Admissible Types-to-PERs Relativization in Higher-Order Logic

ANDREI POPESCU, University of Sheffield, United Kingdom
DMITRIY TRAYTEL, University of Copenhagen, Denmark

Relativizing statements in Higher-Order Logic (HOL) from types to sets is useful for improving productivity when working with HOL-based interactive theorem provers such as HOL4, HOL Light and Isabelle/HOL. This paper provides the first comprehensive definition and study of types-to-sets relativization in HOL, done in the more general form of types-to-PERs (partial equivalence relations). We prove that, for a large practical fragment of HOL which includes container types such as datatypes and codatatypes, types-to-PERs relativization is *admissible*, in that the provability of the original, type-based statement implies the provability of its relativized, PER-based counterpart. Our results also imply the admissibility of a previously proposed axiomatic extension of HOL with local type definitions. We have implemented types-to-PERs relativization as an Isabelle tool that performs relativization of HOL theorems on demand.

CCS Concepts: • **Theory of computation** → **Logic and verification**; **Higher order logic**; **Type structures**.

Additional Key Words and Phrases: higher-order logic (HOL), proof theory, interactive theorem proving, type definition, relativization, Isabelle/HOL, partial equivalence relation

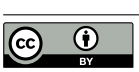
1 INTRODUCTION

Interactive theorem provers, also known as proof assistants, are being increasingly deployed to verify meta-theoretic properties of operating systems, programming languages, compilers and hardware architectures, and to formalize mathematical theories. Major verification successes that make the news, such as CompCert [Leroy 2009] and seL4 [Klein et al. 2010] in computer science and the Four Color [Gonthier 2007] and Kepler's theorems [Hales et al. 2015] in mathematics, are only the tip of the iceberg. The iceberg itself consists of a large body of libraries of formalized results developed by countless enthusiasts within the communities built around the theorem provers. The goal of boosting the users' productivity is therefore key to such successes. This is a multi-faceted goal, whose achievement involves an understanding of how a prover's logical foundation and the abstraction layers built on top of that can be leveraged towards increased automation and expressiveness, so overall decreased "bureaucracy"—without compromising reliability.

We can distinguish two kinds of logical foundations that dominate today's interactive theorem prover landscape. On the one hand, provers based on dependent type theory (DTT) such as Agda [Bove et al. 2009], Coq [Bertot and Castéran 2004], Lean [de Moura et al. 2015] and Matita [Asperti et al. 2011] rely on sophisticated type systems, featuring inductive and sometimes also coinductive datatypes, to manage both the data of interest and the proofs themselves. Developments in these provers often emphasize constructiveness and the computational content of the proofs.

On the other hand, provers based on higher-order logic (HOL) such as HOL4 [Gordon and Melham 1993; Slind and Norrish 2008], HOL Light [Harrison 1996], Isabelle/HOL [Nipkow and Klein 2014; Nipkow et al. 2002], ProofPower-HOL [Arthan and Jones 2005] and HOL Zero [Adams 2010], rely on simple type theory enhanced with rank-1 polymorphism, and a minimalistic set of logical axioms. Unlike in DTT, in HOL there are no built-in datatypes. Instead, new types are introduced from the underlying simple types by carving out subsets determined by predicates—a mechanism similar

Authors' addresses: Andrei Popescu, Department of Computer Science, University of Sheffield, United Kingdom, a.popescu@sheffield.ac.uk; Dmitriy Traytel, Department of Computer Science, University of Copenhagen, Denmark, traytel@di.ku.dk.



to comprehension/separation in set theory. For this reason, HOL is sometimes referred to as "typed set theory". Another difference from DTT is that HOL is a classic rather than intuitionistic logic and relies essentially on the Hilbert choice operator and the corresponding choice axiom.[1]

Both kinds of logical foundations have comparative pros and cons, some of which are discussed by Geuvers [2009]. The highly structured foundation of DTT systems, which natively supports the definition of inductive datatypes, facilitates the study of important meta-theoretic properties such as strong normalization and parametricity [Bernardy et al. 2012; Reynolds 1983], the latter enabling so-called "theorems for free" [Wadler 1989]. By contrast, in HOL everything (including all inductive datatypes) must be bootstrapped from its set-theoretic-like foundation: Thanks to the powerful yet unwieldy combination of HOL's comprehension-based type definitions and Hilbert choice, one has the freedom to perform both structured and highly unstructured constructions. This paper will be concerned with identifying some "structure" in HOL, in order to practically enable and foundationally justify a certain kind of theorems for free that help streamline the proof development.

Background: Set-Based versus Type-Based Developments in HOL. In a recent line of work, types-to-sets relativization was proposed as a way of increasing user productivity in HOL. The starting point was the observation that HOL developments that involve structures of various kinds (algebraic, topological etc.) highly benefit from a lightweight approach: making the assumption that the carriers of these structures are unspecified types. For example, a group can be modeled as a triple $(\alpha, \mathit{times}, e)$, consisting of a multiplication operation $\mathit{times} : \alpha \Rightarrow \alpha \Rightarrow \alpha$ and a neutral element $e : \alpha$ on an unspecified carrier modeled as a type variable α, subject to assumptions reflecting the group axioms. Likewise, a topological space can be modeled as a pair (α, T) where the type variable α represents the set of points and $T : \alpha\ \mathit{set}\ \mathit{set}$ represents the set of open sets, again subject to suitable assumptions.[2] Then the statements one proves about groups or topological spaces are (implicitly) quantified universally over the type variable α. This approach is of course not restricted to the usage of a single type variable. For example, we use two type variables, say α and β, when discussing group homomorphisms, which are functions $f : \alpha \Rightarrow \beta$ subject to suitable assumptions. Formulating and proving statements this way allows one to take advantage of HOL's underlying simple type system.

However, when needing to instantiate the results to particular concrete groups or topological spaces, one faces the problem that these concrete structures are usually defined not on entire types, but on certain *subsets* of types. This problem appears not only at the very bottom of the concreteness chain, but also at intermediary layers. For example, the group of permutations on a given type α has as carrier not the whole type $\alpha \Rightarrow \alpha$ but a subset of it, namely the set $\{f : \alpha \Rightarrow \alpha \mid f \text{ bijective}\}$.

Therefore, many HOL formalizations are performed not type-based as above, but *set-based*. Namely, the carriers of the structures of interest are modeled not as unspecified types α, but as unspecified subsets of unspecified types, $A : \alpha\ \mathit{set}$. For example, groups are modeled as quadruples $(\alpha, A, \mathit{times}, e)$ where $A : \alpha\ \mathit{set}$. Now, the group axioms must be formulated relative to A, i.e., for all elements of type $\alpha\ \mathit{in\ the\ set}\ A$. In addition, one requires that A is closed under e and *times*, i.e., e is in A and *times* preserves membership in A. Also all proved facts about groups are formulated relative to A. For example, whenever we quantify over $x : \alpha$ we assume that x is in A. If multiple type variables are involved, as in the case of group homomorphisms $f : \alpha \Rightarrow \beta$, one uses multiple carrier sets, e.g., $A : \alpha\ \mathit{set}$ and $B : \beta\ \mathit{set}$, and makes the corresponding assumptions, e.g., that f sends elements of A to elements of B. The set-based approach is more general than the type-based one, but comes with a high

[1] There are major interactive theorem provers that do not fall squarely in either of the mentioned categories. For example, PVS [Owre et al. 1992] has dependent types but also subtypes based on set comprehension. Moreover, ACL2 [Kaufmann et al. 2000] is based on a first-order theory of arithmetics, and Mizar [Grabowski et al. 2010] on a first-order set theory.

[2] In HOL, for any type σ, the "powerset" type $\sigma\ \mathit{set}$ is either an abbreviation for, or a copy of, the type of predicates, i.e., functions from σ to *bool*. Section 2.2 gives details.

price in terms of bureaucracy: Additional assumptions about membership must be maintained and discharged when appropriate, which makes the statements heavier and the proofs less automatic.[3]

Kunčar and Popescu [2019] started to address this problem under the motto "prove easily and still be flexible"—the idea being that users should be able to develop their results in a lightweight type-based fashion, and then receive the set-based version of the theorems for free whenever needed. This intuitive goal turned out to be challenging from a formal perspective, for two main reasons:

(1) Inferring the set-based from the type-based theorem cannot be done using HOL's axioms. In other words, set-based relativization cannot be performed via a derived rule in HOL.
(2) Even formulating (let alone proving) the set-based counterpart of a type-based theorem is not possible without a suitable infrastructure that "anticipates" this formulation. For example, a theorem about groups may universally quantify the elements in $\alpha\ \kappa$ where α models the group's carrier (as before) and κ is some type constructor, such as *set* or *list*. For the set-based counterpart, we need a way to lift the carrier set, $A : \alpha\ set$, to a corresponding subset of $\alpha\ \kappa$.

Instead of tackling these foundational problems, Kunčar and Popescu focused on practical progress from the users' perspective. They bypassed problem (1) by adding a new rule to HOL that facilitates the types-to-sets relativization in certain cases: Local Typedef. And they bypassed problem (2) by restricting their scope to type constructors κ for which natural set-lifting functions are known to exist, which include container types such as lists and trees. Some case studies have been performed in Isabelle/HOL [Divasón et al. 2018; Divasón and Thiemann 2022; Immler and Zhan 2019], validating the usefulness of the Local Typedef rule for simplifying formal developments. Tool support for automating the application of these rules has also been recently implemented [Milehins 2022].

While postulating new rules that extend HOL's axiomatic basis (like the Local Typedef rule mentioned above) may seem like an easy way out, it constitutes a foundational compromise—regarded with reluctance by the HOL community of developers and users, who take very seriously Bertrand Russell's aphorism [Russell 1919]: "The method of 'postulating' what we want has many advantages; they are the same as the advantages of theft over honest toil." Indeed, one of the reasons why HOL-based provers are today's some of the most reliable verification tools is the smallness of their axiomatic kernel. And the HOL logic in its current form has been successfully used across different provers (with minor variations of the same rules and axioms) ever since Gordon's introduction of the first HOL system more than 30 years ago [Gordon 1991]. Yet, the naturalness of Local Typedef—which simply allows to define types from sets just like with standard HOL type definitions, but in local proof contexts—could make us wonder whether Gordon did not overlook some important mechanisms when introducing his enduring axiomatization of HOL.

This Paper's Contributions. Here, we tackle the aforementioned foundational challenges raised by types-to-sets relativization. First, we generalize types-to-sets to types-to-PERs relativization, where "PER" means "partial equivalence relation". This allows us to systematically apply relativization to higher-order types in a manner that ensures the desired/expected properties.

We address challenge (2) by delimiting a fragment of HOL, given by what we call *widely-typed* definitional theories, where relativization can be defined. Essentially, wide typedness means that the

[3] An alternative to set-based approach is to introduce type definitions for sets of interest when using type-based results—e.g., instead of working with the set of bijections on α, define a new type of bijections and copy the group operations (function composition and identity) from $\alpha \Rightarrow \alpha$ to this new type. This alternative has several drawbacks. First, it moves the burden from the service (the formal library) to the client (the library user) who has to do more work to use a result. Second, native operators from the old type (e.g., function application) must be replaced with more bureaucratic copies. Finally, one loses the flexibility of handling overlapping sets on the same type, e.g., to work with both a group of bijections (taking advantage of a group library) and the monoid of all functions (taking advantage of a monoid library) one must explicitly convert between different types.

newly introduced types are large enough relatively to the size of the defining predicate's relativization. The widely-typed fragment seems to include all practical HOL developments, and currently all developments we are aware of—though artificial non-widely-typed examples can be constructed.

We address challenge (1) by proving that, for the widely-typed fragment, types-to-PERs (in particular, types-to-sets) relativization is *admissible*, i.e., provided the type-based statement is provable, its PER-based (in particular, set-based) counterpart is also provable. Thus, relativization can be accepted as a shortcut rule to boost user productivity *without extending HOL's axiomatic base*.

We also contribute to the understanding of parametricity in HOL, where both Hilbert choice and comprehension-based type definitions *a priori* are non-parametric. We show that our relativization scheme achieves parametricity with respect to PERs. Finally, we show that the Local Typedef rule is also admissible for the widely-typed fragment. So, after all, this rule was not really overlooked by Gordon, since its admissibility reveals it as an implicit part of the original HOL axiomatization.

This Paper's Structure. We start by recalling the HOL foundations: type system, deduction system, and definitional mechanisms for constants and types (Section 2). Then we discuss, quasi-informally and based on examples, the goal of types-to-sets relativization in the form of a wish list of desirable properties (Section 3). These include Admissibility, but also some natural well-behavedness properties including PER-Parametricity, Recoverability and Connective & Quantifier Suitability. After that, we move to the technical presentation of the relativization process (Section 4), by defining operators for relativizing types and terms and formulating the main theorems asserting the desirable properties of these operators. As a consequence of Admissibility (of relativization) and Recoverability, we show that in the widely-typed fragment, the Local Typedef rule is also admissible (Section 5). An adaptation of our results can also handle the extension of classic HOL with a mechanism for the ad hoc overloading of constant instance definitions, as featured in Isabelle/HOL (Section 6).

Finally, we analyze the relativization's scope—namely, the outreach of our identified widely-typed fragment (Section 7). We show that, notwithstanding their low-level definitions in HOL, all container types, including the inductive and coinductive datatypes, fall within this fragment. As further empirical evidence, we show that the type definitions performed in a large subset of Isabelle/HOL's distribution are all widely-typed. We do this with the help of a publicly available custom tool [Popescu and Traytel 2022b] that automates the relativization and prompts the user to prove wide typedness when necessary. A technical report [Popescu and Traytel 2022a] referenced throughout the paper provides more details on this paper's results, proofs, and additional results.

2 HOL PRELIMINARIES

While its ideas go back a long way (to the work of Church [1940] and beyond), Higher-Order Logic (HOL) as used in the theorem proving community contains a unique blend of features proposed by Gordon at the end of the eighties, inspired by practical verification needs: Its type system is the rank-one polymorphic extension of simple types, generated using the function-space constructor from two base types, *bool* and *ind*; its terms have built-in equality (from which all the usual connectives and quantifiers are derived); deduction, operating on terms of type *bool* called formulas, is regulated by the built-in axioms of Equality, (Hilbert) Choice and Infinity (for the type *ind*).

In addition to this purely logical layer, users can perform constant and type declarations and definitions. Type definitions proceed by indicating a predicate on an existing type and carving out the new type from the subset satisfying the predicate. For accepting a type definition, the system requires a proof that the subset is nonempty. This is because *HOL types are required to be nonempty*—a major design decision, with practical and theoretical ramifications [Gordon and Melham 1993; Paulson 1988]. No new axioms are accepted (more precisely, they are strongly discouraged), besides the aforementioned definitions. This minimalistic, *definitional approach* offers good protection against the accidental introduction of *inconsistency* (the possibility to prove False).

2.1 Syntax

All throughout this paper, we fix an infinite set *TVar*, of *type-variables*, ranged by α, β and an infinite set *Var*, of *(term) variable*, ranged by x, y, z or by other symbols, as suitable for the context (e.g., A for a set or p for a predicate). A *type structure* is a pair $(K, arOf)$ where K is a set of symbols, ranged by κ, called *type constructors*, containing three special symbols: *bool*, *ind* and $\Rightarrow$ (representing the type of booleans, an infinite type of individuals, and the function type constructor) and $arOf : K \to \mathbb{N}$ associates arities to the type constructors, such that $arOf(bool) = arOf(ind) = 0$ and $arOf(\Rightarrow) = 2$.

The *types* associated to $(K, arOf)$, ranged by σ, τ, are defined as: $\sigma ::= \alpha \mid (\sigma_1, \ldots, \sigma_{arOf(\kappa)})\,\kappa$. Thus, a type is either a type-variable or an n-ary type constructor κ postfix-applied to a number of types corresponding to its arity. We write $Type_{(K,arOf)}$ for the set of types associated to $(K, arOf)$.

A *signature* is a tuple $\Sigma = (K, arOf, Const, tpOf)$, where $(K, arOf)$ is a type structure, *Const*, ranged over by c, is a set of symbols called *constants*, containing two special symbols: $=$ and *choice* (aimed at representing equality and Hilbert choice of some element from a type, respectively), and $tpOf : Const \to Type_{(K,arOf)}$ is a function associating a type to every constant, such that:

$$tpOf(=) = \alpha \Rightarrow \alpha \Rightarrow bool \qquad tpOf(choice) = (\alpha \Rightarrow bool) \Rightarrow \alpha$$

Convention 1. For the rest of this paper, we fix a signature $\Sigma = (K, arOf, Const, tpOf)$. We usually write $Type_\Sigma$, or simply *Type*, instead of $Type_{(K,arOf)}$.

Let $\mathsf{TV}(\sigma)$ be the set of type σ's type-variables. A type σ is *ground*, or *monomorphic*, if $\mathsf{TV}(\sigma) = \emptyset$, and *polymorphic* otherwise. Let *GType* be the set of ground types. The *support* of $\rho : TVar \to Type$ is the set of type-variables where ρ is not the identity: $\mathsf{supp}(\rho) = \{\alpha \mid \rho(\alpha) \neq \alpha\}$. A *type-substitution* is a function $\rho : TVar \to Type$ with finite support. The application of ρ to a type σ, written $\sigma[\rho]$, is defined recursively by $\alpha[\rho] = \rho(\alpha)$ and $((\sigma_1, \ldots, \sigma_m)\kappa)[\rho] = (\sigma_1[\rho], \ldots, \sigma_m[\rho])\kappa$. If $\alpha_1, \ldots, \alpha_m$ are all different, we write $\tau_1/\alpha_1, \ldots, \tau_n/\alpha_m$ for the type-substitution that sends α_i to τ_i and each $\beta \notin \{\alpha_1, \ldots, \alpha_m\}$ to β. Type σ is an *instance* of τ *via* ρ, written $\sigma \leq_\rho \tau$, if $\tau[\rho] = \sigma$. Type σ is an *instance* of τ, written $\sigma \leq \tau$, if there exists $\rho \in TSubst$ such that $\sigma \leq_\rho \tau$. Given $\rho_1, \rho_2 \in TSubst$, we write $\rho_1 \cdot \rho_2$ for their *composition*: $(\rho_1 \cdot \rho_2)(\alpha) = (\rho_1(\alpha))[\rho_2]$. For all types σ, it holds that $\sigma[\rho_1 \cdot \rho_2] = \sigma[\rho_1][\rho_2]$.

A *typed variable* is a pair of a variable x and a type σ, written x_σ. We let VarT denote the set of typed variables. A *constant instance* is a pair of a constant and a type, written c_σ, such that $\sigma \leq tpOf(c)$. Let *CInst* denote the set of constant instances. We extend the notion of being an instance ($\leq$) from types to constant instances, by defining $c_\tau \leq d_\sigma$ to mean that $c = d$ and $\tau \leq \sigma$.

