Semantics for Dependently-Typed HOL

Florian Rabe*[0000−0003−3040−3655]

University of Erlangen-Nuremberg, Computer Science, Germany

Abstract. Both higher-order logic and dependent types are popular features for languages used for the formalization of complex domains. Recently DHOL was introduced as a language that provides the expressivity of dependent types while retaining the general feel of higher-order logic. Since then multiple extensions and theorem provers have been developed, and DHOL was adopted as the first TPTP standard for using dependent types in automated theorem provers.
However, the literature lacks a model theoretical semantics for DHOL. The present paper fills in this gap, covering dependent types, rank-1 polymorphism, and functions with preconditions. It introduces both standard models and an appropriate generalization of Henkin-style models and proves soundness and completeness. Considerable care went into keeping the formulation as close to the usual definitions for HOL and elegant enough to serve as a reference point for language extensions and theorem provers for DHOL.

1 Introduction and Related Work

Motivation Recently dependently-typed higher-order logic (DHOL) was introduced in [RRB23,RRB25] and adopted as a TPTP standard in [RKRS25]. It can be seen as an extension of monomorphic HOL [Chu40,Gor88] with dependent function types $\Pi x : A.\, B$ instead of simple function types $A \to B$. It is designed to stay as simple and as close to HOL as possible while meeting the frequent user demand of supporting dependent types. Contrary to typical formulations of dependent type theory such as Martin-Löf type theory [ML74] and implementations in major proof assistants [Nor05,Roc25,dKA+15], DHOL does not employ a sophisticated treatment of equality that keeps typing decidable. Instead, it uses a straightforward formulation of equality at the cost of making typing undecidable, similarly as in *extensional* Martin-Löf type theory. Because theorem proving, the ultimate goal, is undecidable anyway, that cost can be acceptable.

[RRB23] uses a sound and complete translation of DHOL to HOL to obtain theorem proving support via HOL ATPs. Independent DHOL provers were later developed in [RBK24] and [NBK24], and language extensions with choice [RBK24], subtyping [RR25], and polymorphism [RRK26] have been developed. However, a model-theoretical semantics of DHOL has so far been lacking.

* The author was supported by the FAUstairs project (see https://www.faustairs.fau.de/), funded by the Stiftung Innovation in der Hochschullehre under grant StIL:1001-3096.

Dependent types and undecidable typing make the proof theory finicky. Here a sound and complete model theory provides an alternative characterization of the language and acts as an independent check that no proof rules are missing. It can also serve as a reference semantics for building provers and for developing optimized calculi. Moreover, a model theory for DHOL allows transferring model-theoretical techniques from HOL to DHOL. Individual models can be used as counter-examples for disproving conjectures or as example for establishing the consistency of theories. It also allows using DHOL as a dependently typed higher-order specification language. Finally, we hope that it can help apply model-theoretical techniques to prove language translations into or out of DHOL correct. Dependency erasure translations such as the one used in [RRB23] are difficult to state correctly and even harder to prove correct.

Contribution and Overview In a nutshell, we define the missing model theory and prove soundness and completeness. While the presentation feels natural and smooth to read, significant effort went into identifying the sweet spot where the various subtleties can be treated elegantly. Sect. 2 gives a self-contained definition of DHOL, integrating and extending existing definitions. Then we define strict and lax models in Sect. 3 and 4, and show completeness via a term model construction in Sect. 5.

Lax models are necessary because strict models do not yield completeness for higher-order languages as is well-known for HOL [Hen50,GP93]. Our key idea is that a lax model can interpret functions as arbitrary objects as long as it preserves all their observable properties. We also characterize lax models as certain functors out of the category of contexts.

Our definitions go beyond the language introduced in [RRB23] in two ways. Firstly, we add rank-1 polymorphism as introduced for DHOL in [RRK26], bringing DHOL closer to HOL-based proof assistants, where polymorphism is common. Secondly, we allow for functions to use preconditions about the argument that must be discharged when applied. Such partial functions are desirable for both formalizing mathematics and software verification.

Our work also provides a uniform semantics for the HOL-based languages in the TPTP family of theorem prover standards [Sut22]. We obtain monomorphic HOL [BRS08] if all preconditions are omitted and the parameter lists Φ in theories are always empty. (Then all types can be written $A \to B$, and typing is decidable.) From that, we obtain polymorphic HOL [KRS16] if Φ may contain only type variables, and monomorphic DHOL [RKRS25] if the parameters of type symbols may contain only term variables.

Related Work A few versions of HOL with partial functions were introduced, e.g., [Far93]. The dependently-typed setting is trickier because it requires treating partial *types*. Here function preconditions provide a key missing link to extend DHOL with refinement types (as in PVS) or subtype definitions (as in HOL) as well as with quotient types. While refinement and quotient types can be normalized away in simply-typed HOL (in the sense that only one toplevel

refinement/quotient is needed), [RR25] shows that the normal forms in the presence of dependent types need exactly the kind of preconditions that we work out here. [Hur01] previously made a similar observation in the context of PVS.

Our work provides a complete semantics for a significant fragment of PVS [ORS92]. With dependent types, polymorphism, and preconditions, we cover a major part of the one used in the official PVS semantics [OS97], but the latter does not define lax models and thus is not complete. Similarly, various extensions of HOL with, e.g., refinement types (of which our preconditions are a special case) and polymorphism exist that define incomplete strict model theories, e.g., [Hom09,KP19,Völ07]. [Pop25] defines a complete semantics for polymorphic HOL with refinement type definitions and choice, but without dependent types.

All of the above and our work have in common that they use the category of sets as the target of the semantics. Here some kind of lax interpretation of function types is needed to obtain completeness. Our representation of lax models as functors is inspired by categorical semantics [LS86], which has been used for many type theories in particular those based on Martin-Löf type theory [Hof97]. Such functorial models can be generalized by allowing for larger classes of target categories instead of just sets. Then completeness can be obtained using a strict interpretation of function types but allowing for sufficiently large classes of categories, such as the contextual categories introduced for that purpose [Car86].

As a compromise between these approaches, [AR11] defines a complete semantics using strict interpretations in Kripke-models. It also uses an undecidable dependent type theory but without Booleans or polymorphism.

2 Syntax and Proof Theory

DHOL was introduced in [RRB23,RRB25], and we give a self-contained definition in Fig. 1 (syntax), 2 (judgments), 3 (declarations), 4 (expressions), and 5 (proofs). Intuitively, DHOL arises from HOL by replacing the simple function types with **dependent function** types $\Pi x : A.\, B$; functions $\lambda x : A.\, t$ map each argument $x : A$ to a result in $B(x)$.

In **theories**, users declare **dependent type symbols** $a(\Phi) : \texttt{Type}$, typed **constants** $c(\Phi) : A$, and global **assumptions** (axioms) $(\Phi) \triangleright F$. All declarations may take a context Φ of parameters. The parameters are bound at the beginning of each declaration and can be used in particular to declare type variables or the index types of dependent type symbols. When a parametric declaration is used, a substitution φ for the parameters must be provided to form atomic types $a\,\varphi$ or terms $c\,\varphi : A[\varphi]$, or to deduce $F[\varphi]$ (see Rules I_1, I_2, I_3 in Fig. 4). **Contexts** contain the corresponding declarations but without parameters. We could allow dependent type variables $u(x : A) : \texttt{Type}$ but exclude them here for simplicity.

The order of declarations matters as every declared identifier may occur in subsequent declarations, and well-typedness of declarations may depend on the preceding assumptions. Therefore, they must be *lists* in which different declarations may alternate. This is in contrast to HOL where it suffices to model the context as three *sets* of type variables, term variables, and assumptions.

