deceptively minor at first but end up presenting fundamental theoretical issues. They come up already in the elementary expression  $x =_A y \Rightarrow f(x) =_{B(x)} f(y)$  for some dependent function  $f: \Pi x:A. B(x)$ .

**Firstly**, the equality  $f(x) =_{B(x)} f(y)$  is not even well-typed because the terms  $f(x): B(x)$  and  $f(y): B(y)$  do not have the same type. Intuitively, it is obvious that the type system can (and maybe should) be adjusted so that the equality  $x =_A y$  between terms

<sup>1</sup> We do not assume a choice operator or the axiom of infinity.

carries over to an equality  $B(x) \equiv B(y)$  between types.<sup>2</sup> However, this means that the undecidability of equality leaks into the equality of types and thus into type-checking.

While some interactive provers successfully use undecidable type systems [6, 16], most formal systems for DTT commit to keeping type-checking decidable. The typical approach goes back to Martin-Löf type theory [14] and the calculus of constructions [8] and uses two separate equality relations, a decidable meta-level equality for use in the type-checker and a stronger undecidable one subject to theorem proving. Moreover, it favors the propositions-as-types representation and deemphasizes or omits a type of classical Booleans. This approach has been studied extensively [7, 9, 15] and is not the subject of this paper.

Instead, our motivation is to retain a single equality relation and classical Booleans. This is arguably more intuitive to users, especially to those outside the DTT community such as typical HOL users or mathematicians, and it is certainly much closer to the logics of the strongest available ATP systems. This means we have to pay the price of undecidable type-checking. The current paper was prompted by the observation that this price may be acceptable for two reasons:

1. If our ultimate interest is theorem proving, undecidability comes up anyway. Indeed, it is plausible that the cost of showing the well-typedness of a conjecture will be negligible compared to the cost of proving it.
2. As the strength of ATPs for HOL increases, the practical drawbacks of undecidable type-checking decrease, which indicates revisiting the trade-off from time to time. Indeed, if we position DHOL close to an existing HOL ATP, it is plausible that the price will, in practice, be affordable.

**Secondly**, even if we add a rule like “if  $\vdash x =_A y$ , then  $\vdash B(x) \equiv B(y)$ ” to our type system, the above expression is still not well-typed: Above, the equality  $x =_A y$  on the left of  $\Rightarrow$  is needed to show the well-typedness of the equality  $f(x) =_{B(x)} f(y)$  on the right. This intertwines theorem proving and type-checking even further. Concretely, we need a *dependent implication*, where the first argument is assumed to hold while checking the well-typedness of the second one. Formally, this means that to show  $\vdash F \Rightarrow G : \text{bool}$ , we require  $\vdash F : \text{bool}$  and  $F \vdash G : \text{bool}$ . Similarly, we need a dependent conjunction. And if we are classical, we may also opt to add a dependent disjunction  $F \vee G$ , where  $\neg F$  is assumed in  $G$ . Naturally, dependent conjunction and disjunction are not commutative anymore. This may feel disruptive, but similar behavior of connectives is well-known from short-circuit evaluation in programming languages.

The meta-logical properties of dependent connectives are straightforward. However, interestingly, these connectives can no longer be defined from just equality. At least one of them (we will choose dependent implication) must be taken as an additional primitive in DHOL along with  $=_A$ .

**Finally**, the above generalizations require a notion of DHOL-contexts that is more complex than for HOL. HOL-contexts can be stratified into (a) a set of variable declarations

---

<sup>2</sup> Note that while term equality  $=_A$  is a bool-valued connective, type equality  $\equiv$  is not. Instead, in HOL,  $\equiv$  is a judgment at the same level as the typing judgment  $t : A$ .

$x_i : A_i$ , and (b) a set of logical assumptions  $F$  possibly using the variables  $x_i$ . Moreover, the former are often not explicitly listed at all and instead inferred from the remainder of the sequent. But in DHOL, the well-formedness of an  $A_i$  may now depend on previous logical assumptions. To linearize this inter-dependency, DHOL contexts must consist of a single list alternating between variable declarations and assumptions.

*Contribution.* Our contribution is twofold. Firstly, we introduce a new logic DHOL designed along the lines described above. Moreover, we further extend DHOL with predicate subtypes  $A|_p$  for a predicate  $p : A \rightarrow \text{bool}$  on the type  $A$ . Besides dependent types, these constitute a second important source of terms occurring in types. Because they also make typing undecidable, they are often avoided. The most prominent exception is PVS [16], whose kernel essentially arises by adding predicate subtypes to HOL. In current HOL ITPs going back to [10], their use is usually restricted to the subtype definition principle: here a definition  $b := A|_p$  may occur on toplevel and is elaborated into a fresh type  $b$  that is axiomatized to mimic the subtype  $A|_p$ . Because we are committed to undecidable typing anyway, predicate subtypes fit naturally into our approach.

Secondly, we develop and implement a sound and complete translation of DHOL into HOL. This setup allows the use of DHOL as the expressive user-facing language and HOL as the internal theorem-proving language. We position our implementation close to an existing HOL ATP, namely the LEO-III system. From the LEO-III perspective, DHOL serves as an additional input language that is translated into HOL by an external logic embedding tool [21, 22] in the LEO-III ecosystem. Because LEO-III already supports such embeddings and because the TPTP syntax [24] foresees the use of dependent types in ATPs and provides syntax for them (albeit without a normative semantics), we were able to implement the translation with no disruptions to existing workflows.

The general idea of our translation of dependent into simple type theory is not new [3]. In that work, Martin-Löf-style dependent type theory is translated into Gordon’s HOL ITP [10]. This work differs critically from ours because it uses DTT in propositions-as-types style. Our work builds DHOL with classical Booleans and equality predicate, which makes the task of proving the translation sound and complete very different. Moreover, their work targets an interactive prover while ours targets automated ones.

*Overview.* In Sect. 2 we recap the HOL logic. In Sect. 3 we extend it to DHOL and define our translation from DHOL to HOL. In Sect. 4 we add subtyping and predicate subtypes. In Sect. 5 we prove the soundness and completeness of the translation. In Sect. 6 we describe how to use our translation and a HOL ATP to implement a theorem prover for DHOL.

## 2 Preliminaries: Higher-Order Logic

We introduce the syntax and rules of HOL. Our definitions are standard except that we tweak a few details in order to later present the extension to DHOL more succinctly. We use the following grammar for HOL:

$T$	$::=$	$\circ \mid T, a : \text{tp} \mid T, c : A \mid T, c : F$	theories
$\Gamma$	$::=$	$\cdot \mid \Gamma, x : A \mid \Gamma, x : F$	contexts
$A, B$	$::=$	$a \mid A \rightarrow B \mid \text{bool}$	types
$s, t, f, F, G$	$::=$	$c \mid x \mid \lambda x : A. t \mid f t \mid s =_A t \mid F \Rightarrow G$	terms

A theory  $T$  is a list of base type declarations  $a : \text{tp}$ , typed constant declarations  $c : A$ , and named axioms  $c : F$  asserting the formula  $F$ . A context  $\Gamma$  has the same form except that no type variables are allowed. It is not strictly necessary to use named axioms and assumptions, but it makes our extensions to DHOL later on simpler. We write  $\circ$  and  $\cdot$  for the empty theory and context, respectively. At this point, it is possible to normalize contexts into a set of variable declarations followed by a set of assumptions because the well-formedness of a type  $A$  can never depend on a variable or an assumption. But that property will change when going to DHOL, which is why we allow  $\Gamma$  to alternate between variables and assumptions.

