From Types to Sets by Local Type Definitions in Higher-Order Logic

Ondřej Kunčar[1] and Andrei Popescu[2]

[1] Fakultät für Informatik, Technische Universität München, Germany
[2] Department of Computer Science, School of Science and Technology,
Middlesex University, UK

Abstract. Types in Higher-Order Logic (HOL) are naturally interpreted as nonempty sets—this intuition is reflected in the type definition rule for the HOL-based systems (including Isabelle/HOL), where a new type can be defined whenever a nonempty set is exhibited. However, in HOL this definition mechanism cannot be applied *inside proof contexts*. We propose a more expressive type-definition rule that addresses the limitation and we prove its soundness. This higher expressive power opens the opportunity for a HOL tool that relativizes type-based statements to more flexible set-based variants in a principled way. We also address particularities of Isabelle/HOL and show how to perform the relativization in the presence of type classes.

1 Motivation

The proof assistant community is divided in two successful camps. One camp, represented by provers such as Agda [7], Coq [6], Matita [5] and Nuprl [10], uses expressive type theories as a foundation. The other camp, represented by the HOL family of provers (including HOL4 [2], HOL Light [14], HOL Zero [3] and Isabelle/HOL [26]), mostly sticks to a form of classic set theory typed using simple types with rank 1 polymorphism. (Other successful provers, such as ACL2 [19] and Mizar [12], could be seen as being closer to the HOL camp, although technically they are not based on HOL.)

According to the HOL school of thought, a main goal is to acquire a sweet spot: keep the logic as simple as possible while obtaining *sufficient expressiveness*. The notion of sufficient expressiveness is of course debatable, and has been debated. For example, PVS [29] includes dependent types (but excludes polymorphism), HOL-Omega [16] adds first-class type constructors to HOL, and Isabelle/HOL adds ad hoc overloading of polymorphic constants. In this paper, we want to propose a gentler extension of HOL: we do not want to promote new “first-class citizens,” but merely to give better credit to an old and venerable HOL citizen: the notion of types emerging from sets.

The problem we address in this paper is best illustrated by an example. Let $\mathsf{lists} : \alpha\ \mathsf{set} \to \alpha\ \mathsf{list\ set}$ be the constant that takes a set A and returns the set of lists whose elements are in A, and $\mathsf{P} : \alpha\ \mathsf{list} \to \mathsf{bool}$ be another constant (whose definition is not important here). Consider the following statements, where we extend the usual HOL syntax by explicitly quantifying over types at the outermost level:

$$\forall\alpha.\ \exists xs_{\alpha\ \mathsf{list}}.\ \mathsf{P}\ xs \tag{1}$$

$$\forall\alpha.\ \forall A_{\alpha\ \mathsf{set}}.\ A \neq \emptyset \longrightarrow (\exists xs \in \mathsf{lists}\ A.\ \mathsf{P}\ xs) \tag{2}$$

The formula (2) is a relativized form of (1), quantifying not only over all types α, but also over all their nonempty subsets A, and correspondingly relativizing the quantification over all lists to quantification over the lists built from elements of A. We call theorems such as (1) *type-based* and theorems such as (2) *set-based*.

Type-based theorems have obvious advantages compared to the set-based ones. First, they are more concise. Moreover, automatic proof procedures work better for them, thanks to the fact that they encode properties more rigidly and more implicitly, namely, in the HOL types (such as membership to $\alpha\ \mathsf{list}$) and not via formulas (such as membership to the set $\mathsf{lists}\,A$). On the downside, type-based theorems are less flexible, and therefore unsuitable for some developments. Indeed, when working with mathematical structures, it is often the case that they have the desired property only on a proper subset of the whole type. For example, a function f from τ to σ may be injective or continuous only on a subset of τ. When wishing to apply type-based theorems from the library to deal with such situations, users are forced to produce ad hoc workarounds for relativizing them from types to sets. In the most striking cases, the relativization is created manually. For example, in Isabelle/HOL there exists the constant $\mathsf{inj\text{-}on}\ A\ f = (\forall x\, y \in A.\ f\ x = f\ y \longrightarrow x = y)$ together with a small library about functions being injective only on a subset of a type. In summary, while it is easier to reason about type-based statements such as (1), the set-based statements such as (2) are more general and easier to apply.

An additional nuance to this situation is specific to Isabelle/HOL, which allows users to annotate types with Haskell-like type-class constraints. This provides a further level of implicit reasoning. For example, instead of explicitly quantifying a statement over an associative operation $*$ on a type σ, one marks σ as having class $\mathsf{semigroup}$ (which carries implicitly the assumptions). This would also need to be reversed when relativizing from types to sets. If (1) made the assumption that α is a semigroup, as in $\forall(\alpha_{\mathsf{semigroup}}).\ \exists xs_{\alpha\ \mathsf{list}}.\ \mathsf{P}\ xs$, then (2) would need to quantify universally not only over A, but also over a binary operation on A, and explicitly assume it to be associative.

The aforementioned problem, of the mismatch between type-based theorems from libraries and set-based versions needed by users, shows up regularly in requests posted on the Isabelle community mailing lists. Here is an example [32]: *Various lemmas* [from the theory Finite_Set] *require me to show that f* [commutes with $\circ$] *for all x and y. This is a too strong requirement for me. I can show that it holds for all x and y in A, but not for all x and y in general.*

Often, users feel the need to convert entire libraries from type-based theorems to set-based ones. For example, our colleague Fabian Immler writes about his large formalization experience [18, §5.7]: *The main reason why we had to introduce this new type* [of finite maps] *is that almost all topological properties are formalized in terms of type classes, i.e., all assumptions have to hold on the whole type universe. It feels like a cleaner approach* [would be] *to relax all necessary topological definitions and results from types to sets because other applications might profit from that, too.*

A prophylactic alternative is of course to develop the libraries in a set-based fashion from the beginning, agreeing to pay the price in terms of verbosity and lack of automation. And numerous developments in different HOL-based provers do just that [4,8,9,15,23].

In this paper, we propose an alternative that gets the best of both worlds: *prove easily and still be flexible.* More precisely, develop the libraries type-based, but export the results set-based. We start from the observation that, from a set-theoretic semantics standpoint, the theorems (1) and (2) are equivalent: they both state that, for every nonempty collection of elements, there exists a list of elements from that collection for which P holds. Unfortunately, the HOL logic in its current form is blind to one direction of this equivalence: assuming that (1) is a theorem, one cannot prove (2). Indeed, in a proof attempt of (2), one would fix a nonempty set A and, to invoke (1), one would need to define a new type corresponding to A—an action not currently allowed inside a HOL proof context. In this paper, we propose a gentle eye surgery to HOL (and to Isabelle/HOL) to enable proving such equivalences, and show how this can be used to leverage user experience as outlined above.

The paper is organized as follows. In Section 2, we recall the logics of HOL and Isabelle/HOL. In Section 3, we describe the envisioned extension of HOL: adding a new rule for simulating type definitions in proof contexts. In Section 4, we demonstrate how the new rule allows us to relativize type-based theorems to set-based ones in HOL. Due to the presence of type classes, we need to extend Isabelle/HOL's logic further to achieve the relativization—this is the topic of Section 5. Finally, in Section 6 we outline the process of performing the relativization in a principled and automated way.

We created a website [1] associated to the paper where we published the Isabelle implementation of the proposed logical extensions and the Isabelle proof scripts showing examples of applying the new rules to relativize from types to sets (including this paper's introductory example).

2 HOL and Isabelle/HOL Recalled

In this section, we briefly recall the logics of HOL and Isabelle/HOL mostly for the purpose of introducing some notation. For more details, we refer the reader to standard textbooks [11, 25]. We distinguish between the *core logic* and the *definitional mechanisms*.

2.1 Core Logic

The core logic is common to HOL and Isabelle/HOL: it is classical Higher-Order Logic with rank 1 polymorphism, Hilbert choice and the Infinity axioms. A HOL signature consists of a collection of type constructor symbols $k \in K$, which include the binary function type constructor $\rightarrow$ and the nullary bool and ind (for representing the booleans and an infinite type, respectively). The types σ, τ are built from type variables α and type constructors. The signature also contains a collection of constants $c \in C$ together with an indication of their types, $c : \tau$. Among these, we have equality, $= : \alpha \rightarrow \alpha \rightarrow \mathsf{bool}$, and implication, $\longrightarrow : \mathsf{bool} \rightarrow \mathsf{bool} \rightarrow \mathsf{bool}$. The terms t, s are built using typed (term) variables x_σ, constant instances c_σ, application and λ-abstraction. When writing concrete terms, types of variables and constants will be omitted when they can be inferred. HOL typing assigns types to terms, $t : \sigma$, in a standard way. The notation $\sigma \leq \tau$ means that σ is an instance of τ, e.g., $\mathsf{bool}\ \mathsf{list}$ is an instance of $\alpha\ \mathsf{list}$, which itself is an instance of α. A formula is a term of type bool. The formula connectives and quantifiers are defined in a standard way starting from equality and implication.

