Safety and Conservativity of Definitions in HOL and Isabelle/HOL

Ondřej Kunčar
Fakultät für Informatik,
Technische Universität München
Munich, Germany

Andrei Popescu
Department of Computer Science,
School of Science and Technology,
Middlesex University, UK

***Abstract*—Definitions are traditionally considered to be a safe mechanism for introducing concepts on top of a logic known to be consistent. In contrast to arbitrary axioms, definitions should in principle be treatable as a form of abbreviation, and thus compiled away from the theory without losing provability. In particular, definitions should form a conservative extension of the pure logic.**

We prove these properties, namely, safety and conservativity, for Higher-Order Logic (HOL), a logic implemented in several mainstream theorem provers and relied upon by thousands of users. Some unique features of HOL, such as the requirement to give non-emptiness proofs when defining new types and the impossibility to unfold type definitions, make the proof of these properties, and also the very formulation of safety, nontrivial.

Our study also factors in the essential variation of HOL definitions featured by Isabelle/HOL, a popular member of the HOL-based provers family. The current work improves on our recent results which showed a weaker property, consistency of Isabelle/HOL's definitions.

***Index Terms*— higher-order logic (HOL), interactive theorem proving, type definitions, conservative extensions, Isabelle/HOL**

I. Introduction

Higher-Order Logic (HOL) ([29], Section III of this paper) is an important logic in the theorem proving community. It forms the basis of several interactive theorem provers, including HOL4 [11], HOL Light [13], Isabelle/HOL [23], ProofPower-HOL [5] and HOL Zero [4].

While its ideas go back a long way (to the work of Alonzo Church [9] and beyond), HOL contains a unique blend of features proposed by Mike 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 and implication (from which all the usual connectives and quantifiers can be 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, which we shall refer to as *minimal HOL*, 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 (the predicate has a witness). This is because *HOL types are required to be nonempty*—a major design decision, with practical and theoretical ramifications [11], [27]. 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).

Isabelle/HOL [23] is a notable member of the HOL family, and a maverick to some extent. It implements an essential variation of HOL, where constant definitions can be overloaded in an ad hoc manner, for different instances of their types. This flexibility forms the basis of Haskell-style type classes [24], a feature that allows for lighter, suppler formalizations and is partly responsible for Isabelle/HOL's wide popularity and prolificness: hundreds of users in both academia and industry, a large library of formalized results [2], [3], major verification success stories [10], [17], [28].

The founding fathers of HOL have paid special attention to consistency and related properties. Andrew Pitts designed a custom notion of *standard model* [29], aimed at smoothly accommodating both polymorphism and type definitions. He proved that constant and type definitions are *model-theoretically conservative w.r.t. standard models*: Any standard model for a theory can be expanded to a standard model of the theory plus the definitions. This of course implies consistency of HOL with definitions. Surprisingly, the founding fathers have not looked into the more customary notion of *proof-theoretic conservativity*, which we shall simply call *conservativity*. It states that, by adding new constants and types and their definitions, nothing new can be proved in the old language. This does not follow from the model-theoretic version (because of the restriction to *standard* models, for which deduction is not complete). In fact, as we discuss below, it does not even hold in general.

In Isabelle/HOL, the foundational problem is more challenging. Here, even consistency of definitions has not been fully understood until very recently (Section II-B). The culprit is precisely the feature that contributes to Isabelle/HOL's popularity—ad hoc overloading—which has a delicate interaction with type definitions [20, Section 1].

Motivated by the desire to settle the Isabelle foundations, early work by Wenzel [31] formulates criteria for safety of definitions in HOL-like logics. For a theory extension $\Theta_1 \subseteq \Theta_2$, he considers (proof-theoretic) conservativity, a property much

	Over Minimal HOL	Over Arbitrary HOL Theories	Over Arbitrary HOL Theories
Constant Definitions	Yes (from the right)	Yes (Wenzel [31])	Yes (from below)
Constant Definitions Mixed with Type Definitions	**Yes (this paper)**	No (Wenzel [31])	Yes (Pitts [29])
Isabelle-HOL Constant Definitions	Yes (from the right)	Yes (Wenzel [31])	No (Wenzel [31])
Isabelle-HOL Constant Definitions Mixed with Type Definitions	**Yes (this paper)**	No (Wenzel [31])	No (from above)
	(Proof-Theoretic) Conservativity		Model-Theoretic Conservativity w.r.t. Standard Models

Fig. 1. Conservativity of Definitions in HOL and Isabelle/HOL

stronger than preservation of consistency, to be a minimum requirement for deeming a theory extension truly definitional [31, p.7]. In fact, he argues for an even stronger notion, *meta-safety*. Let Σ_1 and Σ_2 be the languages (signatures) of Θ_1 and Θ_2, respectively. (Thus, $\Sigma_1 \subseteq \Sigma_2$.) Meta-safety requires that, whenever an Σ_2-formula φ is deducible from Θ_2, there exists a Σ_1-formula $\varphi[\ldots, t/c, \ldots]$, obtained by replacing all the items $c \in \Sigma_2 \smallsetminus \Sigma_1$ with some suitable Σ_1-terms t, which is deducible from Θ_1. This way, the items c can be considered to be "defined" because they can always be compiled away without losing provability. He also shows that, under appropriate well-formedness restrictions, a set of constant definitions, even overloaded as in Isabelle/HOL, forms a meta-safe extension.

However, as formulated, meta-safety does not apply to type definitions, because in HOL it is impossible to replace a defined type with its defining expression. In fact, Wenzel makes the following observation: *In general, type definitions in HOL are not even consistency-preserving, let alone conservative (let alone meta-safe in any reasonable way)*, as witnessed by the following example. Consider the HOL theory consisting of a single formula stating that no type has precisely three elements, i.e, for all types α, if α has at most three elements x, y, z then two of them have to be equal:

$$\forall x, y, z : \alpha. \, (\forall v : \alpha. \, v = x \lor v = y \lor v = z) \longrightarrow \\ x = y \lor x = z \lor y = z$$

The theory is consistent since it is satisfied by any standard model of minimal HOL (where all finite types are function-space combinations over bool, hence their cardinality is a power of 2). On top of this theory, the extension with the definition of a type having three elements, $\tau = \{0, \mathsf{Suc}\, 0, \mathsf{Suc}(\mathsf{Suc}\, 0)\}$, is clearly inconsistent. This analysis has led Wenzel, who is Isabelle's long-standing lead developer and release manager, to deem type definitions *axiomatic* (i.e., having no consistency or conservativity guarantees attached) rather than definitional. This departure from a well-established HOL tradition has generated confusion and misunderstanding amongst Isabelle/HOL's users and developers [1].

But the above counterexample involves a non-definitional theory. Indeed, it shows that, unlike constant definitions, type definitions do not preserve consistency, a fortiori, are not conservative, *over an arbitrary (axiomatic) theory*. Nonetheless, it is still legitimate to ask:

> *Are arbitrary combinations of constant and type definitions conservative over minimal HOL? And are they even meta-safe (again, over minimal HOL) in a suitable sense?*

We believe these are important questions for deepening our understanding of the nature of HOL and Isabelle/HOL definitions. Conservativity also provides the most compelling way of witnessing consistency: Any proof of False using definitions can be reduced to a proof of False in minimal HOL (the latter being manifestly consistent thanks to its standard set-theoretic semantics). This is especially relevant for the brittle foundational terrain of Isabelle/HOL, where it should help rehabilitating type definitions as genuine, safe definitions.

In this paper, we provide a positive answer to both questions. Figure 2 shows our conservativity results in the context of similar known facts.

First, we focus on traditional HOL, where we formulate meta-safety by defining translation operators for types and terms that unfold the definitions (Section IV). Unfolding a type definition has to be done in an indirect fashion, since HOL does not support comprehension/refinement types (of the form $\{x : \sigma \mid t\ x\}$). Namely, a formula operating on defined types will be relativized to a formula on the original, built-in types that hosted the type definitions; so the "unfolding" of a defined type will be a predicate on its host type. Since type definitions are paired with nonemptiness proofs (in the current contexts, having available all the previously introduced definitions), we are forced to proceed gradually, one definition at a time. Consequently, the proof of meta-safety (also leading to conservativity) is itself gradual, in a feedback loop between preservation of deduction, commutation with substitution, and nonemptiness of the relativization predicates.

We organized the proof development for traditional HOL modularly, separating lemmas about termination of the definitional dependency relation. This allows a smooth upgrade to the more complex case of Isabelle/HOL (Section V), where termination is no longer ensured by the historic order of definitions, but by a more global approach. Due to ad hoc overloading, here the translations no longer commute with type substitution. We recover from this "anomaly" by mining the proofs and weakening the commutation lemma—leading to an Isabelle/HOL version of the results.

II. MORE RELATED WORK

There is a vast literature on the logical foundations of theorem provers, which we will not attempt to survey here. We focus on work that is directly relevant to our present contribution, from the point of view of either the object logic or the techniques used.

A. HOL Foundations

Wiedijk [32] defines *stateless HOL*, a version of HOL where terms and types carry *in their syntax* information about the defined constants and type constructors. Kumar et al. [18] define a set-theoretic (Pitts-style) model for stateless HOL and a translation from standard (stateful) HOL with definitions to stateless HOL, thus proving the consistency of both; the work itself is formalized in the HOL4 theorem prover. Their stateful to stateless HOL translation is similar to our translation, in that they both internalize the definitions (which are part of "the state") into "stateless" formulas; however, for of conservativity, we need to appeal to pure HOL entities, not to syntactically enriched ones.

Kumar et al.'s work is based on pioneering self-verification work by Harrison [14], who uses HOL Light to give semantic proofs of soundness of the HOL logic without definitional mechanisms, in two flavors: either after removing the infinity axiom from the object HOL logic, or after adding a "universe" axiom to HOL Light;

B. Isabelle/HOL Foundations

Wenzel's work cited in the introduction [31] proves meta-safety and conservativeness of constant definitions but leaves type definitions aside. In spite of Wenzel's theoretical observation that orthogonality and termination should ensure meta-safety, overloading of constants remains unchecked in Isabelle/HOL for many years—until Obua [25] looks into the problem and proposes a way to implement Wenzel's observation with an external termination checker. Obua also aims to extend the scope of consistency by by factoring in type definitions. But his syntactic proof misses out possible inconsistencies through delayed overloading intertwined with type definitions. Soon after, Wenzel designs and implements a more structural solution based on work of Haftmann, Obua and Urban (parts of which are reported in [12]).

Our own work on the foundations of Isabelle/HOL starts in 2014, after discovering the aforementioned inconsistencies caused by delayed overloading and type definitions. To address the problem, we define a new dependency relation, operating on constants *and types* (which is part of the system starting from Isabelle2016). We prove that, after these modifications, any definitional theory is consistent. In [20], we give a semantic proof by constructing a nonstandard, ground model based on a syntactic interpretation of polymorphism. In recent work [21], we give an alternative syntactic proof, based on translating HOL to a richer logic, HOLC, having comprehension types as first-class citizens. The current paper improves on these results, by proving properties much stronger than consistency.

C. Other Work

Concerning foundational work on provers outside the HOL family, Barras [7] gives a formal semantics of a fragment of Coq [8] and proves its consistency, and Myreen and Davis [22] do the same for Milawa, a prover based on first-order logic in the style of ACL2 [16]. Owre and Shankar develop the set-theoretic semantics of PVS [26], a logic similar to HOL, but different in that it has dependent types and lacks polymorphism.

Outside the world of theorem proving, conservative extensions are heavily employed in mathematical logic, e.g., in the very popular Henkin technique for proving completeness [15]. They are also employed in algebraic specifications to achieve desirable modularity properties [30]. However, in these fields, *definitional* extensions are often trivially conservative, thanks to their simple equational structure and freshness conditions.

III. HOL PRELIMINARIES

By HOL, we mean classical higher-order logic with Infinity, Choice and rank-one polymorphism, and mechanisms for constant and type definitions and declarations. This section explains all these concepts and features in detail.

A. Syntax

All throughout this paper, we fix the following:

- an infinite set TVar, of *type variables*, ranged by α, β
- an infinite set VarN, of *(term) variable names*, ranged by x, y, z

A *type structure* is a pair $(\mathsf{K}, \mathsf{tpOf})$ where:

- K is a set of symbols, ranged by k, called *type constructors*, containing three special symbols: "bool", "ind" and "$\Rightarrow$" (aimed at representing the type of booleans, an infinite type of individuals and the function type constructor, respectively)
- $\mathsf{arOf} : \mathsf{K} \Rightarrow \mathbb{N}$ is a function associating arities to the type constructors, such that $\mathsf{arOf}(\mathsf{bool}) = \mathsf{arOf}(\mathsf{ind}) = 0$ and $\mathsf{arOf}(\Rightarrow) = 2$.