Types  $A$  are either user-declared types  $a$ , the built-in base type  $\text{bool}$ , or function types  $A \rightarrow B$ . Terms are constants  $c$ , variables  $x$ ,  $\lambda$ -abstractions  $\lambda x : A. t$ , function applications  $f t$ , or obtained from the built-in  $\text{bool}$ -valued connectives  $=_A$  or  $\Rightarrow$ . As usual [1], this suffices to define all the usual quantifiers and connectives  $\text{true}$ ,  $\text{false}$ ,  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\forall$  and  $\exists$ . This includes  $\Rightarrow$ , but we make it a primitive here because we will change it in DHOL. As usual,  $E^{[x/l]}$  denotes the capture-avoiding substitution of the variable  $x$  with the term  $t$  within expression  $E$ .

The type and proof system uses the judgments given below. Note that we need a meta-level judgment for the equality of types because  $\equiv$  is not a  $\text{bool}$ -valued connective. On the contrary, the equality of terms  $\vdash s =_A t$  is a special case of the validity judgment  $\vdash F$ . In HOL,  $\equiv$  is trivial, and the judgment is redundant. But we include it here already because it will become non-trivial in DHOL.

Name	Judgment	Intuition
theories	$\vdash T \text{ Thy}$	$T$ is well-formed theory
contexts	$\vdash_T \Gamma \text{ Ctx}$	$\Gamma$ is well-formed context
types	$\Gamma \vdash_T A \text{ tp}$	$A$ is well-formed type
typing	$\Gamma \vdash_T t : A$	$t$ is a well-formed term of type well-formed type $A$
validity	$\Gamma \vdash_T F$	well-formed Boolean $F$ is provable
equality of types	$\Gamma \vdash_T A \equiv B$	well-formed types $A$ and $B$ are equal

The rules are given in Fig. 1. We assume that all names in a theory or a context are unique without making that explicit in the rules. Following common practice, we further assume that HOL types are non-empty.

Theories and contexts:

$$\frac{}{\vdash \circ \text{Thy}} \quad \frac{}{\vdash T, a \text{ tp Thy}} \quad \frac{}{\vdash T, c : A \text{ Thy}} \quad \frac{}{\vdash T, c : F \text{ Thy}}$$

$$\frac{}{\vdash T \text{ Thy}} \quad \frac{}{\vdash_T \cdot \text{Ctx}} \quad \frac{}{\vdash_T \Gamma, x : A \text{ Ctx}} \quad \frac{}{\vdash_T \Gamma, x : F \text{ Ctx}}$$

Lookup in theory and context:

$$\frac{a : \text{tp in } T \quad \vdash_T \Gamma \text{ Ctx}}{\Gamma \vdash_T a \text{ tp}} \quad \frac{c : A' \text{ in } T \quad \Gamma \vdash_T A' \equiv A}{\Gamma \vdash_T c : A} \quad \frac{c : F \text{ in } T \quad \vdash_T \Gamma \text{ Ctx}}{\Gamma \vdash_T F}$$

$$\frac{x : A' \text{ in } \Gamma \quad \Gamma \vdash_T A' \equiv A}{\Gamma \vdash_T x : A} \quad \frac{x : F \text{ in } \Gamma \quad \vdash_T \Gamma \text{ Ctx}}{\Gamma \vdash_T F}$$

Well-formedness and equality of types:

$$\frac{\vdash_T \Gamma \text{ Ctx}}{\Gamma \vdash_T \text{bool tp}} \quad \frac{\Gamma \vdash_T A \text{ tp} \quad \Gamma \vdash_T B \text{ tp}}{\Gamma \vdash_T A \rightarrow B \text{ tp}} \quad \frac{\Gamma \vdash_T A \text{ tp}}{\Gamma \vdash_T A \equiv A} \quad \frac{\Gamma \vdash_T A \equiv A' \quad \Gamma \vdash_T B \equiv B'}{\Gamma \vdash_T A \rightarrow B \equiv A' \rightarrow B'}$$

Typing:

$$\frac{\Gamma, x : A \vdash_T t : B}{\Gamma \vdash_T (\lambda x : A. t) : A \rightarrow B} \quad \frac{\Gamma \vdash_T f : A \rightarrow B \quad \Gamma \vdash_T t : A}{\Gamma \vdash_T f t : B} \quad \frac{\Gamma \vdash_T s : A \quad \Gamma \vdash_T t : A}{\Gamma \vdash_T s =_A t : \text{bool}}$$

Term equality: congruence, reflexivity, symmetry,  $\beta, \eta$

$$\frac{\Gamma \vdash_T A \equiv A' \quad \Gamma, x : A \vdash_T t =_B t'}{\Gamma \vdash_T \lambda x : A. t =_{A \rightarrow B} \lambda x : A'. t'} \quad \frac{\Gamma \vdash_T t =_A t' \quad \Gamma \vdash_T f =_{A \rightarrow B} f'}{\Gamma \vdash_T f t =_B f' t'}$$

$$\frac{\Gamma \vdash_T t : A}{\Gamma \vdash_T t =_A t} \quad \frac{\Gamma \vdash_T t =_A s}{\Gamma \vdash_T s =_A t} \quad \frac{\Gamma \vdash_T (\lambda x : A. s) t : B}{\Gamma \vdash_T (\lambda x : A. s) t =_B s[x/i]} \quad \frac{\Gamma \vdash_T t : A \rightarrow B \quad x \text{ not in } \Gamma}{\Gamma \vdash_T t =_{A \rightarrow B} \lambda x : A. t x}$$

Rules for implication:

$$\frac{\Gamma \vdash_T F : \text{bool} \quad \Gamma \vdash_T G : \text{bool}}{\Gamma \vdash_T F \Rightarrow G : \text{bool}} \quad \frac{\Gamma \vdash_T F : \text{bool} \quad \Gamma, x : F \vdash_T G}{\Gamma \vdash_T F \Rightarrow G} \quad \frac{\Gamma \vdash_T F \Rightarrow G \quad \Gamma \vdash_T F}{\Gamma \vdash_T G}$$

Congruence for validity, Boolean extensionality, and non-emptiness of types:

$$\frac{\Gamma \vdash_T F =_{\text{bool}} F' \quad \Gamma \vdash_T F'}{\Gamma \vdash_T F} \quad \frac{\Gamma \vdash_T p \text{ true} \quad \Gamma \vdash_T p \text{ false}}{\Gamma, x : \text{bool} \vdash_T p x} \quad \frac{\Gamma \vdash_T F : \text{bool} \quad \Gamma, x : A \vdash_T F}{\Gamma \vdash_T F}$$