In HOL, types represent “rigid” collections of elements. More flexible collections can be obtained using sets. Essentially, a set on a type σ, also called a subset of σ, is given by a predicate $S : \sigma \to \mathsf{bool}$. Then membership of an element a to S is given by $S\ a$ being true. HOL systems differ in the details of representing sets: some consider sets as syntactic sugar for predicates, others use a specialized type constructor for wrapping predicates, yet others consider the “type of subsets of a type” unary type constructor as a primitive. All these approaches yield essentially the same notion.

HOL deduction is parameterized by an underlying theory D. It is a system for infering formulas starting from the formulas in D and HOL axioms (containing axioms for equality, infinity, choice, and excluded middle) and applying deduction rules (introduction and elimination of $\longrightarrow$, term and type instantiation and extensionality).

2.2 Definitional Mechanisms of HOL

HOL has the tradition of not allowing arbitrary underlying theories D, but merely *definitional ones*, containing definitions of constants and types.

A *HOL constant definition* is a formula $c_\sigma = t$, where:

- c is a fresh constant of type σ
- t is a term that is closed (i.e., has no free term variables) and whose type variables are included in those of σ

HOL type definitions are more complex entities. They are based on the notion of a newly defined type α being embedded in an existing type β, i.e., being isomorphic to a given subset S of β via mappings *Abs* and *Rep*. Let ${}_{\alpha}(\beta \approx A)^{Abs}_{Rep}$ denote the formula expressing this:

$$(\forall x_\alpha.\, Rep\ x \in S\,) \;\wedge\; (\forall x_\alpha.\, Abs\ (Rep\ x) = x) \;\wedge\; (\forall y_\beta.\, y \in S \;\longrightarrow Rep\ (Abs\ y) = y)$$

When the user issues a command `typedef` $\tau = S_{\sigma\,\mathsf{set}}$, they are required to discharge the goal $S \neq \emptyset$, after which the system introduces a new type τ and two constants $\mathsf{Abs}^\tau : \sigma \to \tau$ and $\mathsf{Rep}^\tau : \tau \to \sigma$ and adds the axiom ${}_{\sigma}(\tau \approx S\,)^{\mathsf{Abs}^\tau}_{\mathsf{Rep}^\tau}$ to the theory.

2.3 Definitional Mechanisms of Isabelle/HOL

While a member of the HOL family, Isabelle/HOL is special w.r.t. constant definitions. Namely, a constant is allowed to be declared with a given type σ and then “overloaded” on various types τ less general than σ and mutually orthogonal. For example, we can have d declared to have type α, and then $\mathsf{d}_{\mathsf{bool}}$ defined to be True and $\mathsf{d}_{\alpha\ \mathsf{list}}$ defined to be $[\mathsf{d}_\alpha]$. We shall write Δ_c for the collection of all types where c has been overloaded. In the above example, $\Delta_{\mathsf{d}} = \{\mathsf{bool}, \alpha\ \mathsf{list}\}$.

The mechanism of overloaded definitions offers broad expressive power. But with power also comes responsibility. The system has to make sure that the defining equations cannot form a cycle. To guarantee that, a binary constant/type dependency relation $\rightsquigarrow$ on types and constants is maintained, where $u \rightsquigarrow v$ holds true iff one of the following holds:

1. u is a constant c that was declared with type σ and v is a type in σ

2. u is a constant c defined as $c = t$ and v is a type or constant in t
3. u is a type σ defined as $\sigma = A$ and v is a type or constant in A

We write $\rightsquigarrow^{\downarrow}$ for (type-)substitutive closure of the constant/type dependency relation, i.e., if $p \rightsquigarrow q$, the type instances of p and q are in $\rightsquigarrow^{\downarrow}$. The system accepts only overloaded definitions for which $\rightsquigarrow^{\downarrow}$ does not contain an infinite chain.

In addition, Isabelle supports user-defined *axiomatic type classes*, which are essentially predicates on types. They effectively improve the type system with the ability to carry implicit assumptions. For example, we can define the type class $\mathsf{finite}(\alpha)$ expressing that α has a finite number of inhabitants. Then, we can annotate type variables by such predicates, e.g., α_{finite}. Finally, we can substitute a type τ for α_{finite} only if τ has been previously proved to fulfill $\mathsf{finite}(\tau)$.

The axiomatic type classes become truly useful when we use overloaded constants for their definitions. This combination allows the use of Haskell-style type classes. E.g., we can reason about arbitrary semigroups by declaring a global constant $* : \alpha \to \alpha \to \alpha$ and defining the HOL predicate $\mathsf{semigroup}(\alpha)$ stating that $*$ is associative on α.

In this paper, we are largely concerned with results relevant for the entire HOL family of provers, but also take special care with the Isabelle/HOL maverick. Namely, we show that our local typedef proposal can be adapted to cope with Isabelle/HOL's type classes.

3 Proposal of a Logic Extension: Local Typedef

To address the limitation described in Section 1, we propose extending the HOL logic with a new rule for type definition with the following properties:

- It enables type definitions to be emulated inside proofs while avoiding the introduction of dependent types by a simple syntactic check.
- It is natural and sound w.r.t. the standard HOL semantics à la Pitts [27] as well as with the logic of Isabelle/HOL.

To motivate the formulation of the new rule and to understand the intuition behind it, we will first look deeper into the idea behind type definitions in HOL. Let us take a purely semantic perspective and ignore the rank-1 polymorphism for a minute. Then the principle behind type definitions simply states that for all types α and nonempty subsets A of them, there exists a type β isomorphic to A:

$$\forall \alpha.\ \forall A_{\alpha\,\mathsf{set}}.\ A \neq \emptyset \longrightarrow \exists \beta.\ \exists Abs_{\alpha \to \beta}\ Rep_{\beta \to \alpha}.\ {}_{\alpha}(\beta \approx A)^{Abs}_{Rep} \qquad (\star)$$

The typedef mechanism can be regarded as the result of applying a sequence of standard rules for connectives and quantifiers to $(\star)$ in a more expressive logic (notationally, we use Gentzen's sequent calculus):

1. Left $\forall$ rule of α and A with given type σ and term $S_{\sigma\,\mathsf{set}}$ (both provided by the user), and left implication rule:

$$\dfrac{\dfrac{\Gamma \vdash S \neq \emptyset \qquad \Gamma, \exists \beta\, Abs\, Rep.\ {}_{\sigma}(\beta \approx S)^{Abs}_{Rep} \vdash \varphi}{\Gamma, (\star) \vdash \varphi}\ \forall_L, \forall_L, \longrightarrow_L}{\Gamma \vdash \varphi}\ \text{Cut of } (\star)$$

2. Left $\exists$ rule for β, *Abs* and *Rep*, introducing some new/fresh type τ, and functions Abs^τ and Rep^τ:

$$\dfrac{\Gamma \vdash S \neq \emptyset \qquad \dfrac{\Gamma, {}_{\sigma}(\tau \approx S)^{\mathsf{Abs}^\tau}_{\mathsf{Rep}^\tau} \vdash \varphi}{\Gamma, \exists\beta\, Abs\, Rep.\, {}_{\sigma}(\beta \approx S)^{Abs}_{Rep} \vdash \varphi}\;\exists_L, \exists_L, \exists_L}{\dfrac{\Gamma, (\star) \vdash \varphi}{\Gamma \vdash \varphi}\;\text{Cut of } (\star)}\;\forall_L, \forall_L, \longrightarrow_L$$

The user further discharges $\Gamma \vdash S \neq \emptyset$, and therefore the overall effect of this chain is the sound addition of ${}_{\sigma}(\tau \approx S)^{\mathsf{Abs}^\tau}_{\mathsf{Rep}^\tau}$ as an extra assumption when trying to prove an arbitrary fact φ.