The signature's *terms*, ranged over by s, t, are defined by the grammar $t ::= x_\sigma \mid c_\sigma \mid t_1\ t_2 \mid \lambda x_\sigma.\ t$. Thus, a term is either a variable, or a constant instance, or an application, or an abstraction. As usual, we identify terms modulo alpha-equivalence. We let $Term_\Sigma$, or simply *Term*, ranged by s and t, denote the set of terms. Typing relates terms and types, written $t : \sigma$ and defined inductively:

$$\frac{x_\sigma \in \mathsf{VarT}}{x_\sigma : \sigma} \qquad \frac{c_\sigma \in CInst}{c_\sigma : \sigma} \qquad \frac{t_1 : \sigma \Rightarrow \tau \qquad t_2 : \sigma}{t_1\ t_2 : \tau} \qquad \frac{t : \tau}{\lambda x_\sigma.\ t : \sigma \Rightarrow \tau}$$

A term t is called *typable* if there exists a type σ such that $t : \sigma$. If it exists, this type is necessarily unique. Let *TTerm* denote the set of well-typed terms and $\mathsf{FTV}(t)$ the set of t's free typed variables. For example, $\mathsf{FTV}(\lambda x_\sigma.x_\sigma = y_\sigma) = \{y_\sigma\}$. The term t is *closed* if $\mathsf{FTV}(t) = \emptyset$. Let $t[s/x_\sigma]$ be the term obtained from t by capture-free substituting the term s for all free occurrences of x_σ. Let $\mathsf{TV}(t)$ be the set of type-variables occurring in (any type that occurs in) t. A term is *ground*, or *monomorphic*, if $\mathsf{TV}(t) = \emptyset$, and *polymorphic* otherwise. We can apply a type-substitution ρ to a term t, written $t[\rho]$, by applying it to the types of all variables and constant instances occurring in t.

The *support* of $\theta : \mathsf{VarT} \to TTerm$ is the set of typed variables x_σ where θ is not the identity: $\mathsf{supp}(\theta) = \{x_\sigma \in \mathsf{VarT} \mid \theta(x_\sigma) \neq x_\sigma\}$. A *term-substitution* is a function $\delta : \mathsf{VarT} \to TTerm$ of finite support such that $\delta(x_\sigma) : \sigma$ for all $x_\sigma \in \mathsf{VarT}$. The application of δ to the term t, written $t[\delta]$, pro-

ceeds by replacing all free variables x_σ of t with $\delta(x_\sigma)$ in a capture-avoiding fashion. Given distinct typed variables $x_{1\sigma_1}, \ldots, x_{n\sigma_n}$ and terms $t_1, \ldots, t_n$ such that $t_i : \sigma_i$, we write $t_1/x_{1\sigma_1}, \ldots, t_n/x_{n\sigma_n}$ for the term-substitution that sends each $x_{i\sigma_i}$ to t_i and all the other typed variables to themselves. For example, if $t : \sigma$ and $x_\sigma \notin \mathsf{FTV}(t)$, then $(\lambda x_\sigma.\ x_\sigma = y_\sigma)[t/y_\sigma] = (\lambda x_\sigma.\ x_\sigma = t)$.

A *formula* is a term of type *bool*. We let $Fmla_\Sigma$, or simply *Fmla*, ranged by φ and χ, denote the set of formulas. A *unary predicate* is a term of type $\sigma \Rightarrow bool$ for some type σ. Similarly, a *relation* (i.e., binary predicate) is a term of type $\sigma_1 \Rightarrow \sigma_2 \Rightarrow bool$ for some types σ_1 and σ_2, etc.

2.2 More Notations and Abbreviations

We will use a plethora of notations and abbreviations that will make the presentation more readable. We urge the reader to consult this section whenever in doubt about the meaning of some symbols.

The formula connectives (e.g., $\wedge$, $\neg$ and $\longrightarrow$) and quantifiers ($\forall$ and $\exists$) are defined from the HOL primitives. For example, for any type σ, we write $\forall x_\sigma.\ t$ for $all_\sigma\ (\lambda x_\sigma.\ t)$, where all_σ is the term $\lambda p_{\sigma \Rightarrow bool}.\ p = (\lambda x_\sigma.\ true)$. Our technical report gives more details. Given a finite family of formulas $(\varphi_i)_{i \in I}$, we write $\bigwedge_{i \in I} \varphi_i$ for the conjunction of the formulas in the family; and $\bigvee_{i \in I} \varphi_i$ for their disjunction. To avoid confusion with later object-logic definitions, we treat connectives, quantifiers and other concepts introduced below as abbreviations (i.e., meta-level definitions of certain HOL terms).

Usually, HOL introduces sets as an abbreviation for predicates, which is what we will assume for most of this paper. Namely, we use the following notations for powerset (powertype), set membership and set comprehension: We let $\sigma\ set$ be an abbreviation for $\sigma \Rightarrow bool$. Assuming $t_1 : \sigma$ and $t_2 : \sigma\ set$, we let $t_1 \Coloneqq t_2$ be abbreviation for $t_2\ t_1$. Assuming φ is a formula, we let $\{\!|x_\sigma\ .\ \varphi|\!\}$ be an abbreviation for $\lambda x_\sigma\ .\ \varphi$. Some HOL-based provers do it slightly differently but fundamentally equivalently, namely by defining a standalone type of sets that is copy of that of predicates.

We sometimes omit the types if they can be inferred—e.g, we write $\lambda x_\sigma.\ x$ instead of $\lambda x_\sigma.\ x_\sigma$. We will use infix notation, and most of the times omit the type instance, for equality, e.g., $t_1 = t_2$ and membership, e.g., $t \Coloneqq A$. Given terms $b : bool$, $t_1 : \sigma$ and $t_2 : \sigma$, the if-then-else expression, written $if\ b\ then\ t_1\ else\ t_2$, is the term $choice\ (\lambda x_\sigma.\ (b \longrightarrow x_\sigma = t_1)\ \wedge\ (\neg\ b \longrightarrow x_\sigma = t_2))$. Its behavior is the expected one: It equals t_1 if b is *true* and equals t_2 if b is *false*. We will write:

- $t_1 \neq t_2$ for $\neg\ t_1 = t_2$;
- $Some\ y_\tau.\ \varphi$ for $choice\ (\lambda y_\tau.\ \varphi)$;
- $Some\ y_\tau \Coloneqq A$ for $Some\ y_\tau.\ y \Coloneqq A$;
- $\forall x_\sigma \Coloneqq A.\ \varphi$ for $\forall x_\sigma.\ x \Coloneqq A \longrightarrow \varphi$;
- $\exists x_\sigma \Coloneqq A.\ \varphi$ for $\exists x_\sigma.\ x \Coloneqq A\ \wedge\ \varphi$;
- $\emptyset_\sigma$ for $\{\!|x_\sigma\ .\ false|\!\}$;
- σ rel (read "the type of relations on σ") for $\sigma \Rightarrow \sigma \Rightarrow bool$;
- $univ_\sigma$ (read "the universe set for σ") for $\{\!|x_\sigma\ .\ true|\!\}$;
- $\bigwedge \Delta$ for the conjunction of all formulas in Δ (provided Δ is a finite set of formulas).

2.3 Partial Equivalence Relations (PERs)

Let *per* abbreviate the following predicate of type α rel $\Rightarrow bool$: $\lambda r_{\alpha\ \mathsf{rel}}.\ (\forall x_\alpha, y_\alpha.\ r\ x\ y \longrightarrow r\ y\ x) \wedge (\forall x_\alpha, y_\alpha, z_\alpha.\ r\ x\ y \wedge r\ y\ z \longrightarrow r\ x\ z)$. Thus, *per r* says that r is a *partial equivalence relation* (*PER* for short), i.e., a relation that is symmetric and transitive. If r is a PER on σ, we will refer to the set $\{x_\sigma \mid r\ x\ x\}$, which is the same as the set $\{x_\sigma \mid \exists y_\sigma.\ r\ x\ y\}$, as the *domain* of r. Note that r is an equivalence relation on its domain. We let $per_{\neq \emptyset}$, read "non-empty partial equivalence relation", abbreviate the predicate $\lambda r_{\alpha\ \mathsf{rel}}.\ per\ r\ \wedge\ (\exists x_\alpha.\ r\ x\ x)$. In the above formulation, non-emptiness of r refers to the existence of a an element that is related to itself, i.e., to the non-emptiness of r's domain. But since r is also a PER, this is equivalent to saying that there exists a pair of related elements: $\exists x_\alpha, y_\alpha.\ r\ x\ y$.

Given a PER $r : \sigma$ rel and a predicate $t : \sigma \Rightarrow bool$, the *restriction of r to t*, written $r_{\restriction t}$, is defined as $\lambda x_\sigma, y_\sigma.\ t\ x \wedge t\ y \wedge r\ x\ y$. Given two types τ and σ and PERs on these types, $p : \tau$ rel and $q : \sigma$ rel, we are interested in $\sigma \Rightarrow \tau$ that are bijections between the domains of p and q up to the induced equalities. To capture this, we define the *bijUpto* predicate of type σ rel $\Rightarrow \tau$ rel $\Rightarrow (\sigma \Rightarrow \tau) \Rightarrow bool$ to be

$$\lambda p_{\tau\ \mathrm{rel}},\, q_{\sigma\ \mathrm{rel}},\, f_{\sigma\Rightarrow\tau}.\ (\forall x_\tau,\, x'_\tau.\ p\ x\ x' \longrightarrow q\ (f\ x)\ (f\ x')) \wedge (\forall x_\tau,\, x'_\tau.\ p\ x\ x \wedge p\ x'\ x' \wedge q\ (f\ x)\ (f\ x') \longrightarrow p\ x\ x') \wedge (\forall y_\sigma.\ q\ y\ y \longleftrightarrow (\exists x_\tau.\ p\ x\ x \wedge q\ y\ (f\ x)))$$

We read *bijUpto* $f\ p\ q$ as "f is a bijection-up-to between p and q". The three conjuncts that make up the notion of bijection-up-to state the following: (1) f is a "function up to" between p and q, i.e., it preserves the relations, (2) f is an "injection up to" between p and q, i.e., for any two elements in the domain of p, if their images through f are q-related, then they are themselves p-related, and (3) f is a "surjection up to" between p and q, i.e., for any element in the domain of q, there exists an element in the domain of p whose image through f is q-related to it. A bijection-up-to between PERs p and q induces a bijection between the equivalence classes on their domains.

2.4 Theories, Axioms and Deduction

We call *theory* (over Σ) any set of closed (Σ-)formulas. The HOL axioms, forming the set Ax, consist of: (1) the Equality axioms; (2) the Infinity axiom (stating that there exists an injective but non-surjective function between *ind* and itself, which makes the type *ind* infinite); (3) the Choice axiom, which states that the Hilbert choice operator returns an element satisfying its argument predicate (if nonempty): $p_{\alpha\Rightarrow bool}\ x \longrightarrow p\ (choice\ p)$. (Our technical report gives more details.) HOL is classical, since the principle of excluded middle follows from Ax. A *context* Γ is a finite set of formulas. The type-variable α is fresh for Γ when it does not appear in (any formula in) Γ; similarly, x_σ is fresh for Γ when x_σ does not appear *free* in Γ. We define *deduction* as a ternary relation $\vdash$ between theories D, contexts Γ and formulas φ, written $D; \Gamma \vdash \varphi$.

$$\frac{}{D; \Gamma \vdash \varphi}\ [\varphi \in \mathrm{Ax} \cup D]\ (\textsc{Fact}) \qquad \frac{D; \Gamma \vdash \varphi}{D; \Gamma \vdash \varphi[\sigma/\alpha]}\ [\alpha \text{ fresh for } \Gamma]\ (\text{T-}\textsc{Inst}) \qquad \frac{}{D; \Gamma \vdash (\lambda x_\sigma.\ t)\ s = t[s/x_\sigma]}\ (\textsc{Beta})$$

$$\frac{}{D; \Gamma \vdash \varphi}\ [\varphi \in \Gamma]\ (\textsc{Assum}) \qquad \frac{D; \Gamma \vdash \varphi}{D; \Gamma \vdash \varphi[t/x_\sigma]}\ [x_\sigma \text{ fresh for } \Gamma]\ (\textsc{Inst}) \qquad \frac{D; \Gamma \vdash f\ x_\sigma = g\ x_\sigma}{D; \Gamma \vdash f = g}\ [x_\sigma \text{ fresh for } \Gamma]\ (\textsc{Ext})$$

$$\frac{D; \Gamma \cup \{\varphi\} \vdash \chi}{D; \Gamma \vdash \varphi \longrightarrow \chi}\ (\textsc{ImpI}) \qquad \frac{D; \Gamma \vdash \varphi \longrightarrow \chi \quad D; \Gamma \vdash \varphi}{D; \Gamma \vdash \chi}\ (\mathrm{MP})$$

The axioms and the deduction rules we gave here are (a variant of) the standard ones for HOL [Gordon and Melham 1993; Harrison 2009]. Different provers implementing Higher-Order Logic, such as HOL4, HOL Light, HOL-ProofPower, HOL Zero and Isabelle/HOL, use slightly different sets of logical primitives and slightly different rules and axioms. However, these provers implement essentially the same logic, up to logical equivalence.

We write $D \vdash \varphi$ instead of $D; \emptyset \vdash \varphi$ and $\vdash \varphi$ instead of $\emptyset; \emptyset \vdash \varphi$ (i.e., we omit empty contexts and theories). The HOL axioms are not part of the theory D, but are wired together with D in the (Fact) axiom. So $\vdash \varphi$ indicates that φ is provable from the HOL axioms only. (See also our technical report.)

2.5 HOL Definitions

Besides deduction, another main component of the HOL logic is a mechanism for introducing new constants and types by spelling out their definitions. The *built-in type constructors* are *bool*, *ind*, and $\Rightarrow$. The *built-in constants* are $=$ and *choice*. Since the built-in items have an already specified behavior (by the HOL axioms), only non-built-in items can be defined.

Definition 2. Given a non-built-in constant c of type σ, and a closed term $t : \sigma$, we let $c_\sigma \equiv t$ denote the formula $c_\sigma = t$. We call $c_\sigma \equiv t$ a *constant definition* provided $\mathrm{TV}(t) \subseteq \mathrm{TV}(c_\sigma)$.

Note that we introduced constant instance definitions, which is a slightly more general concept than the standard one of constant definition used traditionally in HOL [Gordon and Melham 1993].

Namely, we allow defining an instance of a constant that is less general than its type. This is to also accommodate the Isabelle/HOL definitional mechanisms, where this flexibility enables ad hoc overloading and, based on this, Haskell-style type classes [Nipkow and Snelting 1991].

For example, one can introduce the constant *total* of type $\alpha\ rel \Rightarrow bool$, expressing that a relation on α is (left-)total, using the constant definition $total_{\alpha\ rel \Rightarrow bool} \equiv \lambda r.\ \forall x_\alpha.\ \exists y_\alpha.\ r\ x\ y$.

Definition 3. Given types τ and σ and a closed term $t : \sigma \Rightarrow bool$, we let $\tau \equiv t$ denote the formula $\exists rep_{\tau \Rightarrow \sigma}.\ \mathsf{typedef}_{\tau,\sigma,t,rep}$ where $\mathsf{typedef}_{\tau,\sigma,t,rep}$ denotes the formula

$$(\forall x_\tau, x'_\tau.\ rep\ x = rep\ x' \longrightarrow x = x') \ \wedge\ (\forall y_\sigma.\ t\ y \ \longleftrightarrow\ (\exists x_\tau.\ y = rep\ x))$$

We call $\tau \equiv t$ a *type definition*, provided τ has the form $(\alpha_1, \ldots, \alpha_m)\kappa$ such that κ is a non-built-in type constructor, the α_i's are all distinct type-variables and $\mathsf{TV}(t) \subseteq \{\alpha_1, \ldots, \alpha_m\}$. (Hence, we have $\mathsf{TV}(t) \subseteq \mathsf{TV}(\tau)$, which also implies $\mathsf{TV}(\sigma) \subseteq \mathsf{TV}(\tau)$.)

A type definition expresses the following: The new type $(\alpha_1, \ldots, \alpha_m)\kappa$ is embedded in its host type σ via some one-to-one function *rep* (the first conjunct of the formula), and the image of this embedding consists of the elements of σ for which t holds (the second conjunct). Since types in HOL are required to be nonempty, the definition is only accepted if the user provides a proof that $\exists x_\sigma.\ t\ x$ holds.[4] Thus, to perform a type definition, one must give a nonemptiness proof.

For example, the type definition $\alpha\ totalRel \equiv total$ introduces the type of total relations on α. This requires proving that $\exists\, r_{\alpha\ rel}.\ total\ r$, which is true because, e.g., the equality relation is total.

A HOL development, i.e., a session of interaction with the HOL logic from a user's perspective, intertwines type and constant definitions and (statements and proofs of) theorems. Since theorems are consequences of definitions, we do not model them explicitly, but focus on definitions.

2.6 Signature Extensions and the Initial Signature

In the remainder of this paper, when necessary for disambiguation, we will indicate the signature Σ as a subscript when denoting sets and relations associated to it: $Type_\Sigma$, $Term_\Sigma$, $CInst_\Sigma$, $\vdash_\Sigma$, etc.

Given a signature $\Sigma = (K, arOf, Const, tpOf)$ and an item u, we write $u \in \Sigma$ to mean that $u \in K$ or $u \in Const$. Given signatures $\Sigma = (K, arOf, Const, tpOf)$ and $\Sigma' = (K', arOf', Const', tpOf')$, we say Σ *is included in* Σ', or Σ' *extends* Σ, written $\Sigma \subseteq \Sigma'$, when $K \subseteq K'$, $Const \subseteq Const'$ and the functions $arOf'$ and $tpOf'$ are extensions of $arOf$ and $tpOf$, respectively. We write $u \in \Sigma' \setminus \Sigma$ to mean $u \in \Sigma'$ and $u \notin \Sigma$. If $c \notin Const$ and $\sigma \in Type_\Sigma$, we write $\Sigma \cup \{(c, \sigma)\}$ for the extension of Σ with a new constant c of type σ. Similarly, if $\kappa \notin K$, we write $\Sigma \cup \{(\kappa, n)\}$ for the extension of Σ with a new type constructor κ of arity n. We write Σ_{init} for the *initial signature*, containing only built-in type constructors and constants. Note that, by definition, any signature extends the initial signature.

2.7 HOL Definitional Theories

Let $\Sigma = (K, arOf, Const, tpOf)$ be a signature and let D be a finite theory over Σ.

Definition 4. D is said to be a *definitional theory* if $D = \{def_1, \ldots, def_n\}$, where each def_i is a (type or constant) definition of the form $u_i \equiv t_i$, and there exist the signatures $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$ such that $\Sigma_0 = \Sigma_{\mathsf{init}}$, $\Sigma_n = \Sigma$ and the following hold for all $i \in \{1, \ldots, n\}$:

(1) $t_i \in Term_{\Sigma_{i-1}}$ and Σ_i is the extension of Σ_{i-1} with a fresh item defined by def_i, namely:
  (1.1) If u_i has the form $(\alpha_1, \ldots, \alpha_m)\kappa$, then $\kappa \notin \Sigma_{i-1}$ and $\Sigma_i = \Sigma_{i-1} \cup \{(\kappa, m)\}$
  (1.2) If u_i has the form c_σ, then $c \notin \Sigma^i$ and $\Sigma_i = \Sigma_{i-1} \cup \{(c, \sigma)\}$

[4] Disallowing empty types in HOL is necessary, if we accept that Hilbert choice has the polymorphic type $(\alpha \Rightarrow bool) \Rightarrow \alpha$. In this case, $choice_{(\sigma \Rightarrow bool) \Rightarrow \sigma}(\lambda x_\sigma.\ true)$ is an inhabitant of σ for any type σ. Paulson [1988] discusses this design decision.