**Fig. 1.** HOL Rules

### 3 Dependent Function Types

#### 3.1 Language

We have carefully defined HOL in such a way that only a few surgical changes are needed to define DHOL. A consolidated summary of DHOL is given in Appendix A.2 in the extended preprint [20]. The **grammar** is as follows with unchanged parts shaded out:

$T$	$::=$	$\circ \mid T, a : (\prod x : A. \_)*\text{tp} \mid T, c : A \mid T, c : F$	theories
$\Gamma$	$::=$	$\cdot \mid \Gamma, x : A \mid \Gamma, \text{ass} : F$	contexts
$A, B$	$::=$	$a \ t_1 \dots t_n \mid \prod x : A. B \mid \text{bool}$	types
$s, t, f, F, G$	$::=$	$c \mid x \mid \lambda x : A. t \mid f \ t \mid s =_A t \mid F \Rightarrow G$	terms

Concretely, base types  $a$  may now take term arguments and simple function types  $A \rightarrow B$  are replaced with dependent function types  $\prod x : A. B$ . As usual we will retain the notation  $A \rightarrow B$  for the latter if  $x$  does not occur free in  $B$ . DHOL is a conservative extension of HOL, and we recover HOL as the fragment of DHOL in which all base types  $a$  have arity 0.

*Example 1 (Category Theory).* As a running example, we formalize the theory of a category in DHOL. It declares the base type  $\text{obj}$  for objects and the dependent base type  $\text{mor } a \ b$  for morphisms. Further it declares the constants  $\text{id}$  and  $\text{comp}$  for identity and composition, and the axioms for neutrality. We omit the associativity axiom for brevity.

$$\begin{aligned} \text{obj} &: \text{tp} \\ \text{mor} &: \prod x, y : \text{obj}. \text{tp} \\ \text{id} &: \prod a : \text{obj}. \text{mor } a \ a \\ \text{comp} &: \prod a, b, c : \text{obj}. \text{mor } a \ b \rightarrow \text{mor } b \ c \rightarrow \text{mor } a \ c \\ \text{neutL} &: \forall x, y : \text{obj}. \forall m : \text{mor } x \ y. m \circ \text{id}_x =_{\text{mor } x \ y} m \\ \text{neutR} &: \forall x, y : \text{obj}. \forall m : \text{mor } x \ y. \text{id}_y \circ m =_{\text{mor } x \ y} m \end{aligned}$$

Here we use a few intuitive notational simplifications such as writing  $\prod x, y : \text{obj}.$  for binding two variables of the same type. We also use the notations  $\text{id}_x$  for  $\text{id } x$  and  $h \circ g$  for  $\text{comp } \_ \_ g \ h$  where the  $\_$  denote inferable arguments of type  $\text{obj}$ .

The **judgments** stay the same and we only make minor changes to the **rules**, which we explain in the sequel. Firstly we replace all rules for  $\rightarrow$  with the ones for  $\Pi$ :

$$\begin{array}{c} \frac{\Gamma \vdash_T A \text{ tp} \quad \Gamma, x : A \vdash_T B \text{ tp}}{\Gamma \vdash_T \prod x : A. B \text{ tp}} \quad \frac{\Gamma \vdash_T A \equiv A' \quad \Gamma, x : A \vdash_T B \equiv B'}{\Gamma \vdash_T \prod x : A. B \equiv \prod x : A'. B'} \\ \\ \frac{\Gamma, x : A \vdash_T t : B}{\Gamma \vdash_T (\lambda x : A. t) : \prod x : A. B} \quad \frac{\Gamma \vdash_T f : \prod x : A. B \quad \Gamma \vdash_T t : A}{\Gamma \vdash_T f \ t : B[x/i]} \\ \\ \frac{\Gamma \vdash_T A \equiv A' \quad \Gamma, x : A \vdash_T t =_B t'}{\Gamma \vdash_T \lambda x : A. t =_{\prod x : A. B} \lambda x : A'. t'} \quad \frac{\Gamma \vdash_T t =_A t' \quad \Gamma \vdash_T f =_{\prod x : A. B} f'}{\Gamma \vdash_T f \ t =_B f' \ t'} \end{array}$$

$$\frac{\Gamma \vdash_T t : \Pi x : A. B}{\Gamma \vdash_T t =_{\Pi x A. B} \lambda x : A. t x}$$

Then we replace the rules for declaring, using, and equating base types with the ones where base types are applied to arguments:

$$\frac{\vdash_T x_1 : A_1, \dots, x_n : A_n \text{ Ctx}}{\vdash_T, a : \Pi x_1 : A_1. \dots \Pi x_n : A_n. \text{tp Thy}}$$

$$\frac{\vdash_T \Gamma \text{ Ctx } a : \Pi x_1 : A_1. \dots \Pi x_n : A_n. \text{tp in } T \quad \Gamma \vdash_T t_1 : A_1 \dots \Gamma \vdash_T t_n : A_n[x_1/t_1] \dots [x_{n-1}/t_{n-1}]}{\Gamma \vdash_T a t_1 \dots t_n \text{tp}}$$

$$\frac{\vdash_T \Gamma \text{ Ctx } a : \Pi x_1 : A_1. \dots \Pi x_n : A_n. \text{tp in } T \quad \Gamma \vdash_T s_1 =_{A_1} t_1 \dots \Gamma \vdash_T s_n =_{A_n[x_1/t_1] \dots [x_{n-1}/t_{n-1}]} t_n}{\Gamma \vdash_T a s_1 \dots s_n \equiv a t_1 \dots t_n}$$

The last of these is the critical rule via which term equality leaks into type equality. Thus, typing of expressions may now depend on equality assumptions and thus typing becomes undecidable.

*Example 2 (Undecidability of Typing).* Continuing Example 1, consider terms  $\vdash f : \text{mor } u \ v$  and  $\vdash g : \text{mor } v' \ w$  for terms  $\vdash u, v, v', w : \text{obj}$ . Then  $\vdash g \circ f : \text{mor } u \ w$  holds iff  $\vdash f : \text{mor } u \ v'$ , which holds iff  $\vdash v =_{\text{obj}} v'$ . Depending on the axioms present, this may be arbitrarily difficult to prove.