What we propose is to use a variant of the above (with fewer instantiations) as an actual rule:

- In step 1. we do not ask the user to provide concrete σ and $S_{\sigma\ \mathsf{set}}$, but work with a type σ and a term $A_{\sigma\ \mathsf{set}}$ that can contain type and term *variables*.
- In step 2., we only apply the left $\exists$ rule to the type β and introduce a fresh type *variable* β

We obtain:

$$\dfrac{\Gamma \vdash A \neq \emptyset \qquad \dfrac{\Gamma, \exists Abs\, Rep.\, {}_{\sigma}(\beta \approx A)^{Abs}_{Rep} \vdash \varphi}{\Gamma, \exists\beta\, Abs\, Rep.\, {}_{\sigma}(\beta \approx A)^{Abs}_{Rep} \vdash \varphi}\;[\beta \text{ fresh}]\;\exists_L}{\dfrac{\Gamma, (\star) \vdash \varphi}{\Gamma \vdash \varphi}\;\text{Cut of } (\star)}\;\forall_L, \forall_L, \longrightarrow_L$$

To conclude, the overall rule, written (LT) as in “Local Typedef”, looks as follows:

$$\dfrac{\Gamma \vdash A \neq \emptyset \qquad \Gamma \vdash (\exists Abs\, Rep.\, {}_{\sigma}(\beta \approx A)^{Abs}_{Rep}) \longrightarrow \varphi}{\Gamma \vdash \varphi}\;[\beta \notin A, \varphi, \Gamma]\ \text{(LT)}$$

This rule allows us to locally assume that there is a type β isomorphic to an arbitrary non-empty set A. The syntactic check $\beta \notin A, \varphi, \Gamma$ prevents an introduction of a dependent type (since A can contain term variables in general).

The above discussion merely shows that (LT) is morally correct and more importantly *natural* in the sense that it is an instance of a more general principle, namely the rule ($\star$).

As for any extension of a logic, we have to make sure that the extension is correct.

Proposition 1. HOL extended by the (LT) rule is consistent.

This means that using rules of the HOL deduction system together with the (LT) rule cannot produce a proof of False. The same property holds for Isabelle/HOL.

Proposition 2. Isabelle/HOL extended by the (LT) rule is consistent.[3]

In the next section we will look at how the (LT) rule helps us to achieve the transformation from types to sets in HOL.

[3] The justification for the soundness of this rule in the context Isabelle/HOL’s arcane circularity checks, sketched in the appendix, is based on our new work on proving Isabelle/HOL consistency [21]; more details can be found in the extended technical report from [1].

4 From Types to Sets in HOL

Let us look again at the motivating example from Section 1 and see how the rule (LT) allows us to achieve the relativization from a type-based theorem to a set-based theorem in HOL or Isabelle/HOL without type classes. We assume (1) is a theorem, and wish to prove (2). We fix α and $A_{\alpha\,\mathsf{set}}$ and assume $A \neq \emptyset$. Applying (LT), we obtain a type β (represented by a fresh type variable) such that $\exists Abs\ Rep.\ {}_{\alpha}(\beta \approx A)^{Abs}_{Rep}$, from which we obtain Abs and Rep such that ${}_{\alpha}(\beta \approx A)^{\mathsf{Abs}}_{\mathsf{Rep}}$. From this, (1) with α instantiated to β, and the definition of lists, we obtain

$$\exists xs_{\beta\,\mathsf{list}} \in \mathsf{lists}\ (\mathsf{UNIV}_{\beta\,\mathsf{set}}).\ \mathsf{P}_{\beta\,\mathsf{list}\to\mathsf{bool}}\ xs.$$

Furthermore, using that Abs and Rep are isomorphisms between $A_{\alpha\,\mathsf{set}}$ and $\mathsf{UNIV}_{\beta\,\mathsf{set}}$, we obtain

$$\exists xs_{\alpha\,\mathsf{list}} \in \mathsf{lists}\ A_{\alpha\,\mathsf{set}}.\ \mathsf{P}_{\alpha\,\mathsf{list}\to\mathsf{bool}}\ xs,$$

as desired.[4]

We will consider a general case now. Let us start with a type-based theorem

$$\forall \alpha.\ \varphi[\alpha], \tag{3}$$

where $\varphi[\alpha]$ is a formula containing α. We fix α and $A_{\alpha\,\mathsf{set}}$, assume $A \neq \emptyset$ and “define” a new type β isomorphic to A. Technically, we fix a fresh type variable β and assume

$$\exists Abs\ Rep.\ {}_{\alpha}(\beta \approx A)^{Abs}_{Rep}. \tag{4}$$

From the last formula, we can obtain the isomorphism Abs and Rep between β and A. Having the isomorphisms, we can carry out the relativization along them and prove

$$\varphi[\beta] \longleftrightarrow \varphi^{\mathsf{on}}[\alpha, A_{\alpha\,\mathsf{set}}], \tag{5}$$

where $\varphi^{\mathsf{on}}[\alpha, A_{\alpha\,\mathsf{set}}]$ is the relativization of $\varphi[\beta]$. In the motivational example:

$$\varphi[\beta] = \exists xs_{\beta\,\mathsf{list}}.\ \mathsf{P}\ xs$$
$$\varphi^{\mathsf{on}}[\alpha, A_{\alpha\,\mathsf{set}}] = \exists xs_{\alpha\,\mathsf{list}} \in \mathsf{lists}\ A.\ \mathsf{P}\ xs$$

We postpone the discussion how to derive φ^{on} from φ in a principled way and how to automatically prove the equivalence between them until Section 6. We only appeal to the intuition here: for example, if φ contains the universal quantification $\forall x_\beta$, we replace it by the isomorphic bounded quantification $\forall x_\alpha \in A$ in φ^{on}. Or if φ contains the predicate $\mathsf{inj}\ f_{\beta\to\gamma}$, we replace it by the isomorphic notion of $\mathsf{inj}^{\mathsf{on}}\ A_{\alpha\,\mathsf{set}}\ f_{\alpha\to\gamma}$ in φ^{on}.

Since the left-hand side of the equivalence (5) is an instance of (3), we discharge the left-hand side and obtain $\varphi^{\mathsf{on}}[\alpha, A_{\alpha\,\mathsf{set}}]$, which does not contain the locally “defined” type β anymore. Thus we can discard β. Technically, we use the (LT) rule and remove the assumption (4). Thus we obtain the final result:

$$\forall \alpha.\ \forall A_{\alpha\,\mathsf{set}}.\ A \neq \emptyset \longrightarrow \varphi^{\mathsf{on}}[\alpha, A]$$

[4] We silently assume parametricity of the quantifier $\exists$ and P.

This theorem is the set-based version of $\forall\alpha.\ \varphi[\alpha]$.

We will move to Isabelle/HOL in the next section and explore how the isomorphic journey between types and sets proceeds in the environment where we are allowed to restrict type variables by type-class annotations.

5 From Types to Sets in Isabelle/HOL

Isabelle/HOL goes beyond traditional HOL and extends it by axiomatic type classes and overloading. We will explain in this section how these two features are in conflict with the algorithm described in Section 4 and how to circumvent these complications.

5.1 Local Axiomatic Type Classes

The first complication is the implicit assumptions on types given by the axiomatic type classes. Let us recall that α_{finite} means that α can be instantiated only with a type that we proved to fulfill the conditions of the type class finite, namely that the type must contain finitely many elements.

To explain the complication on an example, let us modify (3) to speak about types of class finite:

$$\forall\alpha_{\mathsf{finite}}.\ \varphi[\alpha_{\mathsf{finite}}] \tag{6}$$

Clearly, the set that is isomorphic to α_{finite} must be some non-empty set A that is *finite*. Thus as a modification of the algorithm from Section 4, we fix a set A and assume that it is non-empty *and* finite. As previously, we locally define a new type β isomorphic to A. Although β fulfills the condition of the type class finite, we cannot add the type into the type class since this action is allowed only at the global theory level in Isabelle and not locally in a proof context.

