Tableaux for Automated Reasoning in Dependently-Typed Higher-Order Logic

Johannes Niederhauser[1]([✉])[0000−0002−8662−6834], Chad E. Brown[2], and Cezary Kaliszyk[1,3][0000−0002−8273−6059]

[1] Department of Computer Science, University of Innsbruck, Innsbruck, Austria
`johannes.niederhauser@uibk.ac.at`
[2] Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Prague, Czech Republic
[3] School of Computing and Information Systems, University of Melbourne, Melbourne, Australia
`cezarykaliszyk@gmail.com`

Abstract. Dependent type theory gives an expressive type system facilitating succinct formalizations of mathematical concepts. In practice, it is mainly used for interactive theorem proving with intensional type theories, with PVS being a notable exception. In this paper, we present native rules for automated reasoning in a dependently-typed version (DHOL) of classical higher-order logic (HOL). DHOL has an extensional type theory with an undecidable type checking problem which contains theorem proving. We implemented the inference rules as well as an automatic type checking mode in Lash, a fork of Satallax, the leading tableaux-based prover for HOL. Our method is sound and complete with respect to provability in DHOL. Completeness is guaranteed by the incorporation of a sound and complete translation from DHOL to HOL recently proposed by Rothgang et al. While this translation can already be used as a preprocessing step to any HOL prover, to achieve better performance, our system directly works in DHOL. Moreover, experimental results show that the DHOL version of Lash can outperform all major HOL provers executed on the translation.

Keywords: Tableaux · Dependent Types · Higher-Order Logic

1 Introduction

Dependent types introduce the powerful concept of types depending on terms. Lists of fixed length are an easy but interesting example. Instead of having a simple type lst we may have a type $\Pi n\colon \mathsf{nat}.\,\mathsf{lst}\,n$ which takes a natural number as argument and returns the type of a list with length n. More generally, lambda terms $\lambda x.s$ now have a dependent type $\Pi x\colon A.B$ which makes the type of $(\lambda x.s)t$ dependent on t. With that, it is possible for example to specify an unfailing version of the tail function by declaring its type to be $\Pi n\colon \mathsf{nat}.\mathsf{lst}(\mathsf{s}\,n) \to \mathsf{lst}\,n$. Many interactive theorem provers for dependent type theory are available [3, 9, 12, 14], most of them implement intensional type theories, i.e., they distinguish between

a decidable judgmental equality (given by conversions) and provable equality (inhabiting an identity type). Notable exceptions are PVS [16] and F* [?] which implement an extensional type theory. In the context of this paper, we say a type theory is *extensional* if judgmental equality and provable equality coincide, as in [?]. The typing judgment in such type theories is usually undecidable, as shown in [?].

The broader topic of this paper is automated reasoning support for extensional type theories with dependent types. Not much has been done to this end, but last year Rothgang et al. [15] introduced an extension of HOL to dependent types which they dub DHOL. In contrast to dependent type theory, automated theorem proving in HOL has a long history and led to the development of sophisticated provers [2,4,17]. Rothgang et al. defined a natural extension of HOL and equipped it with automation support by providing a sound and complete translation from DHOL into HOL. Their translation has been implemented and can be used as a preprocessing step to any HOL prover in order to obtain an automated theorem prover for DHOL. Hence, by committing to DHOL, automated reasoning support for extensional dependent type theories does not have to be invented from scratch but can benefit from the achievements of the automated theorem proving community for HOL.

In this paper, we build on top of the translation from Rothgang et al. to develop a tableau calculus which is sound and complete for DHOL. In addition, dedicated inference rules for DHOL are defined and their soundness is proved. The tableau calculus is implemented as an extension of Lash [6]. The remainder of this paper is structured as follows: Section 2 sets the stage by defining DHOL and the erasure from DHOL to HOL due to Rothgang et al. before Section 3 defines the tableau calculus and provides soundness and completeness proofs. The implementation is described in Section 4. Finally, we report on experimental results in Section 5.

2   Preliminaries

2.1   HOL

We start by giving the syntax of higher-order logic (HOL) which goes back to Church [8]. In order to allow for a graceful extension to DHOL, we define it with a grammar based on [15].

$$
\begin{array}{lr}
T ::= \circ \mid T, a\colon \mathsf{tp} \mid T, x\colon A \mid T, s & \text{(theories)} \\
\Gamma ::= \cdot \mid \Gamma, x\colon A \mid \Gamma, s & \text{(contexts)} \\
A, B ::= a \mid A \to B \mid o & \text{(types)} \\
s, t, u, v ::= x \mid \lambda x\colon A.s \mid s\,t \mid \bot \mid \neg s \mid s \Rightarrow t \mid s =_A t \mid \forall x\colon A.s & \text{(terms)}
\end{array}
$$

A theory consists of base type declarations $a\colon \mathsf{tp}$, typed variable or constant declarations $x\colon A$ and axioms. Contexts are like theories but without base type declarations. In the following, we will often write $s \in T, \Gamma$ to denote that s

occurs in the combination of T and Γ. Furthermore, note that $\circ$ and $\cdot$ denote the empty theory and context, respectively. Types are declared base types a, the base type of booleans o or function types $A \to B$. As usual, the binary type constructor $\to$ is right-associative. Terms are simply-typed lambda-terms (modulo α-conversion) enriched by the connectives $\bot, \neg, \Rightarrow, =_A$ as well as the typed binding operator for $\forall$. All connectives yield terms of type o (formulas). By convention, application associates to the left, so $s\,t\,u$ means $(s\,t)\,u$ with the exception that $\neg\,s\,t$ always means $\neg(s\,t)$. Moreover, we abbreviate $\neg(s =_A t)$ by $s \neq_A t$ and sometimes omit the type subscript of $=_A$ when it is either clear from the context or irrelevant. We write $s[x_1/t_1, \ldots, x_n/t_n]$ to denote the simultaneous capture-avoiding substitution of the x_i's by the t_i's. The set of free variables of a term s is denoted by $\mathcal{V}s$.

A theory T is well-formed if all types are well-formed and axioms have type o with respect to its base type declarations. In that case, we write $\vdash^{\mathsf{s}} T\,\mathsf{Thy}$ where the superscript s indicates that we are in the realm of simple types. Given a well-formed theory T, the well-formedness of a context Γ is defined in the same way and denoted by $\vdash^{\mathsf{s}}_T \Gamma\,\mathsf{Ctx}$. Given a theory T and a context Γ, we write $\Gamma \vdash^{\mathsf{s}}_T A\,\mathsf{tp}$ to state that A is a well-formed type and $\Gamma \vdash^{\mathsf{s}}_T s\colon A$ to say that s has type A. Furthermore, $\Gamma \vdash^{\mathsf{s}}_T s$ denotes that s has type o and is provable from Γ and T in HOL. Finally, we use $\Gamma \vdash^{\mathsf{s}}_T A \equiv B$ to state that A and B are equivalent well-formed types. For HOL this is trivial as it corresponds to syntactic equivalence, but this will change drastically in DHOL.

2.2 DHOL

The extension from HOL to DHOL consists of two crucial ingredients:

- the type constructor $A \to B$ is replaced by the constructor $\Pi x\colon A.B$ which potentially makes the return type B dependent on the actual argument x; we stick to the usual arrow notation if B does not contain x
- base types a can now take term arguments; for an n-ary base type we write $a\colon \Pi x_1\colon A_1.\ \cdots\ \Pi x_n\colon A_n.\,\mathsf{tp}$

Thus, the grammar defining the syntax of DHOL is given as follows:

$$T ::= \circ \mid T, a\colon (\Pi x\colon A.)^* \,\mathsf{tp} \mid T, x\colon A \mid T, s \qquad \text{(theories)}$$
$$\Gamma ::= \cdot \mid \Gamma, x\colon A \mid \Gamma, s \qquad \text{(contexts)}$$
$$A, B ::= a\,t_1\,\ldots\,t_n \mid \Pi x\colon A.B \mid o \qquad \text{(types)}$$
$$s, t, u, v ::= x \mid \lambda x\colon A.s \mid s\,t \mid \bot \mid \neg s \mid s \Rightarrow t \mid s =_A t \mid \forall x\colon A.s \quad \text{(terms)}$$

If a base type a has arity 0, it is called a *simple base type*. Note that HOL is the fragment of DHOL where all base types have arity 0. Allowing base types to have term arguments makes type equality a highly non-trivial problem in DHOL. For example, if $\Gamma \vdash^{\mathsf{d}}_T s\colon \Pi x\colon A.B$ (the d in $\vdash^{\mathsf{d}}$ indicates that we are speaking about DHOL) and $\Gamma \vdash^{\mathsf{d}}_T t\colon A'$ we still want $\Gamma \vdash^{\mathsf{d}}_T (s\,t)\colon B[x/t]$ to hold if $\Gamma \vdash^{\mathsf{d}}_T A \equiv A'$, so checking whether two types are equal is a problem

which occurs frequently in DHOL. Intuitively, we have $\Gamma \vdash_T^{\mathsf{d}} A \equiv A'$ if and only if their simply-typed skeleton consisting of arrows and base types without their arguments is equal and given a base type $a\colon \Pi x_1\colon A_1.\ \cdots\ \Pi x_n\colon A_n.\,\mathsf{tp}$, an occurrence $a\,t_1\ \ldots\ t_n$ in A and its corresponding occurrence $a\,t_1'\ \ldots\ t_n'$ in A', we have $\Gamma \vdash_T^{\mathsf{d}} t_i =_{A_i[x_1/t_1,\ldots,x_{i-1}/t_{i-1}]} t_i'$ for all $1 \leqslant i \leqslant n$. This makes DHOL an extensional type theory where already type checking is undecidable as it requires theorem proving. Another difference from HOL is the importance of the chosen representation of contexts and theories: Since the well-typedness of a term may depend on other assumptions, the order of the type declarations and formulas in a context Γ or theory T is relevant. A formal definition of the judgments $\Gamma \vdash_T^{\mathsf{d}} A\,\mathsf{tp}$, $\Gamma \vdash_T^{\mathsf{d}} s\colon A$, $\Gamma \vdash_T^{\mathsf{d}} s$ and $\Gamma \vdash_T^{\mathsf{d}} A \equiv B$ via an inference system is given in [15]. Since we use more primitive connectives, a minor variant is presented in Figure 2.