Finally, we modify the rule for the non-emptiness of types: we allow the existence of empty dependent types and only require that for each HOL type in the image of the translation there exists one non-empty DHOL type translated to it (rather than requiring all dependent types translated to it to be non-empty). And we replace the typing rule for implication with the dependent one. The proof rules for implications are unchanged.

$$\frac{\Gamma \vdash_T F : \text{bool} \quad \Gamma, x : F \vdash_T G : \text{bool}}{\Gamma \vdash_T F \Rightarrow G : \text{bool}}$$

*Example 3 (Dependent Implication).* Continuing Example 1, consider the formula

$$x : \text{obj}, y : \text{obj} \vdash x =_{\text{obj}} y \Rightarrow \text{id}_x =_{\text{mor } x \ x} \text{id}_y : \text{bool}$$

which expresses that equal objects have equal identity morphisms. It is easy to prove. But it is only well-typed because the typing rule for dependent implication allows using  $x =_{\text{obj}} y$  while type-checking  $\text{id}_x =_{\text{mor } x \ x} \text{id}_y : \text{bool}$ , which requires deriving  $\text{id}_y : \text{mor } x \ x$  and thus  $\text{mor } y \ y \equiv \text{mor } x \ x$ .

All the usual connectives and quantifiers can be defined in any of the usual ways now. However, the details matter for the dependent versions of the connectives. In particular, we choose  $F \wedge G := \neg(F \Rightarrow \neg G)$  and  $F \vee G := \neg F \Rightarrow G$  in order to obtain the dependent versions of conjunction and disjunction, in which the well-formedness of  $G$  may depend on the truth or falsity of  $F$ , respectively.

### 3.2 Translation

We define a translation function  $X \mapsto \bar{X}$  that maps any DHOL-syntax  $X$  to HOL-syntax. Its intuition is to erase type dependencies by translating all types  $a t_1 \dots, t_n$  to  $a$  and replacing every  $\Pi$  with  $\rightarrow$ . To recover the information of the erased dependencies, we additionally define a partial equivalence relation (PER)  $A^*$  on  $\bar{A}$  for every DHOL-type  $A$ .

In general, a PER  $r$  on type  $U$  is a symmetric and transitive relation on  $U$ . This is equivalent to  $r$  being an equivalence relation on a subtype of  $U$ . The intuitive meaning of our translation is that the DHOL-type  $A$  corresponds in HOL to the quotient of the appropriate subtype of  $\bar{A}$  by the equivalence  $A^*$ . In particular, the predicate  $A^* t t$  captures whether  $t$  represents a term of type  $A$ . More formally, the correspondence is:

DHOL	HOL
type $A$	type $\bar{A}$ and PER $A^* : \bar{A} \rightarrow \bar{A} \rightarrow \text{bool}$
term $t : A$	term $\bar{t} : \bar{A}$ satisfying $A^* \bar{t} \bar{t}$

**Definition 1 (Translation).** *We translate DHOL-syntax by induction on the grammar. Theories and contexts are translated declaration-wise:*

$$\bar{\circ} := \circ \quad \overline{T, \bar{D}} := \bar{T}, \bar{D} \quad \bar{\cdot} := \cdot \quad \overline{T, \bar{D}} := \bar{T}, \bar{D}$$

where  $\bar{D}$  is a list of declarations.

The translation  $\overline{a : \Pi x_1 : A_1. \dots \Pi x_n : A_n. \text{tp}}$  of a base type declaration is given by

$$a : \text{tp}, \quad a^* : \bar{A}_1 \rightarrow \dots \rightarrow \bar{A}_n \rightarrow a \rightarrow a \rightarrow \text{bool}$$

$$a_{\text{PER}} : \forall x_1 : \bar{A}_1. \dots \forall x_n : \bar{A}_n. \forall u, v : a. a^* x_1 \dots x_n u v \Rightarrow u =_a v$$

Thus,  $a$  is translated to a base type of the same name without arguments and a trivial PER for every argument tuple. Intuitively,  $a^* \bar{t}_1 \dots \bar{t}_n u u$  defines the subtype of the HOL-type  $a$  corresponding to the DHOL-type  $a t_1 \dots t_n$ .

Constant and variable declarations are translated by adding the assumptions that they are in the PER of their type, and axioms and assumptions are translated straightforwardly:

$$\overline{c : A} := c : \bar{A}, c^* : A^* c c \quad \overline{x : A} := x : \bar{A}, x^* : A^* x x$$



$$\overline{c:F} := c:\overline{F} \quad \overline{x:F} := x:\overline{F}$$

The cases of  $\overline{A}$  and  $A^*$  for types  $A$  are:

$$\begin{aligned} \overline{a t_1 \dots t_n} &:= a & (a t_1 \dots t_n)^* s t &:= a^* \overline{t_1} \dots \overline{t_n} s t \\ \overline{\prod x:A. B} &:= \overline{A} \rightarrow \overline{B} & (\prod x:A. B)^* f g &:= \forall x,y:\overline{A}. A^* x y \Rightarrow B^* (f x) (g y) \\ \overline{\text{bool}} &:= \text{bool} & \text{bool}^* s t &:= s =_{\text{bool}} t \end{aligned}$$

Finally, the cases for terms are straightforward except for, crucially, translating equality to the respective PER:

$$\begin{aligned} \overline{c} &:= c & \overline{x} &:= x & \overline{\lambda x:A. t} &:= \lambda x:\overline{A}. \overline{t} & \overline{f t} &:= \overline{f} \overline{t} \\ \overline{F \Rightarrow G} &:= \overline{F} \Rightarrow \overline{G} & \overline{s =_A t} &:= A^* \overline{s} \overline{t} \end{aligned}$$

*Example 4 (Translating Derived Connectives).* If we define true, false,  $\neg$  as usual in HOL and use the definition for dependent conjunction from above, it is straightforward to show that all DHOL-connectives are translated to their HOL-counterparts. For example, we have (up to logical equivalence in HOL) that  $\overline{F \wedge G} = \overline{F} \wedge \overline{G}$ .

We also define the quantifiers in the usual way, e.g., using  $\forall x:A. F(x) := \lambda x:A. F(x) =_{A \rightarrow \text{bool}} \lambda x:A. \text{true}$ . Then applying our translation yields

$$\begin{aligned} \overline{\forall x:A. F(x)} &= (A \rightarrow \text{bool})^* \overline{\lambda x:A. F(x)} \overline{\lambda x:A. \text{true}} \\ &= \forall x,y:\overline{A}. A^* x y \Rightarrow \text{bool}^* F(x) \text{true} \end{aligned}$$

This looks clunky, but (because  $A^*$  is a PER as shown in Theorem 1) is equivalent to  $\forall x:\overline{A}. A^* x x \Rightarrow F(x)$ . Thus, DHOL- $\forall$  is translated to HOL- $\forall$  relativized using  $A^* x x$ . The corresponding rule  $\overline{\exists x:A. F(x)} = \exists x:\overline{A}. A^* x x \wedge F(x)$  can be shown accordingly.