On the other hand, without adding β into finite we cannot continue since we need to instantiate β for α_{finite} to prove the analog of the equivalence (5). Our solution is to internalize the type class assumption in (6) and obtain

$$\forall\alpha.\ \mathsf{finite}(\alpha) \longrightarrow \varphi[\alpha], \tag{7}$$

where $\mathsf{finite}(\alpha)$ is a term of type bool, which is true if and only if α is a finite type.[5] Now we can instantiate α by β and get $\mathsf{finite}(\beta) \longrightarrow \varphi[\beta]$. Using the fact that the relativization of $\mathsf{finite}(\beta)$ is $\mathsf{finite}\ A$, we apply the isomorphic translation between β and A and obtain

$$\mathsf{finite}\ A \longrightarrow \varphi^{\mathsf{on}}[\alpha, A].$$

Quantifying over the fixed variables and adding the assumptions yields the final result, the set-based version of (6):

$$\forall\alpha.\ \forall A_{\alpha\ \mathsf{set}}.\ A \neq \emptyset \longrightarrow \mathsf{finite}\ A \longrightarrow \varphi^{\mathsf{on}}[\alpha, A]$$

The internalization of type classes (inferring (7) from (6)) is already supported by the kernel of Isabelle—thus no further work is required from us. The rule for internalization of type classes is a result of the work by Haftmann and Wenzel [13, 31].

[5] This is Wenzel's approach [31] to represent axiomatic type classes by internalizing them as predicates on types, i.e., constants of type $\forall\alpha.\ \mathsf{bool}$. As this particular type is not allowed in Isabelle, Wenzel uses instead $\alpha\ \mathsf{itself} \to \mathsf{bool}$, where $\alpha\ \mathsf{itself}$ is a singleton type.

5.2 Local Overloading

In the previous section we addressed implicit assumptions on types given by axiomatic type classes and showed how to reduce the relativization of such types to the original translation algorithm by internalizing the type classes as predicates on types. As we explained in Section 2.3, the mechanism of Haskell-like type classes in Isabelle is more general than the notion of axiomatic type classes since additionally we are allowed to associate operations with every type class. In this respect, the type class finite is somewhat special since there are no operations associated with it.

In this section, we take as an example the type class semigroup defined as

$$\mathsf{semigroup}(\beta) \text{ iff } \forall x_\beta\, y_\beta\, z_\beta.\ (x * y) * z = x * (y * z). \tag{8}$$

The type class semigroup contains the associated operation multiplication, which is represented by the overloaded constant $*$.

Let us relativize $\forall \alpha_{\mathsf{semigroup}}.\ \varphi[\alpha_{\mathsf{semigroup}}]$. The structure that is isomorphic to the variable $\alpha_{\mathsf{semigroup}}$ must be a non-empty set A together with a binary operation f such that A is closed under f and f is associative on A. Formally, we fix $A_{\alpha\ \mathsf{set}}$ and $f_{\alpha\to\alpha\to\alpha}$ such that $A \neq \emptyset$ and $\forall x, y \in A.\ f\ x\ y \in A$ and we assume $\mathsf{semigroup}^{\mathsf{on}}_{\mathsf{with}}\ A\ f$, where

$$\mathsf{semigroup}^{\mathsf{on}}_{\mathsf{with}}\ A\ f = (\forall x\ y\ z \in A.\ f\ (f\ x\ y)\ z = f\ x\ (f\ y\ z)),$$

which we read along the paradigm: *a structure* on *the set* A with *operations* $f_1, \ldots, f_n$. As before, we locally define β to be isomorphic to A via isomorphisms Abs and Rep.

Having defined β, we want to prove that β belongs into semigroup. Using the approach from the previous section, this goal translates into proving $\mathsf{semigroup}(\beta)$, which requires that the overloaded constant $*_{\beta\to\beta\to\beta}$ (see (8)) must be isomorphic to f on A. In other words, we have to locally define $*_{\beta\to\beta\to\beta}$ to be a projection of f onto β, i.e., $x_\beta * y_\beta$ must equal $\mathsf{Abs}(f\ (\mathsf{Rep}\ x)\ (\mathsf{Rep}\ y))$. Although we can locally "define" a new constant (fix a fresh term variable c and assume $c = t$), we cannot overload the global symbol $*$ locally for β. This is not supported by Isabelle.

We will cope with the complication by compiling out the overloaded constant $*$ from

$$\forall \alpha.\ \mathsf{semigroup}(\alpha) \longrightarrow \varphi[\alpha] \tag{9}$$

by the dictionary construction and obtain

$$\forall \alpha.\ \forall f_{\alpha\to\alpha\to\alpha}.\ \mathsf{semigroup}_{\mathsf{with}}\ f \longrightarrow \varphi_{\mathsf{with}}[\alpha, f], \tag{10}$$

where $\mathsf{semigroup}_{\mathsf{with}}\ f_{\alpha\to\alpha\to\alpha} = (\forall x_\alpha\ y_\alpha\ z_\alpha.\ f\ (f\ x\ y)\ z = f\ x\ (f\ y\ z))$ and similarly for φ_{with}. First, we will look at how (10) helps us to finish the relativization and later we will explain how to obtain (10).

Given (10), we will instantiate α with β and obtain

$$\forall f_{\beta\to\beta\to\beta}.\ \mathsf{semigroup}_{\mathsf{with}}\ f \longrightarrow \varphi_{\mathsf{with}}[\beta, f].$$

Recall that the quantification of $\forall f_{\beta\to\beta\to\beta}$ is isomorphic to a bounded quantification over all $f_{\alpha\to\alpha\to\alpha}$s such that $A_{\alpha\ \mathsf{set}}$ is closed under this $f_{\alpha\to\alpha\to\alpha}$. The difference after compiling out the overloaded constant $*$ is that now we are isomorphically relating two lambda abstractions (local variables) and not a global constant $*$ to a local variable.

Thus we reduced the relativization once again to the original algorithm and can obtain the set-based version

$$\begin{aligned}&\forall\alpha.\ \forall A_{\alpha\ \mathsf{set}}.\ A \neq \emptyset \longrightarrow\\ &\quad \forall f_{\alpha\to\alpha\to\alpha}.\ (\forall x_\alpha\ y_\alpha \in A.\ f\ x\ y \in A) \longrightarrow \mathsf{semigroup}^{\mathsf{on}}_{\mathsf{with}}\ A\ f \longrightarrow \varphi^{\mathsf{on}}_{\mathsf{with}}[\alpha, A, f].\end{aligned}$$

Let us get back to the dictionary construction. Its detailed description can be found, for example, in the paper by Krauss and Schropp [20]. We will outline the process only informally here. Our task is to compile out an overloaded constant $*$ from a term s. As a first step, we transform s into s_{with} such that $s = s_{\mathsf{with}}$ and such all occurrences of $*$ on which s depends transitively through definitions of other constants are directly at the top in s_{with}. We proceed for every constant c in s as follows: if c has no definition, we do not do anything. If c was defined as $c = t$, we first apply the construction on t and obtain t_{with} such that $t = t_{\mathsf{with}}$; thus $c = t_{\mathsf{with}}$. Now we define a new constant $c_{\mathsf{with}}\ f = t_{\mathsf{with}}[f/*]$. As $c_{\mathsf{with}}\ * = c$, we replace c in s by $c_{\mathsf{with}}\ *$. At the end, we obtain $s = s_{\mathsf{with}}$ as a theorem. Notice that this procedure works only if there is no type in s that depends on $*$.

Thus the above-described step applied to (9) produces

$$\forall\alpha.\ \mathsf{semigroup}_{\mathsf{with}}\ *_{\alpha\to\alpha\to\alpha} \longrightarrow \varphi_{\mathsf{with}}[\alpha, *_{\alpha\to\alpha\to\alpha}].$$

To finish the dictionary construction, we replace every occurrence of $*_{\alpha\to\alpha\to\alpha}$ by a universally quantified variable $f_{\alpha\to\alpha\to\alpha}$ and obtain (10). This derivation step is not currently allowed in Isabelle. The idea why this is a sound derivation is as follows: since $*_{\alpha\to\alpha\to\alpha}$ is a type-class operation, the constant is overloaded only for instances of $*$ but never for $\alpha \to \alpha \to \alpha$, therefore $*_{\alpha\to\alpha\to\alpha}$ is unrestricted and must behave as a term variable. We formulate a rule (an extension of Isabelle's logic) that allows us to perform the above-described derivation (and prove its soundness).