Theories (each declaration may declare parameters Φ)

T	$::=$	$\cdot$	empty theory
	$\mid$	$T,\ a(\Phi) : \texttt{Type}$	dependent type symbol a
	$\mid$	$T,\ c(\Phi) : A$	typed constant c
	$\mid$	$T,\ (\Phi) \triangleright F$	axiom asserting formula F

Contexts: type variables u, typed variables x, resp. local assumptions

$\Gamma, \Delta, \Phi \quad ::= \cdot \mid \Gamma,\ u : \texttt{Type} \mid \Gamma,\ x : A \mid \Gamma,\ \triangleright F$

Substitutions: types for type variables, terms for term variables

$\gamma, \varphi \quad ::= \cdot \mid \gamma,\ A \mid \gamma,\ t$

Types

A, B	$::=$	$\texttt{bool}$	Booleans/formulas
	$\mid$	$a\ \varphi$	atomic type: type symbol applied to parameters
	$\mid$	u	type variable
	$\mid$	$\Pi x : A \mid F.\ B$	dependent function type, with precondition F

Terms (including formulas)

s, t, F, G	$::=$	$c\ \varphi$	constant applied to parameters
	$\mid$	x	term variable
	$\mid$	$\lambda x : A \mid F.\ t$	function formation, with precondition F
	$\mid$	$t\ s$	function application
	$\mid$	$s =_A t$	equality of terms of type A
	$\mid$	$F \Rightarrow G$	dependent implication

Abbreviation	Definiens
$\texttt{true}$	$(\lambda x : \texttt{bool}.\ x) =_{\texttt{bool}} (\lambda x : \texttt{bool}.\ x)$
$\texttt{false}$	$(\lambda x : \texttt{bool}.\ x) =_{\texttt{bool}} (\lambda x : \texttt{bool}.\ \texttt{true})$
$\neg F$	$F =_{\texttt{bool}} \texttt{false}$
$F \vee G$	$\neg F \Rightarrow G$
$F \wedge G$	$\neg(F \Rightarrow \neg G)$
$\forall x : A.F$	$(\lambda x : A.\ F) =_{A \to \texttt{bool}} (\lambda x : A.\ \texttt{true})$
$\exists x : A.F$	$\neg \forall x : A.\neg F$
$\Pi x : A.\ B$	$\Pi x : A \mid \texttt{true}.\ B$
$\lambda x : A.\ t$	$\lambda x : A \mid \texttt{true}.\ t$
$A \to B$	$\Pi x : A.\ B$ (for x not free in B)

Fig. 1. Grammar and Defined Operations

A **substitution** $\vdash \gamma : \Gamma \to \Delta$ is a list of Δ-types/terms corresponding to the declarations in Γ. For a type/term in context Γ, we write $A[\gamma]$ and $t[\gamma]$ for the simultaneous substitution of all Γ-variables with the types/terms in γ. We simply write the common case where γ only substitutes a term s for the last variable in Γ as $A[s]$ resp. $t[s]$. (Note that some authors write $\gamma : \Delta \to \Gamma$

Judgment	Intuition	Equality Judgment
$\vdash T : \texttt{Thy}$	T is a theory	
$\vdash_T \Gamma\ \texttt{Ctx}$	Γ is a context	$\vdash_T \Gamma \equiv \Gamma'$
$\vdash_T \gamma : \Gamma \to \Delta$	γ substitutes Γ-variables with Δ-terms/types	$\vdash_T \gamma \equiv \gamma' : \Gamma \to \Delta$
$\Gamma \vdash_T \varphi : \Phi$	abbreviation for $\vdash (id_\Gamma, \varphi) : (\Gamma, \Phi) \to \Gamma$	$\Gamma \vdash_T \varphi \equiv \varphi' : \Phi$
$\Gamma \vdash_T A : \texttt{Type}$	A is a type	$\Gamma \vdash_T A \equiv A'$
$\Gamma \vdash_T t : A$	term t has type A	$\Gamma \vdash t =_A t'$
$\Gamma \vdash F$	F is a theorem (including the special case where F is $t =_A t'$)	

equality of terms $s =_A t : \texttt{bool}$ is a term; the $\equiv$ judgments are not terms

Fig. 2. DHOL Judgments (T will be dropped if clear from context)

Theory formation

$$\frac{}{\vdash \cdot : \texttt{Thy}} \qquad \frac{\vdash T\ \texttt{Thy} \quad a \notin T \quad \vdash_T \Phi\ \texttt{Ctx}}{\vdash T, a(\Phi) : \texttt{Type}\ \texttt{Thy}} \qquad \frac{\vdash T\ \texttt{Thy} \quad c \notin T \quad \vdash_T \Phi\ \texttt{Ctx} \quad \Gamma \vdash_T A : \texttt{Type}}{\vdash T, c(\Phi) : A\ \texttt{Thy}} \qquad \frac{\vdash T\ \texttt{Thy} \quad \vdash_T \Phi\ \texttt{Ctx} \quad \Gamma \vdash_T F : \texttt{bool}}{\vdash T, (\Phi) \triangleright F\ \texttt{Thy}}$$

Contexts: formation and congruence:

$$\frac{\vdash T\ \texttt{Thy}}{\vdash_T \cdot\ \texttt{Ctx}} \qquad \frac{\vdash T : \texttt{Thy}}{\vdash_T \cdot \equiv \cdot}$$

$$\frac{\vdash_T \Gamma\ \texttt{Ctx} \quad u \notin \Gamma}{\vdash_T \Gamma, u : \texttt{Type}\ \texttt{Ctx}} \qquad \frac{\vdash_T \Gamma \equiv \Gamma'}{\vdash_T \Gamma, u : \texttt{Type} \equiv \Gamma', u : \texttt{Type}}$$

$$\frac{\vdash_T \Gamma\ \texttt{Ctx} \quad x \notin \Gamma \quad \Gamma \vdash_T A : \texttt{Type}}{\vdash_T \Gamma, x : A\ \texttt{Ctx}} \qquad \frac{\vdash_T \Gamma \equiv \Gamma' \quad \Gamma \vdash_T A \equiv A'}{\vdash_T \Gamma, x : A \equiv \Gamma', x : A'}$$

$$\frac{\vdash_T \Gamma\ \texttt{Ctx} \quad \Gamma \vdash_T F : \texttt{bool}}{\vdash_T \Gamma, \triangleright F\ \texttt{Ctx}} \qquad \frac{\vdash_T \Gamma \equiv \Gamma' \quad \Gamma \vdash_T F =_{\texttt{bool}} F'}{\vdash_T \Gamma, \triangleright F \equiv \Gamma', \triangleright F'}$$

Substitutions: formation and congruence:

$$\frac{\vdash_T \Delta : \texttt{Ctx}}{\vdash_T \cdot : \cdot \to \Delta} \qquad \frac{\vdash_T \Delta : \texttt{Ctx}}{\vdash_T \cdot \equiv \cdot : \cdot \to \Delta}$$

$$\frac{\vdash_T \gamma : \Gamma \to \Delta \quad \Delta \vdash_T A : \texttt{Type}}{\vdash_T (\gamma, A) : (\Gamma, u : \texttt{Type}) \to \Delta} \qquad \frac{\vdash_T \gamma \equiv \gamma' : \Gamma \to \Delta \quad \Delta \vdash_T A \equiv A'}{\vdash_T (\gamma, A) \equiv (\gamma', A') : (\Gamma, u : \texttt{Type}) \to \Delta}$$

$$\frac{\vdash_T \gamma : \Gamma \to \Delta \quad \Gamma \vdash A : \texttt{Type} \quad \Delta \vdash_T t : A[\gamma]}{\vdash_T, (\gamma, t) : (\Gamma, x : A) \to \Delta} \qquad \frac{\vdash_T \gamma \equiv \gamma' : \Gamma \to \Delta \quad \Gamma \vdash A : \texttt{Type} \quad \Delta \vdash_T t =_{A[\gamma]} t'}{\vdash_T (\gamma, t) \equiv (\gamma', t') : (\Gamma, x : A) \to \Delta}$$

$$\frac{\vdash_T \gamma : \Gamma \to \Delta \quad \Gamma \vdash F : \texttt{bool} \quad \Delta \vdash_T F[\gamma]}{\vdash_T \gamma : (\Gamma, \triangleright F) \to \Delta}\ (S3) \qquad \frac{\vdash_T \gamma \equiv \gamma' : \Gamma \to \Delta \quad \Gamma \vdash F : \texttt{bool} \quad \Delta \vdash_T F[\gamma]}{\vdash_T \gamma \equiv \gamma' : (\Gamma, \triangleright F) \to \Delta}$$

Fig. 3. Rules for theories, contexts, substitutions

instead. Our notation $\gamma : \Gamma \to \Delta$ emphasizes the direction in which substitutions transport syntax across contexts.) If Γ contains local assumptions, these must be discharged when forming the substitution γ, see Rule S_3 in Fig. 3.