*Example 5 (Categories in HOL).* We give a fragment of the translation of Example 1:

$$\begin{aligned} \text{obj} &: \text{tp} & \text{obj}^* &: \text{obj} \rightarrow \text{obj} \rightarrow \text{bool} \\ \text{mor} &: \text{tp} & \text{mor}^* &: \text{obj} \rightarrow \text{obj} \rightarrow \text{mor} \rightarrow \text{mor} \rightarrow \text{bool} \\ \text{id} &: \text{obj} \rightarrow \text{mor} & \text{id}^* &: \forall x,y:\text{obj}. \text{obj}^* x y \Rightarrow \text{mor}^* x x (\text{id } x) (\text{id } y) \\ \text{comp} &: \text{obj} \rightarrow \text{obj} \rightarrow \text{obj} \rightarrow \text{mor} \rightarrow \text{mor} \rightarrow \text{mor} \\ \text{neutL} &: \forall x:\text{obj}. \text{obj}^* x x \Rightarrow \forall y:\text{obj}. \text{obj}^* y y \Rightarrow \\ & \quad \forall m:\text{mor}. \text{mor}^* x y m m \Rightarrow \text{mor}^* x y (\text{comp } x x y (\text{id } x) m) m \end{aligned}$$

Here, for brevity, we have omitted  $\text{obj}_{\text{PER}}$ ,  $\text{mor}_{\text{PER}}$ , and  $\text{comp}^*$  and have already used the translation rule for  $\forall$  from Example 4. The result is structurally close to what a native formalization of categories in HOL would look like, but somewhat clunkier.

Typing rules for predicate subtypes:

$$\frac{\Gamma \vdash_T p : \Pi x : A. \text{bool}}{\Gamma \vdash_T A|_p \text{ tp}} \quad \frac{\Gamma \vdash_T t : A \quad \Gamma \vdash_T p t}{\Gamma \vdash_T t : A|_p} \quad \frac{\Gamma \vdash_T t : A|_p}{\Gamma \vdash_T p t}$$

Congruence and variance rule for predicate subtypes:

$$\frac{\Gamma \vdash_T A \equiv A' \quad \Gamma \vdash_T p \equiv_{\Pi x:A. \text{bool}} p'}{\Gamma \vdash_T A|_p \equiv A'|_{p'}} \quad \frac{\Gamma \vdash_T A <: A' \quad \Gamma, x : A \vdash_T p x \Rightarrow p' x}{\Gamma \vdash_T A|_p <: A'|_{p'}}$$

Rules that relate  $A$  and  $A|_p$ :

$$\frac{\Gamma \vdash_T A <: A'}{\Gamma \vdash_T A|_p <: A'} \quad \frac{\Gamma \vdash_T A \text{ tp}}{\Gamma \vdash_T A \equiv A|_{\lambda x:A. \text{true}}} \quad \frac{\Gamma \vdash_T A \text{ tp}}{\Gamma \vdash_T A|_{\lambda x:A. \text{true}} \equiv A}$$

Variance rules for other DHOL types:

$$\frac{\Gamma \vdash_T A \equiv A'}{\Gamma \vdash_T A <: A'} \quad \frac{\Gamma \vdash_T A' <: A \quad \Gamma, x : A' \vdash_T B <: B'}{\Gamma \vdash_T \Pi x : A. B <: \Pi x : A'. B'}$$

Rules for normalizing certain subtypes:

$$\frac{\Gamma \vdash_T A \text{ tp} \quad \Gamma, x : A \vdash_T B \text{ tp} \quad \Gamma, x : A \vdash_T p : \Pi y : B. \text{bool}}{\Gamma \vdash_T \Pi x : A. (B|_p) \equiv (\Pi x : A. B)|_{\lambda f.\forall x:A. p (f x)}} \quad \frac{\Gamma \vdash_T A \text{ tp} \quad \Gamma \vdash_T p : \Pi x : A. \text{bool} \quad \Gamma \vdash_T q : \Pi x : (A|_p). \text{bool}}{\Gamma \vdash_T A|_p|_q \equiv A|_{\lambda x:A. p \wedge q x}}$$

**Fig. 2.** Additional Rules for Predicate Subtypes

## 4 Predicate Subtypes

To add predicate subtypes, we extend the **grammar** with the production  $A ::= A|_F$ . No new productions for terms are needed because the inhabitants of  $A|_F$  use the same syntax as those of  $A$ .

*Example 6 (Isomorphisms).* We continue Example 1 and use predicate subtypes to write the type isomorphisms  $u$  of automorphisms on  $u$  as a subtype of  $\text{mor } u \ u$ . We can define isomorphisms  $u := (\text{mor } u \ u)|_p$  where the predicate  $p$  is given by

$$\lambda m : \text{mor } u \ u. \exists i : \text{mor } u \ u. (i \circ m =_{\text{mor } u \ u} \text{id}_u) \wedge (m \circ i =_{\text{mor } u \ u} \text{id}_u)$$

Adding subtyping requires a few extensions to our type system. First we add a **judgment**  $\Gamma \vdash_T A <: B$  and replace the lookup rules for variables and constants with their subtyping-aware variants:

$$\frac{c : A' \text{ in } T \quad \Gamma \vdash_T A' <: A}{\Gamma \vdash_T c : A} \quad \frac{x : A' \text{ in } \Gamma \quad \Gamma \vdash_T A' <: A}{\Gamma \vdash_T x : A}$$

Then we add the **rules** given in Fig. 2. These induce an algorithm for deciding subtyping relative to an oracle for the undecidable validity judgment. The latter enters the algorithm when two predicate subtypes are compared. Note that the type-equality rule for  $A|_p|_q$  uses a dependent conjunction.

The resulting system is a conservative extension of the variants of HOL and DHOL without subtyping: we recover these systems as the fragments that do not use  $A|_p$ . In particular, in that case  $A <: B$  is trivial and holds iff  $A \equiv B$  holds.

Finally, we extend our **translation** by adding the cases for predicate subtypes:

**Definition 2 (Translation).** *We extend Definition 1 with*

$$\overline{A|_p} := \overline{A} \quad (A|_p)^* s t := A^* s t \wedge \overline{p} s \wedge \overline{p} t$$

## 5 Soundness and Completeness

Now we establish that our translation is faithful, i.e. sound and complete. We will use the terms *sound* and *complete* from the perspective of using a HOL-ATP for theorem proving in DHOL, e.g., *sound* means if  $\overline{F}$  is a HOL-theorem, then  $F$  is a DHOL-theorem, and *complete* is the dual.<sup>3</sup>

The completeness theorem states that our translation preserves all DHOL-judgments. Moreover, the theorem statement clarifies the intuition behind the translations invariants:

**Theorem 1 (Completeness).** *We have*

<i>if in DHOL</i>	<i>then in HOL</i>
$\vdash_T T \text{ Thy}$	$\vdash_{\overline{T}} \overline{T} \text{ Thy}$
$\vdash_T \Gamma \text{ Ctx}$	$\vdash_{\overline{T}} \overline{\Gamma} \text{ Ctx}$
$\Gamma \vdash_T A \text{ tp}$	$\overline{\Gamma} \vdash_{\overline{T}} \overline{A} \text{ tp}$ and $\overline{\Gamma} \vdash_{\overline{T}} A^* : \overline{A} \rightarrow \overline{A} \rightarrow \text{bool}$ and $A^*$ is PER
$\Gamma \vdash_T A \equiv B$	$\overline{\Gamma} \vdash_{\overline{T}} \overline{A} \equiv \overline{B}$ and $\overline{\Gamma}, x, y : \overline{A} \vdash_{\overline{T}} A^* x y =_{\text{bool}} B^* x y$
$\Gamma \vdash_T A <: B$	$\overline{\Gamma} \vdash_{\overline{T}} \overline{A} \equiv \overline{B}$ and $\overline{\Gamma}, x, y : \overline{A} \vdash_{\overline{T}} A^* x y \Rightarrow B^* x y$
$\Gamma \vdash_T t : A$	$\overline{\Gamma} \vdash_{\overline{T}} \overline{t} : \overline{A}$ and $\overline{\Gamma} \vdash_{\overline{T}} A^* \overline{t} \overline{t}$
$\Gamma \vdash_T F$	$\overline{\Gamma} \vdash_{\overline{T}} \overline{F}$

Additionally the substitution lemma holds, i.e.,

$$\Gamma, x : A \vdash_T t : B \text{ and } \Gamma \vdash u : A \quad \text{implies} \quad \overline{\Gamma} \vdash_{\overline{T}} \overline{t[x/u]} =_{\overline{B}} \overline{t}[x/\overline{u}]$$

*Proof.* The proof proceeds by induction and can be found in Appendix B of the extended preprint [20].

<sup>3</sup> If, however, we think of our translation as an interpretation function that maps syntax to semantics, we could also justify swapping the names of the theorems.

The reverse direction is much trickier. To understand why, we look at two canaries in the coal mine that we have used to reject multiple intuitive but untrue conjectures:

*Example 7 (Non-Injectivity of the Translation).* Continuing Example 1, assume terms  $u, v : \text{obj}$  and consider the identify functions  $I_u := \lambda f : \text{mor } u \ u.f$  and  $I_v := \lambda f : \text{mor } v \ v.f$ . Both are translated to the same HOL-term  $\bar{I}_u = \bar{I}_v = \lambda f : \text{mor}.f$  (because  $I_u$  and  $I_v$  only differ in the type indices, which are erased by our translation).

Consequently, the ill-typed DHOL-Boolean  $b := I_u =_{\text{mor } u \ u \rightarrow \text{mor } u \ u} I_v$  is translated to the HOL-Boolean  $\lambda f : \text{mor}.f =_{\text{mor} \rightarrow \text{mor}} \lambda f : \text{mor}.f$ , which is not only well-typed but even a theorem.

To better understand the underlying issue we introduce the notion of *spurious* terms. The well-typed translation  $\bar{t}$  of a DHOL-term  $t$  is called **spurious** if  $t$  is ill-typed (otherwise it is called *proper*). Intuitively, we should be able to use the PERs  $A^*$  to deal with spurious terms: to type-check  $t : A$  in DHOL, we want to use  $A^* \bar{t} \bar{t}$  in HOL. But even that is tricky:

*Example 8 (Trivial PERs for Built-In Base Types).* Consider the property  $\text{bool}^* x x$ . Our translation guarantees  $\text{bool}^* \text{true true}$  and  $\text{bool}^* \text{false false}$ . Thus, we can use Boolean extensionality to prove in HOL that  $\forall x : \text{bool}. \text{bool}^* x x$ , making the property trivial. In particular, we can prove  $\text{bool}^* \bar{b} \bar{b}$  for the spurious Boolean  $b$  from Example 7. Even worse, the property  $(\Pi x : A. B)^* x x$  is trivial in this way whenever it is for  $B$  and thus for all  $n$ -ary bool-valued function types.

More generally, this degeneration effect occurs for every base type that is built into both DHOL and HOL and that is translated to itself. `bool` is the simplest example of that kind, and the only one in the setting described here. But reasonable language extensions like built-in base types  $a$  for numbers, strings, etc. would suffer from the same issue. This is because all of these types would come with built-in induction principles that derive a universal property from its ground instances, at which point  $a^* x x$  becomes trivial.

Note, however, that the degeneration effect does *not* occur for *user-declared* base types. For example, consider a theory that declares a base type  $N$  for the natural numbers and an induction axiom for it.  $N$  would not be translated to itself but to a fresh HOL-type in whose induction axiom the quantifier  $\forall$  is relativized by  $N^* x x$ . Consequently,  $N^* x x$  is not trivial and can be used to reject spurious terms.

These examples show that we cannot expect the reverse directions of the statements in Theorem 1 to hold in general. However, we can show the following property that is sufficient to make our translation well-behaved:

**Theorem 2 (Soundness).** *Assume a well-formed DHOL-theory  $\vdash T$  Thy.*

$$\text{If } \Gamma \vdash_T F : \text{bool} \text{ and } \bar{\Gamma} \vdash_{\bar{T}} \bar{F}, \text{ then } \Gamma \vdash_T F$$

*In particular, if  $\Gamma \vdash_T s : A$  and  $\Gamma \vdash_T t : A$  and  $\bar{\Gamma} \vdash_{\bar{T}} A^* \bar{s} \bar{t}$ , then  $\Gamma \vdash s =_A t$ .*

*Proof.* The key idea is to transform a HOL-proof of  $\overline{F}$  into one that is in the image of the translation, at which point we can read off a DHOL-proof of  $F$ . The full proof is given in Appendix B of the extended preprint [20].

Intuitively, the reverse directions of Theorem 1 holds once we establish that all involved expressions are well-typed in DHOL. Thus, we *can* use a HOL-ATP to prove DHOL-conjectures if we validate independently that the conjecture is well-typed all along. In the remainder of the section, we develop the necessary type-checking algorithm for DHOL.

*Type-Checking.* Inspecting the rules of DHOL, we observe that all DHOL-judgments would be decidable if we had an oracle for the validity judgment  $\Gamma \vdash_T F$ . Indeed, our DHOL-rules are already written in a way that essentially allows reading off a bidirectional type-checking algorithm. It only remains to split the typing judgment  $\Gamma \vdash_T t : A$  into two algorithms for type-inference (which computes  $A$  from  $t$ ) and type-checking (which takes  $t$  and  $A$  and returns yes or no) and to aggregate the rules for subtyping into an appropriate pattern-match.