Example 1. Consider the simple base types $\mathsf{nat}\colon \mathsf{tp}$ and $\mathsf{elem}\colon \mathsf{tp}$ as well as the dependent base type $\mathsf{lst}\colon \Pi x\colon \mathsf{nat}.\,\mathsf{tp}$. The constants and functions

$$0\colon \mathsf{nat} \qquad\qquad \mathsf{s}\colon \mathsf{nat} \to \mathsf{nat}$$
$$\mathsf{nil}\colon \mathsf{lst}\,0 \qquad\qquad \mathsf{cons}\colon \Pi n\colon \mathsf{nat}.\,\mathsf{elem} \to \mathsf{lst}\,n \to \mathsf{lst}\,(\mathsf{s}\,n)$$

provide means to represent their inhabitants. Additionally, we define functions $\mathsf{plus}\colon \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$

$$\forall n\colon \mathsf{nat}.\,\mathsf{plus}\,0\,n =_{\mathsf{nat}} n \qquad \forall n,m\colon \mathsf{nat}.\,\mathsf{plus}\,(\mathsf{s}\,n)\,m =_{\mathsf{nat}} \mathsf{s}\,(\mathsf{plus}\,n\,m)$$

and $\mathsf{app}\colon \Pi n\colon \mathsf{nat}.\,\Pi m\colon \mathsf{nat}.\,\mathsf{lst}\,n \to \mathsf{lst}\,m \to \mathsf{lst}\,(\mathsf{plus}\,n\,m)$:

$$\forall n\colon \mathsf{nat}, x\colon \mathsf{lst}\,n.\,\mathsf{app}\,0\,n\,\mathsf{nil}\,x =_{\mathsf{lst}\,n} x$$
$$\forall n,m\colon \mathsf{nat}, z\colon \mathsf{elem}, x\colon \mathsf{lst}\,n, y\colon \mathsf{lst}\,m.$$
$$\mathsf{app}\,(\mathsf{s}\,n)\,m\,(\mathsf{cons}\,n\,z\,x)\,y =_{\mathsf{lst}\,(\mathsf{s}\,(\mathsf{plus}\,n\,m))} \mathsf{cons}\,(\mathsf{plus}\,n\,m)\,z\,(\mathsf{app}\,n\,m\,x\,y)$$

In the defining equations of app, we annotated the equality sign with the dependent type of the term on the right-hand side. In all cases, the simply-typed skeleton is just lst but for a type check we need to prove the two equalities

$$\forall n\colon \mathsf{nat}.\,\mathsf{plus}\,0\,n =_{\mathsf{nat}} n \qquad \forall n,m\colon \mathsf{nat}.\,\mathsf{plus}\,(\mathsf{s}\,n)\,m =_{\mathsf{nat}} \mathsf{s}\,(\mathsf{plus}\,n\,m)$$

which are exactly the corresponding axioms for plus. Type checking the conjecture

$$\forall n\colon \mathsf{nat}, x\colon \mathsf{lst}\,n.\,\mathsf{app}\,n\,0\,x\,\mathsf{nil} =_{\mathsf{lst}\,n} x$$

would require proving $\forall n\colon \mathsf{nat}.\,\mathsf{plus}\,n\,0 =_{\mathsf{nat}} n$ which can be achieved by induction on natural numbers if we include the Peano axioms.

2.3   Erasure

The following definition presents the translation from DHOL to HOL due to Rothgang et al. [15]. Intuitively, the translation erases dependent types to their

$$\frac{}{\vdash^{\mathsf{d}} \circ \; \mathsf{Thy}}\text{thyEmpty} \qquad \frac{\vdash^{\mathsf{d}}_T x_1\colon A_1,\ldots,x_n\colon A_n \; \mathsf{Ctx}}{\vdash^{\mathsf{d}} T, a\colon \Pi x_1\colon A_1.\;\ldots\;\Pi x_n\colon A_n.\mathsf{tp} \; \mathsf{Thy}}\text{thyType}$$

$$\frac{\vdash^{\mathsf{d}}_T A \; \mathsf{tp}}{\vdash^{\mathsf{d}} T, x\colon A \; \mathsf{Thy}}\text{thyConst} \qquad \frac{\vdash^{\mathsf{d}}_T s\colon o}{\vdash^{\mathsf{d}} T, s \; \mathsf{Thy}}\text{thyAxiom} \qquad \frac{\vdash^{\mathsf{d}} T \; \mathsf{Thy}}{\vdash^{\mathsf{d}}_T \cdot \; \mathsf{Ctx}}\text{ctxEmpty}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T A \; \mathsf{tp}}{\vdash^{\mathsf{d}}_T \Gamma, x\colon A \; \mathsf{Ctx}}\text{ctxVar} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s\colon o}{\vdash^{\mathsf{d}}_T \Gamma, s \; \mathsf{Ctx}}\text{ctxAssume}$$

$$\frac{a\colon (\cdots \Pi x_i\colon A_i.\cdots .\mathsf{tp}) \in T \quad \vdash^{\mathsf{d}}_T \Gamma \; \mathsf{Ctx} \quad \cdots \Gamma \vdash^{\mathsf{d}}_T t_i\colon A_i[x_1/s_1 \ldots x_{i-1}/s_{i-1}] \cdots}{\Gamma \vdash^{\mathsf{d}}_T a\; t_1 \; \ldots \; t_n \; \mathsf{tp}}\text{type}$$

$$\frac{x\colon A' \in T \quad \Gamma \vdash^{\mathsf{d}}_T A' \equiv A}{\Gamma \vdash^{\mathsf{d}}_T x\colon A}\text{const} \qquad \frac{s \in T \quad \vdash^{\mathsf{d}}_T \Gamma \; \mathsf{Ctx}}{\Gamma \vdash^{\mathsf{d}}_T s}\text{axiom}$$

$$\frac{x\colon A' \in \Gamma \quad \Gamma \vdash^{\mathsf{d}}_T A' \equiv A}{\Gamma \vdash^{\mathsf{d}}_T x\colon A}\text{var} \qquad \frac{s \in \Gamma \quad \vdash^{\mathsf{d}}_T \Gamma \; \mathsf{Ctx}}{\Gamma \vdash^{\mathsf{d}}_T s}\text{assume} \qquad \frac{\vdash^{\mathsf{d}}_T \Gamma \; \mathsf{Ctx}}{\Gamma \vdash^{\mathsf{d}}_T o \; \mathsf{tp}}\text{bool}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T A \; \mathsf{tp} \quad \Gamma, x\colon A \vdash^{\mathsf{d}}_T B \; \mathsf{tp}}{\Gamma \vdash^{\mathsf{d}}_T \Pi x\colon A.B \; \mathsf{tp}}\text{pi} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T A \equiv A' \quad \Gamma, x\colon A \vdash^{\mathsf{d}}_T B \equiv B'}{\Gamma \vdash^{\mathsf{d}}_T \Pi x\colon A.B \equiv \Pi x\colon A'.B'}\text{cong}\Pi$$

$$\frac{a\colon (\cdots \Pi x_i\colon A_i.\cdots .\mathsf{tp}) \in T \quad \vdash^{\mathsf{d}}_T \Gamma \; \mathsf{Ctx} \quad \cdots \Gamma \vdash^{\mathsf{d}}_T s_i =_{A_i[x_1/s_1\ldots x_{i-1}/s_{i-1}]} t_i \cdots}{\Gamma \vdash^{\mathsf{d}}_T a\; s_1 \; \ldots \; s_n \equiv a\; t_1 \; \ldots \; t_n}\text{congBT}$$

$$\frac{\Gamma, x\colon A \vdash^{\mathsf{d}}_T t\colon B}{\Gamma \vdash^{\mathsf{d}}_T (\lambda x\colon A.t)\colon \Pi x\colon A.B}\text{lambda} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s\colon \Pi x\colon A.B \quad \Gamma \vdash^{\mathsf{d}}_T t\colon A}{\Gamma \vdash^{\mathsf{d}}_T (s\;t)\colon B[x/t]}\text{appl}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T s\colon A \quad \Gamma \vdash^{\mathsf{d}}_T t\colon A}{\Gamma \vdash^{\mathsf{d}}_T (s =_A t)\colon o}\text{=type} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T A \equiv A' \quad \Gamma, x\colon A \vdash^{\mathsf{d}}_T t =_B t'}{\Gamma \vdash^{\mathsf{d}}_T \lambda x\colon A.t =_{\Pi x\colon A.B} \lambda x\colon A'.t'}\text{cong}\lambda$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T s =_{\Pi x\colon A.B} s' \quad \Gamma \vdash^{\mathsf{d}}_T t =_A t'}{\Gamma \vdash^{\mathsf{d}}_T s\;t =_{B[x/t]} s'\;t'}\text{congAppl} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s\colon A}{\Gamma \vdash^{\mathsf{d}}_T s =_A s}\text{refl}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T s =_A t}{\Gamma \vdash^{\mathsf{d}}_T t =_A s}\text{sym} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s\colon (\Pi x\colon A.B) \quad x \notin \mathcal{V}s}{\Gamma \vdash^{\mathsf{d}}_T s =_{\Pi x\colon A.B} \lambda x\colon A.s\;x}\text{eta}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T (\lambda x\colon A.s)\;t\colon B}{\Gamma \vdash^{\mathsf{d}}_T (\lambda x\colon A.s)\;t =_B s[x/t]}\text{beta} \qquad \frac{\vdash^{\mathsf{d}}_T \Gamma \; \mathsf{Ctx}}{\Gamma \vdash^{\mathsf{d}}_T \bot\colon o}\bot\text{type} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s\colon o \quad \Gamma \vdash^{\mathsf{d}}_T \bot}{\Gamma \vdash^{\mathsf{d}}_T s}\bot\text{e}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T s\colon o}{\Gamma \vdash^{\mathsf{d}}_T (\neg s)\colon o}\neg\text{type} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s\colon o \quad \Gamma, s \vdash^{\mathsf{d}}_T \bot}{\Gamma \vdash^{\mathsf{d}}_T \neg s}\neg\text{i} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s \quad \Gamma \vdash^{\mathsf{d}}_T \neg s}{\Gamma \vdash^{\mathsf{d}}_T \bot}\neg\text{e}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T \neg\neg s}{\Gamma \vdash^{\mathsf{d}}_T s}\neg\neg\text{e} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s\colon o \quad \Gamma, s \vdash^{\mathsf{d}}_T t\colon o}{\Gamma \vdash^{\mathsf{d}}_T (s \Rightarrow t)\colon o}\Rightarrow\text{type} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s\colon o \quad \Gamma, s \vdash^{\mathsf{d}}_T t}{\Gamma \vdash^{\mathsf{d}}_T s \Rightarrow t}\Rightarrow\text{i}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T s \Rightarrow t \quad \Gamma \vdash^{\mathsf{d}}_T s}{\Gamma \vdash^{\mathsf{d}}_T t}\Rightarrow\text{e} \qquad \frac{\Gamma, x\colon A \vdash^{\mathsf{d}}_T s\colon o}{\Gamma \vdash^{\mathsf{d}}_T \forall x\colon A.s\colon o}\forall\text{type} \qquad \frac{\Gamma, x\colon A \vdash^{\mathsf{d}}_T s}{\Gamma \vdash^{\mathsf{d}}_T \forall x\colon A.s}\forall\text{i}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T \forall x\colon A.s \quad \Gamma \vdash^{\mathsf{d}}_T t\colon A}{\Gamma \vdash^{\mathsf{d}}_T s[x/t]}\forall\text{e} \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T A \equiv A' \quad \Gamma, x\colon A \vdash^{\mathsf{d}}_T s =_o s'}{\Gamma \vdash^{\mathsf{d}}_T \forall x\colon A.s =_o \forall x\colon A'.s'}\text{cong}\forall$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T s =_o s' \quad \Gamma \vdash^{\mathsf{d}}_T s'}{\Gamma \vdash^{\mathsf{d}}_T s}\text{cong}\vdash \qquad \frac{\Gamma \vdash^{\mathsf{d}}_T s\bot \quad \Gamma \vdash^{\mathsf{d}}_T s(\neg\bot)}{\Gamma \vdash^{\mathsf{d}}_T \forall x\colon o.sx}\text{boolExt}$$

$$\frac{\Gamma \vdash^{\mathsf{d}}_T s\colon o \quad \Gamma, x\colon A \vdash^{\mathsf{d}}_T s \quad A \; \text{simple type}}{\Gamma \vdash^{\mathsf{d}}_T s}\text{nonempty}$$

Fig. 1. Natural Deduction Calculus for DHOL

simply typed skeletons by ignoring arguments of base types. The thereby lost information on concrete base type arguments is restored with the help of a partial equivalence relation (PER) A^* for each type A. A PER is a symmetric, transitive relation. The elements on which it is also reflexive are intended to be the members of the original dependent type, i.e., $\Gamma \vdash_T^{\mathsf{d}} s\colon A$ if and only if $\overline{\Gamma} \vdash_{\overline{T}}^{\mathsf{s}} A^* \, \overline{s} \, \overline{s}$.

