Subtyping in Dependently-Typed Higher-Order Logic

Colin Rothgang[1,2] [0000−0001−9751−8989] and Florian Rabe[3] [0000−0003−3040−3655]

[1] Imdea Software Institute, Madrid, Spain
[2] Universidad Politécnica de Madrid, Madrid, Spain
[3] Computer Science, FAU Erlangen-Nürnberg, Germany

Abstract. The recently introduced dependent typed higher-order logic (DHOL) offers an interesting compromise between expressiveness and automation support. It sacrifices the decidability of its type system in order to significantly extend its expressiveness over standard HOL. Yet it retains strong automated theorem proving support via a sound and complete translation to HOL.

We leverage this design to extend DHOL with refinement and quotient types. Both of these are commonly requested by practitioners but rarely provided by automated theorem provers. This is because they inherently require undecidable typing and thus are very difficult to retrofit to decidable type systems. But with DHOL already doing the heavy lifting, adding them is not only possible but elegant and simple.

Concretely, we add refinement and quotient types as special cases of subtyping. This turns the associated canonical inclusion resp. projection maps into identity maps and thus avoids costly changes in representation. We present the syntax, semantics, and translation to HOL for the extended language, including the proofs of soundness and completeness.

1 Introduction and Related Work

Motivation Recently dependently typed higher-order logic (DHOL) was introduced [22]. It is a variant of HOL [6,2] that uses dependent function types $\Pi x{:}A.\ B$ instead of simple function types $A \to B$. It is designed to remain as simple and as close to HOL and ATPs as possible while meeting the frequent user demand of dependent types. Notably, contrary to typical formulations of dependent type theory, DHOL features a straightforward equality and classical Booleans at the cost of making typing undecidable.

Concretely, DHOL uses a type bool of propositions in the style of HOL, and equality $s =_A t \colon \mathsf{bool}$ of typed terms is a proposition, whose truth may depend on axioms in the theory or assumptions in the context. Equality $A \equiv B$ of types (which is not a proposition but a meta-level judgment) uses a straightforward congruence rule: if a dependent type constructor is applied to equal arguments, it produces equal types. Thus, equality of types and typing are undecidable. To yield practical tool support, DHOL reduces typing judgments to a series of proof

obligations, and [22] gives a sound and complete translation to HOL that allows using existing automated theorem provers for HOL to discharge these.

While undecidable typing is not used by most current ATPs or ITPs, it is justified by the pragmatic consideration that the ultimate task of theorem proving is undecidable anyway, and the difficulty of typing-related proof obligations is often small in comparison. Follow-up work on DHOL includes a native ATP for DHOL [17] and extensions with a choice operator [20] and polymorphism [21].

Contribution The present paper leverages this key design choice and extends DHOL's expressivity at low cost by adding refinement and quotient types. Both work elegantly in languages with undecidable typing so that DHOL, for which the necessary meta-theory and infrastructure already exist, is a good base to support them. Indeed, the necessary changes to DHOL's syntax and semantics turned out to be few and simple — only the extension of the proof was difficult. We see DHOL as an intermediate between the automation support of HOL and the more expressive type theories of interactive provers such as those based on richer dependent type theories. In this sense, the present paper pushes the boundary of ATP-near languages a little further.

Refinement types $A|_p$ is the subtype of A consisting of the terms satisfying the predicate $p{:}A \to \mathsf{bool}$. They correspond to comprehension in set theory. They were already proposed in [22] (with ad-hoc subtyping rules), and here we give a systematic treatment. Critically, our refinement types avoid a change in representation: the injection $A|_p \to A$ is always a no-op, i.e., an identity map. This is in contrast to encodings of refinement types in decidable type theories, such as using the type $\Sigma x{:}A.p\,x$, or in set-theory, where, e.g., the injection $(A \to B) \to (A|_p \to B)$ is not a no-op.

Quotient types A/r, intuitively, consist of all equivalence classes of the equivalence relation $r{:}A \to A \to \mathsf{bool}$. Again we avoid representation changes: We use all terms of type A as terms of type A/r and adjust the equality $=_{A/r}$ to obtain the quotient semantics. Thus, the projection $A \to A/r$ is a no-op and we have the subtyping relation $A \prec{:} A/r$. In contrast, the usual definitions in set theory (via equivalence classes) or in decidable type theories (via setoids) require explicit changes in representation.

The statement $A \prec{:} A/r$ may look odd. It is sound because we use a different equality relation at the two types: $x =_A y$ implies $x =_{A/r} y$ but not the other way round. This approach captures the mathematical practice of using elements of A as elements of the quotient, often to the point that readers do not even notice anymore they are technically working with equivalence classes.

Together, this yields a subtype hierarchy of refinements and quotients of A: $A|_{\lambda x:A.\ \mathsf{false}} \prec{:} \ldots \prec{:} A|_p \prec{:} \ldots \prec{:} A|_{\lambda x:A.\ \mathsf{true}} \equiv A \equiv A/=_A \prec{:} \ldots \prec{:} A/r \prec{:} \ldots \prec{:} A/\lambda x, y{:}A.\ \mathsf{true}$ from initial (empty) type to terminal (singleton) type.

Related Work Systems based on HOL [11] use the subtype definition principle to introduce definitional refinement types. General refinement types in HOL were

considered in [15]. But neither solution has a general notion of subtyping. Definitional quotient types for HOL were considered in [12] and [13] and, combined with refinement types, in [19]. Definitional solutions differ from ours in that they introduce new types that are isomorphic to the refinement/quotient types and then require explicit representation-changing injection/projections. [19] uses PERs to relativize type-based to set-based theorems.

In formal systems, the approach of quotients as supertypes has been adopted occasionally, e.g., in NuPRL's quotients [7] or in Quotient Haskell [3]. NuPRL's type theory in particular features refinement and quotient types similar to ours and uses essentially the same PER semantics [1]. The main difference to our approach is that DHOL tries to be as close to HOL (and thus HOL ATPs) as possible whereas NuPRL uses a very rich type theory.

PVS [18] subsumes DHOL and refinement types with polymorphism. It does not support quotients, but its horizontal subtyping between records resembles our quotient subtyping. Notably, it treats refinement subtyping and record subtyping as two separate judgments with slightly different rules.

In soft type systems, all types are refinements of a fixed universe of objects. For example, Mizar's [4] type system is inherently undecidable and supports dependent types and refinement types. It supports quotient types but only with a change in representation and not as supertypes.

Most ITPs allow users to construct refinement and quotient types at the cost of representation changes. Systems based on decidable dependent type theory like Coq [8] or Lean [9] typically use Σ-types for refinement and setoids for quotients. Lean provides some kernel support for quotients.

[21] adds polymorphism to DHOL, and in future work we want to combine both features. The key challenge will be to support *subtype-bounded* polymorphism.

Overview We give a self-contained definition of DHOL in Sect. 2. Then we add subtyping in Sect. 3, refinements in Sect. 4, and quotients in Sect. 5. We develop the meta-theory in Sect. 6 (type normalization) and Sect. 7 (soundness/completeness). We present an application to typed set theory in Sect. 8.

2 Preliminaries: Dependently Typed Higher-Order Logic

The DHOL [22] **grammar** uses terms and types. A theory T declares typed constants $\mathtt{c}{:}A$, axioms $\triangleright F$, and dependent type symbols $\mathtt{a}{:}\Pi x_1{:}A_1.\ \ldots \Pi x_n{:}A_n.\ \mathsf{tp}$, which are applied to terms to obtain base types $\mathtt{a}\ t_1\ \ldots\ t_n$. Contexts declare typed variables $x{:}A$ and local assumptions $\triangleright F$ (but no new types). Dependent functions $\lambda x{:}A.\ t$ of type $\Pi x{:}A.\ B$ (written $A \to B$ if x does not occur free in B) map terms $x{:}A$ to terms of type $B(x)$. We recover HOL as the fragment in

which all base types $\mathtt{a}$ have arity 0, in which case all function types are simple.