The construction is routine, and we have implemented the resulting algorithm in our MMT/LF logical framework [12, 19].<sup>4</sup> The oracle for the validity judgment is provided by our translation and a theorem prover for HOL (see Sect. 6). It remains to show that whenever the algorithm calls the oracle for  $\Gamma \vdash_T F$ , we do in fact have that  $\Gamma \vdash_T F : \text{bool}$  so that Theorem 2 is applicable. Formally, we show the following:

**Theorem 3.** *Relative to an oracle for  $\Gamma \vdash_T F$ , consider a derivation of some DHOL-judgment, in which the children of each node are ordered according to the left-to-right order of the assumptions in the statement of the applied rule.*

*If the oracle calls are made in depth-first order, then each such call satisfies  $\Gamma \vdash_T F : \text{bool}$ .*

*Proof.* We actually prove, by induction on derivations, the more general statement requires that each rule preserves the following preconditions:

Judgment	Precondition
$\vdash_T \Gamma \text{Ctx}$	$\vdash T \text{Thy}$
$\Gamma \vdash_T A \text{tp}$	$\vdash_T \Gamma \text{Ctx}$
$\Gamma \vdash_T t : A$	$\Gamma \vdash_T A \text{tp}$ (post-condition when used as type-inference)
$\Gamma \vdash_T F$	$\Gamma \vdash_T F : \text{bool}$
$\Gamma \vdash_T A \equiv B$ or $\Gamma \vdash_T A <: B$	$\Gamma \vdash_T A \text{tp}$ and $\Gamma \vdash_T B \text{tp}$

<sup>4</sup> The formalization of DHOL in MMT is available at [https://gl.mathhub.info/MMT/LATIN2/-/blob/devel/source/logic/hol\\_like/dhol.mmt](https://gl.mathhub.info/MMT/LATIN2/-/blob/devel/source/logic/hol_like/dhol.mmt). The example theories given throughout this paper and a few example conjectures are available at <https://gl.mathhub.info/MMT/LATIN2/-/blob/devel/source/casestudies/2023-cade>.

Note that rules whose conclusion is a validity judgment can be ignored because they are replaced by the oracle anyway.

The most interesting case is the rule for  $\Gamma \vdash_T a s_1 \dots s_n \equiv a t_1 \dots t_n$ . Here, the left-to-right order of assumptions is critical because  $\Gamma \vdash_T s_1 =_{A_1} t_1$  may be needed to show, e.g.,  $\Gamma \vdash_T s_2 =_{A_2[t_1/h_1]} t_2 : \text{bool}$ .

## 6 Theorem Prover Implementation

We have integrated our translation as a preprocessor to the HOL ATP LEO-III [23]. We chose this ATP because its existing preprocessor infrastructure already includes a powerful logic embedding tool [21, 22]. However, with a little more effort, other HOL ATPs work as well.

Furthermore, we developed a bridge between the MMT logical framework [19] and LEO-III (both of which are written in the same programming language). This allows us to use our MMT-based type-checker for DHOL with our Leo-III-based theorem prover to obtain a full-fledge implementation of DHOL. Moreover, this system can immediately use MMT's logic-independent frontend features like IDE and module system.

Alternatively, we can use LEO-III as a general purpose DHOL-ATP that accepts input in TPTP. Even though TPTP does not officially sanction DHOL as a logic, it anticipates dependent function types and already provides syntax for them (although—to our knowledge—no ATP system has made use of it so far). Concretely, TPTP represents the type  $\Pi x : A. B$  as  $!>[X:A]:B$  and a base type  $a t_1 \dots t_n$  as  $a @ t_1 \dots @ t_n$ . TPTP does not yet provide syntax for predicate subtypes, i.e., this approach is currently limited to the no-subtyping fragment of DHOL. But extending the TPTP syntax with predicate subtypes would be straightforward, e.g., by using  $A ?| p$  to represent the type  $A|_p$ .

The encoding of the conjecture given in Example 3 using the theory from Example 1 is given at <https://gl.mathhub.info/MMT/LATIN2/-/blob/devel/source/casestudies/2023-cade/CategoryTheory/category-theory-lemmas-dhol.p> (which also includes further example conjectures relative to the same theory). Running the logic embedding tool translates it into the TPTP TH0 problem given at <https://gl.mathhub.info/MMT/LATIN2/-/blob/devel/source/casestudies/2023-cade/CategoryTheory/category-theory-lemmas-hol.p>. Unsurprisingly, LEO-III can prove this simple theorem easily.

*Practical Evaluation.* In order to evaluate the practical usefulness of the translation we studied various example conjectures about function composition in set theory and category theory. We considered 5 further lemmas based on the theory in Example 1 which are written directly in TPTP and can all be proven by E, Vampire and cvc5. We also studied various harder lemmas about function composition and category theory. Those examples are written in MMT and take advantage of advanced MMT features to improve readability, such as definitions, user-defined notations, and implicit arguments that are inferred by the prover.

The examples can be found at <https://gl.mathhub.info/MMT/LATIN2/-/blob/development/source/casestudies/2023-cade>. The MMT prover successfully type-checks all problems and translates them into TPTP problems to be solved by HOL ATPs.

Since LEO-III can solve none of the 6 function composition examples, we also tested other HOL ATPs on the generated TPTP problems. Running all HOL ATP provers supported at <https://www.tptp.org/cgi-bin/SystemOnTPTP> on the function composition problems shows that many provers can solve 3 of the problems, Vampire can solve 4 of them, and 5 out of the 6 conjectures can be solved by at least one HOL ATP.

We also studied 6 more difficult theorems about limits in category theory including the uniqueness, commutativity, and associativity of some limits. To better evaluate the usefulness of our translation, we also formalized these lemmas in native HOL (in MMT) and compared the results. Naturally, the DHOL formalization is significantly more readable and benefits from the more expressive type system that can help spot mistakes in the formalization. Running the HOL ATPs from <https://www.tptp.org/cgi-bin/SystemOnTPTP> on the generated TPTP problems (with 60 s timeout) yields the results in the table below (where we omit provers that proved none of the theorems in either formalization).

HOL ATP	lemma 1 proven		lemma 2 proven		lemma 3 proven	
	DHOL	native HOL	DHOL	native HOL	DHOL	native HOL
agsyHOL	yes	no	no	no	yes	no
cocATP	yes	no	no	no	no	no
cvc5	yes	yes	no	no	yes	no
cvc5-SAT	yes	no	no	no	no	no
E	yes	yes	no	no	no	yes
HOLyHammer	yes	yes	no	no	yes	yes
Lash	yes	yes	no	no	no	no
LEO-II	yes	no	no	no	no	no
Leo-III	yes	yes	no	no	no	no
Leo-III-SAT	yes	yes	no	no	no	no
Satallax	yes	yes	no	no	yes	no
Vampire	yes	yes	no	no	no	yes
Zipperpin	yes	yes	no	no	yes	yes
total	13	9	0	0	5	4

HOL ATP	lemma 4 proven		lemma 5 proven		lemma 6 proven	
	DHOL	native HOL	DHOL	native HOL	DHOL	native HOL
agsyHOL	no	no	no	no	no	no
cocATP	no	no	no	no	no	no
cvc5	no	yes	no	no	no	no
cvc5-SAT	no	no	no	no	no	no
E	no	yes	no	yes	no	yes
HOLyHammer	no	yes	no	no	no	yes
Lash	no	no	no	no	no	no
LEO-II	no	no	no	no	no	no
Leo-III	no	no	no	no	no	no
Leo-III-SAT	no	no	no	no	no	no
Satallax	no	no	yes	no	no	no
Vampire	no	yes	no	yes	no	yes
Zipperpin	no	yes	yes	yes	no	yes
total	0	5	2	3	0	4

Overall more problems generated from the native HOL formalization can be solved by some HOL ATP (5/6 compared to 3/6 for the DHOL formalization). The HOL ATPs found 25 successful proofs for the native HOL problems and 20 for the DHOL problems. This suggests that current HOL ATPs can prove native HOL problems somewhat better than their translated DHOL counterparts, but not much better. In 8 cases a prover can prove the DHOL conjecture but not the native HOL analogue, indicating that the two formalizations have different advantages.