(2) If def_i is a type definition, i.e., u_i is a type and $t_i : \sigma \Rightarrow bool$, then $\{def_1, \ldots, def_{i-1}\} \vdash_{\Sigma_{i-1}} \exists x_\sigma.\ t_i\ x$

These conditions express that the theory D consists of intertwined type and constant definitions def_i that are introduced in a well-founded manner, yielding a chain of signature extensions $\Sigma_{\mathsf{init}} = \Sigma_0 \subseteq \Sigma_1 \subseteq \Sigma_2 \ldots \subseteq \Sigma_n = \Sigma$ that start from the initial signature Σ_{init} and end with Σ. Each Σ_i extends Σ_{i-1} with u_i, the unique item being defined by def_i, i.e., as $u_i \equiv t_i$. As shown by condition (2), in the case of type definitions, we also require proofs of non-emptiness of the defining predicate t (from the definitions available so far). Note that we do not allow declared (but undefined) types and constants. But Section 6 discusses a modification of our results to cope with such declared-only entities.

2.8 Definitional Dependency Relation

Since HOL with definitions is well-known to be consistent [Pitts 1993], one would expect that definitions cannot introduce infinite (including cyclic) chains of dependencies. Next, we will make this rigorous, following Kunčar and Popescu [2018].

We let $Type^\bullet_\Sigma$ be the set of Σ-types that have a non-built-in type constructor at the top, and $CInst^\bullet_\Sigma$ be the set of instances of non-built-in constants. Given any term t, we let $types^\bullet(t)$ be the set of all types from $Type^\bullet_\Sigma$ appearing in t and $cinsts^\bullet(t)$ be the set of all constant instances from $CInst^\bullet_\Sigma$ appearing in t. The definitions are given below. The $types^\bullet$ operator is overloaded for types and terms.

$$types^\bullet(\alpha) = \{\alpha\} \quad types^\bullet(bool) = types^\bullet(ind) = \emptyset \quad types^\bullet(\sigma_1 \Rightarrow \sigma_2) = types^\bullet(\sigma_1) \cup types^\bullet(\sigma_2)$$
$$types^\bullet((\sigma_1, \ldots, \sigma_n)\kappa) = \{(\sigma_1, \ldots, \sigma_n)\kappa\} \cup types^\bullet(\sigma_1) \ \ldots \cup \ \ldots types^\bullet(\sigma_n) \text{ if } \kappa \notin \{\Rightarrow, bool, ind\}$$
$$types^\bullet(x_\sigma) = types^\bullet(c_\sigma) = types^\bullet(\sigma) \quad types^\bullet(t_1\ t_2) = types^\bullet(t_1) \cup types^\bullet(t_2)$$
$$types^\bullet(\lambda x_\sigma.\ t) = types^\bullet(\sigma) \cup types^\bullet(t)$$
$$cinsts^\bullet(x_\sigma) = \emptyset \quad cinsts^\bullet(c_\sigma) = \begin{cases} \{c_\sigma\} & \text{if } c_\sigma \in CInst^\bullet \\ \emptyset & \text{otherwise} \end{cases} \quad cinsts^\bullet(t_1\ t_2) = cinsts^\bullet(t_1) \cup cinsts^\bullet(t_2) \quad cinsts^\bullet(\lambda x_\sigma.\ t) = cinsts^\bullet(t)$$

Definition 5. Let D be a HOL definitional theory. The *dependency relation* $\rightsquigarrow$ associated to D on $Type^\bullet_\Sigma \cup CInst^\bullet_\Sigma$ is defined as follows: $u \rightsquigarrow v$ means that there exists in D a definition of the form $u \equiv t$ such that $v \in cinsts^\bullet(t) \cup types^\bullet(t)$.

We write $\rightsquigarrow^\downarrow$ for the (type-)substitutive closure of $\rightsquigarrow$, defined as follows: $u \rightsquigarrow^\downarrow v$ means that there exist u', v' and a type substitution ρ such that $u = u'[\rho]$, $v = v'[\rho]$ and $u' \rightsquigarrow v'$.

Prop 6. [Kunčar and Popescu 2018] Let D be a HOL definitional theory. Then $\rightsquigarrow^\downarrow$ is terminating.

3 DISCUSSION

Next, we discuss the goal of relativizing HOL statements and its challenges, using examples to which we will refer all throughout the paper (Section 3.1). We synthesize some properties one should reasonably expect from relativization (Section 3.2), which become targets for our technical development.

3.1 Some Working Examples

Let us consider in more detail the example from the introduction—of modeling in HOL the notion of a group. In the lightweight, type-based approach, we define the polymorphic predicate $group : (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow bool$ with $group\ times\ e$ stating that $(\alpha, times, e)$ forms a group, i.e., $times$ is associative with neutral element e, and any element has an inverse:

$$(\forall x_\alpha, y_\alpha, z_\alpha.\ times\ (times\ x\ y)\ z = times\ x\ (times\ y\ z)) \wedge$$
$$(\forall x_\alpha.\ times\ x\ e = x \ \wedge \ times\ e\ x = x) \wedge$$
$$(\forall x_\alpha.\ \exists y_\alpha.\ times\ x\ y = e \ \wedge \ times\ y\ x = e)$$

After defining what a group is, we can prove properties about it. Let us pick three examples: the first-order property of left-cancellation

$$(1)\quad \begin{array}{l} group\ times\ e\ \longrightarrow \\ (\forall x_\alpha, y_\alpha, z_\alpha.\ times\ x\ y = times\ x\ z \longrightarrow y = z) \end{array}$$

the second-order property of left-inverse function uniqueness

$$(2)\quad \begin{array}{l} group\ times\ e\ \longrightarrow \\ (\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha}.\ (\forall x_\alpha.\ times\ (inv\ x)\ x = e) \wedge (\forall x_\alpha.\ times\ (inv'\ x)\ x = e) \longrightarrow inv = inv') \end{array}$$

and a property involving a defined type (here, that of lists)

$$(3)\quad \begin{array}{l} group\ times\ e\ \longrightarrow \\ (\forall xs_{\alpha\ list}, ys_{\alpha\ list}.\ fold\ times\ e\ (append\ xs\ ys) = times\ (fold\ times\ e\ xs)\ (fold\ times\ e\ ys)) \end{array}$$

where $append : \alpha\ list \Rightarrow \alpha\ list \Rightarrow \alpha\ list$ and $fold : (\beta \Rightarrow \alpha \Rightarrow \beta) \Rightarrow \beta \Rightarrow \alpha\ list \Rightarrow \beta$ are the polymorphic operators for appending two lists and respectively left-folding a list with a function.

Let us now see what the above becomes under the more bureaucratic (yet more flexible) set-based approach. We work in a context that fixes a set $A : \alpha\ set$ and assumes that A is closed under the considered operations: $closed_2\ A\ times$, defined as $\forall x_\alpha, y_\alpha.\ x \in A \wedge y \in A \longrightarrow times\ x\ y \in A$, and $closed_0\ A\ e$, defined as $e \in A$. The group notion is now expressed as $group^{rlt}\ A\ times\ e$, defined as

$$\begin{array}{l} (\forall x_\alpha \in A, y_\alpha \in A, z_\alpha \in A.\ times\ (times\ x\ y)\ z = times\ x\ (times\ y\ z)) \ \wedge \\ (\forall x_\alpha \in A.\ times\ x\ e = x \ \wedge \ times\ e\ x = x) \ \wedge \\ (\forall x_\alpha \in A.\ \exists y_\alpha \in A.\ times\ x\ y = e \ \wedge \ times\ y\ x = e) \end{array}$$

Henceforth, we work *relative to the carrier set A*. In this carrier-relative world, left-cancellation becomes, as one might expect:

$$(1')\quad \begin{array}{l} closed_2\ A\ times \ \wedge \ closed_0\ A\ e \longrightarrow group^{rlt}\ A\ e\ times \longrightarrow \\ (\forall x_\alpha \in A, y_\alpha \in A, z_\alpha \in A.\ times\ x\ y = times\ x\ z \longrightarrow y = z) \end{array}$$

Above, the relativization process was conceptually straightforward, in that we simply used relativized quantifiers. As we move to making relativized statements of a higher order, we need more invasive modifications. For example, left-inverse function uniqueness becomes:

$$(2')\quad \begin{array}{l} closed_2\ A\ times \ \wedge \ closed_0\ A\ e \longrightarrow group^{rlt}\ A\ e\ times \longrightarrow \\ (\forall inv_{\alpha\Rightarrow\alpha} \in A \Rightarrow A,\ inv'_{\alpha\Rightarrow\alpha} \in A \Rightarrow A. \\ \quad (\forall x_\alpha \in A.\ times\ (inv\ x)\ x = e) \wedge (\forall x_\alpha \in A.\ times\ (inv'\ x)\ x = e) \longrightarrow inv =^A inv') \end{array}$$

Highlighted are two less trivial changes to the original statement. First, the two functions inv and inv', which by their types send elements of α to elements of α, must be assumed to also belong to the set of functions that send elements of A to elements of A, i.e., of functions under which A is closed (which we denote by $A \Rightarrow A$, by analogy with the function-space type constructor). But this is not enough! Another modification is that we cannot conclude the strict equality of inv and inv', but only their equality on the elements of A, written $=^A$. So we see that, for items of a compound polymorphic type such as $\alpha \Rightarrow \alpha$, when relativizing the statements we must consider both a *subset* of the type, here $A \Rightarrow A$, as well as a *more abstract notion of equality* on it, here $=_A$.[5]

[5]Admittedly, saying that we *must* consider both subsets and abstract equalities may sound too strong. An alternative that avoids resorting to abstract equalities with the price of complicating the treatment of functions, and ultimately failing to properly cover container types, is discussed in our technical report.

This relativization infrastructure comes naturally for the built-in type of functions, but how about the other types, obtained via HOL type definitions? To use them in relativized statements, they must provide similar infrastructure. The third example statement involves lists, and its relativized form is

$$\begin{array}{ll} & closed_2\ A\ times\ \wedge\ closed_0\ A\ e \longrightarrow group^{rlt}\ A\ e\ times \longrightarrow \\ (3') & (\forall xs_{\alpha\ list} \Leftarrow list(A),\ ys_{\alpha\ list} \Leftarrow list(A). \\ & \quad fold\ times\ e\ (append\ xs\ ys) = times\ (fold\ times\ e\ xs)\ (fold\ times\ e\ ys)) \end{array}$$

Above, we overloaded the *list* notation to also denote the internalization to sets of the *list* type constructor, namely the operator of type $\alpha\ set \Rightarrow \alpha\ list\ set$ that takes any set A to the set of lists xs such that all the elements occurring in xs are in A.

Unlike in statement (2') where we needed a custom equality for functions, in statement (3'), for lists, we used bare equality. This is because we only had to deal with bare equality, which when lifted becomes again bare equality. But in general, when relativizing statements involving $\sigma\ list$ for some type σ (which could be, for instance, $\alpha \Rightarrow \alpha$), we would need to lift to $\sigma\ list$ whatever custom equality we reached for σ; in the case of lists, the natural lifting occurs componentwise.

Overall, we need a way to lift along type constructors both sets and "abstract equalities". Both these concepts are particular cases of PERs: an equivalence relation is obviously a PER, and a set A induces the PER $eqOf\ A$, read "the equality of A" (holding for two elements when they are equal and belong to A; not to be confused with $=^A$, used earlier for the equality of two functions when applied to A's elements). This suggests focusing on PER-based relativization as a generalization of set-based relativization. And if we also take into account that the liftings of equivalence relations along function spaces are not always equivalences but are always PERs, we are compelled to follow this suggestion.

3.2 What We Want from Relativization

We are ready to formulate our wish list for relativization. First, let's identify the relevant translations:

- We want to produce, from any HOL formula, or more generally from any HOL term t, a relativized counterpart $\mathsf{RLT}(t)$ that has the same type as t—so that, in particular, if t is a formula then $\mathsf{RLT}(t)$ is also a formula. In the recursive process of producing the relativized term, we must relativize all the constants occurring in the original term.
- Equality stood out as requiring special treatment, in that equality on a type σ should be mapped to a PER on σ. And the relativization of equality on compound types such as $\sigma\ list$ and $\sigma \Rightarrow \tau$ depends on that of their components (σ, or σ and τ). Therefore, whereas relativization initially proceeds structurally recursive on terms, when equality is encountered we must switch to structural recursion on types. So to separate concerns, we can speak about the *relational interpretation* of a type σ, denoted $\mathsf{RIN}(\sigma)$, which will be a PER on σ. The relativization of equality on σ, $\mathsf{RLT}(=_{\sigma\ rel})$, will be taken to be $\mathsf{RIN}(\sigma)$.

The main result that we are after is an Admissibility theorem: If the original type-based statements φ are provable in HOL, then their relativized versions, $\mathsf{RLT}(\varphi)$, are also provable in HOL under suitable relativization assumptions—let us denote these assumptions by Δ^φ. This allows us, for our working examples from Section 3.1, to conclude (1'), (2') or (3') as soon as we have proved (1), (2) or (3), while staying within the boundaries of HOL's axiomatic base. In these examples, φ is the original statement (x) (where x is 1, 2 or 3), Δ^φ is the first line of (x') (assuming closedness w.r.t. A) and $\mathsf{RLT}(\varphi)$ is the statement on the remaining lines. We will actually formulate Admissibility a bit more generally: We don't need to start with sets $A : \alpha\ set$, but can start with PERs $R : \alpha\ rel$, obtaining the theorem for sets as a particular case by taking R to be $eqOf(A)$.

While being the main goal, Admissibility is certainly not our only goal. We want relativization to "make sense", i.e., to satisfy some expected/desired properties. Importantly, it should satisfy what

we will refer to as Connective & Quantifier Suitability, i.e., behave as expected for the connectives and quantifiers. These are not primitive in HOL but are defined using λ-abstraction and equality; for example, $\forall x_\sigma.\ \varphi$ is defined as $(\lambda x_\sigma.\ \varphi) = (\lambda x_\sigma.\ \mathit{true})$. As it turns out, our choice to interpret equality on σ as the PER $\mathsf{RIN}(\sigma)$ frees us from having to do anything special when relativizing λ-abstraction, so we can simply set $\mathsf{RLT}(\lambda x_\sigma.\ \varphi)$ to be $\lambda x_\sigma.\ \mathsf{RLT}(\varphi)$. Indeed, for the case of universal quantification, $\mathsf{RLT}(\forall x_\sigma.\ \varphi)$ then becomes $\mathsf{RIN}(\sigma \Rightarrow \sigma)\ (\lambda x_\sigma.\ \mathsf{RLT}(\varphi))\ (\lambda x_\sigma.\ \mathit{true})$, which means $\forall x_\sigma, y_\sigma.\ \mathsf{RIN}(\sigma)\ x\ y \longrightarrow \mathsf{RLT}(\varphi)$ with y_σ fresh for φ. Since $\mathsf{RIN}(\sigma)$ will be (guaranteed to be) a PER, the last formula is equivalent to $\forall x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \longrightarrow \mathsf{RLT}(\varphi)$. And since $\mathsf{RIN}(\sigma)\ x\ x$ says that x is in the domain of $\mathsf{RIN}(\sigma)$, this is precisely what one wants from the relativization of $\forall$. Similarly, we obtain $\mathsf{RLT}(\exists x_\sigma.\ \varphi)$ provably equal to $\exists x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge \mathsf{RLT}(\varphi)$.

Another natural property we wish to have is Recoverability: Instantiating the relational parameters of a relativized term $\mathsf{RLT}(t)$ to be the equality relations should provide a term that is provably equal to the original term t. In particular, a relativized statement should be a generalization of the original statement, meaning that the converse of Admissibility should also hold.

The relational interpretation RIN is of course an incarnation of the types-as-relations interpretation paradigm [Reynolds 1983]. In fact, already implicit in the above discussion is that we want the relativized terms to belong to the domain of their types' relational interpretation: Given $t : \sigma$, we want that $\mathsf{RIN}(\sigma)\ \mathsf{RLT}(t)\ \mathsf{RLT}(t)$ is provable (in HOL) assuming a suitable relativization context, which fixes a PER variable for each type variable in t. This is a restricted form of parametricity, namely parametricity with respect to PERs. So relativization will essentially be a mechanism for turning terms and constants into counterparts that are parametric with respect to PERs. The items that are already parametric, such as *fold* and *append* in statement (3), will therefore not be affected by relativization: They persist unchanged in the relativized statement (3').

Lists, as well as many container types (such as sets, bags, trees of various kinds, etc.), can be naturally interpreted relationally [Hoogendijk and de Moor 2000; Traytel et al. 2012; Wadler 1989]. But what about types defined in HOL in general, via comprehension with predicates? We need to identify conditions under which a well-behaved relational interpretation, one that in particular brings well-behaved (including admissible) relativization, is possible for defined types.

Even for container types, which have a well-understood natural relational interpretation, a HOL-specific problem arises: In HOL, these types are not built-in, but are also defined via comprehension. This "humble", low-level origin can be forgotten by the users (and is usually hidden from the average users) after the high-level properties of container types (injectiveness and exhaustiveness of the constructors, induction and recursion principles, etc.) have been established. But for proving Admissibility we obviously cannot afford to forget it! So it is desirable that our notion of HOL relativization, which will be proved to be admissible, is compatible with the standard relational interpretation of container types—we call this property Compatibility. It would ensure that the expected relativization of formulas involving container types is itself admissible in HOL.

4 PER RELATIVIZATION IN HOL

This section will develop systematically the ideas explored in Section 3.2. We start by defining the syntactic operators that perform relativization, namely the relational interpretation operator RIN on types and the relativization operator RLT on terms (§4.1). Special care is required for the Hilbert choice constant and the defined types. For Hilbert choice, we must ensure that the relativized version is consistent with the original version (i.e., satisfies Recoverability) even outside its intended domain, where the argument predicate holds. For defined types, it turns out that a well-behaved relational interpretation is only possible for them if a certain favorable condition holds. Defined types τ are, as discussed, copies of subsets of existing types σ given by predicates $t : \sigma \Rightarrow \mathit{bool}$. The relational interpretation of σ, $\mathsf{RIN}(\sigma)$, and t's relativization, $\mathsf{RLT}(t)$, should therefore both play a

role in defining τ's relational interpretation. The two are connected by the property that, due to t's PER-parametricity, $\mathsf{RLT}(t)$ is compatible with $\mathsf{RIN}(\sigma)$, meaning that it can be regarded as operating on equivalence classes of $\mathsf{RIN}(\sigma)$. This suggests taking $\mathsf{RIN}(\tau)$ to be a copy on τ of the restriction of $\mathsf{RIN}(\sigma)$ to elements where $\mathsf{RLT}(t)$ is true—which only works provided τ is wide enough! Thus, to be able to extend the relational interpretation to defined types, we will need that their definitions satisfy a "wide typedness" property. Then we prove that the relativization operators satisfy most of Section 3.2's desired properties, including Admissibility, Connective & Quantifier Suitability, Recoverability and PER-Parametricity (§4.2). (The only desirable property that we do not prove at this stage is Compatibility, which we will address later, in Section 7.) Finally, we look back at our working examples from Section 3.1 and show that our PER relativization handles them correctly (§4.3).