First, let us recall that $\rightsquigarrow^{\downarrow}$ is substitutive closure of the constant/type dependency relation $\rightsquigarrow$ from Section 2.3 and $\varDelta_c$ is the set of all types for which c was overloaded. The notation $\sigma \not\lesssim S$ means that σ is not an instance of any type in S. We shall write R^+ for the transitive closure of R. Now we can formulate the Unoverloading Rule (UO):

$$\frac{\varphi}{\forall x_\sigma.\ \varphi[x_\sigma/c_\sigma]}\ [\neg(u \rightsquigarrow^{\downarrow+} c_\sigma) \text{ for any type or constant } u \text{ in } \varphi;\ \sigma \not\lesssim \varDelta_c] \quad \text{(UO)}$$

This means that we can replace a constant c_σ by a universally quantified variable x_σ under these two side-conditions:

1. All types and constant instances in φ do not semantically depend on c_σ through a chain of constant and type definitions. This guarantees that the term substitution $[x_\sigma/c_\sigma]$ replaces all occurrences of c_σ on which φ semantically depends. This constraint is fulfilled in the first step of the dictionary construction since for example $\varphi_{\mathsf{with}}[\alpha, *]$ does not contain any hidden $*$s due to the construction of φ_{with}.[6]

[6] Unless there is a type depending on $*$.

2. There is no matching definition for c_σ. In our use case, c_σ is a type-class operation with the most general type (e.g., $*_{\alpha\to\alpha\to\alpha}$) and therefore there does not exist any matching definition for σ.

Proposition 3. Isabelle/HOL extended by the (UO) rule is consistent.[7]

Notice that the (UO) rule suggests that even in presence of *ad-hoc* overloading, the polymorphic overloaded constants retain parametricity under some conditions.

In the next section, we will look at a concrete example of relativization of a formula with type classes.

5.3 Example: Relativization of Topological Spaces

Coming back to Immler's experience with topological spaces (see Section 1), we will show an example of relativization of a type-based theorem with type classes in a set-based theorem from the field of topologies. The type class in question will be a topological space, which has one associated operation $\mathsf{open} : \alpha\ \mathsf{set} \to \mathsf{bool}$, a predicate defining the open subsets of α. We require that the whole space is open, finite intersections of open sets are open, finite or infinite unions of open sets are open and that every two distinct points can be separated by two open sets that contain them. Such a topological space is called T2 space and therefore we call the type class T2-space.

One of the basic properties of T2 spaces is the fact that every compact set is closed:

$$\forall \alpha_{\mathsf{T2\text{-}space}}.\ \forall S_{\alpha\ \mathsf{set}}.\ \mathsf{compact}\ S \longrightarrow \mathsf{closed}\ S \tag{11}$$

A set is compact if every open cover of it has a finite subcover. A set is closed if its complement is open. i.e., $\mathsf{closed}\ S = \mathsf{open}\ (-S)$. Recall that our main motivation is to solve the problem when we have a T2 space on a proper subset of α. Let us show the translation of (11) into a set-based variant, which amends the problem. We will observe what happens to the predicate closed during the translation.

We will first internalize the type class T2-space and then abstract over its operation open via the first step of the dictionary construction. As a result, we obtain

$$\forall \alpha.\ \mathsf{T2\text{-}space_{with}}\ \mathsf{open} \longrightarrow \forall S_{\alpha\ \mathsf{set}}.\ \mathsf{compact_{with}}\ \mathsf{open}\ S \longrightarrow \mathsf{closed_{with}}\ \mathsf{open}\ S,$$

where $\mathsf{closed_{with}}\ \mathit{open}\ S = \mathit{open}\ (-S)$. Let us apply (UO) and generalize over open:

$$\begin{gathered}\forall \alpha.\ \forall \mathit{open}_{\alpha\ \mathsf{set}\to\mathsf{bool}}.\\ \mathsf{T2\text{-}space_{with}}\ \mathit{open} \longrightarrow \forall S_{\alpha\ \mathsf{set}}.\ \mathsf{compact_{with}}\ \mathit{open}\ S \longrightarrow \mathsf{closed_{with}}\ \mathit{open}\ S\end{gathered} \tag{12}$$

The last formula is a variant of (11) after we internalized the type class T2-space and compiled out its operation. Now we reduced the task to the original algorithm (using Local Typedef) from Section 4. As always, we fix a non-empty set $A_{\alpha\ \mathsf{set}}$, locally define β to be isomorphic to A and transfer the β-instance of (12) onto the $A_{\alpha\ \mathsf{set}}$-level:

$$\begin{gathered}\forall \alpha.\ \forall A_{\alpha\ \mathsf{set}}.\ A \neq \emptyset \longrightarrow \forall \mathit{open}_{\alpha\ \mathsf{set}\to\mathsf{bool}}.\ \mathsf{T2\text{-}space^{on}_{with}}\ A\ \mathit{open} \longrightarrow\\ \forall S_{\alpha\ \mathsf{set}} \subseteq A.\ \mathsf{compact^{on}_{with}}\ A\ \mathit{open}\ S \longrightarrow \mathsf{closed^{on}_{with}}\ A\ \mathit{open}\ S\end{gathered}$$

[7] Again, the rigorous justification for this based on our work on Isabelle/HOL's consistency [21].

This is the set-based variant of the original theorem (11). Let us show what happened to $\mathsf{closed}_{\mathsf{with}}$: its relativization is defined as $\mathsf{closed}^{\mathsf{on}}_{\mathsf{with}}\ A\ \mathit{open}\ S = \mathit{open}\ (-S \cap A)$. Notice that we did not have to restrict *open* while moving between β and A (since the function does not produce any values of type β), whereas S is restricted since subsets of β correspond to subsets of A.

5.4 General Case

Having seen a concrete example, let us finally aim for the general case. Let us assume that Υ is a type class depending on the overloaded constants $*_1, \ldots, *_n$, written $\overline{*}$. We write $A \downarrow \overline{f}$ to mean that A is closed under operations $f_1, \ldots, f_n$.

The following derivation tree shows how we derive from the type based theorem $\vdash \forall \alpha_{\Upsilon}.\ \varphi[\alpha_{\Upsilon}]$ (the topmost formula in the tree) its set-based version (the bottommost formula). Explanation of the derivation steps follows after the tree.

$$\begin{array}{cl}
\vdash \forall \alpha_{\Upsilon}.\ \varphi[\alpha_{\Upsilon}] & \\ \cline{1-1}
\vdash \forall \alpha.\ \Upsilon(\alpha) \longrightarrow \varphi[\alpha] & (1) \\ \cline{1-1}
\vdash \forall \alpha.\ \Upsilon_{\mathsf{with}}\ \overline{*}[\alpha] \longrightarrow \varphi_{\mathsf{with}}[\alpha, \overline{*}] & (2) \\ \cline{1-1}
\vdash \forall \alpha.\ \forall \overline{f}[\alpha].\ \Upsilon_{\mathsf{with}}\ \overline{f} \longrightarrow \varphi_{\mathsf{with}}[\alpha, \overline{f}] & (3) \\ \cline{1-1}
A_{\alpha\ \mathsf{set}} \neq \emptyset,\ {}_{\alpha}(\beta \approx A)^{\mathit{Abs}}_{\mathit{Rep}} \vdash \forall \alpha.\ \forall \overline{f}[\alpha].\ \Upsilon_{\mathsf{with}}\ \overline{f} \longrightarrow \varphi_{\mathsf{with}}[\alpha, \overline{f}] & (4) \\ \cline{1-1}
A_{\alpha\ \mathsf{set}} \neq \emptyset,\ {}_{\alpha}(\beta \approx A)^{\mathit{Abs}}_{\mathit{Rep}} \vdash \forall \overline{f}[\beta].\ \Upsilon_{\mathsf{with}}\ \overline{f} \longrightarrow \varphi_{\mathsf{with}}[\beta, \overline{f}] & (5) \\ \cline{1-1}
A_{\alpha\ \mathsf{set}} \neq \emptyset,\ {}_{\alpha}(\beta \approx A)^{\mathit{Abs}}_{\mathit{Rep}} \vdash \forall \overline{f}[\alpha].\ A \downarrow \overline{f} \longrightarrow \Upsilon^{\mathsf{on}}_{\mathsf{with}}\ A\ \overline{f} \longrightarrow \varphi^{\mathsf{on}}_{\mathsf{with}}[\alpha, A, \overline{f}] & (6) \\ \cline{1-1}
A_{\alpha\ \mathsf{set}} \neq \emptyset \vdash \forall \overline{f}[\alpha].\ A \downarrow \overline{f} \longrightarrow \Upsilon^{\mathsf{on}}_{\mathsf{with}}\ A\ \overline{f} \longrightarrow \varphi^{\mathsf{on}}_{\mathsf{with}}[\alpha, A, \overline{f}] & (7) \\ \cline{1-1}
\vdash \forall \alpha.\ \forall A_{\alpha\ \mathsf{set}}.\ A \neq \emptyset \longrightarrow \forall \overline{f}[\alpha].\ A \downarrow \overline{f} \longrightarrow \Upsilon^{\mathsf{on}}_{\mathsf{with}}\ A\ \overline{f} \longrightarrow \varphi^{\mathsf{on}}_{\mathsf{with}}[\alpha, A, \overline{f}] & (8)
\end{array}$$