Furthermore, our translation has so far been engineered for generality and soundness/completeness and not for ATP efficiency. Indeed, future work has multiple options to boost the ATP performance on translated DHOL, e.g., by

- developing sufficient criteria for when simpler HOL theories can be produced
- inserting lemmas into the translated theories that guide proof search in ATPs, e.g., to speed up equality reasoning
- adding definitions to translated DHOL problems and developing better criteria when to expand them

Thus, we consider the test results to be very promising. In particular, the translation could serve as a useful basis for type-checkers and hammer tools for DHOL ITPs.

## 7 Conclusion and Future Work

We have combined two features of standard languages, higher-order logic HOL and dependent type theory DTT, thereby obtaining the new dependently-typed higher-order



logic DHOL. Contrary to HOL, DHOL allows for *dependent* function types. Contrary to DTT, DHOL retains the simplicity of classical Booleans and standard equality.

On the downside, we have to accept that DHOL, unlike both HOL and DTT, has an undecidable type system. Further work will show how big this disadvantage weighs in practical theorem proving applications. But we anticipate that the drawback is manageable, especially if, as in our case, an implementation of DHOL is coupled tightly with a strong ATP system. We accomplish this with a sound and complete translation from DHOL into HOL that enables using existing HOL ATPs to discharge the proof obligations that come up during type-checking. We have implemented our novel translation as a TPTP-to-TPTP preprocessor for HOL ATP systems and outlined the implementation of a type-checker and hammer tool for DHOL based on the resulting prover.

Moreover, once this design is in place, it opens up the possibility to add certain type constructors to DHOL that are often requested by users but difficult to provide for system developers because they automatically make typing undecidable. We have shown an extension of DHOL with predicate subtypes as an example. Quotients, partial functions, or fixed-length lists are other examples that can be supported in future work.

We expect our translation remains sound and complete if DHOL is extended with other features underlying common HOL systems such as built-in types for numbers, the axiom of infinity, or the subtype definition principle. How to extend DHOL with a choice operator remains a question for future work — if solved, this would allow extending existing HOL ITPs to DHOL.

**Acknowledgment.** Chad Brown and Alexander Steen provided valuable feedback on earlier versions of this paper.

## References

1. Andrews, P.: An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof. Academic Press, Cambridge (1986)
2. Andrews, P., Bishop, M., Issar, S., Nesmith, D., Pfenning, F., Xi, H.: TPS: a theorem-proving system for classical type theory. *J. Autom. Reasoning* **16**(3), 321–353 (1996)
3. Jacobs, B., Melham, T.: Translating dependent type theory into higher order logic. In: Bezem, M., Groote, J.F. (eds.) TLCA 1993. LNCS, vol. 664, pp. 209–229. Springer, Heidelberg (1993). <https://doi.org/10.1007/BFb0037108>
4. Brown, C.E.: Satallax: an automatic higher-order prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. Lecture Notes in Computer Science, vol. 7364, pp. 111–117. Springer, Berlin, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31365-3\\_11](https://doi.org/10.1007/978-3-642-31365-3_11)
5. Church, A.: A formulation of the simple theory of types. *J. Symbolic Logic* **5**(1), 56–68 (1940)
6. Constable, R., et al.: Implementing Mathematics with the Nuprl Development System. Prentice-Hall, Hoboken (1986)
7. Coq Development Team: The Coq Proof Assistant: Reference Manual. Technical report, INRIA (2015)
8. Coquand, T., Huet, G.: The calculus of constructions. *Inf. Comput.* **76**(2/3), 95–120 (1988)

9. de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The lean theorem prover (system description). In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 378–388. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-21401-6\\_26](https://doi.org/10.1007/978-3-319-21401-6_26)
10. Gordon, M.: HOL: a proof generating system for higher-order logic. In: Birtwistle, G., Subrahmanyam, P. (eds.) VLSI Specification, Verification and Synthesis, pp. 73–128. Kluwer-Academic Publishers (1988)
11. Gordon, M., Pitts, A.: The HOL logic. In: Gordon, M., Melham, T. (eds.) Introduction to HOL, Part III, pp. 191–232. Cambridge University Press (1993)
12. Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. *J. Assoc. Comput. Mach.* **40**(1), 143–184 (1993)
13. Harrison, J.: HOL light: a tutorial introduction. In: Srivas, M., Camilleri, A. (eds.) FMCAD 1996. LNCS, vol. 1166, pp. 265–269. Springer, Heidelberg (1996). <https://doi.org/10.1007/BFb0031814>
14. Martin-Löf, P.: An intuitionistic theory of types: predicative part. In: Proceedings of the 2073 Logic Colloquium, North-Holland, pp. 73–118 (1974)
15. Norell, U.: The Agda Wiki (2005). <https://wiki.portal.chalmers.se/agda>
16. Owre, S., Rushby, J.M., Shankar, N.: PVS: a prototype verification system. In: Kapur, D. (ed.) CADE 1992. LNCS, vol. 607, pp. 748–752. Springer, Heidelberg (1992). [https://doi.org/10.1007/3-540-55602-8\\_217](https://doi.org/10.1007/3-540-55602-8_217)
17. Paulson, L.C.: Isabelle. LNCS, vol. 828. Springer, Heidelberg (1994). <https://doi.org/10.1007/BFb0030541>
18. Pfenning, F., Schürmann, C.: System description: twelf — a meta-logical framework for deductive systems. In: CADE 1999. LNCS (LNAI), vol. 1632, pp. 202–206. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48660-7\\_14](https://doi.org/10.1007/3-540-48660-7_14)
19. Rabe, F.: A modular type reconstruction algorithm. *ACM Trans. Comput. Logic* **19**(4), 1–43 (2018)
20. Rothgang, C., Rabe, F., Benz Müller, C.: Theorem proving in dependently-typed higher-order logic - extended preprint (2023). [arXiv:2305.15382](https://arxiv.org/abs/2305.15382)
21. Steen, A.: An extensible logic embedding tool for lightweight non-classical reasoning (2022). [arXiv:2203.12352](https://arxiv.org/abs/2203.12352)
22. Steen, A.: Logic embedding tool 1.7 (2022). <https://doi.org/10.5281/zenodo.6139916>
23. Steen, A., Benz Müller, C.: Extensional higher-order paramodulation in Leo-III. *J. Autom. Reasoning* **65**(6), 775–807 (2021)
24. Sutcliffe, G.: The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0. *J. Autom. Reasoning* **43**(4), 337–362 (2009)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