4.1 The Relativization Operators

4.1.1 Supply of Fresh Relational Variables, and Relativization Contexts. For every type-variable α, we fix a variable $R^\alpha \in \mathit{Var}$. We assume that the variables R^α are mutually distinct (for any two distinct type-variables) and do not occur in the input terms to the operators we define. Usually, we will use these to form typed variables of type α rel, written $R^\alpha_{\alpha\ \mathsf{rel}}$; but we will occasionally also perform type-substitutions on them, obtaining typed variables such as $R^\alpha_{\sigma\ \mathsf{rel}}$. We will use these relation-variables to provide context for relativization, as described next.

Definition 7. For a type σ, let Δ^σ denote the context consisting of an assumption $\mathit{per}_{\neq\emptyset}\ R^\alpha_{\alpha\ \mathsf{rel}}$ for each $\alpha \in \mathsf{TV}(\sigma)$. For any term t, let Δ^t denote the context consisting of the following assumptions:

(1) $\mathit{per}_{\neq\emptyset}\ R^\alpha_{\alpha\ \mathsf{rel}}$ for each $\alpha \in \mathsf{TV}(t)$ (2) $\mathsf{RIN}(\sigma)\ x_\sigma\ x_\sigma$ for each $x_\sigma \in \mathsf{FTV}(t)$

For any context Γ, let Δ^Γ be $\bigcup_{\varphi\in\Gamma} \Delta^\varphi$. We call Δ^σ, Δ^t, and Δ^Γ the *relativization context* of σ, t, and Γ, respectively.

Note that, taking $\mathsf{TV}(\Gamma)$ and $\mathsf{FTV}(\Gamma)$ to be the unions of the corresponding operators applied to all formulas in Γ, we have that Δ^Γ consists of the following assumptions:

(1) $\mathit{per}_{\neq\emptyset}\ R^\alpha_{\alpha\ \mathsf{rel}}$ for each $\alpha \in \mathsf{TV}(\Gamma)$ (2) $\mathsf{RIN}(\sigma)\ x_\sigma\ x_\sigma$ for each $x_\sigma \in \mathsf{FTV}(\Gamma)$

Let $\Sigma = (K, \mathit{arOf}, \mathit{Const}, \mathit{tpOf})$ be a signature and D a definitional theory for this signature. All the concepts of this section are defined relative to Σ and D.

We let the term *frel* denote the following term of type α rel $\Rightarrow \beta$ rel $\Rightarrow (\alpha \Rightarrow \beta)$ rel:

$$\lambda P_{\alpha\ \mathsf{rel}}, Q_{\beta\ \mathsf{rel}}, \lambda f_{\alpha\Rightarrow\beta}, \lambda f'_{\alpha\Rightarrow\beta}.\ \forall x_\alpha, x'_\alpha.\ P\ x\ x' \longrightarrow Q\ (f\ x)\ (f'\ x')$$

Thus, *frel* P Q is the lifting of the relations P on α and Q on β to the function space $\alpha \Rightarrow \beta$. Reflecting this intuition, we will usually write $P \Rightarrow Q$ instead of *frel* P Q.

4.1.2 Defining the Relativization Operators Everywhere but on the Defined Types.

Definition 8. We define the operators $\mathsf{RIN} : \mathit{Type} \to \mathit{Term}$, where $\mathsf{RIN}(\sigma)$ is called the *relational interpretation* of the type σ, and $\mathsf{RLT} : \mathit{Term} \to \mathit{Term}$, where $\mathsf{RLT}(t)$ is called the *relativization* of the term t, recursively by the following clauses:

$$\mathsf{RIN}(\alpha) = R^\alpha_{\alpha\ \mathsf{rel}} \qquad \mathsf{RIN}(\sigma \Rightarrow \tau) = \mathsf{RIN}(\sigma) \Rightarrow \mathsf{RIN}(\tau)$$
$$\mathsf{RIN}(\mathit{bool}) = (=_{\mathit{bool}\ \mathsf{rel}}) \qquad \mathsf{RIN}(\mathit{ind}) = (=_{\mathit{ind}\ \mathsf{rel}}) \qquad \mathsf{RLT}(=_{\sigma\Rightarrow\sigma\Rightarrow\mathit{bool}}) = \mathsf{RIN}(\sigma)$$
$$\mathsf{RLT}(x_\sigma) = x_\sigma \qquad \mathsf{RLT}(t_1\ t_2) = \mathsf{RLT}(t_1)\ \mathsf{RLT}(t_2) \qquad \mathsf{RLT}(\lambda x_\sigma.\ t) = \lambda x_\sigma.\ \mathsf{RLT}(t)$$
$$\mathsf{RLT}(\mathit{choice}_{(\sigma\Rightarrow\mathit{bool})\Rightarrow\sigma}) = \lambda p_{\sigma\Rightarrow\mathit{bool}}.\ \mathit{if}\ (\exists x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge p\ x)$$
$$\mathit{then}\ (\mathit{Some}\ x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge p\ x)$$
$$\mathit{else}\ (\mathit{Some}\ x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge (\mathsf{RIN}(\sigma) \neq (=_{\sigma\ \mathit{rel}})))$$
$$\mathsf{RLT}(c_\tau) = \mathsf{RLT}(t'[\rho]) \quad \text{if } c \text{ is a non-built-in constant, } c_{\tau'} \equiv t' \text{ is in } D \text{ and } \tau \leq_\rho \tau'$$

Note that, in the definitional clause for defined constants c_τ, since $\mathsf{TV}(t') \subseteq \mathsf{TV}(c_{\tau'})$, the definition of RLT does not depend on the choice of ρ such that $\tau = \tau'[\rho]$.

The clauses of the above definition are mostly the expected ones, in light of our previously discussed design decisions. The clause for the choice operator is the only one requiring additional explanations. Recall that the *choice* function applied to a predicate $p_{\sigma \Rightarrow bool}$ returns an element of σ satisfying p provided such an element exists, and a completely arbitrary element otherwise. Consequently, it is natural to think that the relativization should return an element in $\mathsf{RIN}(\sigma)$'s domain satisfying p provided such an element exists, and an arbitrary element of $\mathsf{RIN}(\sigma)$'s domain otherwise. This explains the if-then-else structure of the clause and "almost works", in that it is compatible with our wish-list properties discussed in Section 3.2 except for Recoverability: We want that, taking $\mathsf{RIN}(\sigma)$ to be the equality on σ, the relativized choice function becomes provably equal to the (original) choice function—but this would not be true for the trivial case of the input predicate p being vacuously false, where we would get a mismatch between *Some* x_σ. *false* and *Some* x_σ. $\mathsf{RIN}(\sigma)\ x\ x$ (whose equality is not provable in HOL). This is a degenerate and somewhat pathological case—of using choice outside its intended domain—but nevertheless we must deal with it (unless we settle for well-behavedness of our relativization only in cases of meaningful application of choice, which is not ideal). It turns out we can restore Recoverability while maintaining the other properties (including PER-parametricity of relativized choice): The highlighted addition of the conjunct $\mathsf{RIN}(\sigma) \neq (=_{\sigma\ rel})$ (saying that $\mathsf{RIN}(\sigma)$ is not the equality relation) fixes this problem. Indeed, when $\mathsf{RIN}(\sigma)$ is not the equality relation the result stays the same; and when it is the equality relation, it again stays the same except for when the predicate is vacuously false, in which case it returns a choice of *false*, as desired.

4.1.3 Defining the Relativization Operators on the Defined Types. Assume the type τ is defined from the type σ via the predicate $t : \sigma \Rightarrow bool$. This means that D contains the type definition $\tau \equiv t$, which states that there exists a bijection *rep* between τ's elements and the elements of σ satisfying t.

To define $\mathsf{RIN}(\tau)$, the relational interpretation of τ, what we have at our disposal is the relational interpretation of σ, $\mathsf{RIN}(\sigma)$, and the relativization of t, $\mathsf{RLT}(t)$, which is a predicate on σ. Thus, given that τ is essentially a "copy" of (the collection of elements satisfying) t from σ, it would be natural to define $\mathsf{RIN}(\tau)$ as some sort of copy of $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$, the restriction of the PER $\mathsf{RIN}(\sigma)$ to $\mathsf{RLT}(t)$. In fact, this would not only be natural, but also mandatory given the properties we wish to prove for our relativization functions, in particular Admissibility: Since $\tau \equiv t$ is provable (from D, via (Fact)), we wish that $\mathsf{RLT}(\tau \equiv t)$ is also provable. And, if we factor in Connective & Quantifier Suitability, the latter says that there exists a bijection-up-to between $\mathsf{RIN}(\tau)$ and $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$. In conclusion, we want that $\mathsf{RIN}(\tau)$ and $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ are not necessarily isomorphic, but isomorphic up to their respective PER-induced "equalities", i.e., we want $\exists f_{\tau \Rightarrow \sigma}.\ bijUpto\ f\ \mathsf{RIN}(\tau)\ \mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ to hold.

Fig. 1 is a graphical illustration of this situation. The elements of type σ are shown as either bullets or crosses. The bullets are the elements in the domain of $\mathsf{RIN}(\sigma)$, and the separating lines show their partitioning into PER-classes (two elements are in the same class when they are related by $\mathsf{RIN}(\sigma)$). The crosses are elements outside the domain of $\mathsf{RIN}(\sigma)$. Among the bullets, the unfilled ones (∘) are those that satisfy the predicate $\mathsf{RLT}(t)$. Note that each PER-class has either all or none of its elements satisfying $\mathsf{RLT}(t)$, i.e., in the picture its elements are either all filled or all unfilled. This is by Admissibility: From the tautological property that t is compatible with the equality on σ, we must have that $\mathsf{RLT}(t)$ is compatible with $\mathsf{RIN}(\sigma)$.

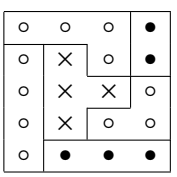


Fig. 1. A type σ, the PER relation $\mathsf{RIN}(\sigma) : \alpha\ rel$, and the predicate $\mathsf{RLT}(t) : \sigma \to bool$ (∘).

To produce such a suitable $\mathsf{RIN}(\tau)$, one may be tempted to make an isomorphic copy of $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ using some representation function *rep* (whose existence is guaranteed by $\tau \equiv t$). However, this is in general not possible, because the predicates t and $\mathsf{RLT}(t)$ are not guaranteed to

be related in any semantically meaningful way. In particular it is not generally the case that $t\ x$ implies $\mathsf{RLT}(t)\ x$ or vice versa; in fact, the sets of elements of σ satisfying these two predicates may even be disjoint. For example, in Fig. 1, t may well correspond to any collection of elements whatsoever—e.g., one red bullet, one black bullet and one cross. In short, the bijection *rep* between τ and σ provided by $\tau \equiv t$ cannot be used to copy $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ from σ to τ.

So how shall we define $\mathsf{RIN}(\tau)$ then? The answer is that *it does not matter, as long as some sanity properties hold*, namely:

- $\mathsf{RIN}(\tau)$ is a PER which becomes actual equality on τ if the relations R^α are assumed to be equalities on α for all type variables in α
- $\exists f_{\tau \Rightarrow \sigma}.\ \mathit{bijUpto}\ f\ \mathsf{RIN}(\tau)\ \mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ holds

This conclusion may be slightly perplexing, but it makes perfect sense once we reflect on what relativization is supposed to provide: By the nature of relativization, the relativized terms will only be considered in connection with domains and induced notions of equality given by the relational interpretations. This means that, when defining RIN for τ (just like for any other type), any two choices of PER will be equally acceptable if they are "isomorphic-up-to", i.e., connected by a bijection-up-to. And, as we have seen, we are forced to choose something in the isomorphic-up-to class of $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$. For the Fig. 1 example, $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ has three PER-classes (the ones containing red bullets), and therefore any definition of $\mathsf{RIN}(\tau)$ as a PER with three equivalence classes would do.

This having been said, for particular defined types τ, some choices of PERs can be more natural than others. And an important, somewhat empirical question is whether such natural choices are indeed isomorphic-up-to with $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$. We will study this question in depth in Section 7.

But here, abstractly, we want to allow ourselves the freedom to make any choice within the isomorphic-up-to constraint. That is, provided at least one choice exists! The following example shows that this may not always be the case:

Example 9. Consider the type definition $\alpha\ \kappa \equiv t$ for $t : \sigma \Rightarrow \mathit{bool}$, σ is $\mathit{ind} \Rightarrow \alpha$ and t states of its argument $u : \sigma$ that u is either surjective or is equal to $\lambda i.\ \mathit{Some}\ a_\alpha.\ \mathit{true}$, i.e., t is

$$\lambda\ u_\sigma.\ (\forall a.\ \exists i.\ u\ i = a)\ \vee\ u = (\lambda i.\ \mathit{Some}\ a_\alpha.\ \mathit{true})$$

For a contradiction, let us assume that RIN can be defined for $\alpha\ \kappa$ so that the discussed properties hold, in particular, under the assumption that $R^\alpha_{\alpha\ \mathsf{rel}}$ is a PER, we have that $\mathsf{RIN}(\alpha\ \kappa)$ is a PER and $\exists f_{\tau \Rightarrow \sigma}.\ \mathit{bijUpto}\ f\ \mathsf{RIN}(\alpha\ \kappa)\ \mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ holds. Let us fix a non-empty set $A : \alpha\ \mathit{set}$ and take $R^\alpha_{\alpha\ \mathsf{rel}}$ to be $\mathit{eqOf}(A)$. Then $\mathsf{RIN}(\sigma)$ is the relation $(=_{\mathit{ind\ rel}}) \Rightarrow \mathit{eqOf}(A)$, which is the same as $\lambda\ u, u'.\ \forall i, i'.\ i = i' \longrightarrow \mathit{eqOf}(A)\ (u\ i)\ (u'\ i')$, which is furthermore the same as $\lambda\ u, u'.\ \forall i.\ u\ i = u'\ i \wedge u\ i \in A$, which is finally the same as $\lambda\ u, u'.\ u = u' \wedge \forall i.\ u\ i \in A$.

Moreover, we have that $\mathsf{RLT}(t)$ is $\lambda\ u_\sigma.\ (\forall a \in A.\ \exists i.\ u\ i = a)\ \vee\ u = (\lambda\ i.\ \mathit{Some}\ a_\alpha.\ \mathit{eqOf}(A)\ a\ a)$ which is the same as $\lambda\ u_\sigma.\ (\forall a \in A.\ \exists i.\ u\ i = a)\ \vee\ u = (\lambda\ i.\ \mathit{Some}\ a_\alpha.\ a \in A)$.

Putting together the above two characterizations of $\mathsf{RIN}(\sigma)$ and $\mathsf{RLT}(t)$ (for the case when $R^\alpha_{\alpha\ \mathsf{rel}}$ is $\mathit{eqOf}(A)$), we obtain that $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ is the relation

$$\lambda\ u_\sigma, u'_\sigma.\ u = u' \wedge \forall i.\ u\ i \in A \wedge ((\forall a \in A.\ \exists i.\ u\ i = a)\ \vee\ u = (\lambda\ i.\ \mathit{Some}\ a_\alpha.\ a \in A))$$

which is the same as

$$\lambda\ u_\sigma, u'_\sigma.\ u = u' \wedge \mathit{image}\ u = A \vee u = (\lambda\ i.\ \mathit{Some}\ a_\alpha.\ a \in A)$$

where $\mathit{image}\ u$ denotes the image of u, namely $\{u\ i \mid i : \mathit{ind}\}$.

Thus, the PER-classes of $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ correspond bijectively to the set of all functions $u : \sigma$ that are either $\lambda i.\ \mathit{Some}\ a_\alpha.\ a \in A$ or have their image equal to A.

Now, let us further instantiate α to a large enough type, say $\mathit{ind} \Rightarrow \mathit{bool}$, and take $A : (\mathit{ind} \Rightarrow \mathit{bool})\ \mathit{set}$ to consist of two elements, say, the functions $\lambda i.\ \mathit{true}$ and $\lambda i.\ \mathit{false}$ (while still assuming

$R^{\alpha}_{(ind \Rightarrow bool)\ \mathsf{rel}}$ to be $eqOf(A)$). Since there are no surjective functions between ind and $ind \Rightarrow bool$, it follows that the corresponding instance of t is satisfied by only one element, namely $\lambda i.\ Some\ a_{ind \Rightarrow bool}.\ true$, which means that the corresponding instance of τ, namely $(ind \Rightarrow bool)\ \kappa$, is a singleton. On the other hand, there exist many functions $u : ind \Rightarrow (ind \Rightarrow bool)$ whose image is equal to A, meaning that the corresponding instance of $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ is not a singleton.

So, for the indicated instances, we have that $\alpha\ \kappa$ is a singleton whereas $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$ is not. We have thus reached a contradiction. Thus, $\alpha\ \kappa$ cannot have a suitable relational interpretation. □

As we can see, the reason why a suitable choice for $\mathsf{RIN}(\tau)$ may not exist is that τ is *not wide enough*, in other words, the extension $\{x_\sigma \mid t\ x\}$ of the defining predicate t is not wide enough (for certain instances). Let us make this precise. For any type σ, predicates $t : \sigma \Rightarrow bool$ and $s : \sigma \Rightarrow bool$, and PER $r : \sigma\ \mathsf{rel}$, we let $t \geq s/r$ denote the formula $\exists f_{\sigma \Rightarrow \sigma}.\ \forall y_\sigma.\ s\ y \longrightarrow \exists x_\sigma.\ t\ x \wedge r\ (f\ x)\ y$. Thus, $t \geq s/r$ says that the extension of (i.e., set of items satisfying) the predicate t is at least as large as the set of r-classes of s, i.e., as the extension of s quotiented by r. This cardinality relationship is witnessed by the function f, which is a surjection between the extension of t and that of s up to r.

Definition 10. A type definition in D of the form $\tau \equiv t$ where $t : \sigma \Rightarrow bool$ is called *wide* if

$$D; \Delta^\tau \vdash t \geq \mathsf{RLT}(t)/\mathsf{RIN}(\sigma)$$

Wide type definitions allow us to obtain a PER on the defined type τ satisfying the desirable properties. In fact, it is easy to see that the converse of the above also holds: If we can find a PER P that generalizes equality in such a way that there exists a bijection-up-to between P and $\mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}$, then the domain of P, in particular τ, hence the extension of t, must be at least as large as $\mathsf{RLT}(t)$ quotiented by $\mathsf{RIN}(\sigma)$.