Derivation steps:

(1) The class internalization from Section 5.1.
(2) The first step of the dictionary construction from Section 5.2.
(3) The Unoverloading rule (UO) from Section 5.2.
(4) We fix fresh α, $A_{\alpha\ \mathsf{set}}$ and assume that A is non-empty. We locally define a new type β to be isomorphic to A; i.e., we fix fresh β, $\mathit{Abs}_{\alpha \to \beta}$ and $\mathit{Rep}_{\beta \to \alpha}$ and assume ${}_{\alpha}(\beta \approx A)^{\mathit{Abs}}_{\mathit{Rep}}$.
(5) We instantiate α in the conclusion with β.
(6) Relativization—see Section 6.
(7) Since *Abs* and *Rep* are present only in ${}_{\alpha}(\beta \approx A)^{\mathit{Abs}}_{\mathit{Rep}}$, we can existentially quantify over them and replace the hypothesis with $\exists \mathit{Abs}\ \mathit{Rep}.\ {}_{\alpha}(\beta \approx A)^{\mathit{Abs}}_{\mathit{Rep}}$, which we discharge by the Local Typedef rule from Section 3, as β is not present elsewhere either (we removed it from the conclusion in the previous step).
(8) We move all hypotheses into the conclusion and quantify over all fixed variables.

As we mentioned, the step (2), the dictionary construction, cannot be performed for types depending on overloaded constants unless we want to compile out such types as well. In the next section, we will explain the last missing step, the relativization step (6).

Notice that with our approach, we can address one of the long-standing user complaints that they are not allowed to provide, for example, two different orders for the same type when using the type class for orders. With our approach, they can still enjoy advantages of type classes while proving abstract properties about orders and only export the final product as a set-based theorem, which quantifies over all possible orders.

6 Transfer: Automated Relativization

In this section, we will describe a procedure that automatically achieves relativization of the type-based theorems. Recall that we are facing the following problem: we have two types β and α such that β is isomorphic to some (non-empty) set $A_{\alpha\,\mathsf{set}}$, a proper subset of α, via two isomorphisms $\mathsf{Abs}_{\alpha\to\beta}$ and $\mathsf{Rep}_{\beta\to\alpha}$. In this setting, given a formula $\varphi[\beta]$, we want to find its isomorphic counterpart $\varphi^{\mathsf{on}}[\alpha, A]$ and prove $\varphi[\beta] \;\longleftrightarrow\; \varphi^{\mathsf{on}}[\alpha, A]$. Thanks to the previous work [17], in which the first author of this paper participated, we can use Isabelle's Transfer tool, which automatically synthesizes the relativized formula $\varphi^{\mathsf{on}}[\alpha, A]$ and proves the equivalence with the original formula $\varphi[\beta]$. We will sketch the main principles of the tool on the following example, where (14) is relativization of (13):

$$\forall f_{\beta\to\gamma}\; xs_{\beta\,\mathsf{list}}\; ys_{\beta\,\mathsf{list}}.\; \mathsf{inj}\; f \longrightarrow (\mathsf{map}\; f\; xs = \mathsf{map}\; f\; ys) \;\longleftrightarrow\; (xs = ys) \tag{13}$$

$$\forall f_{\alpha\to\gamma}.\; \forall xs\; ys \in \mathsf{lists}\; A_{\alpha\,\mathsf{set}}.\; \mathsf{inj_{on}}\; A\; f \longrightarrow (\mathsf{map}\; f\; xs = \mathsf{map}\; f\; ys) \;\longleftrightarrow\; (xs = ys) \tag{14}$$

First of all, we reformulate the problem a little bit. We will not talk about isomorphisms Abs and Rep but express the isomorphism between A and β by a binary relation $\mathsf{T}_{\alpha\to\beta\to\mathsf{bool}}$ such that $\mathsf{T}\; x\; y = (\mathsf{Rep}\; y = x)$. We call T a transfer relation.

To make transferring work, we require some set-up. First of all, we assume that there exists a relator for every non-nullary type constructor in φ. Relators lift relations over type constructors: Related data structures have the same shape, with pointwise-related elements (e.g., the relator list_all2 for lists), and related functions map related input to related output. Concrete definitions follow:

$$\begin{aligned}
&\mathsf{list\_all2} : (\alpha \to \beta \to \mathsf{bool}) \to \alpha\;\mathsf{list} \to \beta\;\mathsf{list} \to \mathsf{bool}\\
&(\mathsf{list\_all2}\; A)\; xs\; ys \equiv (\mathsf{length}\; xs = \mathsf{length}\; ys) \wedge (\forall (x,y) \in \mathsf{set}\;(\mathsf{zip}\; xs\; ys).\;\; A\; x\; y)\\
&\Mapsto : (\alpha \to \gamma \to \mathsf{bool}) \to (\beta \to \delta \to \mathsf{bool}) \to (\alpha \to \beta) \to (\gamma \to \delta) \to \mathsf{bool}\\
&(A \Mapsto B)\; f\; g \equiv \forall x\, y.\; A\; x\; y \longrightarrow B\; (f\; x)\; (g\; y)
\end{aligned}$$

Moreover, we need a transfer rule for every constant present in φ. The transfer rules express the relationship between constants on β and α. Let us look at some examples:

$$\begin{aligned}
&((\mathsf{T} \Mapsto =) \Mapsto =)\; (\mathsf{inj_{on}}\; A)\; \mathsf{inj}\\
&((\mathsf{T} \Mapsto =) \Mapsto =)\; (\forall\_ \in A)\; (\forall)\\
&((\mathsf{list\_all2}\; \mathsf{T} \Mapsto =) \Mapsto =)\; (\forall\_ \in \mathsf{lists}\; A)\; (\forall)\\
&((\mathsf{T} \Mapsto =) \Mapsto \mathsf{list\_all2}\; \mathsf{T} \Mapsto \mathsf{list\_all2} =)\; \mathsf{map}\; \mathsf{map}\\
&(\mathsf{list\_all2}\; \mathsf{T} \Mapsto \mathsf{list\_all2}\; \mathsf{T} \Mapsto =)\; (=)\; (=)
\end{aligned}$$

As we already mentioned, the universal quantification on β corresponds to a bounded quantification over A on α ($\forall\_ \in A$). The relation between the two constants is obtained purely syntactically: we start with the type (e.g., $(\beta \to \gamma) \to \mathsf{bool}$ for inj) and replace every type that does not change (γ and bool) by the identity relation $=$, every non-nullary type constructor by its corresponding relator ($\to$ by $\Mapsto$ and list by $\mathsf{list\_all2}$) and every type that changes by the corresponding transfer relation (β by T).

To derive the equivalence theorem between (13) and (14), we use the above-stated transfer rules (they are leaves in the derivation tree) and combine them with the rules for variables (they are at the leaves as well), application and lambda abstraction:

$$\frac{A\,x\,y \in \Gamma}{\Gamma \vdash A\,x\,y} \qquad \frac{\Gamma_1 \vdash (A \Mapsto B)\,f\,g \qquad \Gamma_2 \vdash A\,x\,y}{\Gamma_1 \cup \Gamma_2 \vdash B\,(f\,x)\,(g\,y)} \qquad \frac{\Gamma,\, A\,x\,y \vdash B\,(f\,x)\,(g\,y)}{\Gamma \vdash (A \Mapsto B)\,(\lambda x.\, f\,x)\,(\lambda y.\, g\,y)}$$

Similarity of the rules to those for typing of the simply typed lambda calculus is not a coincidence. A typing judgment here involves two terms instead of one, and a binary relation takes the place of a type. The environment Γ collects the local assumptions for bound variables. Thus since (13) and (14) are of type bool, the procedure produces $(13) = (14)$ as the corresponding relation for bool is $=$.