Definition 1. *The translation from DHOL to HOL is given by the erasure function $\overline{s}$ as well as A^* which computes the formula representing the corresponding PER of a type A. The functions are mutually defined by recursion on the grammar of DHOL. The erasure of a theory (context) is defined as the theory (context) which consists of its erased components.*

$$\overline{o} \;=\; o \qquad\qquad\qquad \overline{a\,t_1\,\ldots\,t_n} \;=\; a$$
$$\overline{\Pi x\colon A.B} \;=\; \overline{A} \to \overline{B} \qquad\qquad \overline{x} \;=\; x$$
$$\overline{\lambda x\colon A.s} \;=\; \lambda x\colon \overline{A}.\,\overline{s} \qquad\qquad \overline{s\,t} \;=\; \overline{s}\,\overline{t}$$
$$\overline{\bot} \;=\; \bot \qquad\qquad\qquad \overline{\neg s} \;=\; \neg\overline{s}$$
$$\overline{s \Rightarrow t} \;=\; \overline{s} \Rightarrow \overline{t} \qquad\qquad \overline{s =_A t} \;=\; A^* \, \overline{s}\,\overline{t}$$
$$\overline{\forall x\colon A.s} \;=\; \forall x\colon \overline{A}.\, A^* \, x\,x \Rightarrow \overline{s} \qquad\qquad \overline{x\colon A} \;=\; x\colon \overline{A}, A^* \, x\,x$$

$$\overline{a\colon \Pi x_1\colon A_1.\,\cdots\,\Pi x_n\colon A_n.\,\mathsf{tp}} \;=\; a\colon \mathsf{tp},\, a^*\colon \overline{A_1} \to \cdots \to \overline{A_n} \to a \to a \to o,\, a_{\mathsf{per}}$$
$$o^* \, s\,t \;=\; s =_o t$$
$$(a\,t_1\,\ldots\,t_n)^* \, s\,t \;=\; a^* \, \overline{t_1}\,\ldots\,\overline{t_n}\,s\,t$$
$$(\Pi x\colon A.B)^* \, s\,t \;=\; \forall x,y\colon \overline{A}.\, A^* \, x\,y \Rightarrow B^* \, (s\,x)\,(t\,y)$$

Here, a_{per} is defined as follows:

$$a_{\mathsf{per}} = \forall x_1\colon \overline{A_1}.\,\ldots\forall x_n\colon \overline{A_n}.\,\forall u,v\colon a.\, a^* \, x_1\,\ldots\,x_n\,u\,v \Rightarrow u =_a v$$

Theorem 1 (Completeness [15]).

- if $\Gamma \vdash_T^{\mathsf{d}} A\colon \mathsf{tp}$ then $\overline{\Gamma} \vdash_{\overline{T}}^{\mathsf{s}} \overline{A}\colon \mathsf{tp}$ and A^* is a PER over $\overline{A}$
- if $\Gamma \vdash_T^{\mathsf{d}} A \equiv B$ then $\overline{\Gamma} \vdash_{\overline{T}}^{\mathsf{s}} \forall x,y\colon \overline{A}.\, A^* \, x\,y =_o B^* \, x\,y$
- if $\Gamma \vdash_T^{\mathsf{d}} s\colon A$ then $\overline{\Gamma} \vdash_{\overline{T}}^{\mathsf{s}} \overline{s}\colon \overline{A}$ and $\overline{\Gamma} \vdash_{\overline{T}}^{\mathsf{s}} A^* \, \overline{s}\,\overline{s}$
- if $\Gamma \vdash_T^{\mathsf{d}} s$ then $\overline{\Gamma} \vdash_{\overline{T}}^{\mathsf{s}} \overline{s}$

Theorem 2 (Soundness [15]).

- if $\Gamma \vdash_T^{\mathsf{d}} s\colon o$ and $\overline{\Gamma} \vdash_{\overline{T}}^{\mathsf{s}} \overline{s}$ then $\Gamma \vdash_T^{\mathsf{d}} s$
- if $\Gamma \vdash_T^{\mathsf{d}} s\colon A$ and $\Gamma \vdash_T^{\mathsf{d}} t\colon A$ and $\overline{\Gamma} \vdash_{\overline{T}}^{\mathsf{s}} A^* \, \overline{s}\,\overline{t}$ then $\Gamma \vdash_T^{\mathsf{d}} s =_A t$

Note that the erasure treats simple types and dependent types in the same way. In the following, we define a post-processing function Φ on top of the original erasure [15] which allows us to erase to simpler but equivalent formulas. The goal of Φ is to replace $A^* \, s\,t$ where A is a simple type by $s =_A t$. As a consequence, the guard $A^* \, x\,x$ in $\overline{\forall x\colon A.s}$ for simple types A can be removed. The following definition gives a presentation of Φ as a pattern rewrite system [11].

Definition 2. *Given a HOL term s, we define $\Phi(s)$ to be the HOL term which results from applying the following pattern rewrite rules exhaustively to all subterms in a bottom-up fashion:*

$$
\begin{aligned}
a^* F G &\rightarrow F =_a G \\
\forall x, y \colon A. \, (x =_A y) \Rightarrow (F\,x =_B G\,y) &\rightarrow F =_{A \rightarrow B} G \\
\forall x \colon A. \, (x =_A x) \Rightarrow F\,x &\rightarrow \forall x \colon A. \, F\,x
\end{aligned}
$$

Here, F, G are free variables for terms, a^* denotes the constant for the PER of a simple base type a and A, B are placeholders for simple types. Given a HOL theory T, there are finitely many instances for a^* but infinite choices for A and B, so the pattern rewrite system is infinite.

Lemma 1. *Assume $\Gamma \vdash^{\mathsf{d}}_T s \colon o$. $\Gamma \vdash^{\mathsf{d}}_T s$ if and only if $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} \Phi(\overline{s})$.*

Proof. Since the erasure is sound and complete (Theorem 2 and Theorem 1), it suffices to show that $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} \Phi(\overline{s})$ if and only if $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} \overline{s}$. Consider the rules from Definition 2. $\Phi(\overline{s})$ is well-defined: Clearly, the rules terminate and confluence follows from the lack of critical pairs [11]. Hence, it is sufficient to prove $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} l$ if and only if $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} r$ for every rule in Definition 2. For the first rule, assume $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} a^* F G$. Since $a_{\mathsf{per}} \in \overline{T}$, we have $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} F =_a G$. Now assume $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} F =_a G$. Since F has type a, we obtain $\Gamma \vdash^{\mathsf{d}}_T F =_a F$. Completeness of the erasure yields $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} a^* F F$. Now, the assumption allows us to replace equals by equals, so we conclude $\overline{\Gamma} \vdash^{\mathsf{s}}_{\overline{T}} a^* F G$. The desired result for the second rule follows from extensionality. Finally, the third rule is an easy logical simplification. □

Given a theory T (context Γ) we write $\Phi(\overline{T})$ ($\Phi(\overline{\Gamma})$) to denote its erased version where formulas have been simplified with Φ.

Corollary 1. *Assume $\Gamma \vdash^{\mathsf{d}}_T s \colon o$. $\Gamma \vdash^{\mathsf{d}}_T s$ if and only if $\Phi(\overline{\Gamma}) \vdash^{\mathsf{s}}_{\Phi(\overline{T})} \Phi(\overline{s})$.*

Example 2. Consider again the axiom recursively defining app from Example 1

$$\forall n, m \colon \mathsf{nat}, z \colon \mathsf{elem}, x \colon \mathsf{lst}\, n, y \colon \mathsf{lst}\, m.$$
$$\mathsf{app}\,(\mathsf{s}\,n)\,m\,(\mathsf{cons}\,n\,z\,x)\,y =_{\mathsf{lst}\,(\mathsf{s}\,(\mathsf{plus}\,n\,m))} \mathsf{cons}\,(\mathsf{plus}\,n\,m)\,z\,(\mathsf{app}\,n\,m\,x\,y)$$

which we refer to as s_{app}. Its post-processed erasure $\Phi(\overline{s_{\mathsf{app}}})$ is given by the following formula which is simpler than $\overline{s_{\mathsf{app}}}$:

$$\forall n, m \colon \mathsf{nat}, z \colon \mathsf{elem}, x \colon \mathsf{lst}. \, \mathsf{lst}^*\, n\,x\,x \Rightarrow \forall y \colon \mathsf{lst}. \, \mathsf{lst}^*\, m\,y\,y \Rightarrow \mathsf{lst}^*\, \big(\mathsf{s}\,(\mathsf{plus}\,n\,m)\big)$$
$$\big(\mathsf{app}\,(\mathsf{s}\,n)\,m\,(\mathsf{cons}\,n\,z\,x)\,y\big)\,\big(\mathsf{cons}\,(\mathsf{plus}\,n\,m)\,z\,(\mathsf{app}\,n\,m\,x\,y)\big)$$

3 Tableau Calculus for DHOL

3.1 Rules

The tableau calculus from [1,7] is the basis of Satallax [4] and its fork Lash [6]. We present an extension of this calculus from HOL to DHOL by extending

the rules to DHOL as well as providing tableau rules for the translation from DHOL to HOL. A *branch* is a 3-tuple (T, Γ, Γ') which is *well-formed* if $\vdash^{\mathsf{d}} T\,\mathsf{Thy}$, $\vdash^{\mathsf{d}}_T \Gamma\,\mathsf{Ctx}$ and $\vdash^{\mathsf{s}}_{\Phi(\overline{T})} \Gamma'\,\mathsf{Ctx}$. Intuitively, the theory contains the original problem and remains untouched while the contexts grow by the application of rules. Furthermore, DHOL and HOL are represented separately: For DHOL, the theory T and context Γ are used while HOL has a separate context Γ' with respect to the underlying theory $\Phi(\overline{T})$. In particular, each rule in Figure 1 really stands for two rules: one that operates in DHOL and the original version that operates in HOL. Except for the erasure rules $\mathcal{T}_{\mathrm{ER}_1}$ and $\mathcal{T}_{\mathrm{ER}_2}$ which add formulas to the HOL context based on information from the DHOL theory and context, the rules always stay in DHOL or HOL, respectively. More formally, a *step* is an $n + 1$-tuple $\langle (T, \Gamma, \Gamma'), (T, \Gamma_1, \Gamma'_1), \ldots, (T, \Gamma_n, \Gamma'_n) \rangle$ of branches where $\bot \notin T, \Gamma, \Gamma'$ and either $\Gamma \subset \Gamma_i$ and $\Gamma' = \Gamma'_i$ for all $1 \leqslant i \leqslant n$ or $\Gamma = \Gamma_i$ and $\Gamma' \subset \Gamma'_i$ for all $1 \leqslant i \leqslant n$. Given a step $\langle A, A_1, \ldots, A_n \rangle$, the branch A is called its *head* and each A_i is an *alternative*.