We conclude that wide type definitions are what we want *and* need for hoping to interpret defined types relationally while retaining the main results about relativization (those from Section 4.2). As for the particular PER that acts as a relational interpretation of defined types, we prefer to not commit to any choice of such a PER, but parameterize our definition by an unspecified choice.

Definition 11. Given a type definition in D of the from $\tau \equiv t$ where $t : \sigma \Rightarrow bool$ and τ has the form $(\alpha_1, \ldots, \alpha_m)\kappa$, a *relational witness* for it is a ground term $relwit^\kappa$ of type $\alpha_1\ \mathsf{rel} \Rightarrow \ldots\ \alpha_m\ \mathsf{rel} \Rightarrow (\alpha_1, \ldots, \alpha_m)\kappa\ rel$ such that

$$\begin{array}{l} D \vdash (\bigwedge \Delta^\tau \longrightarrow per\ (relwit^\kappa\ R^{\alpha_1}_{\alpha_1\ \mathsf{rel}} \ldots R^{\alpha_m}_{\alpha_m\ \mathsf{rel}})) \wedge \\ \quad relwit^\kappa (=_{\alpha_1\ \mathsf{rel}}) \ldots (=_{\alpha_m\ \mathsf{rel}}) = (=_{\tau\ \mathsf{rel}}) \wedge \\ \quad (\bigwedge \Delta^\tau \longrightarrow \exists f_{\tau \Rightarrow \sigma}.\ bijUpto\ f\ (relwit^\kappa\ R^{\alpha_1}_{\alpha_1\ \mathsf{rel}} \ldots R^{\alpha_m}_{\alpha_m\ \mathsf{rel}})\ \mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}) \end{array}$$

Note that we have $\Delta^t \subseteq \Delta^\tau$ and Δ^τ consists of the assumptions $per_{\neq \emptyset}\ R^{\alpha_1}_{\alpha_1\ \mathsf{rel}}, \ldots, per_{\neq \emptyset}\ R^{\alpha_m}_{\alpha_m\ \mathsf{rel}}$.

A wide type definition always has a relational witness given by the choice operator, to which we will refer as the *default relational witness*:

Prop 12. Given a wide type definition $\tau \equiv t$ in D where $t : \sigma \Rightarrow bool$ and τ has the form $(\alpha_1, \ldots, \alpha_m)\kappa$, the following is a relational witness for it:

$$\begin{array}{l} Some\ P_{\alpha_1\ \mathsf{rel} \Rightarrow \ldots \Rightarrow \alpha_m\ \mathsf{rel} \Rightarrow \tau\ \mathsf{rel}}.\ \forall R^{\alpha_1}_{\alpha_1\ \mathsf{rel}}, \ldots, R^{\alpha_m}_{\alpha_m\ \mathsf{rel}}. \\ \quad (\bigwedge \Delta^\tau \longrightarrow per\ (P\ R^{\alpha_1}_{\alpha_1\ \mathsf{rel}} \ldots R^{\alpha_m}_{\alpha_m\ \mathsf{rel}})) \wedge \\ \quad P\ (=_{\alpha_1\ \mathsf{rel}}) \ldots (=_{\alpha_m\ \mathsf{rel}}) = (=_{\tau\ \mathsf{rel}}) \wedge \\ \quad (\bigwedge \Delta^\tau \longrightarrow \exists f_{\tau \Rightarrow \sigma}.\ bijUpto\ f\ (P\ R^{\alpha_1}_{\alpha_1\ \mathsf{rel}} \ldots R^{\alpha_m}_{\alpha_m\ \mathsf{rel}})\ \mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)}) \end{array}$$

Our results will apply to definitional theories whose types are wide, and which optionally come with relational witnesses for some of their type definitions—for those with no provided relational witnesses, we can use the default witnesses.

Definition 13. A definitional theory D is said to be *widely-typed* if all its type definitions $(\alpha_1, \ldots, \alpha_m)\kappa \equiv t$ in D are wide and come with (possibly default) relational witnesses $relwit^\kappa$.

Now we are on solid ground for extending the relational interpretation to defined types:

Definition 14. Assuming D is widely-typed, we extend Def. 8 with a clause for defined types as follows. Assume $(\alpha_1, \ldots, \alpha_m)\kappa \equiv t$ is a type definition in D, and let $(\sigma_1, \ldots, \sigma_m)\kappa$ be an instance of $(\alpha_1, \ldots, \alpha_m)\kappa$. We define

$$\mathsf{RIN}((\sigma_1, \ldots, \sigma_m)\kappa) = relwit^\kappa[\sigma_1/\alpha_1, \ldots, \sigma_m/\alpha_m]\ \mathsf{RIN}(\sigma_1) \ldots\ \mathsf{RIN}(\sigma_m)$$

The definitions of RLT and RIN are mutually recursive. The reason why these definitions are correct, i.e., terminating (thus defining two total functions) is that they combine structural recursion and well-founded recursion along the definitional dependency relation—our technical report gives details.

4.2 Properties of Relativization

For the following results, we work in the same setting as in Section 4.1—namely with a fresh supply of relation-variables R^α, assumed (in relativization contexts) to denote nonempty PERs.

The next proposition states that the relational interpretations (which essentially "lift" those basic PERs to entire types σ) are also PERs.

Prop 15. For all types σ, we have $D; \Delta^\sigma \vdash per\ \mathsf{RIN}(\sigma)$.

Relativization behaves as desired with respect to the connectives and quantifiers:

Theorem 16. **(Connective & Quantifier Suitability)** The following hold for all formulas φ and ψ and typed variables x_σ:
(1) $D \vdash \mathsf{RLT}(true) = true$ and $D \vdash \mathsf{RLT}(false) = false$,
(2) $D \vdash \mathsf{RLT}(\neg\, \varphi) = \neg\, \mathsf{RLT}(\varphi)$, $D \vdash \mathsf{RLT}(\varphi \wedge \psi) = (\mathsf{RLT}(\varphi) \wedge \mathsf{RLT}(\varphi))$,
$D \vdash \mathsf{RLT}(\varphi \vee \psi) = (\mathsf{RLT}(\varphi) \vee \mathsf{RLT}(\varphi))$, and $D \vdash \mathsf{RLT}(\varphi \longrightarrow \psi) = (\mathsf{RLT}(\varphi) \longrightarrow \mathsf{RLT}(\varphi))$,
(3) $D; \Delta^\sigma \vdash \mathsf{RLT}(\forall x_\sigma.\ \varphi) = (\forall x_\sigma.\ \mathsf{RIN}(\sigma)\ x_\sigma\ x_\sigma \longrightarrow \mathsf{RLT}(\varphi))$, and
(4) $D; \Delta^\sigma \vdash \mathsf{RLT}(\exists x_\sigma.\ \varphi) = (\exists x_\sigma.\ \mathsf{RIN}(\sigma)\ x_\sigma\ x_\sigma \wedge \mathsf{RLT}(\varphi))$.

The relational interpretation $\mathsf{RIN}(\sigma)$ can recover the notion of equality on the original type, $=_{\sigma\ \mathrm{rel}}$, by substituting equality for the relational variables associated to type-variables of the σ; and, in a similar way, relativization $\mathsf{RLT}(t)$ can recover the original term t:

Theorem 17. **(Recoverability)** The following hold for all types σ, typable terms t and term-substitutions δ such that $\mathrm{supp}(\delta) \subseteq \{R^\alpha_{\alpha\ \mathrm{rel}} \mid \alpha \in TVar\}$ and $\delta(R^\alpha_{\alpha\ \mathrm{rel}}) = (=_{\alpha\ \mathrm{rel}})$ for all $\alpha \in \mathsf{TV}(\sigma)$:
(1) $D \vdash \mathsf{RIN}(\sigma)[\delta] = (=_{\sigma\ \mathrm{rel}})$ and (2) $D \vdash \mathsf{RLT}(t)[\delta] = t$.

The relational interpretation of a type is non-empty:

Prop 18. **(Nonemptiness)** For all types σ, we have $D; \Delta^\sigma \vdash \exists\, x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x$.
Hence (thanks to Prop. 15), we also have $D; \Delta^\sigma \vdash per_{\neq\emptyset}\ \mathsf{RIN}(\sigma)$.

We also have a restricted form of parametricity: Relativized terms are related to themselves via the relational interpretation of their type. Since the starting relations (on the type variables) are assumed to be PERs, we call this PER-parametricity.

Theorem 19. **(PER-Parametricity)** For all types σ and terms t, if $t : \sigma$, then

$$D; \Delta^t \vdash \mathsf{RIN}(\sigma)\ \mathsf{RLT}(t)\ \mathsf{RLT}(t).$$

Here is more context about PER-parametricity. This is a restriction of the concept of parametricity from relations between two types to relations on a single type that are additionally assumed to be PERs. While expressing general parametricity requires a richer infrastructure (see our technical report), PER-parametricity can be immediately expressed using our infrastructure: A term t of type σ is called *PER-parametric* when $D; \Delta^t \vdash \mathsf{RIN}(\sigma)\ t\ t$. So the above result states that the relativized terms $\mathsf{RLT}(t)$ are PER-parametric—in an apparently stronger form, using a smaller relativization context, Δ^t, instead of $\Delta^{\mathsf{RLT}(t)}$. Indeed, Δ^t is smaller because t has fewer free variables than $\mathsf{RLT}(t)$, which additionally contains some relational variables $R^{\alpha}_{\alpha\ rel}$ for $\alpha \in \mathsf{TV}(\sigma)$. But this alternative statement of the PER-parametricity of $\mathsf{RLT}(t)$ is actually equivalent to the original. Indeed, $\Delta^{\mathsf{RLT}(t)}$ only differs from Δ^t in some assumptions of the form $\mathsf{RIN}(\alpha\ rel\ rel)\ R^{\alpha}_{\alpha\ rel}\ R^{\alpha}_{\alpha\ rel}$, which follow from the assumptions that $R^{\alpha}_{\alpha\ rel}$ are PERs. In short, we have $\vdash (\bigwedge \Delta^t) \longleftrightarrow (\bigwedge \Delta^{\mathsf{RLT}(t)})$, meaning that using Δ^t has the same effect as using $\Delta^{\mathsf{RLT}(t)}$.

In preparation for our main result, we also need a substitutivity property for contexts. Given a context Γ and a type-substitution ρ, $\Gamma[\rho]$ is defined (as one would expect) to be $\{\varphi[\rho] \mid \varphi \in \Gamma\}$; and similarly for term-substitutions. Moreover, given two contexts Γ and Γ', we write $D; \Gamma \vdash \Gamma'$ to express that $D; \Gamma \vdash \varphi'$ for all $\varphi' \in \Gamma$. Finally, for a type-substitution ρ such that $\mathsf{supp}(\rho) = \{\alpha_1, \ldots, \alpha_n\}$, we define its associated term-substitution $\rho^{\#}$ to be $\mathsf{RIN}(\rho(\alpha_1))/R^{\alpha_1}_{\rho(\alpha_1)\ \mathsf{rel}}, \ldots, \mathsf{RIN}(\rho(\alpha_n))/R^{\alpha_n}_{\rho(\alpha_n)\ \mathsf{rel}}$.

Prop 20. **(Context Substitutivity)** For all contexts Γ and type-substitutions ρ, we have

$$D, \Delta^{\Gamma[\rho]} \vdash \Delta^{\Gamma}[\rho][\rho^{\#}].$$

Our main theorem states that, provided a formula is deducible, its relativization is also deducible in the relativized context. Given a context Γ, we write $\mathsf{RLT}(\Gamma)$ for $\{\mathsf{RLT}(\psi) \mid \psi \in \Gamma\}$.

Theorem 21. **(Admissibility)** For all contexts Γ and formulas φ, if $D; \Gamma \vdash \varphi$, then

$$D; \Delta^{\Gamma \cup \{\varphi\}}, \mathsf{RLT}(\Gamma) \vdash \mathsf{RLT}(\varphi).$$

Thanks to the Recoverability theorem, the converse of Admissibility also holds, in the strong form stating that the relativized formula is a more general statement than the original formula:

Theorem 22. **(Generality)** For all formulas φ, we have that

$$D \vdash (\forall_{\alpha \in \mathsf{TV}(\varphi)} R^{\alpha}_{\alpha\ rel}.\ (\bigwedge \Delta^{\varphi}) \longrightarrow \mathsf{RLT}(\varphi)) \longrightarrow \varphi.$$

In particular, $D \vdash (\bigwedge \Delta^{\varphi}) \longrightarrow \mathsf{RLT}(\varphi)$ implies $D \vdash \varphi$.

Above, $\forall_{\alpha \in \mathsf{TV}(\varphi)} R^{\alpha}_{\alpha\ rel}$ is a sequence of universal quantifications, one over $R^{\alpha}_{\alpha\ rel}$ for each $\alpha \in \mathsf{TV}(\varphi)$.

Our technical report gives detailed proofs of the above theorems. They proceed either by structural induction on terms and types or by well-founded induction on the dependency relation induced by constant and type definitions. The proof of Admissibility makes use of most of the other results.

4.3 Working Examples Revisited

Let us now look at some of the working examples from Section 3.1 in light of our PER-relativization process. So *group* denotes the term $\lambda\ times_{\alpha \Rightarrow \alpha \Rightarrow \alpha},\ e_{\alpha}.\ \psi$, where ψ is the following formula:

$$\begin{array}{l}(\forall x_{\alpha}, y_{\alpha}, z_{\alpha}.\ times\ (times\ x\ y)\ z = times\ x\ (times\ y\ z)) \wedge \\ (\forall x_{\alpha}.\ times\ x\ e = x\ \wedge\ times\ e\ x = x) \wedge \\ (\forall x_{\alpha}.\ \exists y_{\alpha}.\ times\ x\ y = e\ \wedge\ times\ y\ x = e)\end{array}$$

Thus, α is a type variable and *group* has type $(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow bool$.

Let us now apply the definitions and results on relativization from the previous sections. In the theory, we wrote R^α for the unique variable associated to the type variables α—here, we will write R instead of R^α. Using the definition of RLT for λ-expressions, we have that $\mathsf{RLT}(\mathit{group})$ is the term $\lambda \mathit{times}_{\alpha\Rightarrow\alpha\Rightarrow\alpha},\, e_\alpha.\ \mathsf{RLT}(\psi)$. Moreover, using the Connective & Quantifier Suitability property together with the definition of relativization of equality (namely $\mathsf{RLT}\ (=_{\sigma\Rightarrow\sigma\Rightarrow \mathit{bool}}) = \mathsf{RIN}(\sigma)$) and the definition of relational interpretation for type variables (namely $\mathsf{RIN}(\alpha) = R^\alpha$), we have that $\mathsf{RLT}(\psi)$ (and also $\mathsf{RLT}(\mathit{group}\ \mathit{times}\ e)$) is HOL-provably equivalent to the following formula:

$$(\forall x_\alpha, y_\alpha, z_\alpha.\ R\ x\ x \wedge R\ y\ y \wedge R\ z\ z \longrightarrow R\ (\mathit{times}\ (\mathit{times}\ x\ y)\ z)\ (\mathit{times}\ x\ (\mathit{times}\ y\ z))) \ \wedge$$
$$(\forall x_\alpha.\ R\ x\ x \longrightarrow R\ (\mathit{times}\ x\ e)\ x \ \wedge\ R\ (\mathit{times}\ e\ x)\ x) \ \wedge$$
$$(\forall x_\alpha.\ R\ x\ x \longrightarrow \exists y_\alpha.\ R\ y\ y \ \wedge\ R\ (\mathit{times}\ x\ y)\ e \ \wedge\ R\ (\mathit{times}\ y\ x)\ e)$$

The case of set relativization (which is the one discussed in Section 3.1) can be obtained as a particular case of our PER-relativization, considering set variables $A : \alpha\ \mathit{set}$ and instantiating the relational variable R to be $\mathit{eqOf}(A)$. Indeed, it is easy to see that

$$\vdash \forall A, \mathit{times}, e.\ \mathsf{RLT}(\mathit{group})\ A\ \mathit{times}\ e \ \longleftrightarrow\ \psi'$$

(i.e., the shown statement is HOL-provable), where ψ' is the set-relativized term from Section 3.1:

$$(\forall x_\alpha \in A, y_\alpha \in A, z_\alpha \in A.\ \mathit{times}\ (\mathit{times}\ x\ y)\ z = \mathit{times}\ x\ (\mathit{times}\ y\ z)) \ \wedge$$
$$(\forall x_\alpha \in A.\ \mathit{times}\ x\ e = x \ \wedge\ \mathit{times}\ e\ x = x) \ \wedge$$
$$(\forall x_\alpha \in A.\ \exists y_\alpha \in A.\ \mathit{times}\ x\ y = e \ \wedge\ \mathit{times}\ y\ x = e)$$

Let us now revisit one of the theorems about groups discussed in Section 3.1, namely left-cancellation (example (1)). Formally, say D is a definitional theory (any would work here, including the empty one). We have $D \vdash \varphi$ where φ is the formula

$$\mathit{group}\ \mathit{times}\ e \ \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha.\ \mathit{times}\ x\ y = \mathit{times}\ x\ z \longrightarrow y = z)$$

Then, using again the Connective & Quantifier Suitability property together with the definitions of relativization and relational interpretation, we have that $\mathsf{RLT}(\varphi)$ is HOL-provably equal to

$$\mathsf{RLT}(\mathit{group})\ R\ e\ \mathit{times} \ \longrightarrow$$
$$(\forall x_\alpha, y_\alpha, z_\alpha.\ R\ x\ x \wedge R\ y\ y \wedge R\ z\ z \longrightarrow R\ (\mathit{times}\ x\ y)\ (\mathit{times}\ x\ z) \longrightarrow R\ y\ z)$$

The Admissibility theorem applied to this case (where the context Γ is empty) gives us

$$D; \Delta^{\{\varphi\}} \vdash \mathsf{RLT}(\varphi)$$

where the relativization context $\Delta^{\{\varphi\}}$ consists of the following, factoring in the fact that $\mathsf{RIN}(\alpha \Rightarrow \alpha \Rightarrow \alpha)$ is $R \Rightarrow R \Rightarrow R$:

- the assumption $\mathit{per}_{\neq\emptyset}\ R$ (corresponding to the only type variable in φ, namely α)
- the assumptions $(R \Rightarrow R \Rightarrow R)\ \mathit{times}\ \mathit{times}$ and $R\ e\ e$ (corresponding to the two free variables in φ, namely *times* and *e*)

In other words, Admissibility gives us:

$$D; \mathit{per}_{\neq\emptyset}\ R,\ (R \Rightarrow R \Rightarrow R)\ \mathit{times}\ \mathit{times},\ R\ e\ e \ \vdash \mathsf{RLT}(\varphi)$$

or, converting this to a context-free version:

$$D \vdash \mathit{per}_{\neq\emptyset}\ R \ \wedge\ (R \Rightarrow R \Rightarrow R)\ \mathit{times}\ \mathit{times} \ \wedge\ R\ e\ e \longrightarrow \mathsf{RLT}(\varphi)$$

Expanding $\mathsf{RLT}(\varphi)$, we obtain:

$$D \vdash \mathit{per}_{\neq\emptyset}\ R \ \wedge\ (R \Rightarrow R \Rightarrow R)\ \mathit{times}\ \mathit{times} \ \wedge\ R\ e\ e \longrightarrow \mathsf{RLT}(\mathit{group})\ R\ e\ \mathit{times} \longrightarrow$$
$$(\forall x_\alpha, y_\alpha, z_\alpha.\ R\ x\ x \wedge R\ y\ y \wedge R\ z\ z \longrightarrow R\ (\mathit{times}\ x\ y)\ (\mathit{times}\ x\ z) \longrightarrow R\ y\ z)$$

The set-based relativized statement (labeled (1') in Section 3.1) is a consequence of the above PER-based statement, by considering sets A and taking R to be $eqOf\ (A)$. In this case, $(R \Rightarrow R \Rightarrow R)\ times$ and $R\ e\ e$ become equivalent to what in Section 3.1 we denoted by $closed_2\ A\ times$ and $closed_0\ A\ e$.