The *types* associated to $(\mathsf{K}, \mathsf{arOf})$, ranged by σ, τ, are defined as follows:

$$\sigma \ ::= \ \alpha \ \mid \ (\sigma_1, \ldots, \sigma_{\mathsf{arOf}(k)})\, k$$

Thus, a type is either a type variable or an n-ary type constructor k postfix-applied to a number of types corresponding to its arity. We write $\mathsf{Type}_{(\mathsf{K},\mathsf{arOf})}$ for the set of types associated to $(\mathsf{K}, \mathsf{arOf})$.

A *signature* is a tuple $\Sigma = (\mathsf{K}, \mathsf{arOf}, \mathsf{Const}, \mathsf{tpOf})$, where:

- $(\mathsf{K}, \mathsf{arOf})$ is a type structure
- Const, ranged over by c, is a set of symbols called *constants*, containing five special symbols: "$\longrightarrow$", "$=$", "ε", "zero" and "suc" (aimed at representing logical implication, equality, Hilbert choice of some element from a type, zero and successor, respectively)
- $\mathsf{tpOf} : \mathsf{Const} \Rightarrow \mathsf{Type}$ is a function associating a type to every constant, such that:

$$\mathsf{tpOf}(\longrightarrow) = \mathsf{bool} \Rightarrow \mathsf{bool} \Rightarrow \mathsf{bool}$$
$$\mathsf{tpOf}(=) = \alpha \Rightarrow \alpha \Rightarrow \mathsf{bool} \quad \mathsf{tpOf}(\varepsilon) = (\alpha \Rightarrow \mathsf{bool}) \Rightarrow \alpha$$
$$\mathsf{tpOf}(\mathsf{zero}) = \mathsf{ind} \qquad \mathsf{tpOf}(\mathsf{suc}) = \mathsf{ind} \Rightarrow \mathsf{ind}$$

For the rest of this section, we fix a signature $\Sigma = (\mathsf{K}, \mathsf{arOf}, \mathsf{Const}, \mathsf{tpOf})$. We usually write Type_Σ, or simply Type, instead of $\mathsf{Type}_{(\mathsf{K},\mathsf{arOf})}$.

$\mathsf{TV}(\sigma)$ is the set of type variables of a type σ. A *type substitution* is a function $\rho : \mathsf{TVar} \Rightarrow \mathsf{Type}$. We let TSubst denote the set of type substitutions. The application of ρ to a type σ, written $\sigma[\rho]$, is defined recursively by $\alpha[\rho] = \rho(\alpha)$ and $((\sigma_1, \ldots, \sigma_m)\, k)[\rho] = (\sigma_1[\rho], \ldots, \sigma_m[\rho])\, k$. If $\alpha_1, \ldots, \alpha_m$ are all different, we write $\tau_1/\alpha, \ldots, \tau_n/\alpha_m$ for the type substitution that sends α_i to τ_i and each $\beta \notin \{\alpha_1, \ldots, \alpha_m\}$ to β. Thus, $\sigma[\tau_1/\alpha, \ldots, \tau_n/\alpha_m]$ is obtained from σ by substituting, for each i, τ_i for all occurrences of α_i.

We say that σ is an *instance* of τ *via* ρ, written $\sigma \leq_\rho \tau$, if $\tau[\rho] = \sigma$. We say that σ is an *instance* of τ, written $\sigma \leq \tau$, if there exists $\rho \in \mathsf{TSubst}$ such that $\sigma \leq_\rho \tau$. Two types σ_1 and σ_2 are called *orthogonal*, written $\sigma_1 \,\#\, \sigma_2$, if they have no common instance; i.e., for all τ it holds that $\tau \not\leq \sigma_1$ or $\tau \not\leq \sigma_2$.

Given $\rho_1, \rho_2 \in \mathsf{TSubst}$, we write $\rho_1 \cdot \rho_2$ for their *composition*, defined as $(\rho_1 \cdot \rho_2)(\alpha) = (\rho_1(\alpha))[\rho_2]$. It is easy to see that, 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 name x and a type σ, written x_σ. Let Var denote the set of variables. A *constant instance* is a pair of a constant and a type, written c_σ, such that $\sigma \leq \mathsf{tpOf}(c)$. We let CInst denote the set of constant instances. We extend the notions of being an instance ($\leq$) and being orthogonal (#) from types to constant instances:

$$\begin{array}{lll} c_\tau \leq d_\sigma & \text{iff} & c = d \text{ and } \tau \leq \sigma \\ c_\tau \,\#\, d_\sigma & \text{iff} & c \neq d \text{ or } \tau \,\#\, \sigma \end{array}$$

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_Σ, or simply Term, ranged by s and t, denote the set of terms. Typing is defined as a binary relation between terms and types, written $t : \sigma$, inductively as follows:

$$\frac{x_\sigma \in \mathsf{Var}}{x_\sigma : \sigma} \qquad \frac{c_\sigma \in \mathsf{CInst}}{c_\sigma : \sigma}$$

$$\frac{t_1 : \sigma \Rightarrow \tau \qquad t_2 : \sigma}{t_1\ t_2 : \tau} \qquad \frac{t : \tau}{\lambda x_\sigma.\ t : \sigma \Rightarrow \tau}$$

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 with the usual renaming of bounded variables if they get captured. $\mathsf{FV}(t)$ is the set of t's free variables. The term t is called *closed* if it has no free variables: $\mathsf{FV}(t) = \emptyset$. We write $t[s/x_\sigma]$ for the term obtained from t by capture-free substituting the term s for all free occurrences of x_σ.

A *formula* is a term of type bool. The formula connectives and quantifiers are defined in the usual way, starting from the implication and equality primitives—the appendix gives details. The if-then-else construct, if_t_e, is defined as follows, given $b : \mathsf{bool}$, $t_1 : \sigma$ and $t_2 : \sigma$

$$\mathsf{if\_t\_e}\ b\ t_1\ t_2 \ = \ \varepsilon\,(\lambda x_\sigma.\ (b \longrightarrow x_\sigma = t_1) \wedge (\neg\, b \longrightarrow x_\sigma = t_2))$$

Its behavior is the expected one: it equals t_1 if b is True and equals t_2 if b is False.

To avoid confusion with the object-logic definitions that we discuss later, we will treat all these as mere abbreviations (i.e., meta-level definitions of certain HOL terms). When writing terms, we sometimes omit the types of variables if they can be inferred. For example, we shall write $\lambda x_\sigma.\ x$ instead of $\lambda x_\sigma.\ x_\sigma$. A *theory* (over Σ) is a set of closed (Σ-)formulas.

B. *Axioms and Deduction*

The HOL axioms, forming the set Ax, are the usual Equality axioms, the Infinity axioms (stating that suc is different from 0 and is injective, which makes the type ind infinite), the classical Excluded Middle and the Choice axiom, which states that the Hilbert choice operator returns an element satisfying its argument predicate (if nonempty): $p_{\alpha\Rightarrow\mathsf{bool}}\ x \longrightarrow p\ (\varepsilon\ p)$.

A *context* Γ is a finite set of formulas. We write $\alpha \notin \Gamma$ to indicate that the type variable α does not appear in any formula from Γ; similarly, $x_\sigma \notin \Gamma$ will indicate that x_σ does not appear free in any formula from Γ. We define *deduction* as a ternary relation $\vdash$ between theories D, contexts Γ and formulas φ, written $D; \Gamma \vdash \varphi$.

$$\frac{}{D; \Gamma \vdash \varphi}\,(\text{FACT})\ [\varphi \in \mathsf{Ax} \cup D] \qquad \frac{}{D; \Gamma \vdash \varphi}\,(\text{ASSUM})\ [\varphi \in \Gamma]$$

$$\frac{D; \Gamma \vdash \varphi}{D; \Gamma \vdash \varphi[\sigma/\alpha]}\,(\text{T-INST})\ [\alpha \notin \Gamma] \qquad \frac{D; \Gamma \vdash \varphi}{D; \Gamma \vdash \varphi[t/x_\sigma]}\,(\text{INST})\ [x_\sigma \notin \Gamma]$$

$$\frac{}{D; \Gamma \vdash (\lambda x_\sigma.\ t)\ s = t[s/x_\sigma]}\,(\text{BETA})$$

$$\frac{D; \Gamma \vdash f\ x_\sigma = g\ x_\sigma}{D; \Gamma \vdash f = g}\,(\text{EXT})\ [x_\sigma \notin \Gamma]$$

$$\frac{D; \Gamma \cup \{\varphi\} \vdash \chi}{D; \Gamma \vdash \varphi \longrightarrow \chi}\,(\text{IMPI}) \qquad \frac{D; \Gamma \vdash \varphi \longrightarrow \chi \quad D; \Gamma \vdash \varphi}{D; \Gamma \vdash \chi}\,(\text{MP})$$

The axioms and the deduction rules we gave here are (a variant of) the standard ones for HOL (as in, e.g., [11], [14]). We write $D \vdash \varphi$ instead of $D; \emptyset \vdash \varphi$ and $\vdash \varphi$ instead of $\emptyset; \emptyset \vdash \varphi$ (that is, we omit empty contexts and theories). Note that the HOL axioms are not part of the parameter 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.

C. *HOL Definitions and Declarations*

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 $\longrightarrow$, $=$, ε, zero and suc. Since the built-in items have an already specified behavior (by the HOL axioms), only non-built-in items can be defined.

Def 1. Constant Definitions: Given a non-built-in constant c such that $\mathsf{tpOf}(c) = \sigma$ 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 $\mathsf{TV}(t) \subseteq \mathsf{TV}(c_\sigma)$ (i.e., $\mathsf{TV}(t) \subseteq \mathsf{TV}(\sigma)$).

Type Definitions: Given types τ and σ and a closed term $t : \sigma \Rightarrow \mathsf{bool}$, we let $\tau \equiv t$ denote the formula

$$\exists rep_{\tau \Rightarrow \sigma}.\ \mathsf{One\_One}_{rep} \wedge (\forall y_\sigma.\ t\ y \leftrightarrow (\exists x_\tau.\ y = rep\ x))$$

where $\mathsf{One\_One}_{rep}$ is the formula stating that *rep* is one-to-one (injective), namely, $\forall x_\tau, y_\tau.\ rep\ x = rep\ y \longrightarrow x = y$. We call $\tau \equiv t$ a *type definition*, provided τ has the form $(\alpha_1, \dots, \alpha_m)\, k$ such that k is a non-built-in type constructor and the α_i's are all distinct type variables and $\mathsf{TV}(t) \subseteq \{\alpha_1, \dots, \alpha_m\}$. (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 $(\alpha_1, \dots, \alpha_m)\, k$ is embedded in its host type σ via some one-to-one function *rep*, and the image of this embedding consists of the elements of σ for which t holds. 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. Thus, *to perform a type definition, one needs to give a nonemptiness proof.*

Type and Constant Declarations: Declarations in HOL are a logical extension mechanism which is significantly milder than definitions—they simply add new items to the signature as "uninterpreted," without proving any definition.

D. Signature Extensions and the Minimal Signature

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

Given a signature $\Sigma = (\mathsf{K}, \mathsf{arOf}, \mathsf{Const}, \mathsf{tpOf})$ and an item u, we write $u \in \Sigma$ to mean that $u \in \mathsf{K}$ or $u \in \mathsf{Const}$.

Given signatures $\Sigma = (\mathsf{K}, \mathsf{arOf}, \mathsf{Const}, \mathsf{tpOf})$ and $\Sigma' = (\mathsf{K}', \mathsf{arOf}', \mathsf{Const}', \mathsf{tpOf}')$, we say Σ *is included in* Σ', or Σ' *extends* Σ, written $\Sigma \subseteq \Sigma'$, if $\mathsf{K} \subseteq \mathsf{K}'$, $\mathsf{Const} \subseteq \mathsf{Const}'$ and the functions arOf' and tpOf' are extensions of arOf and tpOf, respectively. We write $u \in \Sigma' \smallsetminus \Sigma$ to mean $u \in \Sigma'$ and $u \notin \Sigma$. If $c \notin \mathsf{Const}$ and $\sigma \in \mathsf{Type}_\Sigma$, we write $\Sigma \cup \{(c, \sigma)\}$ for the extension of Σ with a new constant c of type σ. Similarly, if $k \notin \mathsf{K}$, we write $\Sigma \cup \{(k, n)\}$ for the extension of Σ with a new type constructor k of arity n.

We write Σ_{min} for the *minimal signature*, containing only built-in type constructors and constants. Note that, by definition, any signature extends the minimal signature.

IV. Conservativity of HOL Definitions

A HOL development, i.e., a session of interaction with the HOL logic from a user's perspective, consists of intertwining definitions, declarations and theorems. Since theorems are merely consequences of definitions, we will not model them explicitly, but focus on definitions and declarations.