A *rule* is a set of steps defined by a schema. For example, the rule $\mathcal{T}_\Rightarrow$ from Figure 1 indicates the set of steps $\langle (T, \Gamma, \Gamma'), (T, \Gamma_1, \Gamma'_1), (T, \Gamma_2, \Gamma'_2) \rangle$ where $\bot \notin T, \Gamma, \Gamma'$ and either $s \Rightarrow t \in T, \Gamma$ or $s \Rightarrow t \in \Phi(\overline{T}), \Gamma'$. In the former case, we have $\Gamma_1 = \Gamma, \neg s$ and $\Gamma_2 = \Gamma, t$ as well as $\Gamma' = \Gamma'_1 = \Gamma'_2$. The latter case is the same but with the primed and unprimed variants swapped.

In the original tableau calculus [1, 7], normalization is defined with respect to an axiomatized generic operator $[\cdot]$. As one would expect, one of these axioms states that the operator does not change the semantics of a given term. Since there is no formal definition of DHOL semantics yet, we simply use $[s]$ to denote the $\beta\eta$-normal form of s which is in accordance with our implementation.

A rule *applies* to a branch A if some step in the rule has A as its head. A tableau calculus is a set of steps. Let $\mathcal{T}$ be the tableau calculus defined by the rules in Figure 1. The side condition of freshness in $\mathcal{T}_{\neg\forall}$ means that for a given step with head (T, Γ, Γ') there is no type A such that $y \colon A \in T, \Gamma$ or $y \colon A \in \Phi(\overline{T}), \Gamma'$ and we additionally require that there is no name x such that $\neg[s\,x] \in T, \Gamma$ or $\neg[s\,x] \in \Phi(\overline{T}), \Gamma'$. In practice, this means that to every formula, $\mathcal{T}_{\neg\forall}$ can be applied at most once. Furthermore, the side condition $t \colon A$ in the rule $\mathcal{T}_\forall$ means that either $\Gamma \vdash^{\mathsf{d}}_T t \colon A$ or $\Gamma' \vdash^{\mathsf{s}}_{\Phi(\overline{T})} t \colon A$ depending on whether the premise is in T, Γ or $\Phi(\overline{T}), \Gamma'$. The side condition $\overline{s} \colon o$ in the rule $\mathcal{T}_{\mathrm{ER}_1}$ means that $\Gamma' \vdash^{\mathsf{s}}_{\Phi(\overline{T})} \overline{s} \colon o$. This is to prevent application of $\mathcal{T}_{\mathrm{ER}_1}$ before the necessary type information is obtained by applying $\mathcal{T}_{\mathrm{ER}_2}$.

The set of $\mathcal{T}$-*refutable* branches is defined inductively: If $\bot \in T, \Gamma, \Gamma'$, then (T, Γ, Γ') is refutable. If $\langle A, A_1, \ldots, A_n \rangle$ is a step in $\mathcal{T}$ and every alternative A_i is refutable, then A is refutable.

The rules in Figure 1 strongly resemble the tableau calculus from [1]. In order to support DHOL, we replaced simple types by their dependent counterparts. To that end, we tried to remain as simple as possible by only allowing syntactically equivalent types in $\mathcal{T}_\forall$ and $\mathcal{T}_{\mathrm{CON}}$: Adding a statement like $A \equiv A'$ as a premise would change the tableau calculus as well as the automated proof search signif-

$$\mathcal{T}_\neg \; \frac{s, \neg s}{\bot} \qquad \mathcal{T}_\neq \; \frac{s \neq_{at_1 \ldots t_n} s}{\bot} \qquad \mathcal{T}_{\neg\neg} \; \frac{\neg\neg s}{s} \qquad \mathcal{T}_\Rightarrow \; \frac{s \Rightarrow t}{\neg s \mid t} \qquad \mathcal{T}_{\neg\Rightarrow} \; \frac{\neg(s \Rightarrow t)}{s, \neg t}$$

$$\mathcal{T}_\forall \; \frac{\forall x \colon A.s \qquad t \colon A}{[s[x/t]]} \qquad \mathcal{T}_{\neg\forall} \; \frac{\neg\forall x \colon A.s}{y \colon A, \neg[s[x/y]]} \; y \text{ fresh} \qquad \mathcal{T}_{\mathrm{BE}} \; \frac{s \neq_o t}{s, \neg t \mid \neg s, t}$$

$$\mathcal{T}_{\mathrm{BQ}} \; \frac{s =_o t}{s, t \mid \neg s, \neg t} \qquad \mathcal{T}_{\mathrm{FE}} \; \frac{s \neq_{\Pi x \colon A.B} t}{\neg[\forall x. \; sx = tx]} \; x \notin \mathcal{V}s \cup \mathcal{V}t \qquad \mathcal{T}_{\mathrm{FQ}} \; \frac{s =_{\Pi x \colon A.B} t}{[\forall x. \; sx = tx]} \; x \notin \mathcal{V}s \cup \mathcal{V}t$$

$$\mathcal{T}_{\mathrm{MAT}} \; \frac{x \, s_1 \; \ldots \; s_n, \neg \, x \, t_1 \; \ldots \; t_n}{s_1 \neq t_1 \mid \cdots \mid s_n \neq t_n} \; n \geqslant 1 \qquad \mathcal{T}_{\mathrm{DEC}} \; \frac{x \, s_1 \; \ldots \; s_n \neq_{au_1 \ldots u_n} x \, t_1 \; \ldots \; t_n}{s_1 \neq t_1 \mid \cdots \mid s_n \neq t_n} \; n \geqslant 1$$

$$\mathcal{T}_{\mathrm{CON}} \; \frac{s =_{at_1 \ldots t_n} t, u \neq_{at_1 \ldots t_n} v}{s \neq u, t \neq u \mid s \neq v, t \neq v} \qquad \mathcal{T}_{\mathrm{ER}_1} \; \frac{s}{[\Phi(\overline{s})]} \; \overline{s} \colon o \qquad \mathcal{T}_{\mathrm{ER}_2} \; \frac{x \colon A}{x \colon \overline{A}, A^* \, x \, x}$$

Fig. 2. Tableau rules for DHOL

icantly, so these situations are handled by the erasure for which the additional rules $\mathcal{T}_{\mathrm{ER}_1}$, $\mathcal{T}_{\mathrm{ER}_2}$ are responsible.

It is known that the restriction of $\mathcal{T}$ to HOL (without $\mathcal{T}_{\mathrm{ER}_1}$ and $\mathcal{T}_{\mathrm{ER}_2}$) is sound and complete with respect to Henkin semantics [1, 7]. Furthermore, due to Corollary **??**, the rules $\mathcal{T}_{\mathrm{ER}_1}$ and $\mathcal{T}_{\mathrm{ER}_2}$ define a sound and complete translation from DHOL to HOL with respect to Rothgang et al.'s definition of provability in DHOL [15].

3.2 Soundness and Completeness

In general, a soundness result based on the refutability of a branch (T, Γ, Γ') is desirable. If there were a definition of semantics for DHOL which is a conservative extension of Henkin semantics, the proof could just refer to satisfiability of T, Γ, Γ'. Unfortunately, this is not the case. Note that an appropriate definition of semantics is out of the scope of this paper: In addition to its conception, we would have to prove soundness and completeness of $\vdash^{\mathsf{d}}$ on top of the corresponding proofs for our novel tableau calculus. Therefore, soundness and completeness of the tableau calculus will be established with respect to provability in DHOL or HOL. Unfortunately, this requirement complicates the proof tremendously as a refutation can contain a mixture of DHOL, erasure and HOL rules. Therefore, we have to consider both HOL and DHOL and need to establish a correspondence between Γ and Γ' which is difficult to put succinctly and seems to be impossible without further restricting the notion of a well-formed branch. Therefore, we prove soundness and completeness with respect to a notion of refutability which has three stages: At the beginning, only DHOL rules are applied, the second stage is solely for the erasure and in the last phase, only HOL rules are applied. Note that this notion of refutability includes the sound but incomplete strategy of only using native DHOL rules as well as the sound and complete strategy of exclusively working with the erasure.

Definition 3. *A branch* (T, Γ, Γ') *is* s-refutable *if it is refutable with respect to the HOL rules.*

Lemma 2. *A well-formed branch* (T, Γ, Γ') *is s-refutable* $\iff \Gamma' \vdash^{\mathsf{s}}_{\Phi(\overline{T})} \bot$.

Proof. Immediate from soundness and completeness of the original HOL calculus as well as soundness and completeness of $\vdash^{\mathsf{s}}$. □

Definition 4. *The set of* e-refutable *branches is inductively defined as follows: If* (T, Γ, Γ') *is s-refutable and* $\Gamma' \subseteq \Phi(\overline{\Gamma})$, *then it is e-refutable. If* $\langle A, A_1 \rangle \in \mathcal{T}_{\mathrm{ER}_1} \cup \mathcal{T}_{\mathrm{ER}_2}$ *and* A_1 *is e-refutable, then* A *is e-refutable.*

Lemma 3. *If* (T, Γ, Γ') *is well-formed and e-refutable then* $\Phi(\overline{\Gamma}) \vdash^{\mathsf{s}}_{\Phi(\overline{T})} \bot$.

Proof. Let (T, Γ, Γ') be well-formed and e-refutable. We proceed by induction on the definition of e-refutability. If (T, Γ, Γ') is s-refutable then $\Gamma' \vdash^{\mathsf{s}}_{\Phi(\overline{T})} \bot$ by Lemma 2. Since $\Gamma' \subseteq \Phi(\overline{\Gamma})$ we also have $\Phi(\overline{\Gamma}) \vdash^{\mathsf{s}}_{\Phi(\overline{T})} \bot$. For the induction step, let $\langle (T, \Gamma, \Gamma'), (T, \Gamma, \Gamma'_1) \rangle$ be a step with either $\mathcal{T}_{\mathrm{ER}_1}$ or $\mathcal{T}_{\mathrm{ER}_2}$ and assume that the branch (T, Γ, Γ'_1) is e-refutable. Since well-formedness of (T, Γ, Γ'_1) follows from the well-formedness of (T, Γ, Γ'), the induction hypothesis yields $\Phi(\overline{\Gamma}) \vdash^{\mathsf{s}}_{\Phi(\overline{T})} \bot$ as desired. □

Definition 5. *The set of* d-refutable *branches is inductively defined as follows: If* $(T, \Gamma, \cdot)$ *is e-refutable or* $\bot \in T, \Gamma$, *then it is d-refutable. If* $\langle A, A_1, \ldots, A_n \rangle \in \mathcal{T} \setminus (\mathcal{T}_{\mathrm{ER}_1} \cup \mathcal{T}_{\mathrm{ER}_2})$ *and every alternative* A_i *is d-refutable, then* A *is d-refutable.*

Next, we have to prove soundness of every DHOL rule. For most of the rules, this is rather straightforward. We show soundness of $\mathcal{T}_{\mathrm{FE}}$, $\mathcal{T}_{\mathrm{FQ}}$ and $\mathcal{T}_{\mathrm{DEC}}$ as representative cases and start with an auxiliary lemma.