To keep the terms small, consider the first conjunct ψ_1 of the formula ψ introduced above; namely $\forall x_\alpha, y_\alpha, z_\alpha.\ times\ (times\ x\ y)\ z = times\ x\ (times\ y\ z)$. Then $\mathsf{RLT}(\psi_1)$ is the corresponding subformula of ψ': $\forall x_\alpha, y_\alpha, z_\alpha.\ R\ x\ x \wedge R\ y\ y \wedge R\ z\ z \longrightarrow R\ (times\ (times\ x\ y)\ z)\ (times\ x\ (times\ y\ z))$.

Applying the Recoverability theorem to the term *group* and the term-substitution ρ that sends R to $=_{\alpha\ \mathsf{rel}}$, we obtain $D \vdash \mathsf{RLT}(\psi_1)[\rho] = \psi_1$, which is

$$\begin{aligned} D \vdash\ & (\forall x_\alpha, y_\alpha, z_\alpha.\ x = x \wedge y = y \wedge z = z \longrightarrow times\ (times\ x\ y)\ z = times\ x\ (times\ y\ z)) \\ =\ & (\forall x_\alpha, y_\alpha, z_\alpha.\ times\ (times\ x\ y)\ z = times\ x\ (times\ y\ z)) \end{aligned}$$

Recoverability is a special case of the aforementioned process of instantiating the PERs to sets, taking the instantiating sets to be the entire types. This discussion also illustrates why a relativized formula such as $\mathsf{RLT}(\psi)$ or $\mathsf{RLT}(\psi_1)$ represents a more general statement than ψ or ψ_1 (precisely because the latter can be obtained from the former by instantiating R with $=_\alpha$), which is what Generality asserts.

5 ADMISSIBILITY OF LOCAL TYPE DEFINITIONS

As mentioned in the introduction, Local Typedef, a rule for enabling local definitions, has been proposed as an extension of HOL's axiomatic basis in order to enable relativization.

The Local Typedef (LT) rule is the following:

$$\frac{D; \Gamma \vdash A \neq \emptyset \qquad D; \Gamma \vdash (\exists rep_{\beta \Rightarrow \sigma}.\ \mathsf{typedef}_{\beta,\sigma,A,rep}) \longrightarrow \varphi}{D; \Gamma \vdash \varphi}\ \text{(LT)}\ [\beta \text{ fresh for } \Gamma, A, \varphi]$$

It provides the local assumption that there is a type β isomorphic to a nonempty set A. The condition that β is fresh for A prevents the introduction of a dependent type (since A may have term variables).

The Local Typedef rule expresses in a roundabout fashion the property that, for every nonempty set A, there exists a type β that is equipollent to it, in that there exists a bijection between them. In a more expressive logic, where *existential* quantification over types is allowed, this could be a mere axiom: $\forall A_{\alpha\ set}.\ A \neq \emptyset \longrightarrow \exists\ \beta.\ \exists rep_{\beta \Rightarrow \alpha}.\ \mathsf{typedef}_{\beta,\alpha,A,rep}$. Indeed, one can see that applying the Local Typedef rule has the same effect as using this axiom in conjunction with the following standard elimination rules (in this order): $\forall$ elimination for type variables, $\longrightarrow$ elimination, $\forall$ elimination for term variables, and $\exists$ elimination for type variables. Kunčar and Popescu [2019] give more details.

We have shown that relativization, which was something that Local Typedef helped to achieve, is admissible. Now we can affirm to the admissibility of Local Typedef itself, which has been open:

Theorem 23. The Local Typedef rule is admissible for widely-typed definitional theories. More precisely, let D be a widely-typed definitional theory, Γ a context, σ a type, A a term of type $\sigma\ set$, φ a formula, and β a type-variable that is fresh for Γ, A, φ. Assume $D; \Gamma \vdash A \neq \emptyset$ and $D; \Gamma \vdash (\exists rep_{\beta \Rightarrow \sigma}.\ \mathsf{typedef}_{\beta,\sigma,A,rep}) \longrightarrow \varphi$. Then $D; \Gamma \vdash \varphi$.

This is proved by first applying the Admissibility theorem to

$$D; \Gamma \vdash (\exists rep_{\beta \Rightarrow \sigma}.\ \mathsf{typedef}_{\beta,\sigma,A,rep}) \longrightarrow \varphi$$

and then performing the following substitutions:

- $=_{\alpha\ \mathsf{rel}}$ for $R^{\alpha}_{\alpha\ \mathsf{rel}}$ for all type variables in Γ, A, ψ (which are known not to contain β)
- σ for β, which in particular will turn $R^{\beta}_{\beta\ \mathsf{rel}}$ into $R^{\beta}_{\sigma\ \mathsf{rel}}$
- the equality on A for $R^{\beta}_{\sigma\ \mathsf{rel}}$

Thanks to the freshness assumptions, and using the Recoverability theorem, we obtain that the relativized and substituted version of the above is equivalent to

$$D; \Gamma \vdash (\exists rep_{\sigma \Rightarrow \sigma}.\ \chi) \longrightarrow \varphi$$

where χ states that *rep* is an endo-bijection on A. Since $\exists rep_{\sigma \Rightarrow \sigma}.\ \chi$ is a tautology (taking *rep* to be $\lambda x_\sigma.\ x$), we can drop it. We obtain $D; \Gamma \vdash \varphi$, as desired. (Our technical report gives a detailed proof.)

In conclusion, for the widely-typed definitional theories, relativization allowed us to eliminate the typedef assumption from the Local Typedef rule, thus trivializing it—which shows its admissibility. We do not yet have a counterexample showing that Local Typedef fails to be admissible for a definitional theory that is not widely-typed. It may well be the case that a partial relativization version of our admissibility result for PER-relativization, which avoids (i.e., leaves untouched) a selection of the type variables, could still do the job and prove the admissibility of Local Typedef under milder conditions. We will investigate this in the future.

6 RELATIVIZATION FOR EXTENSIONS AND VARIATIONS OF HOL

In this section, we sketch a way to extend our results to allow declared-only constants (and consequently defined types depending on them) (§ 6.1), and also overloaded definitions of instances of defined constants, as featured by Isabelle/HOL (§6.2).

We first extend the notion of a definitional theory to incorporate constant declarations.

Definition 24. D is said to be an *extended definitional theory* if $D = \{def_1, \ldots, def_n\}$, where each def_i is a (type or constant) definition of the form $u_i \equiv t_i$, and there exist the signatures $\Sigma^1, \ldots, \Sigma^n$ and $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$ such that $\Sigma_0 = \Sigma_{\mathsf{init}}$, $\Sigma_n = \Sigma$ and the following hold for all $i \in \{1, \ldots, n\}$:

(1) $t_i \in Term_{\Sigma^i}$ and Σ_i is the extension of Σ_{i-1} with a fresh item defined by def_i, namely:
  (1.1) If u_i has the form $(\alpha_1, \ldots, \alpha_m)\kappa$, then $\kappa \notin \Sigma_{i-1}$ and $\Sigma_i = \Sigma_{i-1} \cup \{(\kappa, m)\}$
  (1.2) If u_i has the form c_σ, then $c \notin \Sigma^i$ and $\Sigma_i = \Sigma^i \cup \{(c, \sigma)\}$
(2) If def_i is a type definition, meaning u_i is a type and $t_i : \sigma \Rightarrow bool$, then $\{def_1, \ldots, def_{i-1}\} \vdash_{\Sigma_{i-1}} \exists x_\sigma.\ t_i\ x$
(3) Σ^i is an extension of Σ_{i-1} with a (possibly empty collection of) constants.

Note that the difference from Definition 4 of definitional theories are the intermediate signatures Σ^i that introduce declarations, whereas Σ_i continue to introduce definitions. The notion of dependency relation associated to a definitional theory remains the same for extended definitional theories. In what follows, we fix an extended definitional theory $D = \{def_1, \ldots, def_n\}$.

6.1 Relativizing Declared-Only Constants

To relativize a constant that is declared-only (i.e., such that D does not contain a definition for it) we only have two options to contemplate. First, we can think of relativizing them into themselves, but then properties such as PER-Parametricity cannot be proved to hold. The other option, which is the one we will follow, dwells on the idea that any instance of a declared-only constant really behaves like a variable—so we can relativize the constant to a variable, and assume the corresponding instance of PER-Parametricity in the relativization context.

For every constant c in Σ that is declared-only, let us fix a variable $z^c \in Var$. Similarly to the relational variables R^α, we will assume that the variables z^c are mutually distinct (for any two distinct constants) and do not occur in the terms we consider as input to the operators on types and terms that we will define. We will also assume, of course, that they are distinct from the variables R^α. We will use these to form typed variables of type τ, written z^c_τ, where τ is an instance of σ such that D does not contain a definition with lefthand side c_τ. We plan to use these variables to replace any undefined instances of constants during the relativization process.

To produce the right variables z^c in the relativization context of a term t, we need to know which are those declared-only constants that will be reached in the process of relativizing t. These are exactly those on which t depends definitionally. And since, due to type definitions, the type relational interpretation also depends on term relativization, we need to consider the definitional dependencies of types as well. To this end, we define:

- for any type σ, its set of dependencies $Deps(\sigma)$ to be $\{u \in Type^\bullet_\Sigma \cup CInst^\bullet_\Sigma \mid \text{there exists } v \in types^\bullet(\sigma) \text{ such that } v \rightsquigarrow^{\downarrow *} u\}$
- similarly, for any term t, its set of dependencies $Deps(t)$ to be $\{u \in Type^\bullet_\Sigma \cup CInst^\bullet_\Sigma \mid \text{there exists } v \in types^\bullet(t) \cup cinsts^\bullet(t) \text{ such that } v \rightsquigarrow^{\downarrow *} u\}$

where $\rightsquigarrow^{\downarrow *}$ is the transitive closure of $\rightsquigarrow^{\downarrow}$ (the substitutive closure of the definitional dependency relation, introduced in Section 2.8).

If u is either a type or a term, we define its *set of declared-only dependencies* $DeclaredOnlyDeps(u)$ to be $\{c_\tau \in Deps(u) \mid c \text{ is a declared-only constant}\}$. The definition also extends to contexts Γ, taking $DeclaredOnlyDeps(\Gamma)$ to be $\bigcup_{\varphi \in \Gamma} DeclaredOnlyDeps(\varphi)$. Thus, the declared-only dependencies are those items on which our given item depends and which have no further dependencies, i.e., are leaves in the direct-dependency tree.

Definition 25. We extend Def. 7 of the relativization contexts Δ^u where u is either a type, or a term, or a context, to also include assumptions $\mathsf{RIN}(\tau)\ z^c_\tau\ z^c_\tau$ for all $c_\tau \in DeclaredOnlyDeps(u)$.

Definition 26. We extend Def. 8 with a clause for declared-only constants as follows. Assume c is a declared-only constant of type σ such that $\tau \leq \sigma$. We define $\mathsf{RIN}(c_\tau) = z^c_\tau$.

These extended definitions remain correct:

Prop 27. The functions RIN and RLT are well-defined as total functions in $Type \to Term$ and $Term \to Term$.

The notion of widely-typed definitional theory remains the same. In what follows, we assume D to be a widely-typed definitional theory. With these preparations, we can generalize Section 4's results to HOL enriched with-declared-only constants. (Our technical report gives more details.)

Theorem 28. Then all the results from Section 4.2, including Connective & Quantifier Suitability, Recoverability, Nonemptiness, PER-Parametricity, Admissibility and Generality, still hold in this more general context (for any widely-typed definitional theory D), provided that:

- For both points in the Recoverability theorem, say letting u denote either σ or t, we amend one assumption as highlighted below: $\mathsf{supp}(\delta) \subseteq \{R^\alpha_{\alpha\ \mathsf{rel}} \mid \alpha \in TVar\} \cup \{z^\tau_c \mid c_\tau \in DeclaredOnlyDeps(u)\}$ and add the assumption that $\delta(z^\tau_c) = c_\tau$ for all $c_\tau \in DeclaredOnlyDeps(u)$.
- One of the conclusions of the Generality theorem is amended as highlighted below: $D \vdash (\forall_{\alpha \in \mathsf{TV}(\varphi)} R^\alpha_{\alpha\ rel}.\ \forall_{c_\sigma \in DeclaredOnlyDeps(\varphi)} z^c_\sigma.\ (\bigwedge \Delta^\varphi) \longrightarrow \mathsf{RLT}(\varphi)) \longrightarrow \varphi$.

Note also that now Δ^φ contains more assumptions, so $\bigwedge \Delta^\varphi$, which is used in both the context-free version of Admissibility and in Generality, now denotes:

$$(\bigwedge_{\alpha \in \mathsf{TV}(\varphi)} per_{\neq \emptyset}\ R^\alpha_{\alpha\ rel}) \wedge (\bigwedge_{x_\sigma \in \mathsf{FTV}(\varphi)} \mathsf{RIN}(\sigma)\ x_\sigma\ x_\sigma) \wedge (\bigwedge_{c_\sigma \in DeclaredOnlyDeps(\varphi)} \mathsf{RIN}(\sigma)\ z^c_\sigma\ z^c_\sigma)$$

6.2 Relativization for Overloaded Constants

Definition 29. Given a non-built-in constant c, a type $\sigma \leq tpOf(c)$ and a closed term $t : \sigma$, we let $c_\sigma \equiv t$ denote $c_\sigma = t$. We call $c_\sigma \equiv t$ a *constant-instance definition* provided $\mathsf{TV}(t) \subseteq \mathsf{TV}(c_\sigma)$.

Definition 30. An *Isabelle/HOL definitional theory* is a set D of type and constant-instance definitions over Σ such that:

- It satisfies all the conditions of Def. 24, except that it is *not* required that, in condition (1.2), c be fresh, i.e., it is *not* required that $c \notin \Sigma^i$
- It is orthogonal: For all constants c, if c_σ and c_τ appear in two definitions in D, then neither σ is an instance of τ nor τ is an instance of σ (i.e., $\sigma \not\leq \tau \wedge \tau \not\leq \sigma$).
- Its induced dependency relation $\rightsquigarrow_n$ is terminating

The difference from Def. 24 is that now condition (1.2) does not require that the new defined item c_σ be fresh; rather, c_σ can be an instance of a previously declared constant, say, c_τ where $\sigma \leq \tau$. This type of flexibility, called ad hoc overloading, is allowed in Isabelle/HOL. There, for example, one can declare a polymorphic constant, such as $+ : \alpha \to \alpha \to \alpha$, and later define different, non-overlapping instances of it, such as $+_{nat}$ or $+_{real}$.

But allowing ad hoc overloading has its price, which is factored in the above definition as the last two conditions. Orthogonality expresses that the ad hoc overloaded instances do not overlap. This, together with the termination of the dependency relation, are able to ensure consistency even in the absence of freshness [Kunčar 2015; Kunčar and Popescu 2015].

We fix an Isabelle/HOL definitional theory D. A constant-instance c_σ is said to be *declared-only* if it is not an instance of a constant-instance that is defined in D.

Definition 31. We adapt Def. 8's and Def. 26's clauses for defined and declared-only constants to the more general setting as follows. Assume c_σ is a constant-instance. We have two cases:

- If there exist τ, t and ρ such that $\sigma \leq \tau$ is $c_\tau \equiv t$ is a constant-instance definition in D, we take $\mathsf{RLT}(c_\sigma) = \mathsf{RLT}(t[\rho])$.
- Otherwise, i.e., if the c_σ is declared-only, we take $\mathsf{RIN}(c_\sigma) = z^c_\sigma$.

(Everything else in the definitions of RIN and RLT from Sections 4 and 6 stays the same.)

Theorem 32. Theorem 28 also holds for Isabelle/HOL definitional theories.

7 THE SCOPE OF RELATIVIZATION

In this section we explore the scope of our PER relativization scheme in connection with practical HOL developments. First, we look at a specific kind of defined types—the container types, including inductive and coinductive datatypes—which form the vast majority of types defined in HOL, and conclude that the categorical infrastructure developed in HOL to support these types is compatible with PER relativization (§7.1). Armed with the Compatibility result, we show how to handle Section 3.1's example involving container types, using Admissibility in conjunction with Compatibility and PER-Parametricity (§7.2). Then we describe our experience with relativizing a large fragment of the Isabelle/HOL distribution (§7.4), with the help of a tool we have implemented in Isabelle's meta-programming language (§7.3). Our empirical formal experiments support the thesis that the prerequisite for our results, namely wide typedness, holds quite pervasively in HOL developments.

7.1 Relativization versus Container Types in HOL

As already noted, unlike the dependent type theories used in theorem proving, HOL does not have inductive datatypes (usually simply called *datatypes*) and coinductive datatypes (also known as *codatatypes*) as primitives. Instead, these are defined using the (nonrecursive) HOL constant and type definition mechanisms via some elaborate constructions, which are automated by specific tools. Melham [1989] was the first to formalize such a construction for datatypes (and implemented it in the HOL prover), and a similar construction was formalized by Berghofer and Wenzel [1999] (and implemented in Isabelle/HOL). The first construction of codatatypes in the HOL logic was described

by Traytel et al. [2012] (also implemented in Isabelle/HOL [Blanchette et al. 2014]). In what follows, we will focus on Traytel et al.'s construction, which generalizes the previous constructions, and offers a uniform view of (co)datatypes via the concept of bounded natural functor (BNF).

BNFs are a variant of container types [Abbott et al. 2005; Hoogendijk and de Moor 2000] that are well-behaved in HOL, in that they permit the construction in HOL of their initial algebras (yielding datatypes) and final coalgebras (yielding codatatypes). An n-ary BNF consists of a tuple $(F, Fmap, Frel, Fset_1, \ldots, Fset_n, Fbd)$ where:

- F is an n-ary type constructor (more precisely a type depending on n variables),
- $Fmap$ is a functorial action on functions for F, i.e., a *mapper* (making $(F, Fmap)$ a functor on the category of types and functions),
- $Frel$ is a functorial action on relations, i.e., a *relator* (making $(F, Frel)$ a functor on the category of types and relations),
- $Fset_i$ are natural transformations from F to the powerset functor (giving a concept of *support*, i.e., the set of elements "appearing in" an element of F),
- Fbd is a cardinal bound on the size of the support.