Of course, it is impractical to provide transfer rules for every instance of a given constant and for every particular transfer relation (in our example T). In general, we are solving the transfer problem for some relation $R_{\alpha\to\beta\to\mathsf{bool}}$ such that R is right-total ($\forall y.\, \exists x.\, R\ x\ y$), right-unique ($\forall x\ y\ z.\, R\ x\ y \longrightarrow R\ x\ z \longrightarrow y = z$) and left-unique ($\forall x\, y\, z.\, R\ x\ z \longrightarrow R\ y\ z \longrightarrow x = y$). Notice that our concrete T fulfills all those three conditions. We automatically derive those specific transfer rules from general parametrized transfer rules[8] talking about basic polymorphic constants of HOL, for example:

$$\mathsf{left\_unique}\,A \longrightarrow \mathsf{right\_unique}\,A \longrightarrow (A \Mapsto A \Mapsto =)\ (=)\ (=)$$
$$\mathsf{right\_total}\,A \longrightarrow ((A \Mapsto =) \Mapsto =)\ (\forall\_ \in (\mathsf{Domain}\,A))\ (\forall)$$

These rules are part of Isabelle's library. Notice that we do not look at type constructors in the Transfer tool only as sets of elements but that we need to impose an additional structure on them. For example, we required relators for type constructors and we implicitly used the knowledge that "lists whose elements are in A" can be expressed by $\mathsf{lists}\,A$. For space constraints, we cannot describe the structure in detail here but let us note that the Transfer tool generates automatically the structure for every type constructor that is a natural functor (sets, finite sets, all algebraic datatypes and codatatypes). More could be found in the thesis of the first author [22, §4].

Overall, the tool is able to perform the relativization completely automatically.

7 Conclusion

In this paper, we proposed extending Higher-Order Logic with a Local Typedef (LT) rule. We showed that the rule is not an ad hoc, but a natural addition to HOL in that it incarnates a semantic perspective characteristic to HOL: for every non-empty set A, there

[8] These rules are related to Reynolds's relational parametricity [28] and Wadler's free theorems [30]. The Transfer tool is a working implementation of Mitchell's representation independence [24] and it demonstrates that transferring of properties across related types can be organized and largely automated using the relational parametricity.

must be a type that is isomorphic to A. At the same time, (LT) is careful not to introduce dependent types, which in HOL would be considered to be a heresy. We demonstrated how the rule allows for more flexibility in the proof development: with (LT) in place, the HOL users can enjoy the abstraction provided by types during the proof activity, while still having access to the more widely applicable, set-based theorems. Being natural, semantically well justified and useful, we believe that the Local Typedef rule is a good candidate for HOL citizenship. We have implemented this extension in Isabelle/HOL, but its implementation should be straightforward and noninvasive in any HOL prover. And in a more expressive prover, such as HOL-Omega [16], this rule could simply be added as an axiom in the user space.

In addition, we showed that our method for relativizing theorems is applicable to types restricted by type classes as well, provided we extend the logic by a rule for compiling out overloading constants (UO). With (UO) in place, the Isabelle users can reason abstractly using type classes, while at the same time having access to different instances of the relativized result.

All along according to the motto: *Prove easily and still be flexible.*

References

1. From Types to Sets – Implementation and Examples, `http://www21.in.tum.de/~kuncar/documents/types-to-sets/`
2. The HOL4 Theorem Prover, `http://hol.sourceforge.net/`
3. Adams, M.: Introducing HOL Zero - (Extended Abstract). In: Fukuda, K., van der Hoeven, J., Joswig, M., Takayama, N. (eds.) ICMS 2010, LNCS, vol. 6327, pp. 142–143. Springer (2010)
4. Aransay, J., Ballarin, C., Rubio, J.: A Mechanized Proof of the Basic Perturbation Lemma. J. Autom. Reasoning 40(4), 271–292 (2008)
5. Asperti, A., Ricciotti, W., Coen, C.S., Tassi, E.: The Matita interactive theorem prover. In: CADE-23. pp. 64–69 (2011)
6. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. An EATCS Series, Springer (2004)
7. Bove, A., Dybjer, P., Norell, U.: A Brief Overview of Agda - A Functional Language with Dependent Types. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009, LNCS, vol. 5674, pp. 73–78. Springer (2009)
8. Chan, H., Norrish, M.: Mechanisation of AKS Algorithm: Part 1 – The Main Theorem. In: Urban, C., Zhang, X. (eds.) ITP 2015, LNCS, vol. 9236, pp. 117–136. Springer (2015)
9. Coble, A.R.: Formalized Information-Theoretic Proofs of Privacy Using the HOL4 Theorem-Prover. In: Borisov, N., Goldberg, I. (eds.) PETS 2008, LNCS, vol. 5134, pp. 77–98. Springer (2008)
10. Constable, R.L., Allen, S.F., Bromley, H.M., Cleaveland, W.R., Cremer, J.F., Harper, R.W., Howe, D.J., Knoblock, T.B., Mendler, N.P., Panangaden, P., Sasaki, J.T., Smith, S.F.: Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1986)
11. Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)
12. Grabowski, A., Kornilowicz, A., Naumowicz, A.: Mizar in a Nutshell. J. Formalized Reasoning 3(2), 153–245 (2010)

13. Haftmann, F., Wenzel, M.: Constructive Type Classes in Isabelle. In: Altenkirch, T., McBride, C. (eds.) TYPES 2006, LNCS, vol. 4502, pp. 160–174. Springer (2006)
14. Harrison, J.: HOL Light: A Tutorial Introduction. In: K., Srivas, M., Camilleri, A.J. (eds.) FMCAD '96, LNCS, vol. 1166, pp. 265–269. Springer (1996)
15. Hölzl, J., Heller, A.: Three Chapters of Measure Theory in Isabelle/HOL. In: van Eekelen, M.C.J.D., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) ITP 2011, LNCS, vol. 6898, pp. 135–151. Springer (2011)
16. Homeier, P.V.: The HOL-Omega logic. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 244–259. Springer (2009)
17. Huffman, B., Kunčar, O.: Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In: Gonthier, G., Norrish, M. (eds.) CPP 2013, LNCS, vol. 8307, pp. 131–146. Springer (2013)
18. Immler, F.: Generic Construction of Probability Spaces for Paths of Stochastic Processes. Master's thesis, Institut für Informatik, Technische Universität München (2012)
19. Kaufmann, M., Manolios, P., Moore, J.S.: Computer-Aided Reasoning: An Approach. Kluwer Academic Publishers (2000)
20. Krauss, A., Schropp, A.: A Mechanized Translation from Higher-Order Logic to Set Theory. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010, LNCS, vol. 6172, pp. 323–338. Springer (2010)
21. Kunčar, O., Popescu, A.: Comprehending Isabelle/HOL's Consistency (2016), submitted to ITP 2016. Available at `http://andreipopescu.uk/pdf/kuncar-popescu-IsaCons2016.pdf`
22. Kunčar, O.: Types, Abstraction and Parametric Polymorphism in Higher-Order Logic. Ph.D. thesis, Fakultät für Informatik, Technische Universität München (2016), `http://www21.in.tum.de/~kuncar/documents/kuncar-phdthesis.pdf`, submitted draft
23. Maggesi, M.: A formalisation of metric spaces in HOL Light (2015), `http://www.cicm-conference.org/2015/fm4m/FMM_2015_paper_3.pdf`, Presented at the Workshop Formal Mathematics for Mathematicians. CICM 2015. Published online.
24. Mitchell, J.C.: Representation Independence and Data Abstraction. In: POPL '86, pp. 263–276. ACM (1986)
25. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
26. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Part of the Isabelle2015 distribution (2015), `https://isabelle.in.tum.de/dist/Isabelle2015/doc/tutorial.pdf`
27. Pitts, A.: Introduction to HOL: A Theorem Proving Environment for Higher Order Logic, chap. The HOL Logic, pp. 191–232. In: Gordon and Melham [11] (1993)
28. Reynolds, J.C.: Types, Abstraction and Parametric Polymorphism. In: IFIP Congress, pp. 513–523 (1983)
29. Shankar, N., Owre, S., Rushby, J.M.: PVS Tutorial. Computer Science Laboratory, SRI International (1993)
30. Wadler, P.: Theorems for Free! In: FPCA '89, pp. 347–359. ACM (1989)
31. Wenzel, M.: Type Classes and Overloading in Higher-Order Logic. In: Gunter, E.L., Felty, A.P. (eds.) TPHOLs '97, LNCS, vol. 1275, pp. 307–322. Springer (1997)
32. Wickerson, J.: Isabelle Users List (Feb 2013), `https://lists.cam.ac.uk/mailman/htdig/cl-isabelle-users/2013-February/msg00222.html`

APPENDIX

The following proof is based on the set theoretical model for HOL developed by A. Pitts [27].