The most consequential rule of DHOL is the **congruence rule for atomic types**, Rule C in Fig. 4: It says that two instances $a\ \varphi$ and $a\ \varphi'$ of a type symbol

Types: formation and congruence:

$$\frac{a(\Phi) : \mathtt{Type} \in T \qquad \Gamma \vdash_T \varphi : \Phi}{\Gamma \vdash_T a\ \varphi : \mathtt{Type}}(I_1) \qquad \frac{\Gamma \vdash_T \varphi \equiv \varphi' : \Phi}{\Gamma \vdash_T a\ \varphi \equiv a\ \varphi'}(C)$$

$$\frac{\vdash_T \Gamma\ \mathtt{Ctx} \qquad u : \mathtt{Type} \in \Gamma}{\Gamma \vdash_T u : \mathtt{Type}} \qquad \frac{\vdash_T \Gamma\ \mathtt{Ctx} \qquad u : \mathtt{Type} \in \Gamma}{\Gamma \vdash_T u \equiv u}$$

$$\frac{\Gamma \vdash_T A : \mathtt{Type} \qquad \Gamma, x : A \vdash_T F : \mathtt{bool} \qquad \Gamma, x : A, \triangleright F \vdash_T B : \mathtt{Type}}{\Gamma \vdash \Pi x : A | F.\ B : \mathtt{Type}}(P_1)$$

$$\frac{\Gamma \vdash_T A \equiv A' \qquad \Gamma, x : A \vdash_T F =_{\mathtt{bool}} F' \qquad \Gamma, x : A, \triangleright F \vdash_T B \equiv B'}{\Gamma \vdash \Pi x : A | F.\ B \equiv \Pi x : A' | F'.\ B'}$$

$$\frac{\vdash_T \Gamma\ \mathtt{Ctx}}{\Gamma \vdash_T \mathtt{bool} : \mathtt{Type}} \qquad \frac{\vdash_T \Gamma\ \mathtt{Ctx}}{\Gamma \vdash_T \mathtt{bool} \equiv \mathtt{bool}}$$

Terms: identifiers and congruence

$$\frac{c(\Phi) : A \in T \qquad \Gamma \vdash_T \varphi : \Phi}{\Gamma \vdash_T c\ \varphi : A[\varphi]}(I_2) \qquad \frac{c(\Phi) : A \in T \qquad \Gamma \vdash_T \varphi \equiv \varphi' : \Phi}{\Gamma \vdash_T c\ \varphi =_{A[\varphi]} c\ \varphi'}$$

$$\frac{x : A \in \Gamma \qquad \vdash_T \Gamma\ \mathtt{Ctx}}{\Gamma \vdash_T x : A} \qquad \frac{x : A \in \Gamma \qquad \vdash_T \Gamma\ \mathtt{Ctx}}{\Gamma \vdash_T x =_A x}$$

Terms: function formation, applications, congruence, function checking, Booleans

$$\frac{\Gamma \vdash_T A : \mathtt{Type} \qquad \Gamma, x : A \vdash_T F : \mathtt{bool} \qquad \Gamma, x : A, \triangleright F \vdash_T t : B}{\Gamma \vdash \lambda x : A | F.\ t : \Pi x : A | F.\ B}(P_2)$$

$$\frac{\Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash_T F =_{\mathtt{bool}} F' \qquad \Gamma, x : A, \triangleright F \vdash_T t =_B t'}{\Gamma \vdash \lambda x : A | F.\ t =_{\Pi x : A | F.\ B} \lambda x : A' | F'.\ t'}(\xi)$$

$$\frac{\Gamma \vdash_T t : \Pi x : A | F.\ B \qquad \Gamma \vdash_T s : A \qquad \Gamma \vdash_T F[s]}{\Gamma \vdash_T t\ s : B[s]}(P_3)$$

$$\frac{\Gamma \vdash_T t =_{\Pi x : A | F.\ B} t' \qquad \Gamma \vdash_T s =_A s' \qquad \Gamma \vdash_T F[s]}{\Gamma \vdash_T t\ s =_{B[s]} t'\ s'}$$

$$\frac{\Gamma, x : A, \triangleright F \vdash_T t\ x : B}{\Gamma \vdash_T t : \Pi x : A | F.\ B}(P_4) \qquad \frac{\Gamma \vdash_T t : A \qquad \Gamma \vdash_T t' : A}{\Gamma \vdash t =_A t' : \mathtt{bool}} \qquad \frac{\Gamma \vdash_T F : \mathtt{bool} \qquad \Gamma, \triangleright F \vdash_T G : \mathtt{bool}}{\Gamma \vdash_T F \Rightarrow G : \mathtt{bool}}(D)$$

Fig. 4. Rules for types and terms

$$\frac{(\Phi) \triangleright\ F \in T \qquad \Gamma \vdash_T \varphi : \Phi}{\Gamma \vdash_T F[\varphi]}(I_3) \qquad \frac{\triangleright F \in \Gamma \qquad \vdash_T \Gamma\ \mathtt{Ctx}}{\Gamma \vdash_T F} \qquad \frac{\Gamma \vdash_T t : A \qquad \Gamma \vdash_T A \equiv A'}{\Gamma \vdash t : A'}$$

$$\frac{\Gamma, \triangleright F \vdash_T G \qquad \Gamma, \triangleright G \vdash_T F}{\Gamma \vdash_T F =_{\mathtt{bool}} G}(\mathrm{PE}) \qquad \frac{\Gamma \vdash_T F =_{\mathtt{bool}} G \qquad \Gamma \vdash_T F}{\Gamma \vdash_T G} \qquad \frac{\Gamma \vdash_T s =_A t}{\Gamma \vdash_T t =_A s} \qquad \frac{\Gamma \vdash_T s =_A t \qquad \Gamma \vdash_T t =_A u}{\Gamma \vdash_T s =_A u}$$

$$\frac{\Gamma, \triangleright F \vdash_T G}{\Gamma \vdash_T F \Rightarrow G} \qquad \frac{\Gamma \vdash_T F \Rightarrow G \qquad \Gamma \vdash_T F}{\Gamma \vdash_T G} \qquad \frac{\Gamma \vdash_T F : \mathtt{bool}}{\Gamma \vdash_T F \vee \neg F}(\mathrm{TND})$$

$$\frac{\Gamma, x : A, \triangleright F \vdash_T t : B \qquad \Gamma \vdash_T s : A \qquad \Gamma \vdash_T F[s]}{\Gamma \vdash_T (\lambda x : A | F.\ t)\ s =_{B[s]} t[s]}(\beta) \qquad \frac{\Gamma \vdash_T t : \Pi x : A | F.\ B}{\Gamma \vdash t =_{\Pi x : A | F.\ B} \lambda x : A | F.\ (t\ x)}(\eta)$$

Fig. 5. Proof rules

are equal if their arguments are. This makes equality of types depend on equality of terms, which makes typing undecidable.