These different components are further related in specific ways: namely, the relator is the extension of the mapper and is determined by the mapper and the setter, and a congruence rule relates the mapper to the support (saying that two functions are mapped to the same result if their actions on the support is the same). Traytel et al. [2012] gives full details.

Datatypes and codatatypes emerge as the initial algebras and final coalgebras of BNFs. Let us consider a binary BNF $(F, Fmap, Frel, Fset_1, Fset_2, Fbd)$, where F is a binary type constructor $(\alpha_1, \alpha_2)F$, the relator $Frel$ has type $(\alpha_1 \Rightarrow \alpha_1' \Rightarrow bool) \Rightarrow (\alpha_2 \Rightarrow \alpha_2' \Rightarrow bool) \Rightarrow ((\alpha_1, \alpha_2)F \Rightarrow (\alpha_1', \alpha_2')F \Rightarrow bool)$, etc. Its initial algebra on the second variable gives rise to a unary BNF $(IF, IFmap, IFrel, IFset_1, IFbd)$. Thus, IF is a unary type constructor $\alpha_1\ IF$, the relator $IFrel$ has type $(\alpha_1 \Rightarrow \alpha_1' \Rightarrow bool) \Rightarrow (\alpha_1\ IF \Rightarrow \alpha_1'\ IF \Rightarrow bool)$, etc.

The components of the initial algebra BNF are defined from those of the original BNF. The most laborious part is the construction of the type constructor $\alpha_1\ IF$ and the constructor constant $ctor : (\alpha_1, (\alpha_1, \alpha_1\ IF)F) \Rightarrow \alpha_1\ IF$, which we describe next. Given a set $A : \alpha_2\ set$ and a function $s : (\alpha_1, \alpha_2)F \Rightarrow \alpha_2$, the pair (A, s) is said to be an *F-algebra on α_2 with parameters from α_1*, or an *algebra on α_2* for short (when α_1 is fixed), if A is closed under s, in that $\forall x : (\alpha_1, \alpha_2)F.\ Fset_2\ x \subseteq A \longrightarrow s\ x \in A$. The components A and s are called the algebra's *carrier* and *operation*, respectively. Let us write Alg for the set of all algebras on α_2. One chooses a monomorphic type σ_2 that is large enough to accommodate all algebras (on all types) up to isomorphism—this is possible thanks to the cardinal bound Fbd. The product of all algebras on σ_2 is defined in a standard way, as an algebra (P, p) whose carrier P is the set product $\prod_{(A,s) \in Alg} A$, and whose operation p is defined using the algebraic structure of the components and the universal property of set products. Note that the elements of $\prod_{(A,s) \in Alg} A$ are functions from algebras on σ_2 to (elements of) σ_2. Therefore, (P, p) is an algebra on the type $(\sigma_2\ set \times ((\alpha_1, \sigma_2)F \Rightarrow \sigma_2)) \Rightarrow \sigma_2$; let us denote this type by $\alpha_1 T$.

Then, by fixpoint induction, one defines $M \subseteq P$, the smallest subset of P that is closed under p, i.e., such that (M, p) forms an algebra on $\alpha_1 T$. This last algebra is proved to be initial. Finally, $\alpha_1\ IF$ is defined using a HOL type definition with base type $\alpha_1 T$ and predicate $\lambda x.\ x \in M$, and $ctor$ is defined as a copy of p from $\alpha_1\ T$ to $\alpha_1\ IF$; in other words, the initial algebra (M, p) is turned into an algebra operating on an entire type, the newly introduced type $\alpha_1\ IF$. Other components have easier definitions. Notably, $IFrel$ is defined inductively (as a least fixpoint) by the rule $\frac{Frel\ R\ (IFrel\ R)\ x\ y}{IFrel\ R\ (ctor\ x)\ (ctor\ y)}$.

The construction of final coalgebras for BNFs, yielding codatatypes, proceeds essentially dually. Again for the case of binary BNFs F, it gives a BNF $(\mathcal{J}F, \mathcal{J}Fmap, \mathcal{J}Frel, \mathcal{J}Fset_1, \mathcal{J}Fbd)$, where the type constructor $\alpha_1\ \mathcal{J}F$ and a destructor constant $dtor : \alpha_1\ \mathcal{J}F \Rightarrow (\alpha_1, (\alpha_1, \alpha_1\ \mathcal{J}F)F)$ are produced by the

following route: (1) taking the quotient F-coalgebra to the largest F-bisimulation of the sum of all F-coalgebras on a large enough type σ_2 (a construction also described by Rutten [2000]), and then (2) using a HOL type definition for copying this quotient coalgebra to a coalgebra on an entire type. Here, the base type $\alpha_1 T$ (from which the type $\alpha_1\ \mathit{JF}$ is carved out) is $((\sigma_2\ \mathit{set} \times (\sigma_2 \Rightarrow (\alpha_1, \sigma_2)F)) \times \sigma_2)\,\mathit{set}$ because $(\sigma_2\ \mathit{set} \times (\sigma_2 \Rightarrow (\alpha_1, \sigma_2)F)) \times \sigma_2$ is the underlying type of the sum of all F-coalgebras on σ_2, and the additional *set* type constructor at the top comes from considering the quotient.

Example 33. Let $(\alpha_1, \alpha_2)F$ be $\mathit{unit} + \alpha_1 \times \alpha_2$. Then $\alpha_1\ \mathit{IF}$ is $\alpha_1\,\mathit{list}$, the type of lists over α_1, and $\mathit{ctor} : \mathit{unit} + \alpha_1 \times \alpha_1\,\mathit{list} \Rightarrow \alpha_1\,\mathit{list}$ is the sum-case combination of *Nil* (for *unit*) and *Cons*. Moreover, *IFmap* is the usual list-map and *IFrel* is the usual list relator *listrel*—with *listrel R* relating two lists if they have equal length and their elements are pointwise R-related. On the other hand, $\alpha_1\ \mathit{JF}$ is $\alpha_1\ \mathit{lazyList}$, the type of "lazy" (i.e., possibly infinite) lists over α_1, and $\mathit{dtor} : \alpha_1\,\mathit{lazyList} \Rightarrow \mathit{unit} + \alpha_1 \times \alpha_1\ \mathit{lazyList}$ produces the element of *unit* for the empty list and the pair of applied *head* and *tail* otherwise. □

In Isabelle/HOL (and similarly in other HOL-based provers), a tool extracts the high-level operations, such as *Nil* and *Cons* above, and proves their properties. The user does not see the intricate low-level types $\alpha_1 T$ from the construction process, but only the final product: the datatype $\alpha_1\ \mathit{IF}$ or codatatype $\alpha_1\ \mathit{JF}$, and a standalone collection of constants and properties on this (co)datatype. (However, these low-level aspects *are* relevant in regards to this paper's goals—as we explain below.)

In addition to being closed under the initial algebra and final coalgebra constructions, the BNFs include the type constructors of sum, product, projection and constant type—in that these type constructors form BNFs with their standard mappers and relators. Let us call these *basic BNFs*. Consequently, starting from basic BNFs and iterating the initial algebra and final coalgebra constructions, one obtains a rich collection of (possibly nested) (co)datatypes which we will simply refer to as *container types*. Thus, in this paper, the container types are defined to be all the type constructors that can be obtained through the aforementioned iterative process.

The BNF-based construction gives the expected relators for all container types, not just for lists. Therefore, it is important that our PER-relativization be compatible with the container types' relators. Let us make precise what we mean by Compatibility, again restricting ourselves to unary container types to ease the notation. Any container type $\alpha\ F$, which in particular has a unary BNF structure $(F, \mathit{Fmap}, \mathit{Frel}, \mathit{Fset}_1, \mathit{Fbd})$, is defined via a HOL type definition (like all non-primitive types in HOL)—and indeed, above we have sketched how this type definition looks like in case F emerges as the initial algebra or final coalgebra of another BNF. Now, remember that, in order for our results to apply, yielding well-behaved and admissible PER-relativization, we require a relational witness relwit_F of type $\alpha\ \mathit{rel} \Rightarrow (\alpha\ F)\mathit{rel}$, i.e., $(\alpha \Rightarrow \alpha \Rightarrow \mathit{bool}) \Rightarrow (\alpha\ F \Rightarrow \alpha\ F \Rightarrow \mathit{bool})$, which makes the type definition wide. This has almost the same type as *Frel*, but less general in that it only considers relations between a type and itself, not between different types like *Frel* does; and is only required to behave well on PERs, not on arbitrary relations. Compatibility means that the restriction of *Frel* can play the role of relwit_F. And indeed we can prove that this is the case:

Theorem 34. **(Compatibility)** Every container type F has a wide type definition in HOL, in such a way that relwit_F is the restriction of its BNF-based relator *Frel*.

The proof of this theorem proceeds as follows: First we prove that the basic container types have wide type definitions. Then, for each container type F, we assume that F had a type definition within a widely typed definitional theory, and prove that the type definitions of its datatype and codatatype container types *IF* and *JF* are also wide. This involves showing that there exists a bijection-up-to between the entire type $\alpha\ \mathit{IF}$ endowed with the relation $\mathit{Frel}\ R_\alpha$ and the subset $\mathsf{RLT}(\{x.\ x \Leftarrow M\})$ of the base type $\alpha\ T$ endowed with the relation $\mathsf{RIN}(\alpha\ T)$, where M is the minimal F-algebra

described above. (And similarly for the codatatype $\alpha\ \mathcal{JF}$.) The delicate proof of this crucial step—from the wideness of a BNF's type definition to that of its (co)datatype BNF—has been formalized in Isabelle/HOL and is provided as supplementary material.

The existence of such bijections-up-to for (co)datatypes shows that, as far as PERs are concerned, simply (A) copying the relator from a low-level base type according to an (essentially) set-theoretic construction of a (co)datatype, is as good as (B) defining the relator using a high-level (co)inductive definition principle. This is somewhat surprising, since alternative (A) is an instance of a general-purpose scheme working for any type with a wide definition in HOL, whereas alternative (B) uses domain-specific knowledge from category theory.

7.2 More Revisiting of the Working Examples

We are now ready to look at statement (3) from Section 3.1. Applying the Recoverability theorem (and again writing R instead of R^α), we obtain

$$\begin{array}{ll} & (R \Rightarrow R \Rightarrow R)\ times\ times\ \wedge\ R\ e\ e \longrightarrow \mathsf{RLT}(group\ times\ e) \longrightarrow \\ (3'') & (\forall xs_{\alpha\ list},\ ys_{\alpha\ list}.\ \mathsf{RIN}(\alpha\ list)\ xs\ xs\ \wedge\ \mathsf{RIN}(\alpha\ list)\ ys\ ys \longrightarrow \\ & \quad R\ (\mathsf{RLT}(fold)\ times\ e\ (\mathsf{RLT}(append)\ xs\ ys)) \\ & \quad\ (\mathsf{RLT}(times)\ (\mathsf{RLT}(fold)\ times\ e\ xs)\ (\mathsf{RLT}(fold)\ times\ e\ ys))) \end{array}$$

where $\mathsf{RLT}(group\ times\ e)$ was already discussed in Section 4.3.

According to the Compatibility theorem, the relational interpretation of $\alpha\ list$ is given by the list relator (discussed in Example 33), i.e., $\mathsf{RIN}(\alpha\ list)$ is $listrel\ R$ (in the relativization context, where R is fixed). Moreover, all the high-level operators on lists, including the constructors, as well as *append* and *fold*, are parametric, in particular PER-parametric. Hence, as a consequence of the PER-Parametricity theorem (see our technical report for details), they are PER-related to their relativized counterparts. In particular, we have $\mathsf{RIN}\ (\alpha\ list \Rightarrow \alpha\ list \Rightarrow \alpha\ list)\ append\ \ \mathsf{RLT}(append)$, i.e., $(listrel\ R \Rightarrow listrel\ R \Rightarrow listrel\ R)\ append\ \ \mathsf{RLT}(append)$ (and similarly for *fold*). Hence, thanks to the properties of PERs and the fact that the relational interpretations are all PERs, we can replace in (3") the relativized versions of *append* and *fold* with the originals, obtaining:

$$\begin{array}{ll} & (R \Rightarrow R \Rightarrow R)\ times\ times\ \wedge\ R\ e\ e \longrightarrow \mathsf{RLT}(group\ times\ e) \longrightarrow \\ (3'') & (\forall xs_{\alpha\ list},\ ys_{\alpha\ list}.\ listrel\ R\ xs\ xs\ \wedge\ listrel\ R\ ys\ ys \longrightarrow \\ & \quad R\ (fold\ times\ e\ (append\ xs\ ys)) \\ & \quad\ (times\ (fold\ times\ e\ xs)\ (fold\ times\ e\ ys))) \end{array}$$

Finally, taking R to be $eqOf\ A$ gives us the expected set-based relativization:

$$\begin{array}{ll} & closed_2\ A\ times\ \wedge\ closed_0\ A\ e \longrightarrow group^{rlt}\ A\ e\ times \longrightarrow \\ (3') & (\forall xs_{\alpha\ list} \Leftarrow list(A),\ ys_{\alpha\ list} \Leftarrow list(A). \\ & \quad fold\ times\ e\ (append\ xs\ ys) = times\ (fold\ times\ e\ xs)\ (fold\ times\ e\ ys)) \end{array}$$

This is because, as explained in Section 4.3, closedness of an item under the set A is equivalent the the item being self-related via (the lifting of) the PER $eqOf\ A$. Similarly, membership of an item xs to $list(A)$ is equivalent to $listrel\ (eqOf\ A)\ xs\ xs$; this happens to be a generic property connecting the support and the relator of any BNF.

In conclusion, Compatibility and PER-Parametricity yield a particularly simple (while still HOL-admissible) relativization of statements involving container types.

7.3 Relativization Tool

We have implemented our PER relativization scheme in Isabelle/HOL, as a tool in Isabelle/ML [Wenzel 2022] that automatically produces the relativization of HOL theorems on demand.[6] It proceeds recursively on terms and types as prescribed in our Def. 8, while preserving the definitional abstraction layer—in that new constants are defined for each relativized defined constant and type.

More specifically, whenever relativization encounters a defined type (e.g., those introduced using the typedef command of Isabelle/HOL), it looks up its relational witness in the database of widely-typed types and fails if none exists. After encountering the failure, the user may register the type as widely-typed by providing the relational witness and exhibiting the desired bijection-up-to (and proving their properties). We automate the registration and the proofs (without any user involvement) for type copies of widely-typed types σ, i.e., types defined using the $\lambda x_\sigma.\ \mathit{true}$ predicate.

The relativized versions of given theorems are introduced as axioms, relying on Admissibility (Theorem 21) as a meta-justification. Theorem 21 allows for an axiom-free implementation of relativization in principle. Yet, we would need to analyze the proofs of non-relativized theorems to go axiom-free. By default Isabelle does not record proofs. There is a facility for enabling proof terms due to Berghofer and Nipkow [2000]. A future axiom-free implementation of relativization could take advantage of that work: either by a comprehensive translation of the proof terms following our proof of Admissibility, or by only looking at the facts used in the proof term and performing proof reconstruction in the style of Sledgehammer [Paulson and Blanchette 2010], hopefully even reusing the existing Sledgehammer infrastructure. In spite of aggressive compression techniques employed by Berghofer and Nipkow, proof terms are currently severely hampering Isabelle's performance and, thus, cannot be used in day-to-day developments. In the future, we envision the lightweight (axiomatic) relativization as the default combined with full proofs as an extra check that can be left to run overnight.

Our tool automatically registers datatypes and codatatypes as widely-typed, by axiomatizing the required properties. As with Admissibility, this is a compromise—this time based on Compatibility (Theorem 34) as meta-justification. We formalized in Isabelle the main step involved in the Compatibility theorem, proving that, if a BNF has a wide type definition, then its initial algebra (least fixed point) and final coalgebra (greatest fixed point) also have wide type definitions. In the future it will be possible to turn our proofs into Isabelle/ML tactics that are applied dynamically to any user-defined (co)datatype, and thus remove our reliance on the Compatibility meta-justification.

7.4 Formal Empirical Study: Relativization in the Wild

So far, we have applied our tool to relativize large chunks of the Isabelle/HOL distribution (the HOL session [Isabelle Community 2022a]) and of the standard HOL library theories (the HOL-Library session [Isabelle Community 2022b])—comprising over 186 000 lines of definitions and proofs. In addition to (co)datatypes, this code base contains some types defined directly using HOL type definitions. These include the basic BNFs discussed in Section 7.1, as well as others, shown in Fig. 2. In some cases, these types have been previously proved to be BNFs, hence they had relators. In others, the types are not BNFs, but they had previously defined relator-like relation lifting operators that were proved to preserve equality and commute with relation composition.

Not only were we able to prove that the definitions of these types are wide, but in all cases where the types had previously defined relation-lifting operators (BNF relators or otherwise), we proved wide typedness taking the relational witness to be the relation-lifting operator. (In the cases where no relation-lifting operators were available, we proved wide typedness plugging in the default relational witness described in Prop. 12.) The proofs were occasionally delicate, requiring the

[6] Both the tool and the formal experiments performed with its help, in particular our relativized HOL theories including the proofs of wide typedness, are provided as supplementary material.

α *fset**	finite sets over α	α *filter**	filters on the powerset of α
α *cset**	countable sets over α		
α *multiset**	multisets (bags) over α	α *poly*	polynomials with α-coefficients
(α, β) *fmap**	partial functions of finite support between α and β	α *fls*	formal Laurent series with α-coefficients
α *biject*	bijections on α	(α, β) *comm**	commutative α-operators with values in β
α *dlist**	non-repetitive lists over α		
(α, β) *alist**	association lists with values in α and keys in β	α *comparator*	comparison functions on α
(α, β) *node*	the pre-datatype universe by Berghofer and Wenzel [1999]	α *bit0*, α *bit1*	finite types used for binary representation

Fig. 2. Types in the Isabelle/HOL distribution proved to have wide type definitions. A star ($*$) on their names means that they already had a relation-lifting operator defined.

understanding of some mathematical subtleties about the concepts represented by these types. This was not surprising, since, similarly to the case of containers, we had to relate via bijections-up-to some custom relation-lifting operators with our general-purpose ones.

In conclusion, our experiments so far support the empirical conjecture that defined types in HOL developments tend to be widely-typed, with the following addendum: The relation lifting operators that have been custom-defined for the types tend to work as their relational witnesses, and therefore fit into our relativization scheme.

8 MORE RELATED WORK

The works proximally related to ours are that of Kunčar and Popescu [2019] who proposed a form of types-to-sets relativization supported by an axiomatic extension of HOL (the Local Typedef rule), and the follow-ups by Immler and Zhan [2019] and Milehins [2022] who further improved the automation of the relativization process. In addition to extending the axiomatic basis, these works apply relativization in an ad hoc manner, for a fragment of HOL that is both restricted and not clearly specified: It merely works for types that have relators via which the necessary transfer theorems can be proved. Our current work (largely) removes both restrictions: It applies relativization comprehensively for the entire HOL, provided a widely applicable sanity property holds for the defined types, and it proves that relativization is admissible thus removing the need for the additional rule.