Lemma 4. *Assume* $\Gamma \vdash^{\mathsf{d}}_T s : o$. *We have* $\Gamma \vdash^{\mathsf{d}}_T s$ *if and only if* $\Gamma \vdash^{\mathsf{d}}_T [s]$.

Proof. By the beta and eta rules, we have $\Gamma \vdash^{\mathsf{d}}_T s =_o [s]$. Using cong$\vdash$ we obtain the desired result in both directions. □

Lemma 5 ($\mathcal{T}_{\mathrm{FE}}$). *Let* (T, Γ, Γ') *be a well-formed branch. Choose* x *such that* $x \notin \mathcal{V}s \cup \mathcal{V}t$ *and assume* $s \neq_{\Pi x : A.B} t \in T, \Gamma$. *If* $\Gamma, \neg[\forall x : A.sx = tx] \vdash^{\mathsf{d}}_T \bot$ *then* $\Gamma \vdash^{\mathsf{d}}_T \bot$.

Proof. From the assumptions and Lemma 7, we obtain $\Gamma \vdash^{\mathsf{d}}_T s \neq_{\Pi x : A.B} t$ and $\Gamma \vdash^{\mathsf{d}}_T \forall x : A.sx =_B tx$. Furthermore, an application of $\forall$e yields $\Gamma, x : A \vdash^{\mathsf{d}}_T sx =_B tx$. Using cong$\lambda$, we get $\Gamma \vdash^{\mathsf{d}}_T (\lambda x : A.sx) =_{\Pi x : A.B} (\lambda x : A.tx)$. Hence, we can apply eta $(x \notin \mathcal{V}s \cup \mathcal{V}t)$, sym and the admissible rule trans [?] which says that equality is transitive to get $\Gamma \vdash^{\mathsf{d}}_T s =_{\Pi x : A.B} t$ and therefore $\Gamma \vdash^{\mathsf{d}}_T \bot$. □

Lemma 6 ($\mathcal{T}_{\mathrm{FQ}}$). *Let* (T, Γ, Γ') *be a well-formed branch. Assume* $s =_{\Pi x : A.B} t \in T, \Gamma$ *and* $x \notin \mathcal{V}s \cup \mathcal{V}t$. *If* $\Gamma, [\forall x : A.sx = tx] \vdash^{\mathsf{d}}_T \bot$ *then* $\Gamma \vdash^{\mathsf{d}}_T \bot$.

Proof. From the assumptions, $\Gamma \vdash^{\mathsf{d}}_T \neg[s] =_o [\neg s]$, cong$\vdash$ and Lemma 7, we obtain $\Gamma \vdash^{\mathsf{d}}_T s =_{\Pi x:\ A.B} t$ and $\Gamma \vdash^{\mathsf{d}}_T \neg \forall x:\ A.sx =_B tx$. Furthermore, we have $\Gamma, x:A \vdash^{\mathsf{d}}_T sx =_B tx$ by refl and congAppl. Hence, $\forall$i yields $\Gamma \vdash^{\mathsf{d}}_T \forall x:\ A.sx =_B tx$ and we conclude by an application of $\neg$e. $\square$

Lemma 7 ($\mathcal{T}_{\mathrm{DEC}}$). *Let* (T, Γ, Γ') *be a well-formed branch. Assume*

$$x\ s_1\ \ldots\ s_n \neq_{a u_1 \ldots u_m} x\ t_1\ \ldots\ t_n \in T, \Gamma$$

and $\Gamma \vdash^{\mathsf{d}}_T x:\Pi y_1:\ A_1 \cdots \Pi y_n:\ A_n.a\ u'_1\ \ldots\ u'_m$ *where* $u_i = u'_i[y_1/s_1 \ldots y_n/s_n]$ *for* $1 \leqslant i \leqslant m$. *If* $\Gamma, s_i \neq_{A_i[x_1/s_1,\ldots,x_{i-1}/s_{i-1}]} t_i \vdash^{\mathsf{d}}_T \bot$ *for all* $1 \leqslant i \leqslant n$ *then* $\Gamma \vdash^{\mathsf{d}}_T \bot$.

Proof. From the assumptions, we obtain $\Gamma \vdash^{\mathsf{d}}_T s_i =_{A_i[x_1/s_1,\ldots,x_{i-1}/s_{i-1}]} t_i$ for all $1 \leqslant i \leqslant n$ and $\Gamma \vdash^{\mathsf{d}}_T x =_{\Pi y_1:\ A_1 \cdots \Pi y_n:\ A_n.a u'_1 \ldots u'_m} x$. Hence, n applications of the congruence rule for application yield $\Gamma \vdash^{\mathsf{d}}_T x\ s_1\ \ldots\ s_n =_{a u_1 \ldots u_m} x\ t_1\ \ldots\ t_n$. Since we also have $\Gamma \vdash^{\mathsf{d}}_T x\ s_1\ \ldots\ s_n \neq_{a u_1 \ldots u_m} x\ t_1\ \ldots\ t_n$, we obtain $\Gamma \vdash^{\mathsf{d}}_T \bot$. $\square$

Now we are ready to prove the soundness result for $\mathcal{T}$.

Theorem 3. *If* $(T, \Gamma, \cdot)$ *is well-formed and d-refutable then* $\Gamma \vdash^{\mathsf{d}}_T \bot$.

Proof. Let $(T, \Gamma, \cdot)$ be well-formed and d-refutable. We proceed by induction on the definition of d-refutability. If $(T, \Gamma, \cdot)$ is e-refutable, the result follows from Lemma 3 together with Corollary **??**. If $\bot \in T, \Gamma$ then clearly $\Gamma \vdash^{\mathsf{d}}_T \bot$. For the inductive case, consider a step $\langle (T, \Gamma, \cdot), (T, \Gamma_1, \cdot), \ldots, (T, \Gamma_n, \cdot) \rangle$ with some DHOL rule. Since $(T, \Gamma, \cdot)$ is d-refutable, all alternatives must be d-refutable. If we manage to show well-formedness of every alternative, we can apply the induction hypothesis to obtain $\Gamma_i \vdash^{\mathsf{d}}_T \bot$ for all $1 \leqslant i \leqslant n$. Then, we can conclude $\Gamma \vdash^{\mathsf{d}}_T \bot$ by soundness of the DHOL rules. Hence, it remains to prove well-formedness of the alternatives. In most cases, this is straightforward. We only show one interesting case, namely $\mathcal{T}_{\mathrm{DEC}}$.

Instead of proving $\Gamma, s_i \neq_{A_i[x_1/s_1,\ldots,x_{i-1}/s_{i-1}]} t_i \vdash^{\mathsf{d}}_T \bot$ for all $1 \leqslant i \leqslant n$ we show that $\Gamma \vdash^{\mathsf{d}}_T s_i =_{A_i[x_1/s_1,\ldots,x_{i-1}/s_{i-1}]} t_i$ for all $1 \leqslant i \leqslant n$. Since $(T, \Gamma, \cdot)$ is a well-formed branch, both s_1 and t_1 have type A_1. Hence, $(T, (\Gamma, s_1 \neq_{A_1} t_1), \cdot)$ is well-formed and our original induction hypothesis yields $\Gamma, s_1 \neq_{A_1} t_1 \vdash^{\mathsf{d}}_T \bot$ from which we obtain $\Gamma \vdash^{\mathsf{d}}_T s_1 =_{A_1} t_1$. Now let $i \leqslant n$ and assume we have $\Gamma \vdash^{\mathsf{d}}_T s_j =_{A_j[x_1/s_1,\ldots,x_{j-1}/s_{j-1}]} t_j$ for all $j < i$ (*). This is only possible if $\Gamma \vdash^{\mathsf{d}}_T t_j:A_j[x_1/s_1,\ldots,x_{j-1}/s_{j-1}]$ for all $j < i$. Since $(T, \Gamma, \cdot)$ is a well-formed branch, it is clear that $\Gamma \vdash^{\mathsf{d}}_T s_i:A_i[x_1/s_1,\ldots,x_{i-1}/s_{i-1}]$ and $\Gamma \vdash^{\mathsf{d}}_T t_i:A_i[x_1/t_1,\ldots,x_{i-1}/t_{i-1}]$. From (*), we obtain

$$\Gamma \vdash^{\mathsf{d}}_T t_i:A_i[x_1/s_1,\ldots,x_{i-1}/s_{i-1}],$$

so $(T, (\Gamma, s_i \neq_{A_i[x_1/s_1,\ldots,x_{i-1}/s_{i-1}]} t_i), \cdot)$ is well-formed. Hence, the original induction hypothesis yields $\Gamma \vdash^{\mathsf{d}}_T s_i =_{A_i[x_1/s_1,\ldots,x_i/s_{i-1}]} t_i$ as desired. $\square$

In the previous proof, we can see that for $\mathcal{T}_{\mathrm{DEC}}$, well-formedness of an alternative depends on refutability of all branches to the left. Note that the same

holds for $\mathcal{T}_{\mathrm{MAT}}$ and $\mathcal{T}_{\Rightarrow}$. This is a distinguishing feature of DHOL as in tableaux, branches are usually considered to be independent.

Finally, completeness is immediate from the completeness of the HOL tableau calculus and the erasure:

Theorem 4. *If $\Gamma \vdash^{\mathsf{d}}_T \bot$ then $(T, \Gamma, \cdot)$ is d-refutable.*

Proof. Let $\Gamma \vdash^{\mathsf{d}}_T \bot$. Using Corollary **??** and Lemma 2 we conclude s-refutability of $(T, \Gamma, \Phi(\overline{\Gamma}))$. By definition, $(T, \Gamma, \Phi(\overline{\Gamma}))$ is also e-refutable. Furthermore, by inspecting $\mathcal{T}_{\mathrm{ER}_1}$ and $\mathcal{T}_{\mathrm{ER}_2}$ we conclude that $(T, \Gamma, \cdot)$ is also e-refutable and therefore d-refutable.   □

4   Implementation

We implemented the tableau calculus for DHOL as an extension of Lash [6] which is a fork of Satallax, a successful automated theorem prover for HOL [4]. By providing an efficient C implementation of terms with perfect sharing as well as other important data structures and operations, Lash outperforms Satallax when it comes to the basic ground tableau calculus which both of them implement. However, Lash removes a lot of the additional features beyond the basic calculus that was implemented in Satallax. Nevertheless, this was actually beneficial for our purpose as we could concentrate on adapting the core part. Note that Lash and Satallax do not just implement the underlying ground tableau calculus but make heavy use of SAT-solving and a highly customizable priority queue to guide the proof search [4, 5].

For the extension of Lash to DHOL, the data structure for terms had to be changed to support dependent function types as well as quantifiers and lambda abstractions with dependent types. Of course, it would be possible to represent everything in the language of DHOL but the formulation of DHOL suggests that the prover should do as much as possible in the HOL fragment and only use "proper" DHOL when it is really necessary. With this in mind, the parser first always tries to produce simply-typed terms and only resorts to dependent types when it is unavoidable. Therefore, the input problem often looks like a mixture of HOL and DHOL even though everything is included in DHOL. A nice side effect of this design decision is that our extension of Lash works exactly like the original version on the HOL fragment except for the fact that it is expected to be slower due to the numerous case distinctions between simple types and dependent types which are needed in this setting.