T	::=	$\circ \mid T, \mathtt{a}{:}(\Pi x{:}A.)^* \mathsf{tp} \mid T, \mathtt{c}{:}A \mid T, \triangleright F$	theories
Γ	::=	$. \mid \Gamma, x{:}A \mid \Gamma, \triangleright F$	contexts
A, B	::=	$\mathtt{a}\ t^* \mid \Pi x{:}A.\ B \mid \mathsf{bool}$	types
s, t, F, G	::=	$\mathtt{c} \mid x \mid \lambda x{:}A.\ t \mid s\ t \mid s =_A t \mid F \Rightarrow G$	terms (incl. propositions)

Following typical HOL-style [2], DHOL defines all connectives and quantifiers from the equality connective $s =_A t$. For example, forall is defined as $\forall\, x{:}A.\ F := \lambda x{:}A.\ F =_{\Pi x:A.\ \mathsf{bool}} \lambda x{:}A.\ \mathsf{true}$. In particular, we can define the usual connectives and quantifiers and derive their usual proof rules. The only subtlety is that, DHOL needs *dependent* binary connectives: in an implication $F \Rightarrow G$, the well-formedness of G may depend on the truth of F, and accordingly for conjunction $F \wedge G$ which is defined as $\neg(F \Rightarrow \neg G)$. Because we see no way to define these from equality, we make dependent implication an additional primitive.

DHOL uses axiomatic equality $s =_A t$ in the style of FOL and HOL with a straightforward congruence rule for base types: type equality $\mathtt{a}\ s_1\ \ldots\ s_n \equiv \mathtt{a}\ t_1\ \ldots\ t_n$ holds if each s_i is equal to t_i. This makes type equality and thus typing undecidable. In line with HOL's simplicity and unlike dependent type theories based on Martin-Löf type theory [16], there is no support for type-valued computation like large elimination.

Example 1 (Lists). As an accessible running example, we show a formalization of lists over some type $\mathtt{obj}$, both plain lists list and lists llist n with fixed length. Notably, the well-typedness of the statement of associativity of lconc now requires the associativity of plus.

nat: tp, zero: nat, succ: nat $\to$ nat, plus: nat $\to$ nat $\to$ nat,
$\mathtt{obj}$: tp, list: tp, nil: list, cons: $\mathtt{obj} \to$ list $\to$ list, conc: list $\to$ list $\to$ list,
llist: nat $\to \mathsf{tp}$, lnil: llist zero,
lcons: Πn:nat. $\mathtt{obj} \to$ llist $n \to$ llist (succ n),
lconc: $\Pi m, n$:nat. llist $m \to$ llist $n \to$ llist (plus m n)

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

Fig. 1. DHOL Judgments

Theories and contexts:

$$\frac{}{\vdash \circ\ \mathsf{Thy}} \qquad \frac{\vdash_T x_1{:}A_1,\ \ldots,\ x_n{:}A_n\ \mathsf{Ctx}}{\vdash T,\ \mathsf{a}{:}\Pi x_1{:}A_1.\ \ldots \Pi x_n{:}A_n.\ \mathsf{tp}\ \mathsf{Thy}} \qquad \frac{\vdash_T A\ \mathsf{tp}}{\vdash T,\ \mathsf{c}{:}A\ \mathsf{Thy}} \qquad \frac{\vdash_T F{:}\mathsf{bool}}{\vdash T,\ \triangleright F\ \mathsf{Thy}}$$

$$\frac{\vdash T\ \mathsf{Thy}}{\vdash_T \mathsf{Ctx}} \qquad \frac{\Gamma \vdash_T A\ \mathsf{tp}}{\vdash_T \Gamma,\ x{:}A\ \mathsf{Ctx}} \qquad \frac{\Gamma \vdash_T F{:}\mathsf{bool}}{\vdash_T \Gamma,\ \triangleright F\ \mathsf{Ctx}}$$

Well-formedness and equality of types:

$$\frac{\mathsf{a}{:}\Pi x_1{:}A_1.\ \ldots \Pi x_n{:}A_n.\ \mathsf{tp}\ \text{in}\ T \qquad \Gamma \vdash_T t_1{:}A_1 \quad \ldots \quad \Gamma \vdash_T t_n{:}A_n[x_1/t_1]\ldots[x_{n-1}/t_{n-1}]}{\Gamma \vdash_T \mathsf{a}\ t_1\ \ldots\ t_n\ \mathsf{tp}} \qquad \frac{\vdash_T \Gamma\ \mathsf{Ctx}}{\Gamma \vdash_T \mathsf{bool}\ \mathsf{tp}} \qquad \frac{\Gamma \vdash_T A\ \mathsf{tp} \quad \Gamma,\ x{:}A \vdash_T B\ \mathsf{tp}}{\Gamma \vdash_T \Pi x{:}A.\ B\ \mathsf{tp}}$$

$$\frac{\mathsf{a}{:}\Pi x_1{:}A_1.\ \ldots \Pi x_n{:}A_n.\ \mathsf{tp}\ \text{in}\ T \qquad \Gamma \vdash_T s_1 =_{A_1} t_1\ \ldots\ \Gamma \vdash_T s_n =_{A_n[x_1/t_1]\ldots[x_{n-1}/t_{n-1}]} t_n}{\Gamma \vdash_T \mathsf{a}\ s_1\ \ldots\ s_n \equiv \mathsf{a}\ t_1\ \ldots t_n} \qquad \frac{\vdash_T \Gamma\ \mathsf{Ctx}}{\Gamma \vdash_T \mathsf{bool} \equiv \mathsf{bool}} \qquad \frac{\Gamma \vdash_T A \equiv A' \quad \Gamma, x{:}A \vdash_T B \equiv B'}{\Gamma \vdash_T \Pi x{:}A.\ B \equiv \Pi x{:}A'.\ B'}$$

Typing:

$$\frac{c{:}A'\ \text{in}\ T \qquad \Gamma \vdash_T A' \equiv A}{\Gamma \vdash_T \mathsf{c}{:}A} \qquad \frac{\Gamma,\ x{:}A \vdash_T t{:}B \qquad A' \equiv A}{\Gamma \vdash_T (\lambda x{:}A.\ t){:}\Pi x{:}A'.\ B} \qquad \frac{\Gamma \vdash_T F{:}\mathsf{bool} \qquad \Gamma,\ \triangleright F \vdash_T G{:}\mathsf{bool}}{\Gamma \vdash_T F \Rightarrow G{:}\mathsf{bool}}$$

$$\frac{x{:}A'\ \text{in}\ \Gamma \qquad \Gamma \vdash_T A' \equiv A}{\Gamma \vdash_T x{:}A} \qquad \frac{\Gamma \vdash_T f{:}\Pi x{:}A.\ B \qquad \Gamma \vdash_T t{:}A}{\Gamma \vdash_T f\ t{:}B[x/t]} \qquad \frac{\Gamma \vdash_T s{:}A \qquad \Gamma \vdash_T t{:}A}{\Gamma \vdash_T s =_A t{:}\mathsf{bool}}$$

Equality: congruence, reflexivity, symmetry, β, η (transitivity and extensionality are derivable):

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

$$\frac{\Gamma \vdash_T t{:}A}{\Gamma \vdash_T t =_A t} \qquad \frac{\Gamma \vdash_T t =_A s}{\Gamma \vdash_T s =_A t} \qquad \frac{\Gamma \vdash_T (\lambda x{:}A.\ s)\ t{:}B}{\Gamma \vdash_T (\lambda x{:}A.\ s)\ t =_B s[x/t]} \qquad \frac{\Gamma \vdash_T t{:}\Pi x{:}A.\ B}{\Gamma \vdash_T t =_{\Pi x:A.\ B} \lambda x{:}A.\ t\ x}$$