DHOL uses a single **equality** $s =_A t$ for typed terms, which behaves like that of FOL or HOL. Following [And86], DHOL defines the usual connectives from equality as shown in Fig. 1. However, contrary to HOL, we have to also make

one binary connective primitive because DHOL requires **dependent binary connectives**: because typing may depend on assumptions, in the dependent implication $F \Rightarrow G$, the well-formedness of G must be allowed to depend on the truth of F, see Rule D in Fig. 4. This is important, e.g., to state a formula like $x =_A y \Rightarrow f(x) =_{B[x]} f(y)$ for a dependent function $f : \Pi x : A.\, B$. While this formula is easy to prove, it is tricky to show that it is well-formed at all: $f(x) =_{B[x]} f(y)$ is only well-formed if $f(x) : B(x)$ and $f(y) : B(y)$ have the same type; that follows from the assumption $x =_A y$ using Rule C; but we need Rule D to actually appeal to that assumption while type-checking $f(x) =_{B[x]} f(y)$. There is no known way to define the dependent connectives from equality alone. Our definitions make the other binary connectives dependent as well: G in $F \wedge G$ may assume F, and G in $F \vee G$ may $\neg F$.

We allow for functions to have **preconditions**: In $\Pi x : A|F.\, B$ and $\lambda x : A|F.\, t$, the values of x are guarded by the condition F, and F may be assumed to show that B or t is well-formed (see Rules P_1, P_2 in Fig. 4). F defaults to `true` when omitted, yielding the usual dependent function types. The precondition must be discharged during function application (see Rule P_3 in Fig. 4), which is another source of undecidability in type-checking. This leads to **precondition-subtyping**: a term of type $\Pi x : A|F.\, B$ also has type $\Pi x : A|G.\, B$ if $G \Rightarrow F$. However, we can handle that via Rule P_4 for type-driven type-checking (which is almost but not quite derivable from η) without any explicit subtype reasoning.

Example 1 (Vectors). The following theory specifies a dependent monomorphic type $\mathtt{MVec}\, n$ of vectors of length n over type U:

$$\begin{array}{l} \mathtt{N} : \mathtt{Type} \quad + : \mathtt{N} \to \mathtt{N} \to \mathtt{N} \quad U : \mathtt{Type} \quad \mathtt{MVec}(n : \mathtt{N}) : \mathtt{Type} \\ \mathtt{cc} : \Pi m : \mathtt{N}.\, \Pi n : \mathtt{N}.\, \mathtt{MVec}\, m \to \mathtt{MVec}\, n \to \mathtt{MVec}\, (m + n) \end{array}$$

Here we simplify and write $\mathtt{N} : \mathtt{Type}$ instead of $\mathtt{N}(\cdot) : \mathtt{Type}$ for a type without parameters. Now associativity for concatenation

$$\mathtt{cc}\, l\, (m + n)\, x\, (\mathtt{cc}\, m\, n\, y\, z) =_{\mathtt{MVec}\, l + (m+n)} \mathtt{cc}\, (l + m)\, n\, (\mathtt{cc}\, l\, m\, x\, y)\, z$$

is well-typed only if associativity $l + (m + n) =_{\mathtt{N}} (l + m) + n$ for addition holds.

For polymorphic vectors, we modify it as follows:

$$\begin{array}{l} \mathtt{PVec}(u : \mathtt{Type}, n : \mathtt{N}) : \mathtt{Type} \\ \mathtt{pconcat}(u : \mathtt{Type}) : \Pi m : \mathtt{N}.\, \Pi n : \mathtt{N}.\, \mathtt{PVec}\, u\, m \to \mathtt{PVec}\, u\, n \to \mathtt{PVec}\, u\, (m + n) \end{array}$$

If we only have integers `Z` instead of `N`, we can mimic natural numbers using assumptions in declarations and preconditions in bindings:

$$\begin{array}{l} \mathtt{MVec}(n : \mathtt{Z}, \triangleright\, n \geq 0) : \mathtt{Type} \\ \mathtt{cc} : \Pi m : \mathtt{Z} | m \geq 0.\, \Pi n : \mathtt{Z} | n \geq 0.\, \mathtt{MVec}\, m \to \mathtt{MVec}\, n \to \mathtt{MVec}\, (m + n) \end{array}$$

Remark 1 (Subtyping). Following [RR25], we can add refinement types $A|p$ for $p : A \to \mathtt{bool}$ and quotient types A/r for $r : A \to A \to \mathtt{bool}$. In contrast to [RR25], we can then normalize all types into the form $A|p/r$ for a DHOL-type A. The key step is to normalize $\Pi x : (A|p).\, B$ by turning p into a precondition.

The **proof rules** in Fig. 5 include the rules for lookup of axioms in theory or context, type conversion, the usual classical proof rules for equality and implication, propositional extensionality (PE), TND, and β and η for functions (functional extensionality is derivable). The usual proof rules for defined connectives are derivable. As usual for dependently-typed languages and contrary to HOL, we allow for *empty types*. Thus, e.g., $(\forall x : A.F) \Rightarrow (\exists x : A.F)$ is not a theorem. This requires no special treatment in the proof system—we would have to add an axiom to exclude empty types.

3 Strict Models

Syntax	Semantics
theory T	class $[\![T]\!]$ of models M
context Γ	class $[\![\Gamma]\!]^M$ of assignments α for Γ into M
substitution $\gamma : \Gamma \to \Delta$	function $[\![\gamma]\!]^M : [\![\Delta]\!]^M \to [\![\Gamma]\!]^M$
type $\Gamma \vdash A : \mathtt{Type}$	set $[\![A]\!]^M_\alpha \in \mathcal{U}$ for $\alpha \in [\![\Gamma]\!]^M$
term $\Gamma \vdash t : A$	element $[\![t]\!]^M_\alpha \in [\![A]\!]^M_\alpha$ for $\alpha \in [\![\Gamma]\!]^M$

Fig. 6. Structure of an interpretation function for universe $\mathcal{U}$

Our definition of model follows the well-known structure of models for higher-order logic [Gor88,GP93]. Essentially, a strict model interprets the identifiers and induces an inductive interpretation function for all expressions. We use a universe $\mathcal{U}$ of sets, whose main role is to limit the range of type variables and ensure that all polymorphic expressions are interpreted as sets (and not classes). Thus, every type variable ranges over $\mathcal{U}$; because $\mathcal{U}$ can be arbitrarily large, this is not a serious restriction. The following definition makes this precise:

Definition 1 (Interpretation Function). *An interpretation function for a theory T consists of a set $\mathcal{U}$ and a family of mappings defined for well-typed contexts, substitutions, types, and terms as described in Fig. 6.*

Because terms can occur in types, and because well-formedness depends on the assumptions in the context, and because contexts may contain alternations of variable declarations and assumptions, the definitions of models and interpretation functions are all mutually recursive, using a joint induction on derivations. However, it is didactically preferable to state all definitions separately. We eventually prove well-definedness in Thm. 1.

First we define a strict model of T as a tuple that contains one interpretation for every T-identifier and satisfies all axioms. Because models and assignments are tuples, we frequently need to **extend a tuple** with an additional element, and we introduce a notation for it: If t is a tuple $(t_1, \ldots, t_n)$, we write $t +^n a$ for the tuple $(t_1, \ldots, t_n, a)$; we omit n from the notation from now on.

Definition 2 (Strict Models). *We define $[\![T]\!]$ by induction on T:*
- $[\![\cdot]\!] = \{()\}$ *(empty tuple models the empty theory)*

- $[\![T,\, a(\Phi) : \texttt{Type}]\!] = \{M + m \text{ where } M \in [\![T]\!],\, m : [\![\Phi]\!]^M \to \mathcal{U}\}$ *(models interpret type constructors a as $[\![\Phi]\!]^M$-indexed sets)*
- $[\![T,\, c(\Phi) : A]\!] = \{M + m \text{ where } M \in [\![T]\!],\, m : (\alpha \in [\![\Phi]\!]^M) \to [\![A]\!]^M_\alpha\}$ *(models interpret constants c as $[\![\Phi]\!]^M$-indexed elements)*
- $[\![T,\, (\Phi) \triangleright F]\!] = \{M \in [\![T]\!] \mid [\![F]\!]^M_\alpha = 1 \text{ for all } \alpha \in [\![\Phi]\!]^M\}$ *(models must satisfy axioms), where $1 \in [\![\texttt{bool}]\!]^M$ is the designated truth value.*

Given $M \in [\![T]\!]$ and a T-identifier a or c, we write a^M resp. c^M for the corresponding component of M.