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

Def 2. D is said to be a *well-formed definitional theory* if:

a) $D = \{def_1, \dots, def_n\}$ with each def_i being a (type or constant) definition of the form $u_i \equiv t_i$ and
b) there exist the signatures $\Sigma^1, \dots, \Sigma^n$ and $\Sigma_1, \dots, \Sigma_n$

such that $\Sigma_n = \Sigma$ and the following hold for all $i \in \{1, \dots, n\}$:

1) $t_i \in \mathsf{Term}_{\Sigma^i}$ and Σ_i is the extension of Σ^i with a fresh item defined by def_i, namely:
1.1) If u_i has the form $(\alpha_1, \dots, \alpha_m)\, k$, then $k \notin \Sigma^i$ and $\Sigma_i = \Sigma^i \cup \{(k, m)\}$
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 $t_i : \sigma \Rightarrow \mathsf{bool}$, it holds that $\{def_1, \dots, def_{i-1}\} \vdash_{\Sigma^i} \exists x_\sigma.\ t_i\ x$.
3) $\Sigma_i \subseteq \Sigma^{i+1}$ (for $i < n$)

These conditions express that the theory D consists of intertwined definitions and declarations. The chain of extensions

$$\Sigma_{\mathsf{min}} \subseteq \Sigma^1 \subseteq \Sigma_1 \subseteq \Sigma^2 \subseteq \Sigma_2 \ldots \subseteq \Sigma^n \subseteq \Sigma_n = \Sigma,$$

starting from the minimal signature and ending with Σ, alternates sets of declarations (the items in $\Sigma^i \setminus \Sigma_{i-1}$) with definitions (the unique item u_i in $\Sigma_i \setminus \Sigma^i$, defined by def_i, i.e., as $u_i \equiv t_i$). In the case of type definitions, we also require proofs of non-emptiness of the defining predicate t (from the definitions available so far).

Def 3. A theory E over Σ is said to be a *(proof-theoretic) conservative extension of minimal HOL* if any formula proved from E that belongs to the minimal signature Σ_{min} could have been proved without E or the types and constants outside of Σ. Formally: For all $\varphi \in \mathsf{Fmla}_{\Sigma_{\mathsf{min}}}$, $E \vdash_\Sigma \varphi$ implies $\vdash_{\Sigma_{\mathsf{min}}} \varphi$.

A. Roadmap

In what follows, we fix a well-formed definitional theory D and use for it the notations introduced in Def. 2, e.g., Σ, Σ_i. We first sketch the main ideas of our development, motivating the choice of the concepts. The more formal definitions and proofs will be given in the following subsections.

Our two main goals are to *formulate and prove D's meta-safety* and to *prove D's conservativity*. As with any respectable notion of its kind, meta-safety will easily yield conservativity, so we concentrate our efforts on the former.

Recall that, for a Σ-formula φ provable from D, meta-safety should allow us to replace all the defined items in φ with items in the minimal signature without losing provability, i.e., obtaining a deducible Σ_{min}-formula φ'. For constants, the procedure is clear: Any defined constant c appearing in φ is replaced with its defining term t, then any defined constant d appearing in t is replaced with its defining term, and so on, until (hopefully) the process terminates and we are left with built-in items only.

But how about for types τ occurring in φ? A HOL type definition $\tau \equiv t$ where $t : \sigma \Rightarrow \mathsf{bool}$, is not an equality (there is no type equality in HOL), but a formula asserting the existence of a bijection between τ and the set of elements of Σ for which the predicate t holds. So it cannot be "unfolded." First, let us make the simplifying assumption that $\sigma \in \mathsf{Type}_{\Sigma_{\mathsf{min}}}$ and $t \in \mathsf{Type}_{\Sigma_{\mathsf{min}}}$. Then the only reasonable Σ_{min}-substitute for τ is its host type σ; however, after the replacement of τ by σ, the formula needs to be adjusted not to refer to the whole σ, but only to the isomorphic copy of τ—in other words, the formula needs to be relativized to the predicate t. In general, σ or t may themselves contain defined types or constants, which will need to be processed similarly, and so on, recursively. In summary:

- for each type σ, we define its host type $\mathsf{HOST}(\sigma) \in \mathsf{Type}_{\Sigma_{\mathsf{min}}}$ and its relativization predicate on that type, $\mathsf{REL}(\sigma) : \mathsf{HOST}(\sigma) \Rightarrow \mathsf{bool}$ (where $\mathsf{REL}(\sigma) \in \mathsf{Term}_{\Sigma_{\mathsf{min}}}$)
- for each term $t : \sigma$, we define its unfolding $\mathsf{UNF}(t) : \mathsf{HOST}(\sigma)$ (where $\mathsf{UNF}(t) \in \mathsf{Term}_{\Sigma_{\mathsf{min}}}$)

For instances c_σ of constants $c : \tau$ defined by equations $c_\tau \equiv t$, $\mathsf{UNF}(c_\sigma)$ will be recursively defined as $\mathsf{UNF}(t[\rho])$ where ρ is the substitution that makes σ an instance of τ (i.e., $\sigma \leq_\rho \tau$). In other words, we unfold c_σ with the appropriately substituted equation defining c.

Since UNF is applied to arbitrary terms, not only to constants, we need to indicate its recursive behavior for all term constructs. Abstraction and application are handled as expected, but variables raise a subtle issue, with global implications on our overall proof strategy. What should $\mathsf{UNF}(x_\sigma)$ be? $x_{\mathsf{HOST}(\sigma)}$ is an immediate candidate. However, this will not work, since a crucial property that we will need about our translation is that it observes membership to types, in that it maps terms of a given type to terms satisfying that type's representing predicate:

(F1) The relativization predicates hold on translated items, i.e., $\mathsf{REL}(\sigma)\,\mathsf{UNF}(t)$ *is deducible (in minimal HOL) for each term* $t : \sigma$.

In particular, $\mathsf{REL}(\sigma)\,\mathsf{UNF}(x_\sigma)$ should be deducible. To enforce this, we define $\mathsf{UNF}(x_\sigma)$ to be either $x_{\mathsf{HOST}(\sigma)}$ if $\mathsf{REL}(\sigma)\,\mathsf{UNF}(x_\sigma)$ or else any item for which $\mathsf{REL}(\sigma)$ holds. This is expressible using the if-then-else and Choice operators: $\mathsf{if\_t\_e}\ (\mathsf{REL}(\sigma)\ x_{\mathsf{HOST}(\sigma)})\ x_{\mathsf{HOST}(\sigma)}\ (\varepsilon\,\mathsf{REL}(\sigma))$. By the Choice axiom, $\mathsf{REL}(\sigma)$ holds for $\varepsilon\,\mathsf{REL}(\sigma)$ just in case $\mathsf{REL}(\sigma)$ is nonempty. So to achieve the goal of ensuring $\mathsf{REL}(\sigma)$ holds for x_σ, we need:

(F2) The relativization predicates are nonempty, i.e., $\exists x_{\mathsf{HOST}(\sigma)}.\ \mathsf{REL}(\sigma)\ x$ *is deducible.*

Another way to look at this property is as a reflection of the HOL types being nonempty—a faithful relativization should of course follow suite.

Because of the way we apply these definitions recursively to the type and term constructs, the desired Σ_{min}-formula φ' corresponding to φ will be $\mathsf{UNF}(\varphi)$. For example, as one would expect, the unfoldings of $\forall x_\sigma.\ \varphi\ x$ and $\exists x_\sigma.\ \varphi\ x$ will be (deduction-equivalent to) $\forall x_{\mathsf{HOST}(\sigma)}.\ \mathsf{REL}(\sigma)\ x \longrightarrow \mathsf{UNF}(\varphi)\ x$ and $\exists x_{\mathsf{HOST}(\sigma)}.\ \mathsf{REL}(\sigma)\ x \wedge \mathsf{UNF}(\varphi)\ x$, respectively. Hence, for us meta-safety over minimal HOL will mean:

(MS) For all $\varphi \in \mathsf{Fmla}_\Sigma$, $D \vdash_\Sigma \varphi$ *implies* $\vdash_{\Sigma_{min}} \mathsf{UNF}(\varphi)$.

This property is indeed a type-aware version of what Wenzel calls meta-safety: $\mathsf{UNF}(\varphi)$ replaces each defined constant with a term as in Wenzel's concept, and replaces each defined type with a tandem of a host type and a relativization predicate.

To help proving (MS), we will also have lemmas about the good behavior of the translation functions HOST, UNF and REL with respect to the main ingredients of HOL deduction:

(F3) The translation functions preserve variable freshness and commute with substitution.

The order in which we will have to prove these facts has superficially circular dependencies. As discussed, we need (F2) for proving (F1). Moreover, (F1) is needed to prove (F3), more precisely, to make sure that UNF commutes with substitution for the delicate case of variables x_σ. In turn, (F3) is used for (MS). But to prove (F2), the nonemptiness of the relativization predicates, we seem to need (MS). Indeed, for the case of a type τ defined by $\tau \equiv t$ with $t : \sigma \Rightarrow \mathsf{bool}$, the natural choice for $\mathsf{REL}(\tau)$ is the conjunction of $\mathsf{REL}(\sigma)$ and $\mathsf{UNF}(t)$: gathering recursively whatever comes from the potential definition of σ or of its component types and adding the translation of τ's own defining predicate. So, in an inductive proof of (F2), we will need to deduce $\exists x_{\mathsf{HOST}(\sigma)}.\ \mathsf{REL}(\sigma)\ x \wedge \mathsf{UNF}(t)\ x$. The only fact that can help here is that this formula is (equivalent to) $\mathsf{UNF}(\varphi)$, where φ is $\exists x_\sigma.\ t\ x$. Since φ is the non-emptiness claim for the new type τ, it is deducible (according to Def. 2(2)). So we would like to apply (MS) here for obtaining that $\mathsf{UNF}(\varphi)$ is deducible.

In summary, we would need (F2) to prove (MS) and (MS) to prove (F2). The way out of this loop is a *gradual* approach: we will not define a single version of the translation functions, but one version, HOST_i, UNF_i and REL_i, for each subset $\{def_1, \ldots, def_i\}$ of D with $i \leq n$. This way, we can use (MS) for i to prove (F2) for $i+1$.

Finally, we will need to take into account a phenomenon we have ignored so far: the presence of *declarations* in addition to definitions. We cannot eliminate the declared (but not defined) constants and types, so it is reasonable to treat them similarly to the built-in items. In other words, in the statement (MS) of meta-safety we should replace Σ_{min} with a suitable signature Δ containing the declared items. Declarations can be intertwined with definitions, in particular, constants of *defined* types can be declared—so what we need is not only to collect all declared constants, but to also translate their types to the host types.

The rest of this section will unfold the ideas described above. First we illustrate the ideas by some examples, then we formally define and study the translations, culminating with proofs of meta-safety and conservativity.

B. Examples

We start with an extensive example that has only definitions, no declarations:

Example 4. Let Σ be the extension of the minimal signature with:

- the nullary type constructors nat and zfun
- the constants $\mathsf{absnat} : \mathsf{ind} \Rightarrow \mathsf{nat}$, $\mathsf{z} : \mathsf{nat}$ and $\mathsf{repzfun} : \mathsf{zfun} \Rightarrow (\mathsf{nat} \Rightarrow \mathsf{nat})$

Let $D = \{def_i \mid i \in \{1, \dots, 5\}\}$, such that:

- def_1 is $\mathsf{nat} \equiv t_1$, where $t_1 : \mathsf{ind} \Rightarrow \mathsf{bool}$ is a term in the minimal signature (namely, the predicate representing the intersection of all predicates that holds for 0 and are closed under Suc)
- def_2 is $\mathsf{absnat} \equiv t_2$, where t_2 is $\varepsilon\, t_2'$, with $t_2' : (\mathsf{ind} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{bool}$ a predicate (stating that its argument function is surjective and the inverse image of each natural contains an element that satisfies t_2)
- def_3 is $\mathsf{z} \equiv t_3$, where t_3 is $\mathsf{absnat}\ 0$
- def_4 is $\mathsf{zfun} \equiv t_4$, where $t_4 : (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{bool}$ is $\lambda f_{\mathsf{nat}\Rightarrow\mathsf{nat}}.\ f\ \mathsf{z} = \mathsf{z}$
- def_5 is $\mathsf{repzfun} \equiv t_5$, where t_5 is $\varepsilon\, t_5'$ with $t_5' : (\mathsf{zfun} \Rightarrow (\mathsf{nat} \Rightarrow \mathsf{nat})) \Rightarrow \mathsf{bool}$ a predicate stating that its argument is one-to-one and its image is included in t_4

Thus, there are no (non-defined but) declared items, and the chain $\Sigma_{\mathsf{min}} \subseteq \Sigma^1 \subseteq \Sigma_1 \subseteq \ldots \subseteq \Sigma^5 \subseteq \Sigma_5$ consists of the following signatures, where we omit repeating the arities and the types:

$$\begin{array}{ll} \Sigma^1 = \Sigma_{\mathsf{min}} & \Sigma^4 = \Sigma_3 = \Sigma^3 \cup \{\mathsf{z}\} \\ \Sigma^2 = \Sigma_1 = \Sigma^1 \cup \{\mathsf{nat}\} & \Sigma^5 = \Sigma_4 = \Sigma^4 \cup \{\mathsf{zfun}\} \\ \Sigma^3 = \Sigma_2 = \Sigma^2 \cup \{\mathsf{absnat}\} & \Sigma = \Sigma_5 = \Sigma^5 \cup \{\mathsf{repzfun}\} \end{array}$$

Incidentally, this example shows the standard procedure of bootstrapping natural numbers in HOL: The type nat is defined by carving out, from HOL's built-in infinite type ind, the smallest set closed under zero and successor. Using the Choice operator, we define the abstraction function absnat as a surjection that respects nat's defining predicate t_1. (The opposite injection can of course also be defined, but is omitted here.) The version of zero for naturals, $\mathsf{z} : \mathsf{nat}$, is defined by applying the abstraction to the built-in zero from ind.