Rules for validity: lookup, implication, Boolean equality and Boolean extensionality

$$\frac{\triangleright F\ \text{in}\ T \qquad \vdash_T \Gamma\ \mathsf{Ctx}}{\Gamma \vdash_T F} \qquad \frac{\Gamma,\ \triangleright F \vdash_T G}{\Gamma \vdash_T F \Rightarrow G} \qquad \frac{\Gamma \vdash_T F =_{\mathsf{bool}} F' \qquad \Gamma \vdash_T F'}{\Gamma \vdash_T F}$$

$$\frac{\triangleright F\ \text{in}\ \Gamma \qquad \vdash_T \Gamma\ \mathsf{Ctx}}{\Gamma \vdash_T F} \qquad \frac{\Gamma \vdash_T F \Rightarrow G \qquad \Gamma \vdash_T F}{\Gamma \vdash_T G} \qquad \frac{\Gamma \vdash_T p\ \mathsf{true} \qquad \Gamma \vdash_T p\ \mathsf{false}}{\Gamma, x{:}\mathsf{bool} \vdash_T p\ x}$$

Fig. 2. DHOL Rules

DHOL uses the **judgments** given in Fig. 1 and the **rules** listed in Fig. 2. Note that equality of terms is a special case of validity, whereas equality of types is not a Boolean but a separate judgment. Thus, users cannot state axioms equating types, and type equality is defined only by congruence. The rules are straightforward. In particular, type equality is checked structurally and reduced to a set of term equalities, which must then be discharged by an ATP.

Furthermore many well-known admissible HOL rules are also admissible in DHOL, see the extended preprint[5].

The **semantics** for DHOL and a **practical ATP workflow** are given by a sound and complete translation to HOL. The translation is dependency erasure, e.g.,

$$\begin{array}{rl}
\text{Theories and contexts:} & \overline{\circ} := \circ \qquad \overline{\Gamma,\ D} := \overline{\Gamma},\ \overline{D} \qquad \overline{\cdot} := \cdot \qquad \overline{\Gamma,\ D} := \overline{\Gamma},\ \overline{D} \\
& \overline{\mathsf{a}{:}\Pi x_1{:}A_1.\ \ldots\ \Pi x_n{:}A_n.\ \mathsf{tp}} := \mathsf{a}{:}\mathsf{tp},\quad a^*{:}\overline{A_1} \to \ldots \to \overline{A_n} \to \mathsf{a} \to \mathsf{a} \to \mathsf{bool}, \\
& \quad \triangleright \forall x_1{:}\overline{A_1}.\ \ldots \forall x_n{:}\overline{A_n}.\ \forall u,v{:}\mathsf{a}.\ a^*\ x_1\ \ldots\ x_n\ u\ v \Rightarrow u =_a v \\
& \overline{\mathsf{c}{:}A} := \mathsf{c}{:}\overline{A},\ \triangleright A^*\ \mathsf{c}\ \mathsf{c} \qquad \overline{x{:}A} := x{:}\overline{A},\ \triangleright A^*\ x\ x \\
& \overline{\triangleright F} := \triangleright \overline{F} \qquad \overline{\triangleright F} := \triangleright \overline{F} \\
\text{Types:} & \overline{\mathsf{a}\ t_1\ \ldots\ t_n} := \mathsf{a} \qquad (\mathsf{a}\ t_1\ \ldots\ t_n)^*\ s\ t := a^*\ \overline{t_1}\ \ldots\ \overline{t_n}\ s\ t \\
& \overline{\Pi x{:}A.\ B} := \overline{A} \to \overline{B} \qquad (\Pi x{:}A.\ B)^*\ f\ g := \forall x,y{:}\overline{A}.\ A^*\ x\ y \Rightarrow B^*\ (f\ x)\ (g\ y) \\
& \overline{\mathsf{bool}} := \mathsf{bool} \qquad \mathsf{bool}^*\ s\ t := s =_{\mathsf{bool}} t \\
\text{Terms:} & \overline{\mathsf{c}} := \mathsf{c} \qquad \overline{x} := x \qquad \overline{\lambda x{:}A.\ t} := \lambda x{:}\overline{A}.\ \overline{t} \qquad \overline{f\ t} := \overline{f}\ \overline{t} \\
& \overline{s =_A t} := A^*\ \overline{s}\ \overline{t} \qquad \overline{F \Rightarrow G} := \overline{F} \Rightarrow \overline{G}
\end{array}$$

if in DHOL	then in HOL
type A	type $\overline{A}$ and PER $A^*{:}\overline{A} \to \overline{A} \to \mathsf{bool}$
term $t{:}A$	term $\overline{t}{:}\overline{A}$ satisfying $A^*\ \overline{t}\ \overline{t}$

Fig. 3. Definition of the Translation DHOL→HOL

translating dependent types $\mathsf{a}\ t_1\ \ldots\ t_n$ to simple types a, effectively "'merging"' all instances of a dependent type into a large simple type. Fig. 3 shows the details.

Typing and equality at A are recovered by generating a partial equivalence relation (PER) A^* for every HOL-type $\overline{A}$. A PER is a symmetric-transitive relation and the same as an equivalence relation on a subtype of $\overline{A}$. Thus, A corresponds in HOL to the quotient of the appropriate subtype of $\overline{A}$ by A^*.

DHOL terms are translated to their HOL analogues except that equality is translated to the respective PER: $\overline{s =_A t} := A^*\ \overline{s}\ \overline{t}$. In particular, the predicate $A^*\ \overline{t}\ \overline{t}$ captures whether t is a term of type A. For n-ary type symbols a, the translation generates an $n+2$-ary predicate a^* such that $a^*\ \overline{t_1}\ \ldots\ \overline{t_n}$ is the PER for $\mathsf{a}\ t_1\ \ldots\ t_n$. For function types, the PER is the usual condition for logical relations: functions are related if they map related inputs to related outputs.

3 Subtyping

The treatment of quotients as supertypes and the use of different equality relations at different types are subtly difficult. Thus, we first introduce subtyping by its defining extensional property, from which we will derive all subtyping rules:

Definition 1 (Subtyping). $\Gamma \vdash_T A \prec: B$ *abbreviates* $\Gamma, x{:}A \vdash_T x{:}B$.

Lemma 1. *In any extension of DHOL,* $\Gamma \vdash_T A \prec: B$ *iff* $\dfrac{\Gamma \vdash_T t{:}A}{\Gamma \vdash_T t{:}B}$ *is derivable.*

Proof. Left-to-right: We construct the function $(\lambda x{:}A.\ x){:}A \to B$ and derive the desired rule using the typing rule for function application.
Right-to-left: We start with $\Gamma, x{:}A \vdash_T x{:}A$ and apply the derivable rule.

This subtyping relation prevents *incidental* subtype instances, for which the rule from Lem. 1 is admissible but not derivable. For example, $A|_{\lambda x{:}A.\ \mathsf{false}}$ is a subtype of all refinements of A, but not of all types. More generally, this definition precludes using induction on the terms of A to conclude $A \prec: B$. This restriction ensures that subtyping is preserved under, e.g., theory extensions, substitution, or language extensions. Importantly, subtyping preserves equality:

Lemma 2. *Consider some extension of DHOL with productions and rules. Assume (∗) that* $\Gamma \vdash_T x =_B x$ *implies* $\Gamma \vdash_T x : B$*. Let* $F := \forall x{:}A.\ \forall y{:}A.\ x =_A y \Rightarrow x =_B y$*. Then* $\Gamma \vdash_T A \prec: B$ *iff* $\Gamma \vdash_T F : \mathsf{bool}$*; and if these hold, then also* $\Gamma \vdash_T F$*.*

(∗) is a very mild assumption and satisfied by all extensions given in this paper.