Although DHOL is not officially part of TPTP THF, it can be expressed due to the existence of the `!>`-symbol which is used for polymorphism. Hence, a type $\Pi x\colon A.B$ is represented as `!>[X:A]:B`. For simplicity and efficiency reasons, we did not implement dependent types by distinguishing base types from their term arguments but represent the whole dependent type as a term. When parsing a base type a, Lash automatically creates an eponymous constant of type `tp` to be used in dependent types as well as a simple base type a_0 for the erasure and a constant a^* for its PER. The flags `DHOL_RULES_ONLY` and `DHOL_ERASURE_ONLY`

control the availability of the erasure as well as the native DHOL rules, respectively. Note that the implementation is not restricted to d-refutability but allows for arbitrary refutations. In the standard flag setting, however, only the native DHOL rules are used. Clearly, this constitutes a sound strategy. It is incomplete since the confrontation rule only considers equations with syntactically equivalent types. We have more to say about this in Section **??**.

4.1   Type Checking

By default, problems are only type-checked with respect to their simply-typed skeleton. If the option `exactdholtypecheck` is set, type constraints stemming from the term arguments of dependent base types are generated and added to the conjecture. The option `typecheckonly` discards the original conjecture, so Lash just tries to prove the type constraints. Since performing the type check involves proper theorem proving, we added the new SZS ontology statuses `TypeCheck` and `InexactTypecheck` to the standardized output of Lash. Here, the former one means that a problem type checks while the latter one just states that it type checks with respect to the simply-typed skeleton.

   For the generation of type constraints, each formula of the problem is traversed like in normal type checking. In addition, every time a type condition $a\,t_1\ \ldots\ t_n \equiv a\,s_1\ \ldots\ s_n$ comes up and there is some i such that s_i and t_i are not syntactically equivalent, a constraint stating that $s_i = t_i$ is provable is added to the set of type constraints. Note that it does not always suffice to just add $s_i = t_i$ as this equation may contain bound variables or only hold in the context in which the constraint appears. To that end, we keep relevant information about the term context when generating these constraints. Whenever a forall quantifier or lambda abstraction comes up, it is translated to a corresponding forall quantifier in the context since we want the constraint to hold in any case. While details like applications can be ignored, it is important to keep left-hand sides of implications in the context as it may be crucial for the constraint to be met. In general, any axiom may contribute to the typechecking proof.

Example 3. The conjecture

$$\forall n \colon \mathsf{nat}, x \colon \mathsf{lst}\, n.\ n =_{\mathsf{nat}} 0 \Rightarrow \mathsf{app}\, n\, n\, x\, x = x$$

is well-typed if the type constraint

$$\forall n \colon \mathsf{nat}, x \colon \mathsf{lst}\, n.\ n =_{\mathsf{nat}} 0 \Rightarrow \mathsf{plus}\, n\, n =_{\mathsf{nat}} n$$

is provable. Lash can generate this constraint and finds a proof quickly using the axiom $\forall n \colon \mathsf{nat}.\ \mathsf{plus}\, 0\, n =_{\mathsf{nat}} n$.

   Since conjunctions and disjunctions are internally translated to implications, it is important to note that we process formulas from left to right, i.e. for $x \colon \mathsf{lst}\, n$ and $y \colon \mathsf{lst}\, m$, the proposition $m \neq n \vee x = y$ type checks because we can assume $m = n$ to process $x = y$. Consequently, $x = y \vee m \neq n$ does not type check.

As formulas are usually read from left to right, this is a natural adaption of short-circuit evaluation in programming languages. Furthermore, it is in accordance with the presentation of Rothgang et al. [15] as well as the corresponding implementation in PVS [16]. As a matter of fact, PVS handles its undecidable type checking problem in essentially the same way as our new version of Lash by generating so called *type correctness conditions* (TCCs).

4.2   Implementation of the Rules

Given the appropriate infrastructure for dependent types, the implementation of most rules in Figure 1 is a straightforward extension of the original HOL implementation. For $\mathcal{T}_\forall$, the side condition $\Gamma \vdash^{\mathsf{d}}_T t\colon A$ is undecidable in general. It has been chosen to provide a simple characterization of the tableau calculus. Furthermore, it emphasizes that we do not instantiate with terms whose type does not literally match with the type of the quantified variable. In the implementation, we keep a pool of possible instantiations for types A which occur in the problem. The pool gets populated by terms of which we know that they have a given type because this information was available during parsing or proof search. Hence, we only instantiate with terms t for which we already know that $\Gamma \vdash^{\mathsf{d}}_T t\colon A$ holds.

Given an equation $s =_A t$, there are many candidate representations of A modulo type equality. When we build an equation in the implementation, we usually use the type of the left-hand side. Since all native DHOL rules of the tableau calculus enforce syntactically equivalent types, the ambiguity with respect to the type of an equation leads to problems. For example, consider a situation where $\Gamma \vdash^{\mathsf{d}}_T s\colon A$, $\Gamma \vdash^{\mathsf{d}}_T t\colon B$ and $\Gamma \vdash^{\mathsf{d}}_T s =_A t$ which implies $\Gamma \vdash^{\mathsf{d}}_T A \equiv B$. During proof search, it could be that $\Gamma \vdash^{\mathsf{d}}_T t \neq s$ is established. Clearly, this is a contradiction which leads to a refutation, but usually the inequality annotated with the type B which makes the refutation inaccessible for our native DHOL rules. Therefore, we implemented rules along the lines of

$$\mathcal{T}_{\mathrm{SYMCAST}_1} \ \frac{s =_A t}{t =_B s} \ t\colon B \qquad \mathcal{T}_{\mathrm{SYMCAST}_2} \ \frac{s \neq_A t}{t \neq_B s} \ t\colon B$$

which do not only apply symmetry but also change the type of the equality in a sound way. Like in $\mathcal{T}_\forall$, the side condition should be read as $\Gamma \vdash^{\mathsf{d}}_T t\colon B$ which makes it undecidable. However, in practice, we can compute a representative of the type of t given the available type information. While experimenting with the new DHOL version of Lash, the implementation of these rules proved to be very beneficial for refutations which only work with the DHOL rules. For the future, it is important to note that $\mathcal{T}_{\mathrm{SYMCAST}_1}$ and $\mathcal{T}_{\mathrm{SYMCAST}_2}$ are not sound for the extension of DHOL to predicate subtypes as $\Gamma \vdash^{\mathsf{d}}_T s =_A t$ and $\Gamma \vdash^{\mathsf{d}}_T t\colon B$ do not imply $\Gamma \vdash^{\mathsf{d}}_T A \equiv B$ anymore.

4.3   Generating Instantiations

Since Lash implements a ground tableau calculus, it does not support higher-order unification. Therefore, the generation of suitable instantiations is a major

issue. In the case of DHOL, it is actually beneficial that Lash already implements other means of generating instantiations since the availability of unification for DHOL is questionable: There exist unification procedures for dependent type theories (see for example [10]) but for DHOL such a procedure would also have to address the undecidable type equality problem.

For simple base types, it suffices to consider so-called *discriminating* terms to remain complete [1]. A term s of simple base type a is discriminating in a branch A if $s \neq_a t \in A$ or $t \neq_a s \in A$ for some term t. For function terms, completeness is guaranteed by enumerating all possible terms of a given type. Of course, this is highly impractical, and there is the important flag `INITIAL_SUBTERMS_AS_INSTANTIATIONS` which adds all subterms of the initial problem as instantiations. This heuristic works very well in many cases.

For dependent types, we do not check for type equality when instantiating quantifiers but only use instantiations with the exact same type (c.f. $\mathcal{T}_\forall$ in Figure 1) and let the erasure handle the remaining cases.

An interesting feature of this new version of Lash is the possibility to automatically generate instantiations for induction axioms. Given the constraints of the original implementation, the easiest way to sneak a term into the pool of instantiations is to include it into an easily provable lemma and then use the flag `INITIAL_SUBTERMS_AS_INSTANTIATIONS`. However, this adds unnecessary proof obligations, so we modified the implementation such that initial subterms as instantiations also include lambda-abstractions corresponding to forall quantifiers.

Example 4. Consider the induction axiom for lists:

$$\forall p \colon (\Pi n \colon \mathsf{nat}.\ \mathsf{lst}\ n \to o).\ p\ 0\ \mathsf{nil}$$
$$\Rightarrow (\forall n \colon \mathsf{nat}, x \colon \mathsf{elem}, y \colon \mathsf{lst}\ n.\ p\ n\ y \Rightarrow p\ (\mathsf{s}\ n)\ (\mathsf{cons}\ n\ x\ y))$$
$$\Rightarrow (\forall n \colon \mathsf{nat}, x \colon \mathsf{lst}\ n.\ p\ n\ x)$$

Even though it works for arbitrary predicates p, it is very hard for an ATP system to guess the correct instance for a given problem without unification in general. However, given the conjecture $\forall n \colon \mathsf{nat}, x \colon \mathsf{lst}\ n.\ \mathsf{app}\ n\ 0\ x\ \mathsf{nil} =_{\mathsf{lst}\ n} x$ we can easily read off the correct instantiation for p where $\forall$ is replaced by λ.

5   Case Study: List Reversal Is an Involution

Consider the following equational definition of the list reversal function rev:

$$\mathsf{rev}\ 0\ \mathsf{nil} =_{\mathsf{lst}\ 0} \mathsf{nil}$$
$$\forall n \colon \mathsf{nat}, x \colon \mathsf{elem}, y \colon \mathsf{lst}\ n.$$
$$\qquad \mathsf{rev}\ (\mathsf{s}\ n)\ (\mathsf{cons}\ n\ x\ y) =_{\mathsf{lst}\ (\mathsf{s}n)} \mathsf{app}\ n\ (\mathsf{s}\ 0)\ (\mathsf{rev}\ n\ y)\ (\mathsf{cons}\ 0\ x\ \mathsf{nil})$$

The conjecture

$$\forall n \colon \mathsf{nat}, x \colon \mathsf{lst}\ n.\ \mathsf{rev}\ n\ (\mathsf{rev}\ n\ x) =_{\mathsf{lst}\ n} x \qquad\qquad (\texttt{rev-invol})$$

Goal	Number of Problem Files
app-nil	4
app-assoc	8
app-assoc-m1	5
rev-invol-lem	12
rev-invol	5