Subsequently, another type is introduced, zfun, of zero-preserving functions between naturals, defined by carving out from the type $\mathsf{nat} \Rightarrow \mathsf{nat}$ the set of those functions that map z to z. For this type, we define the representation function $\mathsf{repzfun}$ to its defining type $\mathsf{nat} \Rightarrow \mathsf{nat}$. Note that the way to apply an element of zfun to a natural is to apply its representation.

We will focus on evaluating $\mathsf{UNF}(f_{\mathsf{zfun}})$, where we write UNF for the last (widest-reaching) unfolding function UNF_5 (and similarly for HOST and REL). As discussed, since f_{zfun} is a variable, $\mathsf{UNF}(f_{\mathsf{zfun}})$ will be a term for which $\mathsf{REL}(\mathsf{zfun})$ is guaranteed to hold (provided the predicate in nonempty): $\mathsf{if\_t\_e}\ \ (\mathsf{REL}(\mathsf{zfun})\ f_{\mathsf{HOST}(\mathsf{zfun})})\ \ f_{\mathsf{HOST}(\mathsf{zfun})}\ \ (\varepsilon\, \mathsf{REL}(\mathsf{zfun}))$. Now, looking at the types in definitions def_1 and def_4, we can compute the host of zfun:

$$\mathsf{HOST}(\mathsf{zfun}) \ = \ \mathsf{HOST}(\mathsf{nat} \Rightarrow \mathsf{nat}) \ = \\ \mathsf{HOST}(\mathsf{nat}) \Rightarrow \mathsf{HOST}(\mathsf{nat}) \ = \ \mathsf{ind} \Rightarrow \mathsf{ind}$$

Thus, $\mathsf{REL}(\mathsf{zfun})$ is a predicate on $\mathsf{ind} \Rightarrow \mathsf{ind}$. But what does it say? To evaluate $\mathsf{REL}(\mathsf{zfun})$, we again look at the definitions def_1 and def_4, this time also factoring in their terms, t_1 and t_4:

$$\begin{array}{l} \mathsf{REL}(\mathsf{zfun}) = \\ \lambda g_{\mathsf{HOST}(\mathsf{zfun})}.\ \mathsf{REL}(\mathsf{nat} \Rightarrow \mathsf{nat})\ g \ \wedge\ \mathsf{UNF}(t_4)\ g = \\ \lambda g_{\mathsf{HOST}(\mathsf{zfun})}.\ (\mathsf{REL}(\mathsf{nat}) \Rrightarrow \mathsf{REL}(\mathsf{nat}))\ g \ \wedge\ \mathsf{UNF}(t_4)\ g = \\ \lambda g_{\mathsf{ind}\Rightarrow\mathsf{ind}}.\ (\mathsf{UNF}(t_1) \Rrightarrow \mathsf{UNF}(t_1))\ g \ \wedge\ \mathsf{UNF}(t_4)\ g \\ \lambda g_{\mathsf{ind}\Rightarrow\mathsf{ind}}.\ (t_1 \Rrightarrow t_1)\ g \ \wedge\ \mathsf{UNF}(t_4)\ g \end{array}$$

where, for a predicate such as $t_1 : \mathsf{ind} \Rightarrow \mathsf{bool}$, $t_1 \Rrightarrow t_1$ denotes its lifting to functions: $\lambda g_{\mathsf{ind}\Rightarrow\mathsf{ind}}.\ \forall x_{\mathsf{ind}}.\ t_1\ x \longrightarrow t_1\ (g\ x)$. (Note that $\mathsf{UNF}(t_1) = t_1$ since t_1 is in the minimal signature.) Thus, $\mathsf{REL}(\mathsf{zfun})\ g_{\mathsf{ind}\Rightarrow\mathsf{ind}}$ states that g preserves t_1 (the isomorphic image of nat in ind) and that $\mathsf{UNF}(t_4)\ g$ holds, where t_4 is the isomorphic image of zfun in $\mathsf{nat} \Rightarrow \mathsf{nat}$. This shows how, when evaluating REL, nested type definitions lead to the accumulation of their defining predicates, each lifted if necessary along the encountered function-space structure.

We can prove $D \vdash_\Sigma \varphi$, where φ is $\forall f_{\mathsf{zfun}}.\ \mathsf{repzfun}\ f_{\mathsf{zfun}}\ \mathsf{z} = \mathsf{z}$ with f_{zfun} a variable, i.e., that the items in zfun indeed map zero to zero. By our meta-safety result, we will infer $\vdash_{\Sigma_{\mathsf{min}}} \mathsf{UNF}(\varphi)$, which boils down to a tautology: that all functions from ind to ind that preserve the natural-number-predicate and preserve 0 also preserve 0.

We conclude with an example showing how declarations affect the target signature:

Example 5. Consider the following extension of Example 4: After def_4, a declaration of a constant $c : \mathsf{zfun}$ is performed. Thus, Σ^5 is no longer equal to Σ_4, but is $\Sigma_4 \cup \{(c, \mathsf{zfun})\}$.

What should be the signature of $\mathsf{UNF}\ (c_{\mathsf{zfun}})$? Since c has no definition, it will not be compiled away by unfolding. However, we are required to compile away its type zfun, which is a defined type. So it is natural to have $\mathsf{UNF}(c_{\mathsf{zfun}}) = c_{\mathsf{HOST}(\mathsf{zfun})} = c_{\mathsf{ind}\Rightarrow\mathsf{ind}}$. However, none of the existing signatures contains a constant $c : \mathsf{ind} \Rightarrow \mathsf{ind}$.

Consequently, we need to create a signature Δ that extends Σ_{min} with all the declared constants but having HOST-translated types, and, similarly, with all the declared type constructors. In general, the translations will target this signature rather than Σ_{min}.

C. Formal Definition of the Translations and Meta-Safety

We will write D_i for the current definitional theory at moment i, $\{def_1, \dots, def_i\}$. Thus, we have $D = D_n$. As discussed, we will define deduction-preserving translations of the Σ-types and Σ-terms into Δ-types and Δ-terms, where Δ will be a suitable signature that collects all the declared items. We proceed gradually, considering Σ_i one i at a time, eventually reaching $\Sigma = \Sigma_n$.

For each $i \in \{1, \dots, n\}$, we define the signature Δ^i (collecting the declared items from Σ^i with their types translated to their host types), together with the function $\mathsf{HOST}_i : \mathsf{Type}_{\Sigma_i} \Rightarrow \mathsf{Type}_{\Delta^i}$ (producing the host types) as follows:

- Δ^1 is Σ^1
- Δ^{i+1} is Δ^i extended with:
  - all the type constructors $k \in \Sigma^{i+1} \setminus \Sigma_i$
  - for all constants $c \in \Sigma^{i+1} \setminus \Sigma_i$ of type σ, a constant c of type $\mathsf{HOST}_i(\sigma)$
- HOST_i is defined as in Fig. 2, recursively on types

On defined types (i.e., types having a defined type constructor on top, clause (H3)), HOST_i behaves as prescribed in Section IV-A, recursively calling itself for the defining type. Upon encountering built-in or declared type constructors, i.e.,

(H1) $\mathsf{HOST}_i(\alpha) = \alpha$
(H2) $\mathsf{HOST}_i((\sigma_1,\ldots,\sigma_m)\,k) = (\mathsf{HOST}_i(\sigma_1),\ldots,\mathsf{HOST}_i(\sigma_m))\,k$
if $k \in \Sigma^1 \cup \bigcup_{i'=2}^{i}(\Sigma^{i'} \smallsetminus \Sigma_{i'-1})$
(H3) $\mathsf{HOST}_i((\sigma_1,\ldots,\sigma_m)\,k) = \mathsf{HOST}_i(\sigma[\sigma_1/\alpha_1,\ldots,\sigma_m/\alpha_m])$
if $(\alpha_1,\ldots,\alpha_m)\,k \equiv t$ is in D_i and $t : \sigma \Rightarrow \mathsf{bool}$

(U1) $\mathsf{UNF}_i(x_\sigma) = \mathsf{if\_t\_e}\,(\mathsf{REL}_i(\sigma)\,x_{\mathsf{HOST}_i(\sigma)})\,x\,(\varepsilon\,\mathsf{REL}_i(\sigma))$
(U2) $\mathsf{UNF}_i(c_\sigma) = c_{\mathsf{HOST}_i(\sigma)}$ if $c \in \Sigma^1 \cup \bigcup_{i'=2}^{i}(\Sigma^{i'} \smallsetminus \Sigma_{i'-1})$
(U3) $\mathsf{UNF}_i(c_\sigma) = \mathsf{UNF}_i(t[\rho])$
if $c_\tau \equiv t$ is in D_i and $\sigma \leq_\rho \tau$
(U4) $\mathsf{UNF}_i(t_1\,t_2) = \mathsf{UNF}_i(t_1)\,\mathsf{UNF}_i(t_2)$
(U5) $\mathsf{UNF}_i(\lambda x_\sigma.\,t) = \lambda x_{\mathsf{HOST}_i(\sigma)}.\,\mathsf{UNF}_i(t)$

(P1) $\mathsf{REL}_i(\sigma) = \lambda x_\sigma.\,\mathsf{True}$ if $\sigma \in \mathsf{TVar} \cup \{\mathsf{bool},\mathsf{inf}\}$
(P2) $\mathsf{REL}_i(\sigma_1 \Rightarrow \sigma_2) =$
$\lambda f_{\mathsf{HOST}_i(\sigma_1) \Rightarrow \mathsf{HOST}_i(\sigma_2)}.$
$\forall x_{\mathsf{HOST}_i(\sigma_1)}.\,\mathsf{REL}_i(\sigma_1)\,x \longrightarrow \mathsf{REL}_i(\sigma_2)\,(f\,x)$
(P3) $\mathsf{REL}_i((\sigma_1,\ldots,\sigma_m)\,k) =$
$\lambda x_{(\mathsf{HOST}_i(\sigma_1),\ldots,\mathsf{HOST}_i(\sigma_m))\,k}.\mathsf{True}$
if $k \in (\Sigma^1 \smallsetminus \Sigma_{\mathsf{min}}) \cup \bigcup_{i'=2}^{i}(\Sigma^{i'} \smallsetminus \Sigma_{i'-1})$
(P4) $\mathsf{REL}_i((\sigma_1,\ldots,\sigma_m)k) =$
$\lambda x_{\mathsf{HOST}_i(\sigma')}.\,\mathsf{REL}_i(\sigma')\,x \wedge \mathsf{UNF}_i(t')\,x,$
if $(\alpha_1,\ldots,\alpha_m)\,k \equiv t$ is in D_i and $t : \sigma \Rightarrow \mathsf{bool}$,
where $\sigma' = \sigma[\sigma_1/\alpha_1,\ldots,\sigma_m/\alpha_m]$
and $t' = t[\sigma_1/\alpha_1,\ldots,\sigma_m/\alpha_m]$

Fig. 2. Definition of the translation functions

belonging to some $\Sigma^{i'}$ for $i' \leq i$, bot not to the corresponding $\Sigma_{i'-1}$ (clause (H2)), HOST_i delves into the subexpressions.