Proof. Left-to-right: The assumption yields $\Gamma, x{:}A,\ y{:}A,\ \triangleright\, x =_A y \vdash_T (\lambda x{:}A.\ x){:}A \to B$. We get $\Gamma, x{:}A,\ y{:}A,\ \triangleright\, x =_A y \vdash_T (\lambda x{:}A.\ x)\ x =_B (\lambda x{:}A.\ x)\ y$ from congruence of function application and reflexivity and $x =_B y$ by β-reduction.
Right-to-left: Assume $x : A$. Instantiating F twice with x and applying modus ponens with $x =_A x$ yields $x =_B x$, from which we get $x : B$.

Lemma 3 (Preorder of Types). *In any extension of DHOL, subtyping is reflexive (in the sense that* $\Gamma \vdash_T A \equiv B$ *implies* $\Gamma \vdash_T A \prec: B$*) and transitive.*

Proof. Reflexivity: The assumption yields $\Gamma \vdash_T (\lambda x{:}A.\ x){:}A \to B$. Applying both to a term t of type A and β-reducing yields the rule from Lem. 1. Transitivity follows immediately from Lem. 1.

We also want to make subtyping an order. Anti-symmetry with respect to $\equiv$ is not derivable directly, i.e., we might have $\Gamma \vdash_T A \prec: B$ and $\Gamma \vdash_T B \prec: A$, in which case A and B would have the same terms, without being equal. Therefore, we *add* the anti-symmetry rule

$$\frac{\Gamma \vdash_T A \prec: B \qquad \Gamma \vdash_T B \prec: A}{\Gamma \vdash_T A \equiv B}\ \text{STantisym}$$

Notably, this is the only *change* made to DHOL so far — everything before has just been abbreviations. This change is conservative in the following sense:

Theorem 1 (Conservativity). *For DHOL as defined so far (without the extension we introduce below), we have* $A \prec: B$ *iff* $A \equiv B$*.*

Proof. We show by induction on derivations that each term has a unique type up to type equality and that all term equality axioms preserve typing.

Theorem 2 (Variance and Congruence for Function Types). *The usual rules for function types are derivable:*

$$\frac{\Gamma \vdash_T A' \prec: A \qquad \Gamma,\ x{:}A' \vdash_T B \prec: B'}{\Gamma \vdash_T \prod x{:}A.\ B \prec: \prod x{:}A'.\ B'} \qquad \frac{\Gamma \vdash_T A' \equiv A \qquad \Gamma,\ x{:}A' \vdash_T B \equiv B'}{\Gamma \vdash_T \prod x{:}A.\ B \equiv \prod x{:}A'.\ B'}$$

The second rule is primitive in DHOL but derivable in DHOL with subtyping.

Proof. The first rule follows from the definition of subtyping and η-expansion. The second rule is derived by (STantisym), establishing the hypotheses using the variance rule and reflexivity of subtyping.

4 Refinement types

To add refinement types, we add only one production for types. We do not add productions for terms — refinement types only provide new typing properties for the existing terms. Then we add rules for, respectively, formation, introduction, elimination (two rules), and equality:

$$A ::= \quad A|_p \quad \text{type } A \text{ refined by predicate } p \text{ on } A$$

$$\frac{\Gamma \vdash_T p{:}A \to \mathsf{bool}}{\Gamma \vdash_T A|_p\ \mathsf{tp}} \qquad \frac{\Gamma \vdash_T t{:}A \qquad \Gamma \vdash_T p\ t}{\Gamma \vdash_T t{:}A|_p}$$

$$\frac{\Gamma \vdash_T t{:}A|_p}{\Gamma \vdash_T t{:}A} \qquad \frac{\Gamma \vdash_T t{:}A|_p}{\Gamma \vdash_T p\ t} \qquad \frac{\Gamma \vdash_T s =_A t \qquad \Gamma \vdash_T p\ s}{\Gamma \vdash_T s =_{A|_p} t}$$

Example 2 (Refining Lists by Length). We extend Ex. 1 by defining fixed-length lists as a refinement of lists. First, we axiomatize a predicate length on lists:

$$\begin{array}{l} \mathsf{length}\colon \mathsf{list} \to \mathsf{nat} \qquad \triangleright\ \mathsf{length}\ \mathsf{nil} =_{\mathsf{nat}} \mathsf{zero} \\ \triangleright\ \forall x{:}\mathtt{obj}.\ \forall l{:}\mathsf{list}.\ \mathsf{length}\ (\mathsf{cons}\ x\ l) =_{\mathsf{nat}} \mathsf{succ}\ (\mathsf{length}\ l) \end{array}$$

Then we define $\mathsf{llist}\ n\ :=\ \mathsf{list}|_{\lambda l:\mathsf{list}.\ \mathsf{length}\ l =_{\mathsf{nat}} n}$, and we can derive

$$\vdash \mathsf{nil}{:}\mathsf{llist}\ \mathsf{zero} \qquad n{:}\mathsf{nat} \vdash \mathsf{cons} : \Pi x{:}\mathtt{obj}.\ \Pi l{:}\mathsf{llist}\ n.\ \mathsf{llist}\ (\mathsf{succ}\ n)$$

Theorem 3 (Congruence and Variance). *The following rules are derivable if the involved types are well-formed:*

$$\frac{\Gamma \vdash_T A \prec: A' \qquad \Gamma,\ x{:}A,\ \triangleright p\ x \vdash_T p'\ x}{\Gamma \vdash_T A|_p \prec: A'|_{p'}} \qquad \frac{\Gamma \vdash_T A\,\mathsf{tp}}{\Gamma \vdash_T A \equiv A|_{\lambda x:A.\ \mathsf{true}}}$$

$$\frac{\Gamma \vdash_T A \equiv A' \qquad \Gamma \vdash_T p =_{A\to\mathsf{bool}} p'}{\Gamma \vdash_T A|_p \equiv A'|_{p'}} \qquad \frac{\Gamma \vdash_T A|_p\ \mathsf{tp}}{\Gamma \vdash_T A|_p \prec: A}$$

Proof. To derive the first rule, we assume the hypotheses and $x{:}A|_p$. The elimination rules yield $x{:}A$ and $p\ x$, then the hypotheses yield $x{:}A'$ resp. $p'\ x$, finally the introduction rule yields $x{:}A'|_{p'}$.
To derive the second rule, we apply (STantisym) and use the introduction/elimination rules to show the two subtype relationships.
These then imply the other rules.

5 Quotient types

To add quotient types we also extend the grammar with only one production for the type and rules for formation, introduction, elimination, and equality, where $\mathrm{EqRel}(r)$ abbreviates that r is an equivalence relation:

$$A ::= \quad {}^{A}\!/_{r} \quad \text{quotient of } A \text{ by equivalence relation } r$$

$$\frac{\Gamma \vdash_T A\ \mathsf{tp} \qquad \Gamma \vdash_T r{:}A \to A \to \mathsf{bool} \qquad \Gamma \vdash_T \mathrm{EqRel}(r)}{\Gamma \vdash_T {}^{A}\!/_{r}\ \mathsf{tp}} \qquad \frac{\Gamma \vdash_T t{:}A \qquad \Gamma \vdash_T {}^{A}\!/_{r}\ \mathsf{tp}}{\Gamma \vdash_T t{:}{}^{A}\!/_{r}}$$

$$\frac{\Gamma \vdash_T s{:}{}^{A}\!/_{r} \qquad \Gamma,\, x{:}A,\, \triangleright\, x =_{{}^{A}/_{r}} s \vdash_T t{:}B \qquad \Gamma,\, x{:}A,\, x'{:}A,\, \triangleright\, x =_{{}^{A}/_{r}} s,\, \triangleright\, x' =_{{}^{A}/_{r}} s \vdash_T t =_B t[{}^{x}\!/_{x'}]}{\Gamma \vdash_T t[{}^{x}\!/_{s}]{:}B[{}^{x}\!/_{s}]}$$

$$\frac{\Gamma \vdash_T s{:}A \qquad \Gamma \vdash_T t{:}A \qquad \Gamma \vdash_T r{:}A \to A \to \mathsf{bool} \qquad \Gamma \vdash_T \mathrm{EqRel}(r)}{\Gamma \vdash_T (s =_{{}^{A}/_{r}} t) =_{\mathsf{bool}} (r\ s\ t)}$$