Another major theme in our paper is the shift of focus from types-to-sets to the more general types-to-PERs relativization in HOL. At its core, our work is a PER interpretation of the HOL types. On this front, we find ideas roughly similar to ours in dependent type theories (DTTs). Notably, the Nuprl system [Constable et al. 1986] employs subset types and quotient types, which can together represent PERs; a Nuprl foundation variant with first-class PER types was also worked out [Allen 1987]. Setoids and partial setoids (another way to refer to PERs) are essential for developments in proof assistants such as Agda and Coq [Barthe et al. 2003]. Aspects distinguishing our PER interpretation from those in DTTs come from the specificity of HOL (Hilbert choice and types defined by comprehension) and the focus on admissibility. Moreover, major themes related to PER/setoid interpretations in DTTs are intensionality versus extensionality, and proof irrelevance (via Curry-Howard), which are not of interest in HOL. On the other hand, the challenge we have addressed in this paper, which is essentially showing that HOL is self-contained with respect to relativization, may have interesting counterparts in DTTs which could be investigated. Indeed, switching from type-based theorems to more flexible set-based or PER-based theorems without re-proving the latter (and ideally without resorting to

extending provability) seems like a worthwhile goal regardless of the particular foundation. For example, Altenkirch et al. [2019] define a translation of types to setoids for a variant of Martin-Löf type theory, whose conjectured completeness seems to be a counterpart of our Generality theorem.

Beyond proof assistants, the use of PERs to interpret types of λ-calculi and related systems has a rich tradition, going back at least to Myhill and Shepherdson [1955] and Kreisel [1959], and has been particularly successful in the context of domain theory, e.g., [Abadi and Plotkin 1990; Freyd et al. 1992] (also discussed in the monograph by Mitchell [1996]).

Further broadening the scope to general relational interpretations, where the relations are not assumed to be PERs, our work intersects the theme of parametricity [Reynolds 1983; Wadler 1989]. Our translations can be viewed as a mechanism to massage HOL statements into more general yet equiprovable ones made in a PER-parametric fragment of HOL. While our "free theorems" are restricted to PERs, our Compatibility result shows that admissible PER-parametricity in HOL is compatible with full parametricity for container types. Our Recoverability result is a variant of Reynolds [1983]'s Identity Extension Lemma—again, restricted to PERs and applicable to relativized terms only.

General parametricity has been a major research theme in DTT in recent years, showing that parametricity really comes into its own if one adds the expressive power of dependent types [Bernardy et al. 2012; Bernardy and Moulin 2012] (and the same is true [Bernardy and Lasson 2011] for the related concept of realizablity [Kleene 1945; Krivine 1993]). Specifically, Bernardy et al. [2012] describe a parametricity translation and its correctness (a Reynolds-style abstraction theorem) for pure type systems (PTSs), and shows how it can be extended to handle inductive families. Moreover, Bernardy and Moulin [2012] show how the translation and all instances of the abstraction theorem can be internalized in the PTS, after adding a parametricity rule which does not break desirable properties such as Church-Rosser and strong normalization. There are strong similarities between the BNF-based constructions described in Section 7.1, which effectively carve out a well-behaved fragment of HOL, and Bernardy et al. [2012]'s inductive-style variant of the translation for inductive types. Namely, the definition of the BNF-based relator corresponds to the inductive definition of the translated type, where the translated constructors are inhabitants of the original constructor's parametricity statement. For example, Bernardy et al. [2012]'s interpretation $[\![list]\!]$ of *list* is parameterized by two types A_1 and A_2 and a relation R and has constructors $[\![Nil]\!] : [\![list]\!]\ A_1\ A_2\ R\ Nil\ Nil$ and $[\![Cons]\!] : \forall a_1 : A_1,\ \forall a_2 : A_2.\ \forall p : R\ a_1\ a_2.\ \forall as_1 : list\ A_1.\ \forall as_2 : list\ A_2.\ \forall q : [\![list]\!]\ A_1\ A_2\ R\ as_1\ as_2.\ [\![list]\!]\ A_1\ A_2\ R\ (Cons\ a_1\ as_1)\ (Cons\ a_2\ as_2)$. Ignoring the proof variables p and q from the types of these constructors, these are seen to be essentially the inductive definition of the relator for the *list* BNF, and at the same time statements of the parametricity of *Nil* and *Cons* w.r.t. this relator—corresponding closely to the actual definitions and proofs performed by Isabelle/HOL's (co)datatype package [Blanchette et al. 2014]. In short, for the well-behaved fragment of HOL, everything is parametric, and our work is in good company. In fact, Isabelle/HOL's lifting and transfer package [Huffman and Kuncar 2013] also targets this fragment when tracking parametricity for the purpose of transferring theorems. On the other hand, outside of this fragment, where we encounter brittle interactions between Hilbert choice and comprehension-based type definitions, we are "on our own"; here, as discussed, general parametricity fails and the relativized terms can only recover PER-parametricity. As for internalizing in (a suitable extension of) HOL the meta-theory of (PER-)parametricity, this is an interesting prospect, but due to the aforementioned non-uniformities we do not expect any results matching the elegance of those for PTSs.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their *very thorough* reading of our paper, yielding many insightful comments and suggestions that significantly improved our text. We thank Tobias Nipkow for suggesting to look into the admissibility of Local Typedef, which initiated this work.

REFERENCES

Martín Abadi and Gordon D. Plotkin. 1990. A Per Model of Polymorphism and Recursive Types. In *LICS 1990*. IEEE Computer Society, 355–365. https://doi.org/10.1109/LICS.1990.113761

Michael Gordon Abbott, Thorsten Altenkirch, and Neil Ghani. 2005. Containers: Constructing strictly positive types. *Theor. Comput. Sci.* 342, 1 (2005), 3–27. https://doi.org/10.1016/j.tcs.2005.06.002

Mark Adams. 2010. Introducing HOL Zero (Extended Abstract). In *ICMS 2010 (LNCS, Vol. 6327)*, Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama (Eds.). Springer, 142–143. https://doi.org/10.1007/978-3-642-15582-6_25

Stuart F. Allen. 1987. *A Non-Type-Theoretic Semantics for Type-Theoretic Language*. Ph. D. Dissertation. Cornell University, USA.

Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. 2019. Setoid Type Theory - A Syntactic Translation. In *MPC 2019 (LNCS, Vol. 11825)*, Graham Hutton (Ed.). Springer, 155–196. https://doi.org/10.1007/978-3-030-33636-3_7

Rob D. Arthan and Roger Bishop Jones. 2005. Z in HOL in ProofPower. In *The Newsletter of the Formal Aspects of Computing Science (FACS) Specialist Group*. Issue 2005-1. https://web.archive.org/web/20221014122152/https://www.bcs.org/media/3096/facts200503.pdf

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

Gilles Barthe, Venanzio Capretta, and Olivier Pons. 2003. Setoids in type theory. *J. Funct. Program.* 13, 2 (2003), 261–293. https://doi.org/10.1017/S0956796802004501

Stefan Berghofer and Tobias Nipkow. 2000. Proof Terms for Simply Typed Higher Order Logic. In *TPHOLs 2000 (LNCS, Vol. 1869)*, Mark Aagaard and John Harrison (Eds.). Springer, 38–52. https://doi.org/10.1007/3-540-44659-1_3

Stefan Berghofer and Markus Wenzel. 1999. Inductive Datatypes in HOL – Lessons Learned in Formal-Logic Engineering. In *TPHOLs 1999 (LNCS, Vol. 1690)*, Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin-Mohring, and Laurent Théry (Eds.). Springer, 19–36. https://doi.org/10.1007/3-540-48256-3_3

Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. 2012. Proofs for free – Parametricity for dependent types. *J. Funct. Program.* 22, 2 (2012), 107–152. https://doi.org/10.1017/S0956796812000056

Jean-Philippe Bernardy and Marc Lasson. 2011. Realizability and Parametricity in Pure Type Systems. In *FOSSACS 2011 (LNCS, Vol. 6604)*, Martin Hofmann (Ed.). Springer, 108–122. https://doi.org/10.1007/978-3-642-19805-2_8

Jean-Philippe Bernardy and Guilhem Moulin. 2012. A Computational Interpretation of Parametricity. In *LICS 2012*. IEEE Computer Society, 135–144. https://doi.org/10.1109/LICS.2012.25

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

Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. 2014. Truly Modular (Co)datatypes for Isabelle/HOL. In *ITP 2014 (LNCS, Vol. 8558)*, Gerwin Klein and Ruben Gamboa (Eds.). Springer, 93–110. https://doi.org/10.1007/978-3-319-08970-6_7

Ana Bove, Peter Dybjer, and Ulf Norell. 2009. A Brief Overview of Agda – A Functional Language with Dependent Types. In *TPHOLs 2009 (LNCS, Vol. 5674)*, Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel (Eds.). Springer, 73–78. https://doi.org/10.1007/978-3-642-03359-9_6

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

Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, Robert Harper, Douglas J. Howe, Todd B. Knoblock, N. P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. 1986. *Implementing mathematics with the Nuprl proof development system*. Prentice Hall. http://dl.acm.org/citation.cfm?id=10510

Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The Lean Theorem Prover (System Description). In *CADE-25 (LNCS, Vol. 9195)*, Amy P. Felty and Aart Middeldorp (Eds.). Springer, 378–388. https://doi.org/10.1007/978-3-319-21401-6_26

Jose Divasón, Sebastiaan J. C. Joosten, Ondřej Kunčar, René Thiemann, and Akihisa Yamada. 2018. Efficient certification of complexity proofs: formalizing the Perron-Frobenius theorem (invited talk paper). In *CPP 2018*, June Andronick and Amy P. Felty (Eds.). ACM, 2–13. https://doi.org/10.1145/3167103

Jose Divasón and René Thiemann. 2022. A Formalization of the Smith Normal Form in Higher-Order Logic. *J. Autom. Reason.* 66, 4 (2022), 1065–1095. https://doi.org/10.1007/s10817-022-09631-5

Peter J. Freyd, P. Mulry, Giuseppe Rosolini, and Dana S. Scott. 1992. Extensional PERs. *Inf. Comput.* 98, 2 (1992), 211–227. https://doi.org/10.1016/0890-5401(92)90019-C

Herman Geuvers. 2009. Proof assistants: History, ideas and future. *Sadhana* 34, 1 (2009), 3–25. https://doi.org/10.1007/s12046-009-0001-5

Georges Gonthier. 2007. The Four Colour Theorem: Engineering of a Formal Proof. In *ASCM 2007 (LNCS, Vol. 5081)*, Deepak Kapur (Ed.). Springer, 333. https://doi.org/10.1007/978-3-540-87827-8_28

Michael J. C. Gordon. 1991. Introduction to the HOL System. In *TPHOLs 1991*, Myla Archer, Jeffrey J. Joyce, Karl N. Levitt, and Phillip J. Windley (Eds.). IEEE Computer Society, 2–3.

Michael J. C. Gordon and Tom F. Melham (Eds.). 1993. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press. http://www.cs.ox.ac.uk/tom.melham/pub/Gordon-1993-ITH.html

Adam Grabowski, Artur Kornilowicz, and Adam Naumowicz. 2010. Mizar in a Nutshell. *J. Formaliz. Reason.* 3, 2 (2010), 153–245. https://doi.org/10.6092/issn.1972-5787/1980

Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason M. Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. 2015. A formal proof of the Kepler conjecture. *CoRR* abs/1501.02155 (2015). arXiv:1501.02155 http://arxiv.org/abs/1501.02155

John Harrison. 1996. HOL Light: A Tutorial Introduction. In *FMCAD 1996 (LNCS, Vol. 1166)*, Mandayam K. Srivas and Albert John Camilleri (Eds.). Springer, 265–269. https://doi.org/10.1007/BFb0031814

John Harrison. 2009. HOL Light: An Overview. In *TPHOLs 2009 (LNCS, Vol. 5674)*, Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel (Eds.). Springer, 60–66. https://doi.org/10.1007/978-3-642-03359-9_4

Paul F. Hoogendijk and Oege de Moor. 2000. Container types categorically. *J. Funct. Program.* 10, 2 (2000), 191–225. https://doi.org/10.1017/s0956796899003640

Brian Huffman and Ondrej Kuncar. 2013. Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In *CPP 2013 (LNCS, Vol. 8307)*, Georges Gonthier and Michael Norrish (Eds.). Springer, 131–146. https://doi.org/10.1007/978-3-319-03545-1_9

Fabian Immler and Bohua Zhan. 2019. Smooth manifolds and types to sets for linear algebra in Isabelle/HOL. In *CPP 2019*, Assia Mahboubi and Magnus O. Myreen (Eds.). ACM, 65–77. https://doi.org/10.1145/3293880.3294093

The Isabelle Community. 2022a. The HOL-Main session of the Isabelle/HOL distribution. https://isabelle.in.tum.de/website-Isabelle2021-1/dist/library/HOL/HOL/index.html.

The Isabelle Community. 2022b. The Standard Library of the Isabelle/HOL distribution. https://isabelle.in.tum.de/website-Isabelle2021-1/dist/library/HOL/HOL-Library/index.html.

Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. 2000. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers. https://doi.org/10.1007/978-1-4615-4449-4

Stephen Cole Kleene. 1945. On the Interpretation of Intuitionistic Number Theory. *J. Symb. Log.* 10, 4 (1945), 109–124. https://doi.org/10.2307/2269016

Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2010. seL4: formal verification of an operating-system kernel. *Commun. ACM* 53, 6 (2010), 107–115. https://doi.org/10.1145/1743546.1743574

Georg Kreisel. 1959. Interpretation of Analysis by Means of Constructive Functionals of Finite Types. In *Constructivity in Mathematics*, Arend Heyting (Ed.). Amsterdam: North-Holland Pub. Co., 101–128.

Jean-Louis Krivine. 1993. *Lambda-calculus, types and models*. Masson.

Ondřej Kunčar. 2015. Correctness of Isabelle's Cyclicity Checker: Implementability of Overloading in Proof Assistants. In *CPP 2015*, Xavier Leroy and Alwen Tiu (Eds.). ACM, 85–94. https://doi.org/10.1145/2676724.2693175

Ondřej Kunčar and Andrei Popescu. 2015. A Consistent Foundation for Isabelle/HOL. In *ITP 2015 (LNCS, Vol. 9236)*, Christian Urban and Xingyuan Zhang (Eds.). Springer, 234–252. https://doi.org/10.1007/978-3-319-22102-1_16

Ondřej Kunčar and Andrei Popescu. 2018. Safety and conservativity of definitions in HOL and Isabelle/HOL. *Proc. ACM Program. Lang.* 2, POPL (2018), 24:1–24:26. https://doi.org/10.1145/3158112

Ondřej Kunčar and Andrei Popescu. 2019. From Types to Sets by Local Type Definition in Higher-Order Logic. *J. Autom. Reason.* 62, 2 (2019), 237–260. https://doi.org/10.1007/s10817-018-9464-6

Xavier Leroy. 2009. Formal verification of a realistic compiler. *Commun. ACM* 52, 7 (2009), 107–115. https://doi.org/10.1145/1538788.1538814

Thomas F. Melham. 1989. Automating Recursive Type Definitions in Higher Order Logic. In *Current Trends in Hardware Verification and Automated Theorem Proving*. 341–386. https://doi.org/10.1007/978-1-4612-3658-0_9

Mihails Milehins. 2022. An extension of the framework types-to-sets for Isabelle/HOL. In *CPP 2022*, Andrei Popescu and Steve Zdancewic (Eds.). ACM, 180–196. https://doi.org/10.1145/3497775.3503674

John C. Mitchell. 1996. *Foundations for programming languages*. MIT Press.

John Myhill and John Cedric Shepherdson. 1955. Effective operations on partial recursive functions. *Mathematical Logic Quarterly* 1, 4 (1955), 310–317. https://doi.org/10.1002/malq.19550010407

Tobias Nipkow and Gerwin Klein. 2014. *Concrete Semantics – With Isabelle/HOL*. Springer. https://doi.org/10.1007/978-3-319-10542-0

Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. LNCS, Vol. 2283. Springer. https://doi.org/10.1007/3-540-45949-9

Tobias Nipkow and Gregor Snelting. 1991. Type Classes and Overloading Resolution via Order-Sorted Unification. In *FPCA 1991 (LNCS, Vol. 523)*, John Hughes (Ed.). Springer, 1–14. https://doi.org/10.1007/3540543961_1

Sam Owre, John M. Rushby, and Natarajan Shankar. 1992. PVS: A Prototype Verification System. In *CADE-11 (LNCS, Vol. 607)*, Deepak Kapur (Ed.). Springer, 748–752. https://doi.org/10.1007/3-540-55602-8_217

Lawrence C. Paulson. 1988. A formulation of the simple theory of types (for Isabelle). In *COLOG 1988 (LNCS, Vol. 417)*, Per Martin-Löf and Grigori Mints (Eds.). Springer, 246–274. https://doi.org/10.1007/3-540-52335-9_58

Lawrence C. Paulson and Jasmin Christian Blanchette. 2010. Three years of experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers. In *IWIL 2010 (EPiC Series in Computing, Vol. 2)*, Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska (Eds.). EasyChair, 1–11. https://doi.org/10.29007/36dt

Andrew M. Pitts. 1993. *Introduction to HOL: A theorem proving environment for higher order logic*, Chapter The HOL Logic, 191–232. In Gordon and Melham [Gordon and Melham 1993]. http://www.cs.ox.ac.uk/tom.melham/pub/Gordon-1993-ITH.html

Andrei Popescu and Dmitriy Traytel. 2022a. Admissible Types-To-PERs Relativization in Higher-Order Logic (Extended Technical Report). https://doi.org/10.5281/zenodo.7313923

Andrei Popescu and Dmitriy Traytel. 2022b. Formalization and implementation artifact associated with this paper. https://doi.org/10.5281/zenodo.7308911

John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *IFIP 1983*, R. E. A. Mason (Ed.). North-Holland/IFIP, 513–523.

Bertrand Russell. 1919. Descriptions. In *Introducton to Mathematical Philosophy*. George Allen and Unwin Publishers Ltd., 167–180.

Jan J. M. M. Rutten. 2000. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.* 249, 1 (2000), 3–80. https://doi.org/10.1016/S0304-3975(00)00056-6

Konrad Slind and Michael Norrish. 2008. A Brief Overview of HOL4. In *TPHOLs 2008 (LNCS, Vol. 5170)*, Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar (Eds.). Springer, 28–32. https://doi.org/10.1007/978-3-540-71067-7_6

Dmitriy Traytel, Andrei Popescu, and Jasmin Christian Blanchette. 2012. Foundational, Compositional (Co)datatypes for Higher-Order Logic: Category Theory Applied to Theorem Proving. In *LICS 2012*. IEEE Computer Society, 596–605. https://doi.org/10.1109/LICS.2012.75

Philip Wadler. 1989. Theorems for Free!. In *FPCA 1989*, Joseph E. Stoy (Ed.). ACM, 347–359. https://doi.org/10.1145/99370.99404

Makarius Wenzel. 2022. The Isabelle/Isar implementation manual. https://isabelle.in.tum.de/website-Isabelle2021-1/dist/doc/implementation.pdf.