Next, *mutually* recursively on Σ_i-types and Σ_i-terms, we define a function returning the relativization predicate of a type, $\mathsf{REL}_i : \mathsf{Type}_{\Sigma_i} \to \mathsf{Term}_{\Delta^i}$, and one returning the unfolded term, $\mathsf{UNF}_i : \mathsf{Term}_{\Sigma_i} \to \mathsf{Term}_{\Delta^i}$. Their definition is shown in Fig. 2. Again, they behave as prescribed in Section IV-A. In particular, REL_i is naturally lifted to function spaces (clause (P2)) and accumulates defining predicates, as shown in clause (P4)—here, the substitution $\sigma_1/\alpha_1,\ldots,\sigma_m/\alpha_m$ stems from an instance of the defined type, $(\alpha_1,\ldots,\alpha_m)\,k$. Type variables and declared types are treated as black boxes, so REL_i is vacuously true for them, just like for the built-in types bool and ind (clauses (P1) and (P3)). Note that, while (H2) refers to declared or built-in type constructors, (P3) only refers to declared ones—it explicitly excludes Σ_{min}.

As discussed in Section IV-A, UNF_i treats type variables in a "guarded" fashion (clause (U1)), and distributes over application and abstraction (clauses (U4) and (U5)). Moreover, UNF_i calls HOST_i for declared or built-in constants (clause (U2)). Finally, UNF_i unfolds the definitions of defined constants, as shown in clause (U3). In that clause, c_τ and $\rho\!\restriction_{\mathsf{TV}(c_\tau)}$ (the restriction of ρ to $\mathsf{TV}(c_\tau)$) are uniquely determined by c_σ; and since $\mathsf{TV}(t) \subseteq \mathsf{TV}(c_\sigma)$ (by Def. 1), it follows that $t[\rho]$ is also uniquely determined by c_σ.

Obviously, these functions can reach their purpose only if they are well defined, i.e., are total functions. i.e., their recursive evaluation process terminates for all inputs. This is what we prove in the next subsection.

Assuming well-definedness, we have all the prerequisites to formulate meta-safety. We let UNF be UNF_n, the function that unfolds all definitions in $D = D_n$, and Δ be Δ^n, the signature collecting all the declared items in Σ.

Def 6. D is said to be a *meta-safe extension of HOL-with-declarations* if, for all $\varphi \in \mathsf{Fmla}_\Delta$, $D \vdash_\Sigma \varphi$ implies $\vdash_\Delta \mathsf{UNF}(\varphi)$.

D. Well-Definedness of the Translations

The goal of this subsection is to prove:

Prop 7. (1) The function HOST_i is well defined, i.e., its recursive calls terminate.
(2) The functions REL_i and UNF_i are well defined, i.e., their mutually recursive calls terminate.

The concepts we use in the proof of this proposition, in particular, the definitional dependency relation, will be also relevant in Section V, when we attend to Isabelle/HOL.

To prove (1), we need to show that the call graph of HOST_i, namely, the relation $\blacktriangleright\!\!\!\blacktriangleright_i$ defined by:

$$(\sigma_1,\ldots,\sigma_m)\,k \blacktriangleright\!\!\!\blacktriangleright_i \sigma_j \text{ if } k \in \Sigma^i$$
$$(\sigma_1,\ldots,\sigma_m)\,k \blacktriangleright\!\!\!\blacktriangleright_i \sigma[\sigma_1/\alpha_1,\ldots,\sigma_m/\alpha_m]$$
$$\text{if } (\alpha_1,\ldots,\alpha_m)\,k \equiv t \text{ is in } D_i \text{ and } t : \sigma \Rightarrow \mathsf{bool}$$

is terminating. This is easily done by defining a lexicographic order based on the order in which the items were defined, i.e., the indexes of the definitions def_i in which they appear (details in the appendix).

To prove (2), we will exhibit a terminating relation $\blacktriangleright_i$ that captures the mutual call graph of REL_i and UNF_i. We take $\blacktriangleright_i$ to be the union $\equiv_i^{\downarrow} \cup \rhd$, where $\equiv_i^{\downarrow}$ and $\rhd$ are defined below. The relation $\rhd$ consists of the structurally recursive calls of REL_i and UNF_i, from clauses (P2), (U1), (U4) and (U5):

$$\sigma_1 \Rightarrow \sigma_2 \rhd \sigma_1 \qquad \sigma_1 \Rightarrow \sigma_2 \rhd \sigma_2$$
$$x_\sigma \rhd \sigma \qquad t_1 t_2 \rhd t_1 \qquad t_1 t_2 \rhd t_2 \qquad \lambda x_\sigma.\,t \rhd t$$

Moreover, $\equiv_i^{\downarrow}$ captures the recursive calls corresponding to defined items, from (P4) and (U3). Given $u, v \in \mathsf{Type}_{\Sigma_i} \cup \mathsf{Term}_{\Sigma_i}$, $u \equiv_i^{\downarrow} v$ states that there exists a definition $u' \equiv v'$ in D_i and a type substitution ρ such that $u = \rho(u')$ and $v = \rho(v')$.

Thus, the well-definedness of REL_i and UNF_i is reduced to the termination of $\blacktriangleright_i$. In order to prove the latter, we will introduce a more basic relation: the dependency relation between non-built-in items introduced by definitions in D_i. We let $\mathsf{Type}^\bullet_{\Sigma_i}$ be the set of Σ_i-types that have a non-built-in type constructor at the top, and $\mathsf{CInst}^\bullet_{\Sigma_i}$ be the set of instances of non-built-in constants. Given any term t, we let $\mathsf{types}^\bullet(t)$ be the set of all types from $\mathsf{Type}^\bullet_{\Sigma_i}$ appearing in t and $\mathsf{cinsts}^\bullet(t)$ be the set of all constant instances from $\mathsf{CInst}^\bullet_{\Sigma_i}$ appearing in t. (The appendix gives the formal definition of these operators.)

Def 8. The *dependency relation* $\rightsquigarrow_i$ on $\mathsf{Type}^{\bullet}_{\Sigma_i} \cup \mathsf{CInst}^{\bullet}_{\Sigma_i}$ is defined as follows: $u \rightsquigarrow_i v$ iff there exists in D_i a definition of the form $u \equiv t$ such that $v \in \mathsf{cinsts}^{\bullet}(t) \cup \mathsf{types}^{\bullet}(t)$.

We write $\rightsquigarrow_i^{\downarrow}$ for the (type-)substitutive closure of $\rightsquigarrow_i$, defined as follows: $u \rightsquigarrow_i^{\downarrow} v$ iff there exist u', v' and a type substitution ρ such that $u = u'[\rho]$, $v = v'[\rho]$ and $u' \rightsquigarrow_i v'$. Since HOL with definitions is well-known to be consistent, one would expect that definitions cannot introduce infinite (including cyclic) chains of dependencies. This can indeed be proved by a lexicographic argument, again taking advantage of the definitional order:

Lemma 9. The relation $\rightsquigarrow_i^{\downarrow}$ is terminating.

The next observation connects $\blacktriangleright_i$ and $\rightsquigarrow_i^{\downarrow}$, via $\rhd^*$ (the transitive closure of $\rhd$):

Lemma 10. If $u, v \in \mathsf{Type}^{\bullet}_{\Sigma_i} \cup \mathsf{CInst}^{\bullet}_{\Sigma_i}$ and $u \equiv_i^{\downarrow} t \rhd^* v$, then $u \rightsquigarrow_i^{\downarrow} v$

Now we can reduce the termination of $\blacktriangleright_i$ to that of $\rightsquigarrow_i^{\downarrow}$, hence prove the former:

Lemma 11. The relation $\blacktriangleright_i$ is terminating.

This concludes the proof of Prop. 7.

E. Basic Properties of the Translations

As envisioned in Section IV-A the translations are extensions of each other and preserve type membership:

Lemma 12. Assume $i \leq n-1$. The following hold:
(1) If $\sigma \in \mathsf{Type}_{\Sigma_i}$, then $\mathsf{HOST}_{i+1}(\sigma) = \mathsf{HOST}_i(\sigma)$
(2) If $\sigma \in \mathsf{Type}_{\Sigma_i}$, then $\mathsf{REL}_{i+1}(\sigma) = \mathsf{REL}_i(\sigma)$.
(3) If $t \in \mathsf{Term}_{\Sigma_i}$, then $\mathsf{UNF}_{i+1}(t) = \mathsf{UNF}_i(t)$.

Lemma 13. If $\sigma \in \mathsf{Type}_{\Sigma_i}$, $t \in \mathsf{Type}_{\Sigma_i}$ and $t : \sigma$, then $\mathsf{REL}_i(\sigma) : \mathsf{HOST}_i(\sigma) \Rightarrow \mathsf{bool}$ and $\mathsf{UNF}_i(t) : \mathsf{HOST}_i(\sigma)$.

For items in the minimal signature, the behavior of the translations is idle (HOST_i and UNF_i) or trivial (REL_i):

Lemma 14. The following hold:
(1) If $\sigma \in \mathsf{Type}_{\Sigma_{\mathsf{min}}}$, then $\mathsf{HOST}_i(\sigma) = \sigma$
(2) If $\sigma \in \mathsf{Type}_{\Sigma_{\mathsf{min}}}$, then $\vdash_{\Sigma_{\mathsf{min}}} \mathsf{REL}_i(\sigma) = \lambda x_{\mathsf{HOST}_i(\sigma)}.\, \mathsf{True}$
(3) If $t \in \mathsf{Term}_{\Sigma_{\mathsf{min}}}$ and t is well-typed, then $\vdash_{\Sigma_{\mathsf{min}}} \mathsf{UNF}_i(t) = t$

Other easy, but important properties state that the translations do not introduce new variables or type variables and commute with *type* substitution:

Lemma 15. The following hold for all $\sigma \in \mathsf{Type}_{\Sigma_i}$ and $t \in \mathsf{Term}_{\Sigma_i}$:
(1) $\mathsf{TV}(\mathsf{HOST}_i(\sigma)) \subseteq \mathsf{TV}(\sigma)$
(2) $\mathsf{TV}(\mathsf{REL}_i(\sigma)) \subseteq \mathsf{TV}(\sigma)$ and $\mathsf{FV}(\mathsf{REL}_i(\sigma)) = \emptyset$
(3) $\mathsf{TV}(\mathsf{UNF}_i(t)) \subseteq \mathsf{TV}(t)$ and $\mathsf{FV}(\mathsf{UNF}_i(t)) = \{x_{\mathsf{HOST}_i(\sigma)} \mid x_\sigma \in \mathsf{FV}(t)\}$

Lemma 16. The following hold for all $\sigma, \tau \in \mathsf{Type}_{\Sigma_i}$ and $t \in \mathsf{Term}_{\Sigma_i}$:
(1) $\mathsf{HOST}_i(\sigma[\tau/\alpha]) = \mathsf{HOST}_i(\sigma)[\mathsf{HOST}_i(\tau)/\alpha]$
(2) $\mathsf{REL}_i(\sigma[\tau/\alpha]) = \mathsf{REL}_i(\sigma)[\mathsf{HOST}_i(\tau)/\alpha]$
(3) $\mathsf{UNF}_i(t[\tau/\alpha]) = \mathsf{UNF}_i(t)[\mathsf{HOST}_i(\tau)/\alpha]$

F. Main Results

We are now ready to finalize the plan set out in Section IV-A. The following facts in Lemma 17 are stated and proved in the delicate order prescribed there. Fact (4) corresponds to part of (F3) (the remaining parts being covered by Lemmas 15 and 16). Moreover, (2) corresponds to (F2), (3) to (F1), and (5) to (MS). Finally, (1) states deducibility of the translated nonemptiness statement, identified in Section IV-A as an intermediate fact leading to (F2) from (MS).

Lemma 17. Let $i \in \{1, \ldots, n\}$. The following hold for all $\sigma, \tau \in \mathsf{Type}_{\Sigma_i}$, $t, t' \in \mathsf{Term}_{\Sigma_i}$ and $\varphi \in \mathsf{Fmla}_{\Sigma_i}$:
(1) If $\tau \equiv t$ is a type definition in D_i with $t : \sigma \Rightarrow \mathsf{bool}$, then $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma)}.\, \mathsf{REL}_i(\sigma)\ x \wedge \mathsf{UNF}_i(t)\ x$
(2) $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma)}.\, \mathsf{REL}_i(\sigma)\ x$
(3) If $t : \sigma$, then $\vdash_{\Delta^i} \mathsf{REL}_i(\sigma)\ \mathsf{UNF}_i(t)$
(4) If $t' : \sigma$, then
$\vdash_{\Delta^i} \mathsf{UNF}_i(t[t'/x_\sigma]) = \mathsf{UNF}_i(t)[\mathsf{UNF}_i(t')/x_{\mathsf{HOST}_i(\sigma)}]$
(5) If $D_i \vdash_{\Sigma_i} \varphi$, then $\vdash_{\Delta^i} \mathsf{UNF}_i(\varphi)$

Proof. The facts follow by induction on i. More precisely, let $(\mathrm{j})_i$ denote fact (j) for a given layer i. We prove:

- that $(1)_1$ holds;
- that, for any $i \in \{1, \ldots, n\}$:
  - $(1)_i$ implies $(2)_i$ implies $(3)_i$ implies $(4)_i$;
  - $((2)_i$ and $(4)_i$ imply $(5)_i$;
- that, for any $i \in \{1, \ldots, n-1\}$, $(5)_i$ implies $(1)_{i+1}$.