The interpretation of a context Γ is the set $[\![\Gamma]\!]^M$ of assignments that map Γ-variables to appropriate elements in M. An assignment is a tuple that contains one such element for every variable in Γ and satisfies all assumptions in Γ. A substitution $\gamma : \Gamma \to \Delta$ is interpreted contravariantly as the function that composes γ with a Δ-assignment to obtain a Γ-assignment:

Definition 3 (Assignments and Interpretation of Contexts). *We define $[\![\Gamma]\!]^M$ by induction on Γ:*

- $[\![\cdot]\!]^M = \{()\}$ *(empty assignment for the empty context)*
- $[\![\Gamma,\, u : \texttt{Type}]\!]^M = \{\alpha + m \text{ where } \alpha \in [\![\Gamma]\!]^M,\, m \in \mathcal{U}\}$ *(assignments map type variables u to sets in the universe)*
- $[\![\Gamma,\, x : A]\!]^M = \{\alpha + m \text{ where } \alpha \in [\![\Gamma]\!]^M,\, m \in [\![A]\!]^M_\alpha\}$ *(assignments map variables x to elements)*
- $[\![\Gamma, \triangleright F]\!]^M = \{\alpha \in [\![\Gamma]\!]^M \mid [\![F]\!]^M_\alpha = 1\}$ *(assignments must satisfy assumptions)*

Given $\alpha \in [\![\Gamma]\!]^M$ and a Γ-variable x, we write x^α for the corresponding component of α.

For a substitution $\vdash (e_1, \ldots, e_n) : \Gamma \to \Delta$ (where each e_i is a type/term), the mapping $[\![e_1, \ldots, e_n]\!]^M$ is defined by $[\![\Delta]\!]^M \ni \alpha \mapsto ([\![e_1]\!]^M_\alpha, \ldots, [\![e_n]\!]^M_\alpha) \in [\![\Gamma]\!]^M$.

We interpret a partial substitution $\Gamma \vdash \varphi : \Phi$ for $\alpha \in [\![\Gamma]\!]^M$ as the corresponding partial assignment: $[\![\varphi]\!]^M_\alpha = \{(m_1, \ldots, m_n) : \alpha + m_1 + \ldots + m_n \in [\![\Gamma, \Phi]\!]^M\}$.

Example 2 (Vectors). A model M for the theory from Ex. 1 for `MVec` is given by the tuple $\big(\mathbb{N},\ +,\ \mathbb{R},\ n \mapsto \mathbb{R}^n,\ m \mapsto n \mapsto (v \in \mathbb{R}^m) \mapsto (w \in \mathbb{R}^n) \mapsto \binom{v}{w}\big)$. Here we omit $() \mapsto$ when giving the interpretation of a type without arguments.

Types and terms in context Γ are interpreted relative to a model M and an assignment $\alpha \in [\![\Gamma]\!]^M$:

Definition 4 (Interpretation of Types and Terms). *For types and terms, we define $[\![-]\!]^M_\alpha$ by induction:*

- $[\![\texttt{bool}]\!]^M_\alpha$ *is some 2-element set, whose elements we write as 0 and 1,*
- $[\![a\ \varphi]\!]^M_\alpha = a^M([\![\varphi]\!]^M_\alpha)$
- $[\![u]\!]^M_\alpha = u^\alpha$
- $[\![\Pi x : A | F.\, B]\!]^M_\alpha = \{f : \{m \in [\![A]\!]^M \mid [\![F]\!]^M_{\alpha+m} = 1\} \ni m \mapsto f(m) \in [\![B]\!]^M_{\alpha+m}\}$ *(i.e., the set of functions f mapping elements m in the interpretation of A that satisfy F to elements in the interpretation of B at m).*
- $[\![c\ \varphi]\!]^M_\alpha = c^M([\![\varphi]\!]^M_\alpha)$

– $[\![x]\!]^M_\alpha = x^\alpha$
– $[\![\lambda x : A | F.\, t]\!]^M_\alpha = \{m \in [\![A]\!]^M_\alpha \mid [\![F]\!]^M_{\alpha+m} = 1\} \ni m \mapsto [\![t]\!]^M_{\alpha+m}$
– $[\![t\ s]\!]^M_\alpha = [\![t]\!]^M_\alpha([\![s]\!]^M_\alpha)$
– $[\![s =_A t]\!]^M_\alpha = 1$ *iff* $[\![s]\!]^M_\alpha = [\![t]\!]^M_\alpha$
– $[\![F \Rightarrow G]\!]^M_\alpha = 1$ *iff* $[\![F]\!]^M_\alpha = 0$, *or* $[\![F]\!]^M_\alpha = 1$ *and* $[\![G]\!]^M_\alpha = 1$

In the case for implication, note that $[\![F]\!]^M_\alpha = 1$ *implies* $\alpha \in [\![\Gamma, \triangleright F]\!]$, *which is needed to use* α *to interpret* G *(cf. Rule D in Fig. 4).*

We usually use $[\![\texttt{bool}]\!]^M_\alpha := \{0, 1\}$. But keeping its elements unspecified can make it easier to construct models.

As mentioned above, the above definitions are mutually recursive. The devil is in the details, and we must prove well-definedness, the substitution property, and even soundness in a single big induction (given in the appendix):

Theorem 1 (Well-Definedness, Substitution, and Soundness). *Consider a universe* $\mathcal{U}$ *that contains a 2-element set and is closed under subsets and under dependent sums of families of sets. Then the interpretation function induced by a model* $M \in [\![T]\!]$

– *is well-defined (preserves typing): the interpretation function is defined for all well-typed arguments, and it satisfies*

$$\begin{aligned} [\![A]\!]^M_\alpha &\in \mathcal{U} && \text{if } \Gamma \vdash_T A : \texttt{Type} \\ [\![t]\!]^M_\alpha &\in [\![A]\!]^M_\alpha && \text{if } \Gamma \vdash_T t : A \\ [\![\Gamma]\!]^M &\in \mathcal{SET} && \text{if } \vdash_T \Gamma\ \texttt{Ctx} \\ [\![\gamma]\!]^M : [\![\Delta]\!]^M &\to [\![\Gamma]\!]^M && \text{if } \vdash_T \gamma : \Gamma \to \Delta \end{aligned}$$

– *commutes with substitutions* $\vdash \gamma : \Gamma \to \Delta$*: for all* $\beta \in [\![\Delta]\!]^M$

$$\begin{aligned} [\![t[\gamma]]\!]^M_\beta &= [\![t]\!]^M_{[\![\gamma]\!]^M(\beta)} && \text{if } \Gamma \vdash_T t : A \\ [\![A[\gamma]]\!]^M_\beta &= [\![A]\!]^M_{[\![\gamma]\!]^M(\beta)} && \text{if } \Gamma \vdash A : \texttt{Type} \end{aligned}$$

– *preserves equality: whenever the involved expressions are well-typed*

$$\begin{aligned} [\![A]\!]^M_\alpha &= [\![A']\!]^M_\alpha && \text{if } \Gamma \vdash_T A \equiv A' \\ [\![t]\!]^M_\alpha &= [\![t']\!]^M_\alpha && \text{if } \Gamma \vdash_T t =_A t' \\ [\![\Gamma]\!]^M &= [\![\Gamma']\!]^M && \text{if } \vdash_T \Gamma \equiv \Gamma' \\ [\![\gamma]\!]^M &= [\![\gamma']\!]^M && \text{if } \vdash_T \gamma \equiv \gamma' \end{aligned}$$

(where the case for terms is a special case of soundness)

– *is sound: if* $\Gamma \vdash_T F : \texttt{bool}$

$$[\![F]\!]^M_\alpha = 1 \qquad \text{if } \Gamma \vdash_T F$$

where α *ranges over the respective* $[\![\Gamma]\!]^M$ *where applicable.*

4 Lax Models

For HOL, it is relatively easy to define lax models, originally due to Henkin for the monomorphic case [Hen50], and later extended to polymorphic HOL [GP93]. The key idea is allow function types $A \to B$ to be interpreted as sufficiently large *subsets* of the set of functions from $[\![B]\!]^M$ to $[\![A]\!]^M$. This allows retaining the definitions for strict models with few changes. Generalizing this to DHOL is more difficult because DHOL-types cannot be interpreted in isolation and must always be treated in joint inductions with terms (because of dependent types) and provability (because of undecidable typing). After a lot of trial and error, we found an elegant variant, whose key idea is to allow interpreting function types as arbitrary sets as long as their elements still behave like functions.