Table 1. Amount of problem files per (intermediate) goal

is very easy to state, but turns out to be hard to prove automatically. The proof is based on the equational definitions of plus and app given in Example 1 as well as several induction proofs on lists using the axiom from Example 4. In particular, some intermediate goals are needed to succeed:

$$\forall n\colon \mathsf{nat}, x\colon \mathsf{lst}\, n.\ \mathsf{app}\, n\, 0\, x\, \mathsf{nil} =_{\mathsf{lst}\, n}\, x \qquad\qquad (\texttt{app-nil})$$

$$\forall n_1\colon \mathsf{nat}, x_1\colon \mathsf{lst}\, n_1, n_2\colon \mathsf{nat}, x_2\colon \mathsf{lst}\, n_2, n_3\colon \mathsf{nat}, x_3\colon \mathsf{lst}\, n_3. \qquad (\texttt{app-assoc})$$
$$\mathsf{app}\, n_1\, (\mathsf{plus}\, n_2\, n_3)\, x_1\, (\mathsf{app}\, n_2\, n_3\, x_2\, x_3)$$
$$= \mathsf{app}\, (\mathsf{plus}\, n_1\, n_2)\, n_3\, (\mathsf{app}\, n_1\, n_2\, x_1\, x_2)\, x_3$$

$$\forall n\colon \mathsf{nat}, x\colon \mathsf{lst}\, n, y\colon \mathsf{elem}, m\colon \mathsf{nat}, z\colon \mathsf{lst}\, m. \qquad (\texttt{app-assoc-m1})$$
$$\mathsf{app}\, (\mathsf{plus}\, n\, (\mathsf{s}\, 0))\, m\, (\mathsf{app}\, n\, (\mathsf{s}\, 0)\, x\, (\mathsf{cons}\, 0\, y\, \mathsf{nil}))\, z = \mathsf{app}\, n\, (\mathsf{s}\, m)\, x\, (\mathsf{cons}\, m\, y\, z)$$

$$\forall n\colon \mathsf{nat}, x\colon \mathsf{lst}\, n, m\colon \mathsf{nat}, y\colon \mathsf{lst}\, m. \qquad (\texttt{rev-invol-lem})$$
$$\mathsf{rev}\, (\mathsf{plus}\, n\, m)\, (\mathsf{app}\, n\, m\, (\mathsf{rev}\, n\, x)\, y) = \mathsf{app}\, m\, n\, (\mathsf{rev}\, m\, y)\, x$$

Note that for polymorphic lists, this is a standard example of an induction proof with lemmas (see e.g. [13, Section 2.2]). In the dependently-typed case, however, many intermediate equations would be ill-typed in interactive theorem provers like Coq or Lean. In order to succeed in automatically proving these problems, we had to break them down into separate problems for the instantiation of the induction axiom, the base case and the step case of the induction proofs. Often, we further needed to organize these subproblems in manageable steps. Overall, we created 34 TPTP problem files which are distributed over the intermediate goals as shown in Table 5. Note that already type checking these intermediate problems is not trivial: All type constraints are arithmetic equations, and given the Peano axioms, many of them need to be proven by induction themselves. Since we are mainly interested in the dependently-typed part, we added the needed arithmetical facts as axioms. Overall, the problem files have up to 18 axioms including the Peano axioms, selected arithmetical results, the defining equations of plus, app and rev as well as the list induction axiom. We left out unnecessary axioms in many problem files to make the proof search feasible.

With our new modes for DHOL which solely work with the native DHOL rules, Lash can type check and prove all problems easily. If we turn off the native DHOL rules and only work with the erasure using the otherwise same modes

with a 60s timeout, Lash can still typecheck all problems but it only manages to prove 7 out of 34 problems. In order to further evaluate the effectiveness of our new implementation, we translated all problems from DHOL to HOL using the *Logic Embedding Tool*[4], which performs the erasure from [15]. We then tested 16 other HOL provers available on *SystemOnTPTP*[5] on the translated problems with a 60s timeout (without type checking). We found that 5 of the 34 problems could only be solved by the DHOL version of Lash, including one problem where it only needs 5 inference steps. Detailed results as well as means to reproduce them are available on Lash's website[6] together with its source code.

6   Conclusion

Starting from the erasure from DHOL to HOL by Rothgang et al. [15], we developed a sound and complete tableau calculus for DHOL which we implemented in Lash. To the best of our knowledge, this makes it the first standalone automated theorem prover for DHOL. According to the experimental results, configurations where the erasure is performed as a preprocessing step for a HOL theorem prover can be outperformed by our new prover by solely using the native DHOL rules. We hope that this development will raise further interest in DHOL. Possible further work includes theoretical investigations such as the incorporation of choice operators into the erasure as well as a definition of the semantics of DHOL. Furthermore, it is desirable to officially define the TPTP syntax for DHOL which then opens the possibility of establishing a problem data set on which current and future tools can be compared. Finally, we would like to extend Lash to support predicate subtypes. Rothgang et al. already incorporated this into the erasure but there is no corresponding syntactic support in TPTP yet. In particular, this would get us much closer to powerful automation support for systems like PVS.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

---

[4] https://github.com/leoprover/logic-embedding
[5] https://tptp.org/cgi-bin/SystemOnTPTP
[6] http://cl-informatik.uibk.ac.at/software/lash-dhol/

References

1. Backes, J., Brown, C.E.: Analytic tableaux for higher-order logic with choice. Journal of Automated Reasoning **47**, 451–479 (2011). https://doi.org/10.1007/s10817-011-9233-2
2. Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P.: Superposition for higher-order logic. Journal of Automated Reasoning **67**, 10 (2023). https://doi.org/10.1007/s10817-022-09649-9
3. Bove, A., Dybjer, P., Norell, U.: A brief overview of Agda – a functional language with dependent types. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Proc. 22nd International Conference on Theorem Proving in Higher Order Logics. LNCS, vol. 5674, pp. 73–78 (2009). https://doi.org/10.1007/978-3-642-03359-9_6
4. Brown, C.E.: Satallax: An automatic higher-order prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Proc. 6th International Joint Conference on Automated Reasoning. LNAI, vol. 7364, pp. 111–117 (2012). https://doi.org/10.1007/978-3-642-31365-3_11
5. Brown, C.E.: Reducing higher-order theorem proving to a sequence of SAT problems. Journal of Automated Reasoning **51**, 57–77 (2013). https://doi.org/10.1007/s10817-013-9283-8
6. Brown, C.E., Kaliszyk, C.: Lash 1.0 (system description). In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) Proc. 11th International Joint Conference on Automated Reasoning. LNAI, vol. 13385, pp. 350–358 (2022). https://doi.org/10.1007/978-3-031-10769-6_21
7. Brown, C.E., Smolka, G.: Analytic tableaux for simple type theory and its first-order fragment. Logical Methods in Computer Science **6**(2), 1–33 (2010). https://doi.org/10.2168/LMCS-6(2:3)2010
8. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic **5**(2), 56–68 (1940). https://doi.org/10.2307/2266170
9. Coq Development Team: The Coq Reference Manual, Release 8.18.0 (2023)
10. Elliott, C.M.: Higher-order unification with dependent function types. In: Dershowitz, N. (ed.) Proc. 3rd International Conference on Rewriting Techniques and Applications. LNCS, vol. 355, pp. 121–136 (1989). https://doi.org/10.1007/3-540-51081-8_104
11. Mayr, R., Nipkow, T.: Higher-order rewrite systems and their confluence. Theoretical Computer Science **192**(1), 3–29 (1998). https://doi.org/10.1016/S0304-3975(97)00143-6
12. de Moura, L., Ullrich, S.: The Lean 4 theorem prover and programming language. In: Platzer, A., Sutcliffe, G. (eds.) Proc. 28th International Conference on Automated Deduction. LNAI, vol. 12699, pp. 625–635 (2021). https://doi.org/10.1007/978-3-030-79876-5_37
13. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer Berlin Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
14. Pfenning, F., Schürmann, C.: System description: Twelf – a meta-logical framework for deductive systems. In: Ganzinger, H. (ed.) Proc. 16th International Conference on Automated Deduction. LNAI, vol. 1632, pp. 202–206 (1999). https://doi.org/10.1007/3-540-48660-7_14
15. Rothgang, C., Rabe, F., Benzmüller, C.: Theorem proving in dependently-typed higher-order logic. In: Pientka, B., Tinelli, C. (eds.) Proc. 29th International Conference on Automated Deduction. LNAI, vol. 14132, pp. 438–455 (2023). https://doi.org/10.1007/978-3-031-38499-8_25

16. Rushby, J., Owre, S., Shankar, N.: Subtypes for specifications: Predicate subtyping in PVS. IEEE Transactions on Software Engineering **24**(9), 709–720 (1998). https://doi.org/10.1109/32.713327

17. Steen, A., Benzmüller, C.: Extensional higher-order paramodulation in Leo-III. Journal of Automated Reasoning **65**, 775–807 (2021). https://doi.org/10.1007/s10817-021-09588-x

A   Soundness of DHOL Rules

In the following lemmas, we prove the remaining cases of soundness of the DHOL tableau rules with respect to the natural deduction calculus in Figure 2. We begin with two auxiliary lemmas.

Lemma 8. *Let T, Γ and s be such that $\vdash^{\mathsf{d}} T$ Thy, $\vdash^{\mathsf{d}}_T \Gamma$ Ctx and $\Gamma \vdash^{\mathsf{d}}_T s : o$. If $\Gamma, s \vdash^{\mathsf{d}}_T \bot$ and $\Gamma, \neg s \vdash^{\mathsf{d}}_T \bot$, then $\Gamma \vdash^{\mathsf{d}}_T \bot$.*

Proof. Since $\Gamma, s \vdash^{\mathsf{d}}_T \bot$, we have $\Gamma \vdash^{\mathsf{d}}_T \neg s$. Since $\Gamma, \neg s \vdash^{\mathsf{d}}_T \bot$, we have $\Gamma \vdash^{\mathsf{d}}_T s$. Hence, $\Gamma \vdash^{\mathsf{d}}_T \bot$. □

Lemma 9. *If $\Gamma \vdash^{\mathsf{d}}_T (s =_A t) : o$ then $\Gamma \vdash^{\mathsf{d}}_T s : A$ and $\Gamma \vdash^{\mathsf{d}}_T t : A$.*

Proof. Let $\Gamma \vdash^{\mathsf{d}}_T (s =_A t) : o$. An inspection of the proof system in Figure 2 shows that this can only be inferred by the rule =type which requires $\Gamma \vdash^{\mathsf{d}}_T s : A$ and $\Gamma \vdash^{\mathsf{d}}_T t : A$. □

Lemma 10 ($\mathcal{T}_\neg$). *Let (T, Γ, Γ') be a well-formed branch. If $s \in T, \Gamma$ and $\neg s \in T, \Gamma$ then $\Gamma \vdash^{\mathsf{d}}_T \bot$.*

Proof. Clearly, $\Gamma \vdash^{\mathsf{d}}_T s$ and $\Gamma \vdash^{\mathsf{d}}_T \neg s$. Using the $\neg$e rule, we obtain $\Gamma \vdash^{\mathsf{d}}_T \bot$. □

Lemma 11 ($\mathcal{T}_{\neq}$). *Let (T, Γ, Γ') be a well-formed branch. If $s \neq_{at_1 \dots t_n} s \in T, \Gamma$ then $\Gamma \vdash^{\mathsf{d}}_T \bot$.*

Proof. From $s \neq_{at_1 \dots t_n} s \in T, \Gamma$ we obtain $\Gamma \vdash^{\mathsf{d}}_T (s \neq_{at_1 \dots t_n} s) : o$. Note that this can only be the case if $\Gamma \vdash^{\mathsf{d}}_T (s =_{at_1 \dots t_n} s) : o$. Then, Lemma 6 yields $\Gamma \vdash^{\mathsf{d}}_T s : a\, t_1 \dots t_n$. Hence, we can use the refl rule to get $\Gamma \vdash^{\mathsf{d}} s =_{at_1 \dots t_n} s$ and therefore $\Gamma \vdash^{\mathsf{d}}_T \bot$ by $\neg$e. □