We only show proof sketches for the two most crucial of these implications. (The appendix discusses the others.)

$(1)_i$ implies $(2)_i$: Assuming $(1)_i$, we prove $(2)_i$ by structural induction on σ. The only interesting case is when the type is defined, i.e., has a defined type constructor on top (dealt with in clause (P4)). We need to show $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma')}.\, \mathsf{REL}_i(\sigma')\ x \wedge \mathsf{UNF}_i(t')\ x$, where $(\alpha_1, \ldots, \alpha_m)\ k \equiv t$ is in D_i and $t : \sigma \Rightarrow \mathsf{bool}$, $\sigma' = \sigma[(\sigma_j/\alpha_j)_j]$, and $t' = t[(\sigma_j/\alpha_j)_j]$.

By $(1)_i$, we have $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma)}.\, \mathsf{REL}_i(\sigma)\ x \wedge \mathsf{UNF}_i(t)\ x$. By the type substitution rule (T-INST) applied m times (once for each $\mathsf{HOST}_i(\sigma_j)/\alpha_j$), we have $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma)[(\mathsf{HOST}_i(\sigma_j)/\alpha_j)_j]}.\, \mathsf{REL}_i(\sigma)[(\mathsf{HOST}_i(\sigma_j)/\alpha_j)_j]\ x \wedge \mathsf{UNF}_i(t)[(\mathsf{HOST}_i(\sigma_j)/\alpha_j)_j]\ x$. Using Lemma 16 m times (once for each σ_j/α_j), we obtain $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma[(\sigma_j/\alpha_j)_j])}.\, \mathsf{REL}_i(\sigma[(\sigma_j/\alpha_j)_j])\ x \wedge \mathsf{UNF}_i(t[(\sigma_j/\alpha_j)_j])\ x$, i.e., $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma')}.\, \mathsf{REL}_i(\sigma')\ x \wedge \mathsf{UNF}_i(t')\ x$, as desired.

$(2)_i$ and $(4)_i$ imply $(5)_i$: Assume $(2)_i$ and $(4)_i$. By rule induction on the definition of HOL deduction ($\vdash$), we prove a slight generalization of $(5)_i$, namely: We assume $\Gamma \cup \{\varphi\} \subseteq \mathsf{Fmla}_{\Sigma_i}$ and $D_i; \Gamma \vdash_{\Sigma_i} \varphi$, and prove $\emptyset; \mathsf{UNF}_i(\Gamma) \vdash_{\Delta^i} \mathsf{UNF}_i(\varphi)$. We distinguish different cases, according to the last applied rule in inferring $\Gamma \cup \{\varphi\} \subseteq \mathsf{Fmla}_{\Sigma_i}$:

(FACT): We need to prove $\emptyset; \mathsf{UNF}_i(\Gamma) \vdash_{\Delta^i} \mathsf{UNF}_i(\varphi)$, assuming $\varphi \in \mathsf{Ax} \cup D_i$. First, assume $\varphi \in D$. Then $\varphi = u \equiv t \in D_i$. We have two subcases:

(A) u is a constant c_σ. Then $\mathsf{UNF}_i(\varphi)$ is the formula $\mathsf{UNF}_i(c_\sigma) = \mathsf{UNF}_i(t)$. And since $\mathsf{UNF}_i(c_\sigma)$ and $\mathsf{UNF}_i(t)$ are (syntactically) equal, the desired fact follows by the HOL reflexivity rule.

(B) u is a type τ of the form $(\alpha_1, \dots, \alpha_m)\ k$ and $t : \sigma \Rightarrow \mathsf{bool}$. Then, by the definition of UNF_i and of the $\forall$ and $\exists$ constructs, $\mathsf{UNF}_i(\varphi)$ is deduction-equivalent to the formula

$$
\begin{array}{l}
\exists rep_{\mathsf{HOST}_i(\sigma)\Rightarrow\mathsf{HOST}_i(\sigma)}.\\
\quad(\forall x_{\mathsf{HOST}_i(\sigma)}.\ \mathsf{REL}_i(\sigma)\ x \wedge \mathsf{UNF}_i(t)\ x \longrightarrow \mathsf{REL}_i(\sigma)\ (rep\ x))\\
\quad\wedge\\
\quad\forall x_{\mathsf{HOST}(\sigma)}, y_{\mathsf{HOST}(\sigma)}.\\
\qquad\mathsf{REL}_i(\sigma)\ x \wedge \mathsf{UNF}_i(t)\ x \wedge \mathsf{REL}_i(\sigma)\ y \wedge\\
\qquad\mathsf{UNF}_i(t)\ y \wedge rep\ x = rep\ y \longrightarrow x = y\\
\quad\wedge\\
\quad(\forall y_{\mathsf{HOST}_i(\sigma)}.\\
\qquad\mathsf{REL}_i(\sigma)\ y \longrightarrow\\
\qquad(\mathsf{UNF}_i(t)\ y \longleftrightarrow\\
\qquad\ (\exists x_{\mathsf{HOST}_i(\sigma)}.\ \mathsf{REL}_i(\sigma)\ x \wedge \mathsf{UNF}_i(t)\ x \wedge y = rep\ x)))
\end{array}
$$

where the first conjunct comes from the relativization of $\tau \Rightarrow \sigma$, the second from unfolding $\mathsf{One\_One}_{rep}$, and the third from unfolding $\forall y_\sigma.\ t\ y \longleftrightarrow (\exists x_\tau.\ y = rep\ x)$ (in Def. 1). This states the following (in a verbose fashion): There exists $rep : \mathsf{HOST}_i(\sigma) \Rightarrow \mathsf{HOST}_i(\sigma)$ which is one-to-one on the intersection of $\mathsf{REL}_i(\sigma)$ and $\mathsf{UNF}_i(t)$ and the image of this intersection through rep is the intersection itself. This is of course deducible in HOL, taking rep as the identity function.

Now, assume $\varphi \in \mathsf{Ax}$. Then $\varphi \in \mathsf{Fmla}_{\Sigma_{\min}}$, hence, by Lemma 14(3), $\vdash_{\Delta^i} \mathsf{UNF}_i(\varphi) = \varphi$. And since also $\emptyset; \mathsf{UNF}_i(\Gamma) \vdash_{\Delta^i} \varphi$ is true by (FACT), the desired fact follows using the HOL equality rules.

(ASSUM): Follows by applying (ASSUM).

(T-INST): Courtesy of UNF_i commuting with type substitution (Lemma 16(3)) and preserving freshness (Lemma 15(3)).

(INST): Courtesy of UNF_i commuting with substitution (point $(4)_i$) and preserving freshness (Lemma 15(3)).

(BETA), (EXT), (IMPI) and (MP): Courtesy of UNF_i commuting with substitution, preserving freshness, and distributing (by definition) over abstractions, applications and implications. $\square$

As a particular case of this lemma's point (5), we have:

Theorem 18. *D is a meta-safe extension of HOL-with-declarations.*

Thus, we can compile away all the definitions of D, leaving us with types and terms over the signature Δ containing declarations only. With the definitions out of our way, it remains to show that *declarations* are conservative, which is much easier:

Lemma 19. If $\varphi \in \mathsf{Fmla}_{\Sigma_{\min}}$ and $\vdash_\Delta \varphi$, then $\vdash_{\Sigma_{\min}} \varphi$.

Proof of Lemma 19. Assume $\vdash_\Delta \varphi$. In the proof tree for this fact, we replace:

1) all occurrences of any declared constant instance c_σ by a fresh variable x_σ
2) all occurrences of any declared type constructor k of arity m by a built-in type expression of ariry n, e.g., $(\sigma_1, \dots, \sigma_m)k$ is replaced by $\sigma_1 \Rightarrow \ldots \Rightarrow \sigma_m$

Then the resulted proof tree constitutes a proof of $\vdash_{\Sigma_{\min}} \varphi$. $\square$

Finally, we can prove overall conservativity:

Theorem 20. *D is a conservative extension of minimal HOL.*

Proof. Assume $D \vdash_\Sigma \varphi$, where $\varphi \in \mathsf{Fmla}_{\Sigma_{\min}}$. By Theorem 18, we have $\vdash_\Delta \mathsf{UNF}(\varphi)$. Moreover, by Lemma 14(3), we have $\vdash_{\Sigma_{\min}} \mathsf{UNF}(\varphi) = \varphi$, hence, a fortiori, $\vdash_\Delta \mathsf{UNF}(\varphi) = \varphi$. From these two, we obtain $\vdash_\Delta \varphi$. With Lemma 19, we obtain $\vdash_{\Sigma_{\min}} \varphi$, as desired. $\square$

G. Abstract Constant Definition Mechanisms

As definitional schemes for constants, we have only looked into the traditional *equational* ones, implemented in most HOL provers. Two non-equational schemes have also been designed [6], and are available in HOL4, HOL Light and ProofPower-HOL: "new specification" and "gen new specification." They allow for more abstract (under)specification of constants.

However, these schemes have been shown not to increase expressiveness: "new specification" can be over-approximated by traditional definitions and the use of the Choice operator, and "gen new specification" is an admissible rule in HOL with "new specification" [6], [18]. Hence our results cater for them.

V. CONSERVATIVITY OF ISABELLE/HOL DEFINITIONS

As mentioned in the introduction, Isabelle/HOL allows more flexible constant definitions than HOL, in that it enables ad hoc overloaded definitions. For example, one can declare a polymorphic constant, such as $\leq\ : \alpha \Rightarrow \alpha\ \mathsf{bool}$, and at later times (perhaps after some other type and constant definitions and declarations have been performed) define different, non-overlapping instances of it: $\leq_{\mathsf{nat}}$ as the usual order on natural numbers, $\leq_{\mathsf{bool}}$ as implication, etc. Even recursive overloading is allowed, e.g., one can define $\leq_{\alpha\ \mathsf{list}}$ as the component-wise extension of $\leq_\alpha$ to α list:

$$xs \leq_{\alpha\ \mathsf{list}} ys \;\equiv\; \mathsf{len}\ xs = \mathsf{len}\ ys \wedge (\forall i < \mathsf{len}\ xs.\ xs_i \leq_\alpha ys_i)$$

This means that now constant definitions no longer require the constant to be *fresh*. In fact, we are no longer speaking of constant definitions, but of constant *instance* definitions: The above examples do not define the overall constant $\leq$, but various instances of it, $\leq_{\mathsf{nat}}$, $\leq_{\mathsf{bool}}$ and $\leq_{\mathsf{list}}$.

Def 21. Given a non-built-in constant c, a type $\sigma \leq \mathsf{tpOf}(c)$ 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 instance definition* provided $\mathsf{TV}(t) \subseteq \mathsf{TV}(c_\sigma)$.

To compensate for the lack of freshness from constant-instance definitions, the Isabelle/HOL system performs some global syntactic checks, making sure that defined instances do not overlap (i.e., definitions are *orthogonal*) and that the dependency relation $\rightsquigarrow_n$ from Def. 8, terminates [19]–[21]. (Recall that $D = D_n$, hence $\rightsquigarrow_n$ is the dependency induced by D, i.e., by all the considered definitions.) Formally:

Def 22. An *Isabelle/HOL-well-formed definitional theory* is set D of type and constant instance definitions over Σ such that:

- It satisfies all the conditions of Def. 2, 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 $\sigma \# \tau$
- Its induced dependency relation $\rightsquigarrow_n$ is terminating

We wish to prove meta-safety and conservativity results similar to the ones for traditional HOL. To this end, we fix an Isabelle/HOL-well-formed definitional theory D and look into the results of Section IV to see what can be reused—as it turns out, quite a lot.

First, the (type-translated) declaration signatures Δ^i and the translation functions HOST_i, REL_i and UNF_i are defined in the same way. The orthogonality assumption in Def. 22 ensures that, in clause (U3) from the definition of UNF_i, the choice of t is unique (whereas before, this was simply ensured by c appearing on the left in at most one definition). The notion of meta-safety is then defined in the same way. Thanks to $\rightsquigarrow_n$ being terminating, all the dependency relations $\rightsquigarrow_i$, which are included in $\rightsquigarrow_n$, are also terminating. Then all the results in Section IV-D hold, leading to the well-definedness of the translation functions. Furthermore, almost all the lemmas in Section IV-E go through undisturbed, because they do not need the freshness assumption $c \notin \Sigma^i$.