Example 3 (Sets). We extend Ex. 1 by obtaining sets as a quotient of lists. First, we axiomatize a predicate for containing an element:

$\mathrm{contains}\colon \mathrm{list} \to \mathtt{obj} \to \mathsf{bool} \qquad \triangleright\ \forall x{:}\mathtt{obj}.\ \neg(\mathrm{contains}\ \mathrm{nil}\ x)$
$\triangleright\ \forall x{:}\mathtt{obj}.\ \forall y{:}\mathtt{obj}.\ \forall l{:}\mathrm{list}.\ (\mathrm{contains}\ (\mathrm{cons}\ y\ l)\ x) =_{\mathsf{bool}} (x =_{\mathtt{obj}} y \vee \mathrm{contains}\ l\ x)$

Now we can define $\mathrm{set} := {}^{\mathrm{list}}\!/_{\lambda l{:}\mathrm{list}.\ \lambda m{:}\mathrm{list}.\ \forall x{:}\mathtt{obj}.\ \mathrm{contains}\ l\ x =_{\mathsf{bool}} \mathrm{contains}\ m\ x}$ as the type of lists containing the same elements. The equality at set immediately yields extensionality $\vdash \forall x, y{:}\mathrm{set}.\ \ x =_{\mathrm{set}} y \ \Leftrightarrow\ (\forall z{:}\mathtt{obj}.\ \mathrm{contains}\ x\ z =_{\mathsf{bool}} \mathrm{contains}\ y\ z)$.

Any $l{:}\mathrm{list}$ can be used as a representative of the respective equivalence class in set, and operations on sets can be defined via operations on lists, e.g., we can establish $\vdash \mathrm{conc} : \mathrm{set} \to \mathrm{set} \to \mathrm{set}$. To derive this, we assume $u{:}\mathrm{set}$ and apply the elimination rule twice. First we apply it with $B = \mathrm{list} \to \mathrm{set}$ and $t = \mathrm{conc}\ u$; we have to show $\mathrm{conc}\ x =_{\mathrm{list}\to\mathrm{set}} \mathrm{conc}\ x'$ under the assumption that x and x' are equal as sets. That yields a term $\mathrm{conc}\ u : \mathrm{list} \to \mathrm{set}$. We assume $v{:}\mathrm{set}$ and apply the elimination rule again with $B = \mathrm{set}$ to obtain $\mathrm{conc}\ u\ v{:}\mathrm{set}$, and then conclude via λ-abstraction and η-reduction.

The elimination rule above looks overly complex. It can be understood best by comparing it to the following, simpler and more intuitive rule

$$\frac{\Gamma,\, x{:}A \vdash_T t{:}B \qquad \Gamma,\, x{:}A,\, x'{:}A,\, \triangleright\, r\ x\ x' \vdash_T t =_B t[{}^{x}\!/_{x'}]}{\Gamma,\, x{:}{}^{A}\!/_{r} \vdash_T t{:}B}\,(*)$$

This rule captures the well-known condition that a function t on A may be used as a function on ${}^{A}\!/_{r}$ if t maps equivalent representatives x, x' equally. It follows from our elimination rule by putting $s = x$, but is subtly weaker:

Example 4. Continuing Ex. 3, assume a total order on $\texttt{obj}$ and a function $g{:}\mathrm{list}|_{\mathrm{nonEmpty}} \to \texttt{obj}$ picking the maximum from a non-empty list. We should be able to apply g to some $s{:}\mathrm{set}$ that we know to be non-empty. But if we try to apply $(*)$ to obtain $g\ s{:}\,\texttt{obj}$, we get stuck trying to prove $g\ x =_{\texttt{obj}} g\ x'$ for any x, x' that are representatives of an *arbitrary* equivalence class of lists. We cannot use the condition that s is non-empty and thus only non-empty lists need to be considered. Thus, we cannot derive the well-formedness of $g\ x$.

Our elimination rule remedies that: here we need to show $g\ x =_{\texttt{obj}} g\ x'$ for any x, x' that are representatives of *the class of* s. Thus, we can use that x and x' are non-empty and that thus $g\ x$ is well-formed.

In dependent type theory, the two elimination rules are equivalent because we have a type $s = x$ and can use $\Pi s{:}A/r.\ s = x \to B$ as the return type. This is not possible in DHOL where $s = x$ is not a type but a Boolean.

Theorem 4 (Congruence and Variance). *The following rules are derivable if the involved types are well-formed and r, r' are equivalence relations:*

$$\frac{\Gamma \vdash_T A \prec: A' \qquad \Gamma,\ x{:}A,\ y{:}A,\ \triangleright r\ x\ y \vdash_T r'\ x\ y}{\Gamma \vdash_T A/r \prec: A'/r'} \qquad \frac{\Gamma \vdash_T A\ \mathsf{tp}}{\Gamma \vdash_T A \equiv A/\lambda x{:}A.\ \lambda y{:}A.\ x =_A y}$$

$$\frac{\Gamma \vdash_T A/r\ \mathsf{tp}}{\Gamma \vdash_T A \prec: A/r} \qquad \frac{\Gamma \vdash_T A \equiv A' \qquad \Gamma \vdash_T r =_{A\to A\to \mathsf{bool}} r'}{\Gamma \vdash_T A/r \equiv A'/r'}$$

Proof. For the first rule: Assume the hypotheses and $s{:}A/r$. Apply the elimination rule with $B = A'/r'$ and $t = x$. $x{:}A,\ y{:}A,\ \triangleright x =_{A/r} s,\ \triangleright x' =_{A/r} s \vdash_T x =_{A'/r'} x'$ by the equality rule (using $A \prec: A'$ and the second assumption). For the second rule, apply (STantisym) and use the introduction/elimination rules to show the two subtype relationships. The other rules follow from those two.

6 Normalizing Types

To build a type-checker, we derive normalization rules that reduce subtyping conditions to validity conditions that can then be discharged via an ATP. We prove rules for merging consecutive refinements and quotients and for the 4 possible combinations of a function type with a refinement or quotient:

Theorem 5 (Repeated Refinement/Quotient). *The following are derivable whenever the LHS is well-formed*

$$\vdash_T (A|_p)|_{p'} \equiv A|_{\lambda x{:}A.\ p\ x \wedge p'\ x} \tag{RR}$$
$$\vdash_T (A/r)/r' \equiv A/\lambda x{:}A.\ \lambda y{:}A.\ r'\ x\ y \tag{QQ}$$
$$\vdash_T (A/r)|_p \equiv (A|_p)/r \tag{RQ}$$

Proof. (RR): well-formedness of the LHS yields $p{:}A \to \mathsf{bool}$ and $p'{:}A|_p \to \mathsf{bool}$, so $p'\ x$ is well-formed as $\wedge$ is a *dependent* conjunction and $p\ x$ can be assumed while checking $p'\ x$. The well-formedness of the right-hand side (RHS) follows. Verifying the equality is straightforward by showing subtyping in both directions.
(QQ): well-formedness of the LHS yields $r{:}A \to A \to \mathsf{bool}$ and $r'{:}A/r \to A/r \to \mathsf{bool}$, so $A \prec\!: A/r$ implies $r'\ x\ y$ (and thus the RHS) is well-formed. The relation on the RHS is an equivalence relation since r' is. To verify the type equality, we use Lem. 2 and show that both types induce the same equality. In particular, the type of r' already guarantees that it subsumes r.
(RQ): well-formedness of the LHS yields $r{:}A \to A \to \mathsf{bool}$ and $p{:}A/r \to \mathsf{bool}$, implying $r{:}A|_p \to A|_p \to \mathsf{bool}$ and $p{:}A \to \mathsf{bool}$. The well-formedness of the RHS follows. (Note the other direction does not hold in general.) To show the equality, we show both subtyping directions. For LHS$\prec$:RHS, we assume $x{:}A/r$ and $p\ x$ and apply the elimination rule for quotients using $t = x$ and $B = (A|_p)/r$. (Critically, this step would not go through if we had only used the weaker rule $*$ in Sect. 5.) For RHS$\prec$:LHS, we assume $x{:}(A|_p)/r$ and apply the elimination rule for quotients using $t = x$.