Definition 5 (Lax Models). *Consider an interpretation function W for universe $\mathcal{U}$ for a theory T, like in Fig. 6, that is defined for all well-typed syntax. We say that W **preserves typing**, **commutes with substitution**, **preserves equality**, or is **sound** if it satisfies the respective statement of Thm. 1.*

*We say W has **standard variables** if it satisfies the interpretation rules for contexts and substitutions from Def. 3 as well as the interpretation rules for type/term variables from Def. 4. We say W has **standard Booleans** if it satisfies the interpretation rules for* `bool`*, $=_A$, and $\Rightarrow$ from Def. 4.*

*Finally, we call W a **lax model** if it has all of the above properties. We say that a lax model is **extensional** if $[\![f\,x]\!]^W_{\alpha+m} = [\![g\,x]\!]^W_{\alpha+m}$ for all m implies $[\![f]\!]^W_\alpha = [\![g]\!]^W_\alpha$. (In particular, every strict model is an extensional lax model.)*

Notably lacking in Def. 5 is a requirement about function types and terms. A lax model is free to deviate from the interpretation rules for Π, λ, and application arbitrarily. This is not as dramatic as it may seem because the interpretations of functions still behave like functions, at least in the simply-typed case: Using the substitution property (which critically applies to type variables as well), we can define the operation $Fun : \mathcal{U}^2 \to \mathcal{U}$ by $(P, Q) \mapsto [\![u \to v]\!]^W_{(P,Q)}$ in context $u :$ `Type`, $v :$ `Type` and have $[\![A \to B]\!]^W_\alpha = Fun([\![A]\!]^W_\alpha, [\![B]\!]^W_\alpha)$. Accordingly, we can define an operator $App : (P, Q, r, s) \mapsto [\![f\,x]\!]^W_{(P,Q,r,s)}$ in context $u :$ `Type`, $v :$ `Type`, $f : u \to v$, $x : u$ and have $[\![f\,t]\!]^W_\alpha = App([\![A]\!]^W_\alpha, [\![B]\!]^W_\alpha, [\![f]\!]^W_\alpha, [\![t]\!]^W_\alpha)$. Then a term $f : A \to B$ might be interpreted as a non-function but always induces an actual function $\overline{f} : m \mapsto App([\![A]\!]^W_\alpha, [\![B]\!]^W_\alpha, [\![f]\!]^W_\alpha, m)$, and the rules for functions show that $[\![f]\!]^W_\alpha$ and $\overline{f}$ store essentially the same information. For dependent functions, the situation is less clear because the substitution property cannot be applied to constrain the interpretation of the binders Π and λ.

If W is extensional, $[\![f]\!]^W_\alpha$ is entirely determined by $\overline{f}$. But while we have not proved it, we suspect that lax models are not automatically extensional. A candidate for a counter-example is a model that interprets every function term f as a pair of a function and some syntactic information about f.

Because all connectives from Fig. 1 are defined via simple functions and equality, we can show that they have the usual semantics in extensional models:

Theorem 2 (Defined Connectives). *If W is extensional, we have*
- $[\![\texttt{true}]\!]^W_\alpha = 1$
- $[\![\texttt{false}]\!]^W_\alpha = 0$
- $[\![\neg F]\!]^W_\alpha = 1$ *iff* $[\![F]\!]^W_\alpha = 0$
- $[\![F \wedge G]\!]^W_\alpha = 1$ *iff* $[\![F]\!]^W_\alpha = 1$ *and* $[\![G]\!]^W_\alpha = 1$
- $[\![F \vee G]\!]^W_\alpha = 1$ *iff* $[\![F]\!]^W_\alpha = 1$, *or* $[\![F]\!]^W_\alpha = 0$ *and* $[\![G]\!]^W_\alpha = 1$
- $[\![\forall x : A.F]\!]^W_\alpha = 1$ *iff* $[\![F]\!]^W_{\alpha+m} = 1$ *for every* $m \in [\![A]\!]^W_\alpha$
- $[\![\exists x : A.F]\!]^W_\alpha = 1$ *iff* $[\![F]\!]^W_{\alpha+m} = 1$ *for some* $m \in [\![A]\!]^W_\alpha$

It can be seen as an advantage that, while Henkin-models must define $[\![A \to B]\!] \subseteq [\![B]\!]^{[\![A]\!]}$ separately for every function type, lax models interpret function types compositionally using a fixed operation Fun. That makes defining models easier as the following examples show. It becomes particularly helpful when building term models (see Sect. 5): here Henkin-model constructions can require complex mappings (like in [Pop25]) back and forth between (i) function terms (e.g, $\lambda x : A.\, t$) and (ii) the functions on terms (e.g., $s \mapsto t[s]$) that they are interpreted as. But lax term models can easily interpret every function term as itself.

Example 3 (Product Models). We can define a trivial interpretation that interprets every type as the singleton set $\{0\}$ and every term as 0. Similarly, given two lax models V and W, we can define a product $V \times W$ by $[\![A]\!]^{V\times W}_\alpha = [\![A]\!]^V_\alpha \times [\![A]\!]^W_\alpha$ and $[\![t]\!]^{V\times W}_\alpha = ([\![t]\!]^V_\alpha, [\![t]\!]^W_\alpha)$. In particular, $V \times W$ interprets a function as a pair of functions and not, as a strict model would, as a function on pairs. These have all properties of a lax model except for standard Booleans. With a little effort, they can be adapted to have standard Booleans as well.

The requirements of Def. 5 are motivated by the following theorem that enables a high-level characterization of lax models as functors. Specifically, the contexts form a display map category [Jac99], and models can be seen as functors out of it, an approach going back to [Law63] and commonly used both for higher-order logic [LS86] and dependent type theory [Hof97]:

Definition 6 (Category of Contexts). *Given a theory T, the category $\mathcal{CTX}_T$ has as objects the contexts $\vdash_T \Gamma$ $\texttt{Ctx}$ quotiented by $\vdash \Gamma \equiv \Gamma'$, and as morphisms from Γ to Δ the substitutions $\vdash \gamma : \Gamma \to \Delta$ quotiented by $\vdash \gamma \equiv \gamma' : \Gamma \to \Delta$. Identity and composition are defined in the obvious way.*

Theorem 3 (Lax Models as Functors). *For a lax model W of theory T, the function $[\![-]\!]^W$ induces a contravariant functor from $\mathcal{CTX}_T$ to $\mathcal{SET}$.*

Proof. The elements of $\mathcal{CTX}_T$ are equivalence classes, and $[\![-]\!]^W$ is defined on representatives. This is well-defined because W preserves equality. It is well-defined as a mapping of objects/morphisms to objects/morphisms because W preserves typing and has standard variables.