The only losses are parts of Lemmas 12 (extension of the translations from i to $i+1$) and 16 (commutation with type substitution), namely, points (2) and (3) of these lemmas—which deal with REL_i and UNF_i. We first look at Lemma 16. While HOST_i still commutes with substitution, this is no longer the case for REL_i and UNF_i. Essentially, $\mathsf{UNF}_i(\sigma[\tau/\alpha]) = \mathsf{UNF}_i(\sigma)[\mathsf{HOST}_i(\tau)/\alpha]$ now fails because $\mathsf{UNF}_i(\sigma[\tau/\alpha])$ gets to unfold more constant-instance definitions than $\mathsf{UNF}_i(\sigma)$. So the difference is that, for the constance instances $c_{\sigma'}$ occurring in t that happen to have their definition activated by the substitution τ/α, $\mathsf{UNF}_i(\sigma[\tau/\alpha])$ will replace them by $\mathsf{UNF}_i(c_\sigma)$ whereas $\mathsf{UNF}_i(\sigma)[\mathsf{HOST}_i(\tau)/\alpha]$ will keep them as $c_{\mathsf{HOST}_i(\sigma)}$. (And since REL_i depends recursively on UNF_i, the property fails for REL_i as well.)

Example 23. To the signature from Example 4, we add a declared constant c of polymorphic type α and a definition of its nat-instance, $c_{\mathsf{nat}} = \mathsf{z}$. We have $\mathsf{UNF}(c_\alpha[\mathsf{nat}/\alpha]) = \mathsf{UNF}(c_{\mathsf{nat}}) = \mathsf{UNF}(\mathsf{z})$, whereas $\mathsf{UNF}(c_\alpha)([\mathsf{HOST}(\mathsf{nat})/\alpha]) = c_{\mathsf{HOST}(\alpha)}[\mathsf{ind}/\alpha] = c_\alpha[\mathsf{ind}/\alpha] = c_{\mathsf{ind}}$. We do not need to evaluate $\mathsf{UNF}(\mathsf{z})$ in order to see that it cannot be equal, not even HOL-provably equal, to c_{ind} (since the definitions leading to z have nothing to do with c; in fact, they take place over a signature not containing c).

We can amend this mismatch "after the fact" by replacing $c_{\mathsf{HOST}_i(\sigma'')}$ with $\mathsf{UNF}_i(c_{\sigma''})$ in $\mathsf{UNF}_i(\sigma)[\mathsf{HOST}_i(\tau)/\alpha]$ for all instances $c_{\sigma''}$ (with $\sigma'' \leq \sigma'$) of all defined constant instances $c_{\sigma'}$. In the above example, this means replacing c_{ind} with $\mathsf{UNF}(c_{\mathsf{nat}})$, i.e., with $\mathsf{UNF}(\mathsf{z})$. To express this formally, we define a *constant-instance substitution* to be a function $\gamma : \mathsf{CInst}^\bullet_{\Delta^i} \Rightarrow \mathsf{Term}_{\Delta^i}$ such that, for all $c_\sigma \in \mathsf{CInst}^\bullet_{\Delta^i}$, $\gamma(c_\sigma)$ is a closed term and $\mathsf{TV}(\gamma(c)) \subseteq \mathsf{TV}(c)$—thus assigning a term to any instance of a non-built-in, i.e., declared constant in Δ^i. Using a notation similar to variable substitution, we write $\sigma[[\gamma]]$ and $t[[\gamma]]$ for the effect of performing γ everywhere inside the type σ or the term t.

Lemma 24. There exists a constant-instance substitution γ such that:
(1) $\mathsf{REL}_i(\sigma[\tau/\alpha]) = \mathsf{REL}_i(\sigma)[\mathsf{HOST}_i(\tau)/\alpha]\ [[\gamma]]$
(2) $\mathsf{UNF}_i(t[\tau/\alpha]) = \mathsf{UNF}_i(t)[\mathsf{HOST}_i(\tau)/\alpha]\ [[\gamma]]$

Proof. We define γ to map each $c_{\mathsf{HOST}_i(\sigma)}$ to $\mathsf{UNF}_i(c_\sigma)$. Thanks to Lemma 15(3), γ is indeed a constant-instance substitution. Now, points (1) and (2) follow by well-founded induction w.r.t. $\blacktriangleright_i$ (the terminating relation associated to the mutual call graph of REL_i and UNF_i). The only interesting case is that of defined constants (clause (U3) for UNF_i). Assume $\sigma[\tau/\alpha] = \sigma'[\rho]$, such that $c_{\sigma'} \equiv t \in D_i$. We have two cases:

First, assume $\sigma \leq \sigma'$, say, $\sigma = \sigma'[\rho']$ for some ρ'. Then ρ and $\rho' \cdot (\tau/\alpha)$ are equal on $\mathsf{TV}(\sigma')$, a fortiori, on $\mathsf{TV}(t)$. Hence $t[\rho] = t[\rho' \cdot (\tau/\alpha)]$. i.e., $t[\rho] = t[\rho'][\tau/\alpha]$ (*). Both $\mathsf{UNF}_i(c_\sigma[\tau/\alpha])$ and $\mathsf{UNF}_i(c_\sigma)$ will unfold the definitions of their corresponding instances of c, allowing us to infer the desired fact from the induction hypothesis:

$\mathsf{UNF}_i(c_\sigma[\tau/\alpha]) = \mathsf{UNF}_i(c_{\sigma[\tau/\alpha]}) = \mathsf{UNF}_i(c_{\sigma'[\rho]}) =$ (by (U3)) $=$
$\mathsf{UNF}_i(t[\rho]) =$ (by (*)) $= \mathsf{UNF}_i(t[\rho'][\tau/\alpha]) =$
(by the induction hypothesis)
$\mathsf{UNF}_i(t[\rho'])[\mathsf{HOST}_i(\tau)/\alpha][[\gamma]] =$ (by (U3)) $=$
$\mathsf{UNF}_i(c_{\sigma'[\rho']})[\mathsf{HOST}_i(\tau)/\alpha][[\gamma]] = \mathsf{UNF}_i(c_\sigma)[\mathsf{HOST}_i(\tau)/\alpha][[\gamma]]$

Next, assume $\sigma \not\leq \sigma'$. Then only $\mathsf{UNF}_i(c_\sigma[\tau/\alpha])$ would unfold the definition of c'_σ, but γ fixes the mismatch:

$\mathsf{UNF}_i(c_\sigma[\tau/\alpha]) = \mathsf{UNF}_i(c_{\sigma[\tau/\alpha]}) =$ (by def. of γ) $=$
$\gamma(c_{\mathsf{HOST}_i(\sigma[\tau/\alpha])}) = c_{\mathsf{HOST}_i(\sigma[\tau/\alpha])}[[\gamma]] =$
(since HOST_i commutes with substitution)
$c_{\mathsf{HOST}_i(\sigma)[\mathsf{HOST}_i(\tau)/\alpha]}[[\gamma]] =$
$c_{\mathsf{HOST}_i(\sigma)}[\mathsf{HOST}_i(\tau)/\alpha][[\gamma]] =$ (by (U2)) $=$
$\mathsf{UNF}_i(c_\sigma)[\mathsf{HOST}_i(\tau)/\alpha][[\gamma]]$ $\square$

Now, the question is whether the partial consolation offered by Lemma 24, a quasi-commutativity property for REL_i and UNF_i, can replace full commutativity towards the central goal in Lemma 17, namely, point (5) (which ensures meta-safety). The only usage of Lemma 16 was for $(1)_i$ implies $(2)_i$ (which is part of an implication chain leading to $(4)_i$; and both $(2)_i$ and $(4)_i$ are used for $(5)_i$). There, we used Lemma 16 m times to infer $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma')}.\ \mathsf{REL}_i(\sigma')\ x \wedge \mathsf{UNF}_i(t')\ x$ from $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma)}.\ \mathsf{REL}_i(\sigma)\ x \wedge \mathsf{UNF}_i(t)\ x$. So we actually need a weaker statement, which we can prove from Lemma 24:

Lemma 25. If $\vdash_{\Delta^i} \mathsf{UNF}_i(\varphi)$, then $\vdash_{\Delta^i} \mathsf{UNF}_i(\varphi[\sigma/\alpha])$.

Proof. By Lemma 24(2), we have a constant-instance substitution γ such that $\mathsf{UNF}_i(\varphi[\sigma/\alpha]) = \mathsf{UNF}_i(\varphi)[\mathsf{HOST}_i(\sigma)/\alpha]$ $[[\gamma]]$. And since $\vdash_{\Delta^i} \mathsf{UNF}_i(\varphi)[\mathsf{HOST}_i(\sigma)/\alpha]$ follows from $\vdash_{\Delta^i}$

$\mathsf{UNF}_i(\varphi)$ by the type substitution rule (T-INST), it would suffice to have the following: For all constant-instance substitutions γ and Δ^i-formulas φ, $\vdash_{\Delta^i} \varphi$ implies $\vdash_{\Delta^i} \varphi[[\gamma]]$. In words, if we substitute some (undefined) constant instances with terms of the same type we do not lose provability. This follows by routine rule induction on the definition of deduction. $\square$

For Lemma 12, the situation is quite similar to that of Lemma 16. This time, it is not substitution that can enable additional unfoldings, but a newly added instance definition $c_\sigma \equiv t$ at layer $i+1$ for a constant c that already existed at layer i. Moreover, when we look at how we employed Lemma 12 in the proof of our main chain of results in Lemma 17, we discover a similar pattern: We only use that UNF_{i+1} and REL_{i+1} extend UNF_i and REL_i in the proof of $(5)_i$ implies $(1)_{i+1}$, where we needed that deduction at layer $i+1$ is implied by deduction at layer i. By a similar trick as before, this can be proved using a weaker quasi-commutativity property.

Lemma 26. If $\varphi \in \mathsf{Fmla}_{\Sigma_i}$, and $\vdash_{\Delta^i} \mathsf{UNF}_i(\varphi)$, then $\vdash_{\Delta^{i+1}} \mathsf{UNF}_{i+1}(\varphi)$.

Proof. If def_{i+1} is a type definition, then UNF_{i+1} and REL_{i+1} do extend UNF_i and REL_i, so the desired fact follows trivially. Now, assume def_{i+1} is a constant-instance definition $c_\sigma \equiv t$. Similarly to the proof of Lemma 24(2), we obtain a constant-instance substitution γ such that $\mathsf{UNF}_{i+1}(\varphi) = \mathsf{UNF}_i(\varphi)[[\gamma]]$, namely, γ maps each $d_{\mathsf{HOST}(\tau[\rho])}$ to $\mathsf{UNF}_{i+1}(s[\rho])$ where $d_\tau \equiv s$ are the constant definitions in D_{i+1}. (We need to do this replacement to all defined constant instances, not just c_σ, since other definitions from D_i may have already relied on c_σ.) And since, as we have seen, constant-instance substitution preserves deduction, we obtain our desired fact. $\square$

Thus, we were able to recover Lemma 17's point (5), leading to meta-safety. And since the other ingredients in the proof of Theorem 20 are also available (including Lemma 19, which is independent of the definitional mechanisms), we infer conservativity. We obtained:

Theorem 27. Theorems 18 and 20 still hold if we assume that D is an Isabelle/HOL-well-formed definitional theory.

VI. CONCLUSION

We have resolved an open problem, relevant for the foundation of HOL-based theorem provers, including our favorite one, Isabelle/HOL: We showed that the definitional mechanisms in such provers are meta-safe and conservative over pure HOL, i.e., are truly "definitional."

Acknowledgments. We thank Tobias Nipkow, Larry Paulson, Makarius Wenzel, Rob Arthan, Roger Bishop Jones, Ramana Kumar and the members of the Isabelle and HOL mailing lists for inspiring discussions about the logical foundations of theorem proving.