Theorem 6 (Refinement/Quotient in a Function Type). *The following are derivable if either side is well-formed:*

$$\vdash_T \Pi x{:}A.\ (B|_p) \equiv (\Pi x{:}A.\ B)|_{\lambda f:(\Pi x:A.\ B).\ \forall x:A.\ p\ (f\ x)} \quad \text{(RCod)}$$

$$\vdash_T \Pi x{:}A/r.\ B \equiv (\Pi x{:}A.\ B)|_{\lambda f:\Pi x:A.\ B.\ \forall x,y:A.\ r\ x\ y \Rightarrow (f\ x) =_B (f\ y)} \quad \text{(QDom)}$$

$$\vdash_T \Pi x{:}A.\ B/r :\!\succ (\Pi x{:}A.\ B)/\lambda f, g{:}\Pi x{:}A.\ B.\ \forall x{:}A.\ r\ \ (f\ x)\ (g\ x) \quad \text{(QCod)}$$

The following is derivable if the RHS is well-formed:

$$\vdash_T \Pi x{:}A|_p.B :\!\succ (\Pi x{:}A.B)/\lambda f, g{:}\Pi x{:}A.B.\ \forall x{:}A.\ p\ x \Rightarrow (f\ x) =_B (g\ x) \quad \text{(RDom)}$$

Proof. (RCod): Both subtyping directions are straightforward, as terms on either side are given by $\lambda x{:}A.\ t$ where t has type B and satisfies p.
(QDom): Both subtyping directions are straightforward, as both sides are subtypes of $\Pi x{:}A.\ B$ so their elements must preserve r.
(QCod): Assume a term f of RHS-type and show $x{:}A \vdash f\ x{:}B/r$ using the rules for quotients.
(RDom): Assume a term f of RHS-type and show $x{:}A|_p \vdash f\ x{:}B$ using the quotient elimination rule. The well-formedness of the LHS does not imply the well-formedness of the RHS since the well-formedness of B can rely on $p\ x$.

Maybe surprisingly, two of the subtyping laws in Thm. 6 are not equalities. The law for the refined domain *must not* be an equality:

Example 5 (Refined Domain (RDom)). The assumption $p\ x$ makes more terms well-typed, thus there may be functions $\Pi x{:}A|_p.\ B$ that are not a restriction of a function $\Pi x{:}A.\ B$. Consider the theory $\mathsf{a}{:}\mathsf{bool} \to \mathsf{tp}$, $\mathsf{c}{:}\mathsf{a}\,\mathsf{true}$. Then $\mathsf{a}\,\mathsf{false}$

is empty and so are $\Pi x{:}\mathsf{bool}.\ \mathtt{a}\,x$ and its quotients. But with $p = \lambda x{:}\mathsf{bool}.\ x$, we have $\vdash \lambda x{:}\mathsf{bool}|_p.\ \mathtt{c} : \Pi x{:}\mathsf{bool}|_p.\ \mathtt{a}\ x$.

The law for the quotiented codomain *may or may not* be an equality. This is related to the axiom of choice. Consider the two statements

$$\vdash_T \exists\, repr{:}B/r \to B.\ repr =_{B/r \to B/r} \lambda x{:}B/r.\ x$$

$$f{:}\Pi x{:}A.\ B/r \vdash_T \exists\, g{:}\Pi x{:}A.\ B.\ f =_{\Pi x:A.\ B/r} g$$

(Note that the first one is well-typed because $B \prec: B/r$.) Both have a claim to be called the axiom of choice: The first one expresses that every equivalence relation has a system of representatives. The second generalizes this to a family of equivalence relations. The latter implies the former (put $A := B/r$ and $f := \lambda x{:}B/r.\ x$). In the simply-typed case the former also implies the latter (pick $repr \circ f$ for g); but in the dependently-typed case, B and r may depend on x and the implication might not hold.

Both statements construct a new term from an existing one ($repr$ behaves like the identity, and g like f) that has a different type but behaves the same up to quotienting. If the direction $\prec:$ were to hold in the law for the refined codomain, it would not only imply the existence of g from f but also allow using f as a representative of the equivalence class of possible values for g. That is in keeping with our goal of avoiding changes of representation. Therefore:

Definition 2 (Quotiented Codomain). *We adopt the rule below (which is an equality with Thm. 6) as an axiom whenever either side is well-formed:*

$$\vdash_T \Pi x{:}A.\ B/r \prec: (\Pi x{:}A.\ B)/\lambda f, g{:}\Pi x{:}A.\ B.\ \forall\, x{:}A.\ r\ \ (f\ x)\,(g\ x) \qquad (**)$$

Aggregating the above laws, we obtain a normalization algorithm for types:

Theorem 7 (Normalizing Types). *Every type is equal to a type of the form $(A|_p)/r$ where $A, B ::= \mathsf{bool} \mid \mathtt{a}\ t^* \mid \Pi x{:}A|_p.\ B$. In particular, if a type does not use refined domains, it is equal to a quotient of a refinement of a DHOL type.*

Proof. Using Thm. 6 with the axiom from Def. 2, all refinements and quotients can be pushed out of all function types except for a single refinement of the domain; if there is no such refinement, we can use $p := \lambda x{:}A.\ \mathsf{true}$. And using Thm. 5, those can be collected into a single quotient+refinement.

Together with the equality and variance rules from Thm. 2, 3, 4, this induces a **subtype-checking algorithm** that reduces subtyping to validity without ever expanding Def. 1. The latter is important because expanding Def. 1 would recurse into a computationally expensive problem. This algorithm is obviously sound as it only chains derived rules. We are confident, but have not proved yet, that it is also complete in the sense that subtyping can always be derived

by using only our derived rules (i.e., without Def. 1) and a sound and complete theorem prover (which we obtain in Sect. 7).

It may be surprising and certainly complicates subtype-checking that we need to allow for refined domains in the normal forms. This limitation echoes an observation first made in [14] about combining dependent types and refinements. Effectively, the culprits are partial dependent functions that cannot be extended to total functions because the return type is well-defined and non-empty only for the refined domain. For future work, it would be interesting to use a higher-order logic with partial functions as a translation target, like the one of [10]. But we have not considered that option due to the lack of ATP support for such logics.

7 Soundness and Completeness

We extend the translation from Fig. 3 with cases for our two new productions:

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

These definitions are not surprising as PERs in HOL are known to be closed under refinements and quotients.