To show the contravariant functoriality, consider the identity substitution $x_1, \ldots, x_n$ of Γ. $[\![x_1, \ldots, x_n]\!]^W$ maps α to $([\![x_1]\!]^W_\alpha, \ldots, [\![x_n]\!]^W_\alpha)$; because W has

standard variables, this is equal to $(x_1^\alpha, \ldots, x_n^\alpha) = \alpha$; thus $[\![x_1, \ldots, x_n]\!]^W$ is indeed the identify function on $[\![\Gamma]\!]^W$. We proceed accordingly if Γ has type variables.

Now consider substitutions $\gamma = t_1, \ldots, t_n : \Gamma \to \Delta$ and $\delta : \Delta \to \Theta$. Their composition $\gamma; \delta : \Gamma \to \Theta$ is defined as $t_1[\delta], \ldots, t_n[\delta]$. $[\![\gamma; \delta]\!]^W$ maps $\alpha \in [\![\Theta]\!]^W$ to $(\ldots, [\![t_i[\delta]]\!]^W_\alpha, \ldots)$. Because W commutes with substitution, this equals $(\ldots, [\![t_i]\!]^W_{[\![\delta]\!]^W(\alpha)}, \ldots)$, which equals $[\![\gamma]\!]^W([\![\delta]\!]^W(\alpha))$ as needed. The case where the contexts contain type variables proceeds accordingly.

Note that the proof does not actually use the fact W has standard Booleans.

Remark 2 (Functors as lax models). We can classify the lax models as the contravariant functors from $\mathcal{CTX}_T$ to $\mathcal{SET}$ that satisfy a few additional properties. In fact, every contravariant functor Ψ already induces an interpretation function: for $\Gamma \vdash A : \texttt{Type}$, $[\![A]\!]^\Psi_\alpha$ is the set of elements m such that $\alpha + m \in \Psi(\Gamma, x : A)$; and for $\Gamma \vdash t : A$, $[\![t]\!]^\Psi_\alpha$ is the element u such that $\Psi(id_\Gamma, t)(\alpha) = \alpha + m$. This interpretation function already preserves typing and equality, has standard variables, and almost commutes with substitution. We say *almost* because we have proved all necessary properties except for $[\![A]\!]^\Psi_{[\![\gamma]\!]^\Psi_\alpha} \subseteq [\![A[\gamma]]\!]^\Psi_\alpha$. That last condition holds if we additionally require that Ψ preserve pullbacks. In fact, it suffices to preserve pullbacks along display maps. The functors induced by lax models do have that property. So we can say that the lax models are the functors that preserve pullbacks along display maps and have standard Booleans.

Functorial models connect our definition to the categorical models that have been developed for dependent type theory [Jac99,Hof97]. These are functors from $\mathcal{CTX}_T$ into any category that has sufficient structure to define a compositional interpretation function. This yields a trade-off between non-strict models into the standard category $\mathcal{SET}$ (our approach), and strict models into some other category. We expect most models of the latter kind carry over to DHOL and allow giving, e.g., DHOL models in any two-valued Boolean topos.

5 The Term Model and Completeness

We follow the usual structure of a completeness proof. We extend a consistent theory T to a maximally consistent one T^*, then extract a term model from it. The term model interprets every type as the set of its terms, quotiented by provable equality, and makes formulas true iff they are provable. Critically, we use the universe to ensure that type variables only range over those sets of terms, not over arbitrary sets.

The usual construction of T^* enumerates all formulas F and adds axioms and witnesses to give T^* a model-like behavior. We need a few technical adjustments because adding an axiom may increase the set of well-formed formulas.

Definition 7. *We allow for infinite theories T as long as the declarations are well-ordered and every declaration is well-formed over a finite subset of its predecessors. T is* ***complete*** *if $\vdash_T F$ or $\vdash_T \neg F$ for every formula F. T is* ***inconsistent*** *if $\vdash_T F$ and $\vdash_T \neg F$ for some formula F. And $\Gamma \vdash F$ is* ***disprovable****, if adding $(\Gamma) \triangleright F$ would make T inconsistent.*

Theorem 4 (Theory Completion). *Every consistent theory T can be extended to a complete consistent theory T^* in which every open formula is provable iff all its ground instances are.*

Proof. We call a term *possible* if it can be produced by the grammar using the identifiers from T and arbitrarily many free type variables, i.e., possible terms need not be well-typed. There is an enumeration $t_0, t_1, \ldots$ of possible terms in which every element occurs infinitely often. For example, we can enumerate the set $P \times \mathbb{N}$ where P is the set of all possible terms.

We put $T_0 := T$ and define T_{i+1} from T_i as follows. If we do not have $\Gamma \vdash_{T_i} t_i : \mathtt{bool}$, we put $T_{i+1} := T_i$. Otherwise, if $\Gamma \vdash_{T_i} t_i$ is not disprovable, we put $T_{i+1} := T_i$, $(\Gamma) \triangleright t_i$. If t_i is disprovable, we put $T_{i+1} := T_i$ except for the following special case: If t_i is disprovable and t_i is of the form $f =_{\Pi x:A|F.\,B} g$ and there is no term s such that $\Gamma \vdash_{T_i} F[s]$ is provable and $\Gamma \vdash_{T_i} f\,s =_{B[s]} g\,s$ disprovable, we add appropriate witnesses, i.e., put

$$T_{i+1} := T_i,\ u_1 : \mathtt{Type},\ \ldots,\ u_n : \mathtt{Type},\ c : A,\ \triangleright F[c],\ \triangleright \neg f\,c =_{B[c]} g\,c$$

where the u_i are the type variables in Γ (or appropriate renamings if those names are not fresh) and c is fresh. Finally, we define T^* as the union of all T_i.

T^* is consistent: If it were inconsistent, some T_i would be inconsistent because a proof can use only finitely many axioms. But each T_i is consistent by construction: Using the rules of the calculus, an inconsistency in T_i can be turned into an inconsistency in T. In particular, in the special case above, if A were empty or F unsatisfiable, then t_i would be provable; so T_{i+1} is consistent if T_i is.

T^* is complete: If there were an undetermined t over T^*, then there would be some T_i over which t would be well-formed, because the well-formedness of t can only depend on finitely many axioms, and then t would be undetermined over every subsequent T_j. As t occurs in the enumeration some time after i, an axiom for t would have been added in some T_j.

Finally, assume $\Gamma \vdash F : \mathtt{bool}$. If it is provable, then so are its ground instances. If it is not provable, we need to give a ground instance that is not provable. We bind all term variables and assumptions in Γ using λ and equate the resulting function to the constant function returning `true`, obtaining a formula $\Gamma' \vdash F' : \mathtt{bool}$ where Γ' contains only type variables. Because of the rules for functions, $\Gamma' \vdash F'$ is provable iff $\Gamma \vdash F$ is. F' must have occurred in the enumeration (up to renaming type variables) and must therefore be disprovable. We obtain a disprovable ground instance of F by induction on F'. If F' is an implication $G \Rightarrow G'$, the rules for implication show that $\Gamma', \triangleright G \vdash G'$ must be disprovable reducing the case to the induction hypothesis for G'. In F' is an equality, the construction of T^* has added exactly the necessary witnesses to obtain a disprovable ground instance of F.

Theorem 5 (Term Model). *Every consistent theory T has a lax model such that for every $\Gamma \vdash F : \mathtt{bool}$ we have $\Gamma \vdash F$ iff $[\![F]\!]^M_\alpha = 1$ for all α.*

Proof. We construct an extensional lax model W for a universe $\mathcal{U}$ for the infinite theory T^*, which can then be restricted to a lax model for T. For any closed

types $\vdash A : \texttt{Type}$, let $Tm(A)$ be the quotient of the set of all closed terms $\vdash t : A$ by the relation $\vdash s =_A t$ (which is an equivalence relation by the rules for equality). (Except for empty types and the effects of precondition-subtyping, the sets $Tm(A)$ are disjoint for unequal types.) $\mathcal{U}$ is the set containing the sets $Tm(A)$ for all A. We will interpret every A as $Tm(A)$ and every t as its equivalence class. Thus, the type variables effectively range over closed types.