Proof of Proposition 1. Any deduction consisting of the deduction rules of HOL and the (LT) rule is sound.

Proof. Let us fix a model $\mathcal{M}$ and let us assume that the assumptions of the (LT) rule are satisfied in the model, i.e.,

$$\Gamma \models_{\mathcal{M}} A \neq \emptyset \qquad \text{and} \qquad \Gamma \models_{\mathcal{M}} (\exists Abs\; Rep.\; {}_{\sigma}(\beta \approx A)^{Abs}_{Rep}) \longrightarrow \varphi$$

Let us fix a type valuation θ and a compatible term valuation ξ such that $[\psi]_{\theta,\xi} = \mathsf{true}$ for all $\psi \in \Gamma$. Then using the interpretation of $\longrightarrow$, we obtain:

$$[A \neq \emptyset]_{\theta,\xi} = \mathsf{true}, \tag{15}$$

$$[\exists Abs\; Rep.\; {}_{\sigma}(\beta \approx A)^{Abs}_{Rep}]_{\theta,\xi} = \mathsf{true} \;\text{ implies }\; [\varphi]_{\theta,\xi} = \mathsf{true}. \tag{16}$$

From (15) and from the interpretation of sets, we can conclude that

$$[A]_{\theta,\xi} \neq \emptyset. \tag{17}$$

From (16) and the fact that $\beta \notin A$, $\beta \notin \varphi$ and $\beta \notin \Gamma$, we derive

$$(\exists B \in \mathcal{U}.\; [\exists Abs\; Rep.\; {}_{\sigma}(\beta \approx A)^{Abs}_{Rep}]_{\theta[B/\beta],\xi} = \mathsf{true}) \;\text{ implies }\; [\varphi]_{\theta,\xi} = \mathsf{true}. \tag{18}$$

If we were able to prove the antecedent of (18), we would be finished with the proof since we could use Modus Ponens and obtain $[\varphi]_{\theta,\xi} = \mathsf{true}$ and thus $\Gamma \models_{\mathcal{M}} \varphi$.

Following our intuitive understanding of the HOL model theory, we can surely prove

$$\exists B \in \mathcal{U}.\; [\exists Abs\; Rep.\; {}_{\sigma}(\beta \approx A)^{Abs}_{Rep}]_{\theta[B/\beta],\xi} = \mathsf{true}, \tag{19}$$

because we are looking for a set B that is an interpretation of β such that B is isomorphic to the interpretation of A. Needles to say, there exists such an interpretation: it is the interpretation of A. Let us define $B = [A]_{\theta,\xi}$ and observe that $B \in \mathcal{U}$ thanks to (17).

Since $A : \sigma\;\mathsf{set}$, then $B \subseteq [\sigma]_\theta$. Let us define $\mathsf{Abs} : [\sigma]_\theta \to B$ as

$$\mathsf{Abs}(x) = \begin{cases} x & \text{if } x \in B \\ \epsilon([\sigma]_\theta) & \text{otherwise} \end{cases}$$

and $\mathsf{Rep} : B \to [\sigma]_\theta$ as injection. It is a routine to verify ${}_{[\sigma]_\theta}(B \approx [A]_{\theta,\xi})^{\mathsf{Abs}}_{\mathsf{Rep}} = \mathsf{true}$. □

Notice that the bottom line of the proof was to show a semantic analog of ($\star$): given (17), we obtain (19).

We showed that (LT) is sound in HOL. We will move on and show soundness of the (LT) rule in Isabelle/HOL by using the translation of Isabelle/HOL into HOLC. Working in HOLC gives us the advantage to get closer to ($\star$), in the following sense:

for every non-empty set $A : \sigma\,\mathsf{set}$, not only we can postulate that there always exists a type isomorphic to A, we can even directly express such a type in HOLC, namely $\{\sigma \mid A\}$. That is basically what the axiom type_comp tells us. Thus informally speaking, the property ($\star$) is more first-class citizen in HOLC than in HOL. As a consequence, we will not have to appeal to semantics as in the previous proof.

The next two proofs rely on a sound translation between Isabelle/HOL and an extension of HOL with comprehension types (called HOLC), discussed in [21]. The main idea is that the new rules are manifestly sound when translated to HOLC.

Proof of Proposition 2. Any deduction consisting of the deduction rules of Isabelle/HOL and the (LT) rule is sound.

Proof. We will show that for every step

$$\frac{\Gamma \vdash A \neq \emptyset \qquad \Gamma \vdash (\exists Abs\ Rep.\,_{\sigma}(\beta \approx A)^{Abs}_{Rep}) \longrightarrow \varphi}{\Gamma \vdash \varphi}\ [\beta \notin A, \varphi, \Gamma]$$

in a HOL proof, we can construct a step in a HOLC proof of $\mathsf{NF}(\Gamma) \vDash \mathsf{NF}(\varphi)$ given

$$\mathsf{NF}(\Gamma) \vDash \mathsf{NF}(A) \neq \emptyset, \tag{20}$$

$$\mathsf{NF}(\Gamma) \vDash (\exists Abs\ Rep.\,_{\mathsf{NF}(\sigma)}(\beta \approx \mathsf{NF}(A))^{Abs}_{Rep}) \longrightarrow \mathsf{NF}(\varphi). \tag{21}$$

The side-condition of the (LT) rule $\beta \notin A, \varphi, \Gamma$ transfers into HOLC since for every $u \in \mathsf{Type} \cup \mathsf{Term}$ such that $\beta \notin u$, it holds that $\beta \notin \mathsf{NF}(u)$. This follows from the fact that unfolding a (type or constant) definition $v \equiv w$ cannot introduce new type variables since we require $\mathsf{TV}(w) \subseteq \mathsf{TV}(v)$. Thus $\beta \notin \mathsf{NF}(A)$, $\mathsf{NF}(\varphi)$, $\mathsf{NF}(\Gamma)$ and we obtain

$$\mathsf{NF}(\Gamma) \vDash (\exists Abs\ Rep.\,_{\mathsf{NF}(\sigma)}(\{\mathsf{NF}(\sigma) \mid \mathsf{NF}(A)\} \approx \mathsf{NF}(A))^{Abs}_{Rep}) \longrightarrow \mathsf{NF}(\varphi), \tag{22}$$

an instance of (21) where we substituted the witness $\{\mathsf{NF}(\sigma) \mid \mathsf{NF}(A)\}$ for β. As we already argued before the proof, we can use type_comp and discharge the antecedent of (22) by Modus Ponens (with the help of (20)). Thus we obtain the desired $\mathsf{NF}(\Gamma) \vDash \mathsf{NF}(\varphi)$. □

Proof of Proposition 3. Any deduction consisting of the deduction rules of Isabelle/HOL and the (UO) rule is sound.

Proof. We will argue that HOLC + (UO) (without its side-conditions; they are vacuous in HOLC) is still a consistent logic. That means, from φ we can derive $\forall x_\sigma.\ \varphi[x_\sigma/c_\sigma]$ in HOLC + (UO). Since HOLC does not contain any definitions, we interpret c_σ arbitrarily (as long as the value belongs to the interpretation of σ) in the proof of consistency of HOLC. That is to say, the proof of consistency does not rely on the actual value of interpretation of c_σ and thus we can replace c_σ by a term variable x_σ. Therefore the formula $\varphi[x_\sigma/c_\sigma]$ must be fulfilled for every evaluation of x_σ.

The first side conditions of (UO) guarantees that unfolding by NF does not introduce new c_σs and the second one guarantees that NF does not unfold any c_σ. Therefore the substitution $[x_\sigma/c_\sigma]$ commutes with NF, i.e., $\mathsf{NF}(\varphi[x_\sigma/c_\sigma]) = (\mathsf{NF}(\varphi))[x_{\mathsf{NF}(\sigma)}/c_{\mathsf{NF}(\sigma)}]$. Thus NF is a sound embedding of Isabelle/HOL + (UO) into HOLC + (UO). □