Example 6 (PERs for a Quotiented Codomain). We calculate the PERs for both sides of the law for quotiented codomains in Thm. 6:

$$\begin{aligned}&(\Pi x{:}A.\ B/r)^*\ f\ g\\ =\ &\forall x, y{:}\overline{A}.\ A^*\ x\ y \Rightarrow (\overline{r}\ (f\ x)\ (g\ y) \wedge B^*\ (f\ x)\ (f\ x) \wedge B^*\ (g\ y)\ (g\ y))\end{aligned}$$

Both this and $((\Pi x{:}A.\ B)/\lambda f, g{:}\Pi x{:}A.\ B.\ \forall x{:}A.\ r\ (f\ x)\ (g\ x))^*\ f\ g$ simplify to $\forall x{:}\overline{A}.\ A^*\ x\ x \Rightarrow \overline{r}\ (f\ x)\ (g\ x) \wedge B^*\ (f\ x)\ (f\ x) \wedge B^*\ (g\ x)\ (g\ x)$. This justifies adopting axiom (**). The simplification uses the substitution lemma from the extended preprint[5], the well-definedness of r, and the transitivity of $\overline{r}$.

Example 7 (PERs for a Refined Domain). We calculate the PERs for both sides of the law for refined domains in Thm. 6:

$$(\Pi x{:}A|_p.\ B)^*\ f\ g = \forall x, y{:}\overline{A}.\ A^*\ x\ y \wedge \overline{p}\ x \wedge \overline{p}\ y \Rightarrow B^*\ (f\ x)\ (g\ y)$$

$$\begin{aligned}&((\Pi x{:}A.\ B)/\lambda f, g{:}\Pi x{:}A.\ B.\ \forall x{:}A.\ p\ x \Rightarrow (f\ x) =_B (g\ x))^*\ f\ g =\\ &\forall x{:}\overline{A}.\ (A^*\ x\ x \Rightarrow \overline{p}\ x \Rightarrow B^*\ (f\ x)\ (g\ x)) \wedge\\ &\big(\forall x, y{:}\overline{A}.\ A^*\ x\ y \Rightarrow B^*\ (f\ x)\ (f\ y)\big) \wedge\\ &\big(\forall x, y{:}\overline{A}.\ A^*\ x\ y \Rightarrow B^*\ (g\ x)\ (g\ y)\big)\end{aligned}$$

These are indeed not equivalent in line with our observation from Ex. 5.

Like in [22], this translation yields a sound (defined as in the previous paper) and complete theorem prover:

Theorem 8 (Completeness). *We have the invariants from Fig. 4.*

if in DHOL	then in HOL		
$\vdash T$ Thy	$\vdash \overline{T}$ Thy		
$\vdash_T \Gamma$ Ctx	$\vdash_{\overline{T}} \overline{\Gamma}$ Ctx		
$\Gamma \vdash_T A$ tp	$\overline{\Gamma} \vdash_{\overline{T}} \overline{A}$ tp	and	$\overline{\Gamma} \vdash_{\overline{T}} A^* : \overline{A} \to \overline{A} \to$ bool and A^* is a PER
$\Gamma \vdash_T A \equiv B$	$\overline{\Gamma} \vdash_{\overline{T}} \overline{A} \equiv \overline{B}$	and	$\overline{\Gamma}, x, y{:}\overline{A} \vdash_{\overline{T}} A^* \ x \ y =_{\mathsf{bool}} B^* \ x \ y$
$\Gamma \vdash_T A \prec: B$	$\overline{\Gamma} \vdash_{\overline{T}} \overline{A} \equiv \overline{B}$	and	$\overline{\Gamma}, x, y{:}\overline{B} \vdash_{\overline{T}} A^* \ x \ y \Rightarrow B^* \ x \ y$
$\Gamma \vdash_T t{:}A$	$\overline{\Gamma} \vdash_{\overline{T}} \overline{t}{:}\overline{A}$	and	$\overline{\Gamma} \vdash_{\overline{T}} A^* \ \overline{t} \ \overline{t}$
$\Gamma \vdash_T F$	$\overline{\Gamma} \vdash_{\overline{T}} \overline{F}$		

Fig. 4. Invariants of the Translation

Proof. The subtyping claim is a slightly strengthened version of the claim obtained by expanding the definition of $\prec:$. The proof, given in the extended preprint[5], adapts the proof from [22] with additional cases for new productions and rules.

As in [22], the converse of Thm. 8 is much harder to state and prove. First we need a technical assumption: We call a type symbol `a` *inhabited* if at least one of its instances is provably non-empty.

Theorem 9 (Soundness). *In a well-formed DHOL-theory* $\vdash T$ Thy *in which every type symbol is inhabited:*

$$\textit{If } \Gamma \vdash^{DHOL}_{T} F : \mathsf{bool} \textit{ and } \overline{\Gamma} \vdash^{HOL}_{\overline{T}} \overline{F}, \textit{ then } \Gamma \vdash^{DHOL}_{T} F$$

Proof. The key idea is to transform a HOL-proof of $\overline{F}$ into one that is in the image of the translation, at which point we can read off a DHOL-proof of F. The full proof is given in the extended preprint[5]. We expect the inhabitation requirement to be redundant, but have not been able to complete the proof without it yet. In any case, it is harmless because it is satisfied by all practical examples.

It is now straightforward to extend the DHOL implementation we gave in [22]: First run a bidirectional type-checker for DHOL, using the subtyping-checker sketched in Sect. 6, to establish well-typedness of theory and conjecture. Then translate conjecture and generated proof obligations and apply a HOL ATP.

8 Application to Typed Set Theory

DHOL with subtyping enables a novel formalization of typed set-theory:

$$\mathrm{set}{:}\,\mathsf{tp}, \qquad \in : \mathrm{set} \to \mathrm{set} \to \mathsf{bool}, \qquad \mathit{elem}\ s := \mathrm{set}|_{\lambda x:\mathrm{set}.\ x \in s}$$

The key idea is that $\mathit{elem}\,s$ is the DHOL *type* of set-theoretical elements of the *set* s. Leveraging that refinements and quotients do not require change of representation, we obtain a powerful combination of elegant high-level typed formalization and efficient low-level reasoning. All the routine constructions of untyped set theory can be lifted to their typed counterparts. For example, for products, we use $\times$:set $\to$ set $\to$ set and pair:set $\to$ set $\to$ set and the property $\triangleright\ \forall x,y,s,t\text{:set}.\ (x \in s) \wedge (y \in t) \Rightarrow \mathit{pair}\ x\ y \in s \times t$, from which we can show that $\mathit{pair} : \mathit{elem}\ s \to \mathit{elem}\ t \to \mathit{elem}\ (s \times t)$.

We can also use DHOL-functions set $\to$ set as set-theoretical functions between sets s and t without a change in representation as the type $\mathit{Functions}\,s\,t := (\text{set} \to \text{set})|_p\,/r$ where $p\ f = \forall x\text{:set}.\ x \in s \Rightarrow (f\ x) \in t$ and $r\ f\ g = \forall x\text{:set}.\ x \in s \Rightarrow (f\ x) =_{\text{set}} (g\ x)$. This allows us to represent set-theoretical function application and composition $\circ$ directly as DHOL application/composition.