We define the interpretation function for types and terms in context as follows: For a type $\Gamma \vdash A : \texttt{Type}$, $[\![A]\!]^W_\alpha$ is the set $Tm(A[\alpha^!])$. For a term $\Gamma \vdash t : A$, $[\![t]\!]^W_\alpha$ is the equivalence class of $t[\alpha^!]$. Here, by construction, α assigns to every type variable a set of the form $Tm(C)$ for some closed type C, and to every term variable an equivalence class of closed terms. $\alpha^!$ is any substitution that picks for each type variable one of the types C and for each term variable one element of the equivalence class. That is well-defined because the substitution rules for equality guarantee that any choice for $\alpha^!$ leads to $\equiv$-equal types $A[\alpha^!]$, and the rules for typing guarantee that $\equiv$-equal types have the same terms.

The interpretation of contexts and substitutions is determined by the property of standard variables. Then to show standard variables, we check that indeed $[\![x]\!]^W_\alpha = x^\alpha$ and accordingly for type variables. This interpretation function preserves typing and equality by construction, and it commutes with substitution because of the properties of substitutions.

We show that W has standard Booleans. Because T^* is complete, we have $\vdash_{T^*} F =_{\texttt{bool}} \texttt{true}$ or $\vdash_{T^*} F =_{\texttt{bool}} \texttt{false}$ for every closed Boolean F. And because T^* is consistent, $\texttt{true}$ and $\texttt{false}$ are unequal. Thus, $[\![\texttt{bool}]\!]^W_\alpha$ always has two elements. The cases for $=_A$ and $\Rightarrow$ are straightforward (and the latter uses in particular the rules for $\Rightarrow$).

Soundness is subsumed by the second part of the theorem statement. And that claim follows from the properties of T^* because because quantifying over assignments into W is equivalent to quantifying over ground instances.

Theorem 6 (Completeness). *Given $\Gamma \vdash_T F : \texttt{bool}$, if $[\![F]\!]^W_\alpha = 1$ for every lax model W of T and every assignment α, then $\Gamma \vdash_T F$.*

Proof. If T is inconsistent, the result is trivial. Otherwise, if $\Gamma \vdash_T F$ did not hold, the theory T, Γ, $\triangleright \neg F$ (using the declarations from Γ as their counterparts in a theory) would be consistent, and its term model would violate the assumption.

6 Conclusion and Future Work

The present paper can be seen as the reference definition of DHOL. It gives the syntax and proof theory in all detail and develops the model theory with strict and lax models as well as soundness and completeness proofs. It subsumes the languages of [RRB25] and [RRK26] and extends them with function preconditions. The latter anticipates extensions to refinement and quotient types following [RR25].

Future work can use the model theory to better understand and utilize DHOL. We are particularly interested in using the semantics as a pivot to relate different calculi and provers.

References

And86. P. Andrews. *An introduction to mathematical logic and type theory: to truth through proof*. Academic Press, 1986.

AR11. S. Awodey and F. Rabe. Kripke Semantics for Martin-Löf's Extensional Type Theory. *Logical Methods in Computer Science*, 7(3), 2011.

BRS08. C. Benzmüller, F. Rabe, and G. Sutcliffe. THF0 – The core of the TPTP Language for Higher-Order Logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *4th International Joint Conference on Automated Reasoning*, pages 491–506. Springer, 2008.

Car86. J. Cartmell. Generalized algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

Chu40. A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5(1):56–68, 1940.

dKA+15. L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The Lean Theorem Prover (System Description). In A. Felty and A. Middeldorp, editors, *Automated Deduction*, pages 378–388. Springer, 2015.

Far93. W. Farmer. A simple type theory with partial functions and subtypes. *Annals of Pure and Applied Logic*, 64(3):211–240, 1993.

Gor88. M. Gordon. HOL: A Proof Generating System for Higher-Order Logic. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128. Kluwer-Academic Publishers, 1988.

GP93. M. Gordon and A. Pitts. The HOL Logic. In M. Gordon and T. Melham, editors, *Introduction to HOL, Part III*, pages 191–232. Cambridge University Press, 1993.

Hen50. L. Henkin. Completeness in the Theory of Types. *Journal of Symbolic Logic*, 15(2):81–91, 1950.

Hof97. M. Hofmann. Syntax and semantics of dependent types. In A. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997.

Hom09. P. Homeier. The HOL-Omega logic. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics*, pages 244–259. Springer, 2009.

Hur01. J. Hurd. Predicate subtyping with predicate sets. In R. Boulton and P. Jackson, editors, *Theorem Proving in Higher Order Logics*, pages 265–280, 2001.

Jac99. B. Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.

KP19. O. Kuncar and A. Popescu. A consistent foundation for Isabelle/HOL. *Journal of Automated Reasoning*, 62(4):531–555, 2019.

KRS16. C. Kaliszyk, F. Rabe, and G. Sutcliffe. TH1: The TPTP Typed Higher-Order Form with Rank-1 Polymorphism. In P. Fontaine, S. Schulz, and J. Urban, editors, *Workshop on Practical Aspects of Automated Reasoning*, pages 41–55, 2016.

Law63. F. Lawvere. *Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963.

LS86. J. Lambek and P. Scott. *Introduction to Higher-Order Categorical Logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1986.

ML74. P. Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. In *Proceedings of the '73 Logic Colloquium*, pages 73–118. North-Holland, 1974.

NBK24. J. Niederhauser, C. Brown, and C. Kaliszyk. Tableaux for automated reasoning in dependently-typed higher-order logic. In C. Benzmüller, M. Heule, and R. Schmidt, editors, *Automated Reasoning*, pages 86–104, 2024.

Nor05. U. Norell. The Agda WiKi, 2005. http://wiki.portal.chalmers.se/agda.

ORS92. S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Springer, 1992.

OS97. S. Owre and N. Shankar. The formal semantics of PVS. Technical Report SRI-CSL-97-2, SRI International, 1997.

Pop25. A. Popescu. Completing Gordon's Higher-Order Logic. In *Logic in Computer Science, LICS*, pages 1–15. IEEE, 2025.

RBK24. D. Ranalter, C. Brown, and C. Kaliszyk. Experiments with choice in dependently-typed higher-order logic. In N. Bjørner, M. Heule, and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence and Reasoning*, pages 311–320, 2024.

RKRS25. D. Ranalter, C. Kaliszyk, F. Rabe, and G. Sutcliffe. The Dependently Typed Higher-Order Form for the TPTP World. In R. Thiemann and C. Weidenbach, editors, *Frontiers of Combining Systems*. Springer, 2025.

Roc25. Rocq Development Team. The Rocq Proof Assistant. Technical report, INRIA, 2025.

RR25. C. Rothgang and F. Rabe. Subtyping in Dependently-Typed Higher-Order Logic. In R. Thiemann and C. Weidenbach, editors, *Frontiers of Combining Systems*. Springer, 2025.

RRB23. C. Rothgang, F. Rabe, and C. Benzmüller. Theorem Proving in Dependently Typed Higher-Order Logic. In B. Pientka and C. Tinelli, editors, *Automated Dedution*, pages 438–455. Springer, 2023.

RRB25. C. Rothgang, F. Rabe, and C. Benzmüller. Dependently-Typed Higher-Order Logic. *ACM Transactions on Computational Logic*, 27(1), 2025.

RRK26. D. Ranalter, F. Rabe, and C. Kaliszyk. Polymorphism Meets DHOL. In F. Pfenning, editor, *Formal Structures for Computation and Deduction*. Schloss Dagstuhl—Leibniz Center for Informatics, 2026. to appear.

Sut22. G. Sutcliffe. The Logic Languages of the TPTP World. *Logic Journal of the IGPL*, 2022.

Völ07. N. Völker. Hol2p - a system of classical higher order logic with second order polymorphism. In K. Schneider and J. Brandt, editors, *Theorem Proving in Higher Order Logics*, pages 334–351. Springer, 2007.