Lemma 12 ($\mathcal{T}_{\neg\neg}$). *Let (T, Γ, Γ') be a well-formed branch. Assume $\neg\neg s \in T, \Gamma$. If $\Gamma, s \vdash^{\mathsf{d}}_T \bot$ then $\Gamma \vdash^{\mathsf{d}}_T \bot$.*

Proof. From $\Gamma, s \vdash^{\mathsf{d}}_T \bot$ we conclude $\Gamma \vdash^{\mathsf{d}}_T \neg s$. We have $\Gamma \vdash^{\mathsf{d}}_T \neg\neg s$ and therefore $\Gamma \vdash^{\mathsf{d}}_T s$ by $\neg\neg$e. An application of $\neg$e concludes the proof. □

Lemma 13 ($\mathcal{T}_{\Rightarrow}$). *Let (T, Γ, Γ') be a well-formed branch. Assume $s \Rightarrow t \in T, \Gamma$. If $\Gamma, \neg s \vdash^{\mathsf{d}}_T \bot$ and $\Gamma, t \vdash^{\mathsf{d}}_T \bot$ then $\Gamma \vdash^{\mathsf{d}}_T \bot$.*

Proof. From the assumptions, we get $\Gamma \vdash^{\mathsf{d}}_T s$ and $\Gamma \vdash^{\mathsf{d}}_T \neg t$. Together with $\Gamma \vdash^{\mathsf{d}}_T s \Rightarrow t$, we obtain $\Gamma \vdash^{\mathsf{d}}_T t$ and therefore $\Gamma \vdash^{\mathsf{d}}_T \bot$. □

Lemma 14 ($\mathcal{T}_{\neg\Rightarrow}$). *Let (T, Γ, Γ') be a well-formed branch. Assume $\neg(s \Rightarrow t) \in T, \Gamma$. If $\Gamma, s, \neg t \vdash_T^{\mathsf{d}} \bot$ then $\Gamma \vdash_T^{\mathsf{d}} \bot$.*

Proof. From the assumption, we get $\Gamma \vdash_T^{\mathsf{d}} s \Rightarrow t$. Hence, an application of $\neg$e yields $\Gamma \vdash_T^{\mathsf{d}} \bot$. □

Lemma 15 ($\mathcal{T}_\forall$). *Let (T, Γ, Γ') be a well-formed branch. Assume $\forall x \colon A.s \in T, \Gamma$ and $\Gamma \vdash_T^{\mathsf{d}} t \colon A$. If $\Gamma, [s[x/t]] \vdash_T^{\mathsf{d}} \bot$ then $\Gamma \vdash_T^{\mathsf{d}} \bot$.*

Proof. From $\Gamma, [s[x/t]] \vdash_T^{\mathsf{d}} \bot$ we get $\Gamma \vdash_T^{\mathsf{d}} \neg[s[x/t]]$. We also have $\Gamma \vdash_T^{\mathsf{d}} \forall x \colon A.s$ and therefore $\Gamma \vdash_T^{\mathsf{d}} [s[x/t]]$ by $\forall$e and Lemma 7. Finally, $\Gamma \vdash_T^{\mathsf{d}} \bot$ by $\neg$e. □

Lemma 16 ($\mathcal{T}_{\neg\forall}$). *Let (T, Γ, Γ') be a well-formed branch. Assume $\neg\forall x \colon A.s \in T, \Gamma$ and y is fresh for T, Γ. If $\Gamma, y \colon A, \neg[s[x/y]] \vdash_T^{\mathsf{d}} \bot$ then $\Gamma \vdash_T^{\mathsf{d}} \bot$.*

Proof. From $\Gamma, y \colon A, \neg[s[x/y]] \vdash_T^{\mathsf{d}} \bot$ we obtain $\Gamma, y \colon A \vdash_T^{\mathsf{d}} s[x/y]$ using Lemma 7. An application of $\forall$i yields $\Gamma \vdash_T^{\mathsf{d}} \forall y \colon A.s[x/y]$ which is the same as $\Gamma \vdash_T^{\mathsf{d}} \forall x \colon A.s$. Since we also have $\Gamma \vdash_T^{\mathsf{d}} \neg\forall x \colon A.s$, we conclude $\Gamma \vdash_T^{\mathsf{d}} \bot$ by $\neg$e. □

Lemma 17 ($\mathcal{T}_{\mathrm{BE}}$). *Let (T, Γ, Γ') be a well-formed branch. Assume $s \neq_o t \in T, \Gamma$. If $\Gamma, s, \neg t \vdash_T^{\mathsf{d}} \bot$ and $\Gamma, \neg s, t \vdash_T^{\mathsf{d}} \bot$ then $\Gamma \vdash_T^{\mathsf{d}} \bot$.*

Proof. From the assumptions we obtain $\Gamma, s \vdash_T^{\mathsf{d}} t$ and $\Gamma, \neg s \vdash_T^{\mathsf{d}} \neg t$ which is equivalent to $\Gamma, t \vdash_T^{\mathsf{d}} s$. Hence, the rule

$$\frac{\Gamma, s \vdash_T t \quad \Gamma, t \vdash_T s}{\Gamma \vdash_T s =_o t}\mathrm{propExt}$$

which is proven admissible in the appendix of [**?**] (the extended version of [15]) yields $\Gamma \vdash_T^{\mathsf{d}} s =_o t$ and we obtain $\Gamma \vdash_T^{\mathsf{d}} \bot$ with $\neg$e. □

Lemma 18 ($\mathcal{T}_{\mathrm{BQ}}$). *Let (T, Γ, Γ') be a well-formed branch. Assume $s =_o t \in T, \Gamma$. If $\Gamma, s, t \vdash_T^{\mathsf{d}} \bot$ and $\Gamma, \neg s, \neg t \vdash_T^{\mathsf{d}} \bot$ then $\Gamma \vdash_T^{\mathsf{d}} \bot$.*

Proof. By assumption, we have $\Gamma, s \vdash_T^{\mathsf{d}} s =_o t$ and $\Gamma, s \vdash_T^{\mathsf{d}} s$. Hence, we obtain $\Gamma, s \vdash_T^{\mathsf{d}} t$ by cong$\vdash$. Furthermore, $\neg$i applied to $\Gamma, s, t \vdash_T^{\mathsf{d}} \bot$ yields $\Gamma, s \vdash_T^{\mathsf{d}} \neg t$. Hence, we arrive at $\Gamma, s \vdash_T^{\mathsf{d}} \bot$ by an application of $\neg$e. In a similar way, we can derive $\Gamma, \neg s \vdash_T^{\mathsf{d}} \bot$. Therefore, an application of Lemma 5 gives us the desired result. □

Lemma 19 ($\mathcal{T}_{\mathrm{MAT}}$). *Let (T, Γ, Γ') be a well-formed branch. Assume $x s_1 \dots s_n \in T, \Gamma$, $\neg x\, t_1\, \dots\, t_n \in T, \Gamma$ and let $\Gamma \vdash_T^{\mathsf{d}} x \colon \Pi y_1 \colon A_1 \cdots \Pi y_n \colon A_n.o$. If*

$$\Gamma, s_i \neq_{A_i[x_1/s_1, \dots, x_{i-1}/s_{i-1}]} t_i \vdash_T^{\mathsf{d}} \bot$$

for all $1 \leqslant i \leqslant n$ then $\Gamma \vdash_T^{\mathsf{d}} \bot$.

Proof. Note that $x s_1 \dots s_n \in T, \Gamma$ and $\neg x\, t_1\, \dots\, t_n \in T, \Gamma$ implies

$$\Gamma \vdash_T^{\mathsf{d}} x\, s_1\, \dots\, s_n \neq_o x\, t_1\, \dots\, t_n.$$

Since in Lemma 4 it makes no difference whether the assumption is contained in T, Γ or is merely provable from it, Lemma 4 can be used to complete the proof. □

Lemma 20 ($\mathcal{T}_{\text{CON}}$). *Let* (T, Γ, Γ') *be a well-formed branch. Assume* $s =_{at_1 \ldots t_n} t, u \neq_{at_1 \ldots t_n} v \in T, \Gamma$. *If* $\Gamma, s \neq u, t \neq u \vdash^d_T \bot$ *and* $\Gamma, s \neq v, t \neq v \vdash^d_T \bot$ *then* $\Gamma \vdash^d_T \bot$.

Proof. Let $\Gamma, s \neq u, t \neq u \vdash^d_T \bot$ and $\Gamma, s \neq v, t \neq v \vdash^d_T \bot$. By Lemma 5 it is enough to prove $\Gamma, s = u \vdash^d_T \bot$ and $\Gamma, s \neq u \vdash^d_T \bot$.

We first prove $\Gamma, s = u \vdash^d_T \bot$. By Lemma 5 it is enough to prove $\Gamma, s = u, t = v \vdash^d_T \bot$ and $\Gamma, s = u, t \neq v \vdash^d_T \bot$. Using $s = t \in T, \Gamma$ and symmetry and transitivity (which is an admissible rule [?]) we can infer $\Gamma, s = u, t = v \vdash^d_T u = v$. This along with $u \neq v \in T, \Gamma$ gives $\Gamma, s = u, t = v \vdash^d_T \bot$. In order to prove $\Gamma, s = u, t \neq v \vdash^d_T \bot$ we again use Lemma 5 to reduce to proving $\Gamma, s = u, t \neq v, s = v \vdash^d_T \bot$ and $\Gamma, s = u, t \neq v, s \neq v \vdash^d_T \bot$. We know $\Gamma, s = u, t \neq v, s \neq v \vdash^d_T \bot$ already since $\Gamma, t \neq v, s \neq v \vdash^d_T \bot$. By symmetry and transitivity, $\Gamma, s = u, t \neq v, s = v \vdash^d_T u = v$ giving $\Gamma, s = u, t \neq v, s = v \vdash^d_T \bot$. Hence $\Gamma, s = u, t \neq v \vdash^d_T \bot$ and $\Gamma, s = u \vdash^d_T \bot$.

All that remains is to prove $\Gamma, s \neq u \vdash^d_T \bot$. By Lemma 5 it is enough to prove $\Gamma, s \neq u, t = u \vdash^d_T \bot$ since we already know $\Gamma, s \neq u, t \neq u \vdash^d_T \bot$. By Lemma 5 it is enough to prove $\Gamma, s \neq u, t = u, t = v \vdash^d_T \bot$ and $\Gamma, s \neq u, t = u, t \neq v \vdash^d_T \bot$. We have $\Gamma, s \neq u, t = u, t = v \vdash^d_T \bot$ since $\Gamma, s \neq u, t = u, t = v \vdash^d_T u = v$ by symmetry and transitivity. By Lemma 5 all that remains is to prove $\Gamma, s \neq u, t = u, t \neq v, s = v \vdash^d_T \bot$ and $\Gamma, s \neq u, t = u, t \neq v, s \neq v \vdash^d_T \bot$. We know $\Gamma, s \neq u, t = u, t \neq v, s \neq v \vdash^d_T \bot$ since $\Gamma, t \neq v, s \neq v \vdash^d_T \bot$. We finally have $\Gamma, s \neq u, t = u, t \neq v, s = v \vdash^d_T \bot$ as a consequence of $\Gamma, s \neq u, t = u, t \neq v, s = v \vdash^d_T u = v$, which again follows from symmetry and transitivity. □