Consequently, theorem proving in typed set theory becomes very strong because a large share of the proving workload can be outsourced into typing-obligations. For example, the property that the composition of functions $f : \mathit{Functions}\ s\ t$ and $g : \mathit{Functions}\ t\ u$ has type $\mathit{Functions}\ s\ u$ becomes

$$\triangleright \forall s,t,u\text{:set}.\ \forall f\text{:}\mathit{Functions}\ s\ t.\ \forall g\text{:}\mathit{Functions}\ t\ u.\ \forall x\text{:set}.\ x \in s \Rightarrow ((g \circ f)\ x) \in u$$

which yields in HOL the conjecture below that current HOL ATPs solve easily.

$$\begin{aligned}\triangleright\ &\forall s\text{:set}.\ \mathit{set\_rel}\ s\ s \Rightarrow \forall t\text{:set}.\ \mathit{set\_rel}\ t\ t \Rightarrow \forall u\text{:set}.\ \mathit{set\_rel}\ u\ u \Rightarrow\\ &\forall f\text{:set} \to \text{set}.\ \forall x\text{:set}.\ x \in s \Rightarrow \mathit{set\_rel}(f\ x)(f\ x) \wedge (\forall x\text{:set}.\ x \in s \Rightarrow (f\ x) \in t) \Rightarrow\\ &\forall g\text{:set} \to \text{set}.\ \forall x\text{:set}.\ x \in t \Rightarrow \mathit{set\_rel}(g\ x)(g\ x) \wedge (\forall x\text{:set}.\ x \in t \Rightarrow (g\ x) \in u) \Rightarrow\\ &\forall x\text{:set}.\ \mathit{set\_rel}\ u \Rightarrow x \in s \Rightarrow ((g\ (f\ x)) \in u)\end{aligned}$$

Below is the HOL translation of the conjecture that function composition is associative. It is similarly easily proved by current ATPs:

$$\begin{aligned}\triangleright\ &\forall s\text{:set}.\ \mathit{set\_rel}\ s\ s \Rightarrow \forall t\text{:set}.\ \mathit{set\_rel}\ t\ t \Rightarrow \forall u\text{:set}.\ \mathit{set\_rel}\ u\ u \Rightarrow \forall v\text{:set}.\ \mathit{set\_rel}\ v\ v \Rightarrow\\ &\forall f\text{:set} \to \text{set}.\ \forall x\text{:set}.\ x \in s \Rightarrow \mathit{set\_rel}(f\ x)(f\ x) \wedge (\forall x\text{:set}.\ x \in s \Rightarrow (f\ x) \in t) \Rightarrow\\ &\forall g\text{:set} \to \text{set}.\ \forall x\text{:set}.\ x \in t \Rightarrow \mathit{set\_rel}(g\ x)(g\ x) \wedge (\forall x\text{:set}.\ x \in t \Rightarrow (g\ x) \in u) \Rightarrow\\ &\forall h\text{:set} \to \text{set}.\ \forall x\text{:set}.\ x \in u \Rightarrow \mathit{set\_rel}(h\ x)(h\ x) \wedge (\forall x\text{:set}.\ x \in t \Rightarrow (h\ x) \in v) \Rightarrow\\ &\forall x\text{:set}.\ \mathit{set\_rel}\ x\ x \Rightarrow x \in s \Rightarrow (\mathit{set\_rel}\ (h\ (g\ (f\ x)))\ (h\ (g\ (f\ x))))\end{aligned}$$

The corresponding TPTP files are available at `https://gl.mathhub.info/MMT/LATIN2/-/tree/master/source/casestudies/2025-FroCos`.

9 Conclusion and Future Work

DHOL combines higher-order logic with dependent types, obtaining an intuitive and expressive language, albeit with undecidable typing. We double down on this design by elegantly extending DHOL with two practically important type

constructors that thrive in that setting: refinement and quotient types. Like dependent function types, these two require terms occurring in types. Both are near-impossible to add as an afterthought to a type theory with decidable typing.

We translate the resulting logic to HOL, obtaining a practical automated theorem proving workflow for DHOL with refinement and quotient types. Our main result is the proof of soundness and completeness of this translation.

We used an extensional subtyping approach, where $A \prec: B$ holds iff all A-terms also have type B. This allows combining typed representations and efficient reasoning. We established all the expected variance and normalization laws except for function types with refined domains. Future work must investigate how to improve on this to make normalizing types and thus subtype-checking simpler.

We also want to carry our results for DHOL over to existing refinement/quotient type systems for programming languages like Quotient Haskell, where DHOL-like axioms are used as lightweight specifications.

References

1. Allen, S.: A Non-type-theoretic Semantics for Type-theoretic Language. Ph.D. thesis, Cornell University (1987)
2. Andrews, P.: An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof. Academic Press (1986)
3. B. Hewer and G. Hutton: Quotient Haskell: Lightweight Quotient Types for All. Proc. ACM Program. Lang. **8**(POPL) (2024). https://doi.org/10.1145/3632869, https://doi.org/10.1145/3632869
4. Bancerek, G., Byliński, C., Grabowski, A., Korniłowicz, A., R. Matuszewski, A.N., Pak, K., Urban, J.: Mizar: State-of-the-art and beyond. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) Intelligent Computer Mathematics. p. 261–279. Springer International Publishing, Cham (2015)
5. C. Rothgang, F. Rabe: Subtyping in DHOL – Extended preprint (2025), https://arxiv.org/abs/2507.02855
6. Church, A.: A Formulation of the Simple Theory of Types. Journal of Symbolic Logic **5**(1), 56–68 (1940)
7. Constable, R., Allen, S., Bromley, H., Cleaveland, W., Cremer, J., Harper, R., Howe, D., Knoblock, T., Mendler, N., Panangaden, P., Sasaki, J., Smith, S.: Implementing Mathematics with the Nuprl Development System. Prentice-Hall (1986)
8. Coq Development Team: The Coq Proof Assistant: Reference Manual. Tech. rep., INRIA (2015)
9. de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The Lean Theorem Prover (System Description). In: Felty, A., Middeldorp, A. (eds.) Automated Deduction. p. 378–388. Springer International Publishing, Cham (2015)
10. Farmer, W.: A simple type theory with partial functions and subtypes. Annals of Pure and Applied Logic **64**(3), 211–240 (1993)
11. Gordon, M.: HOL: A Proof Generating System for Higher-Order Logic. In: Birtwistle, G., Subrahmanyam, P. (eds.) VLSI Specification, Verification and Synthesis, p. 73–128. Kluwer-Academic Publishers (1988)
12. Homeier, P.V.: A design structure for higher order quotients. In: Hurd, J., Melham, T. (eds.) Theorem Proving in Higher Order Logics. pp. 130–146. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

13. Huffman, B., Kunčar, O.: Lifting and transfer: A modular design for quotients in isabelle/hol. In: Gonthier, G., Norrish, M. (eds.) Certified Programs and Proofs. pp. 131–146. Springer International Publishing, Cham (2013)
14. Hurd, J.: Predicate subtyping with predicate sets. In: Boulton, R., Jackson, P. (eds.) Theorem Proving in Higher Order Logics. pp. 265–280 (2001)
15. Kuncar, O., Popescu, A.: Comprehending isabelle/hol's consistency. In: Yang, H. (ed.) Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10201, pp. 724–749. Springer (2017). https://doi.org/10.1007/978-3-662-54434-1_27, https://doi.org/10.1007/978-3-662-54434-1_27
16. Martin-Löf, P.: An Intuitionistic Theory of Types: Predicative Part. In: Proceedings of the '73 Logic Colloquium, pp. 73–118. North-Holland (1974)
17. Niederhauser, J., Brown, C., Kaliszyk, C.: Tableaux for automated reasoning in dependently-typed higher-order logic. In: Benzmüller, C., Heule, M., Schmidt, R. (eds.) Automated Reasoning. Springer (2024)
18. Owre, S., Rushby, J., Shankar, N.: PVS: A Prototype Verification System. In: Kapur, D. (ed.) 11th International Conference on Automated Deduction (CADE). pp. 748–752. Springer (1992)
19. Popescu, A., Traytel, D.: Admissible types-to-pers relativization in higher-order logic. Proc. ACM Program. Lang. **7**(POPL) (Jan 2023). https://doi.org/10.1145/3571235, https://doi.org/10.1145/3571235
20. Ranalter, D., Brown, C., Kaliszyk, C.: Experiments with choice in dependently-typed higher-order logic. In: Bjørner, N., Heule, M., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence and Reasoning. pp. 311–320 (2024)
21. Ranalter, D., Rabe, F., Kaliszyk, C.: Polymorphic theorem proving for DHOL
22. Rothgang, C., Rabe, F., Benzmüller, C.: Theorem Proving in Dependently Typed Higher-Order Logic. In: Pientka, B., Tinelli, C. (eds.) Automated Deduction. pp. 438–455. Springer (2023)