REFERENCES

[1] Isabelle Foundation & Certification (2015), archived at https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2015-September/thread.html
[2] Isabelle's Archive of Formal Proofs (2016)
[3] The Isabelle Library (2016), https://isabelle.in.tum.de/dist/library/HOL/index.html
[4] Adams, M.: Introducing HOL Zero (Extended Abstract). In: ICMS '10. Springer (2010)
[5] Arthan, R.D.: Some Mathematical Case Studies in ProofPower–HOL. In: TPHOLs 2004
[6] Arthan, R.: HOL constant definition done right. In: ITP. pp. 531–536 (2014)
[7] Barras, B.: Sets in Coq, Coq in Sets. Journal of Formalized Reasoning 3(1) (2010)
[8] Bertot, Y., Casteran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Springer (2004)
[9] Church, A.: A Formulation of the Simple Theory of Types. The Journal of Symbolic Logic 5(2), 56–68 (1940)
[10] Esparza, J., Lammich, P., Neumann, R., Nipkow, T., Schimpf, A., Smaus, J.G.: A fully verified executable LTL model checker. In: CAV. pp. 463–478 (2013)
[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] Haftmann, F., Wenzel, M.: Constructive Type Classes in Isabelle. In: TYPES (2006)
[13] Harrison, J.: HOL Light: A Tutorial Introduction. In: FMCAD '96. Springer (1996)
[14] Harrison, J.: Towards self-verification of HOL Light. In: IJCAR 2006. Springer (2006)
[15] Henkin, L.: The completeness of the first-order functional calculus. J. Symbolic Logic 14(3), 159–166 (09 1949)
[16] Kaufmann, M., Manolios, P., Moore, J.S.: Computer-Aided Reasoning: An Approach. Kluwer Academic Publishers (2000)
[17] Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: formal verification of an operating-system kernel. Commun. ACM 53(6), 107–115 (2010)
[18] Kumar, R., Arthan, R., Myreen, M., Owens, S.: HOL with Definitions: Semantics, Soundness, and a Verified Implementation. In: ITP (2014)
[19] Kunčar, O.: Correctness of Isabelle's cyclicity checker: Implementability of overloading in proof assistants. In: CPP. pp. 85–94 (2015)
[20] Kunčar, O., Popescu, A.: A Consistent Foundation for Isabelle/HOL. In: ITP. pp. 234–252 (2015)
[21] Kunčar, O., Popescu, A.: Comprehending Isabelle/HOL's consistency. In: ESOP (2017), To appear. Preprint available at http://andreipopescu.uk/pdf/compr_IsabelleHOL_cons.pdf
[22] Myreen, M.O., Davis, J.: The reflective Milawa theorem prover is sound - (down to the machine code that runs it). In: ITP. pp. 421–436 (2014)
[23] Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
[24] Nipkow, T., Snelting, G.: Type Classes and Overloading Resolution via Order-Sorted Unification. In: Functional Programming Languages and Computer Architecture (1991)
[25] Obua, S.: Checking Conservativity of Overloaded Definitions in Higher-Order Logic. In: RTA (2006)
[26] Owre, S., Shankar, N.: The formal semantics of PVS (1999), SRI technical report. http://www.csl.sri.com/papers/csl-97-2/
[27] Paulson, L.C.: A formulation of the simple theory of types (for Isabelle). In: COLOG-88. pp. 246–274 (1990)
[28] Paulson, L.C.: A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle. J. Autom. Reasoning 55(1), 1–37 (2015)
[29] 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)
[30] Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. Monographs in Theoretical Computer Science. An EATCS Series, Springer (2012)
[31] Wenzel, M.: Type Classes and Overloading in Higher-Order Logic. In: TPHOLS '97
[32] Wiedijk, F.: Stateless HOL. In: TYPES. pp. 47–61 (2009)

Appendix

A. More details on HOL

It is well-known (and easy to prove) that substitution respects typing:

Lemma 28. If $t : \sigma$, then $t[\rho] : \sigma[\rho]$.

When writing concrete terms or formulas, we take the following conventions:

- We omit redundantly indicating the types of the variables, e.g., we shall write $\lambda x_\sigma.\, x$ instead of $\lambda x_\sigma.\, x_\sigma$.
- We omit redundantly indicating the types of the variables and constants in terms if they can be inferred by typing rules, e.g., we shall write $\lambda x.\, (y_{\sigma\Rightarrow\tau}\ x)$ instead of $\lambda x_\sigma.\, (y_{\sigma\Rightarrow\tau}\ x)$ or $\varepsilon(\lambda x_\sigma.\, P\ x)$ instead of $\varepsilon_{(\sigma\Rightarrow\mathsf{bool})\Rightarrow\sigma}(\lambda x_\sigma.\, P_{\sigma\Rightarrow\mathsf{bool}}\ x)$.
- We write $\lambda x_\sigma\ y_\tau.\, t$ instead of $\lambda x_\sigma.\, \lambda y_\tau.\, t$
- We apply the constants $\longrightarrow$ and $=$ in an infix manner, e.g., we shall write $t_\sigma = s$ instead of $= t_\sigma\ s$. We use ε as a binder, i.e., we shall write $\varepsilon x_\sigma.\, t$ instead of $\varepsilon\ (\lambda x_\sigma.\, t)$.

The formula connectives and quantifiers are defined as abbreviations in the usual way, starting from the implication and equality primitives:

$$
\begin{array}{lcl}
\mathsf{True} & = & (\lambda x_{\mathsf{bool}}.\, x) = (\lambda x_{\mathsf{bool}}.\, x) \\
\mathsf{All} & = & \lambda p_{\alpha\Rightarrow\mathsf{bool}}.\, (p = (\lambda x.\, \mathsf{True})) \\
\mathsf{Ex} & = & \lambda p_{\alpha\Rightarrow\mathsf{bool}}.\, \mathsf{All}\ (\lambda q.\, (\mathsf{All}\ (\lambda x.\, p\ x \longrightarrow q)) \longrightarrow q) \\
\mathsf{False} & = & \mathsf{All}\ (\lambda p_{\mathsf{bool}}.\, p) \\
\mathsf{not} & = & \lambda p.\, p \longrightarrow \mathsf{False} \\
\mathsf{and} & = & \lambda p\, q.\, \mathsf{All}\ (\lambda r.\, (p \longrightarrow (q \longrightarrow r)) \longrightarrow r) \\
\mathsf{or} & = & \lambda p\, q.\, \mathsf{All}\ (\lambda r.\, (p \longrightarrow r) \longrightarrow ((q \longrightarrow r) \longrightarrow r))
\end{array}
$$

It is easy to see that the above terms are closed and well-typed as follows:

- $\mathsf{True}, \mathsf{False} : \mathsf{bool}$
- $\mathsf{not} : \mathsf{bool} \Rightarrow \mathsf{bool}$
- $\mathsf{and}, \mathsf{or} : \mathsf{bool} \Rightarrow \mathsf{bool} \Rightarrow \mathsf{bool}$
- $\mathsf{All}, \mathsf{Ex} : (\alpha \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{bool}$

As customary, we shall write:

- $\forall x_\alpha.\, t$ instead of $\mathsf{All}\ (\lambda x_\alpha.\, t)$
- $\exists x_\alpha.\, t$ instead of $\mathsf{Ex}\ (\lambda x_\alpha.\, t)$
- $\neg\, \varphi$ instead of $\mathsf{not}\, \varphi$
- $\varphi \wedge \chi$ instead of $\mathsf{and}\, \varphi\, \chi$
- $\varphi \vee \chi$ instead of $\mathsf{or}\, \varphi\, \chi$

The HOL axioms, forming the set Ax, are the following:

- Equality Axioms:
  refl: $x_\alpha = x$
  subst: $x_\alpha = y \longrightarrow P\ x \longrightarrow P\ y$
  iff: $(p \longrightarrow q) \longrightarrow (q \longrightarrow p) \longrightarrow (p = q)$
- Infinity Axioms:
  suc_inj: $\mathsf{suc}\ x = \mathsf{suc}\ y \longrightarrow x = y$
  suc_not_zero: $\neg\, \mathsf{suc}\ x = \mathsf{zero}$
- Excluded Middle:
  True_or_False: $(b = \mathsf{True}) \vee (b = \mathsf{False})$
  Choice:
  some_intro: $p_{\alpha\Rightarrow\mathsf{bool}}\ x \longrightarrow p\ (\varepsilon\ p)$

Above, refl and subst axiomatize equality and iff ensures that equality on the bool type behaves as a logical equivalence. suc_inj and suc_not_zero ensure that ind is an infinite type. some_intro regulates the behavior of the Hilbert Choice operator. Finally, True_or_False makes the logic classical.

B. Detailed Definition of the Operators used in the Dependency Relation

Note that the $\mathsf{types}^\bullet$ operator is overloaded for types and terms.

$$
\begin{array}{l}
\mathsf{types}^\bullet(\alpha) = \{\alpha\} \\
\mathsf{types}^\bullet(\mathsf{bool}) = \mathsf{types}^\bullet(\mathsf{ind}) = \emptyset \\
\mathsf{types}^\bullet(\sigma_1 \Rightarrow \sigma_2) = \mathsf{types}^\bullet(\sigma_1) \cup \mathsf{types}^\bullet(\sigma_2) \\
\mathsf{types}^\bullet(\overline{\sigma}\ k) = \{\overline{\sigma}\ k\},\ \text{ if } k \neq \Rightarrow, \mathsf{bool}, \mathsf{ind}
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{cinsts}^\bullet(x_\sigma) & = & \emptyset \\
\mathsf{cinsts}^\bullet(c_\sigma) & = & \begin{cases} \{c_\sigma\} & \text{if } c_\sigma \in \mathsf{CInst}^\bullet \\ \emptyset & \text{otherwise} \end{cases} \\
\mathsf{cinsts}^\bullet(t_1\ t_2) & = & \mathsf{cinsts}^\bullet(t_1) \cup \mathsf{cinsts}^\bullet(t_2) \\
\mathsf{cinsts}^\bullet(\lambda x_\sigma.\, t) & = & \mathsf{cinsts}^\bullet(t)
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{types}^\bullet(x_\sigma) & = & \mathsf{types}^\bullet(\sigma) \\
\mathsf{types}^\bullet(c_\sigma) & = & \mathsf{types}^\bullet(\sigma) \\
\mathsf{types}^\bullet(t_1\ t_2) & = & \mathsf{types}^\bullet(t_1) \cup \mathsf{types}^\bullet(t_2) \\
\mathsf{types}^\bullet(\lambda x_\sigma.\, t) & = & \mathsf{types}^\bullet(\sigma) \cup \mathsf{types}^\bullet(t)
\end{array}
$$

C. Proof Sketches

In the proofs, we will use several induction schemas, fit for the purpose:

- *Well-founded induction* on types and/or terms with respect to one of the (known to be terminating) relations $\blacktriangleright\!\!\!\blacktriangleright_i$ and $\blacktriangleright_i$: Given u, we can assume the property holds for all items u' such that $u \blacktriangleright\!\!\!\blacktriangleright_i u'$ (or $u \blacktriangleright_i u'$) and need to prove it for u. So whenever we indicate a proof by well-founded induction, we will implicitly refer to one of these two, namely, to the first when proving something about HOST_i and to the second when proving something about REL_i and/or UNF_i.
- *Structural induction* on types and/or terms: Given u, we can assume the property holds for all immediate subtypes/subterms of u and need to prove it for u.
- *Rule induction* with respect to the definition of typing or the definition of HOL deduction: To conclude that typing or deduction implies a property, we prove that the property is closed under the rules defining typing or deduction.

In all these schemas, (IH) denotes the induction hypothesis.

Proof of point (1) of Prop. 7. We first define, for any type constructor $k \in \Sigma_i$, the operator $\mathsf{depth}_k : \mathsf{Type}_{\Sigma_i} \Rightarrow \mathbb{N}$ to return, for any type, the length of the longest nesting of k's appearing in it, namely:

$\mathsf{depth}_k(\alpha) = 0$

$$\mathsf{depth}_k((\sigma_1,\ldots,\sigma_m)\ l) = \begin{cases} 1+\max\{\mathsf{depth}_k(\sigma_i) \mid i\in\{1,\ldots,m\}\} & \text{if } l=k \\ \max\{\mathsf{depth}_k(\sigma_i) \mid i\in\{1,\ldots,m\}\} & \text{if } l\neq k \end{cases}$$

Let K^i and K_i be the sets of type constructors of $\Sigma_1 \cup \bigcup_{i'=1}^{i} \Sigma^{i'} \setminus \Sigma_{i'-1}$ and Σ_i, respectively. Note that $\mathsf{K}^i \subseteq \mathsf{K}_i$ and that $\mathsf{K}_i \setminus \mathsf{K}^i$ contains the defined type constructors, whereas K^i contains the declared and built-in ones (up to moment i). We chose an arbitrary total order $\succ$ on K^i, and then extend it to a homonymous total order on K_i, as follows:

- If $k \in \mathsf{K}^i$ and $l \in \mathsf{K}_i$, then $l \succ k$
- If $k_1, k_2 \in \mathsf{K}_i$, then $k_1 \succ k_i$ if k_1 was introduced later than k_2, i.e., if the unique $j_1 \leq i$ such that k_1 appears in the lefthand side of def_{j_1} is greater than the unique $j_2 \leq i$ such that k_2 appears on the lefthand side of def_{j_2}

Since K_i is finite, it has the form $\{k_1,\ldots,k_p\}$ with $k_1 \succ \ldots \succ k_p$. We define the measure $\mathsf{meas} : \mathsf{Type}_{\Sigma_i} \to \mathbb{N}^p$ by $\mathsf{meas}(\sigma) = (\mathsf{depth}_{k_1},\ldots,\mathsf{depth}_{k_p})$. Finally, we note that $\blacktriangleright\!\!\!\blacktriangleright_i$ decreases this measure w.r.t. the lexicographic order on $\mathbb{N}^p$ (which ensures its termination). Indeed, we consider the two cases in the definition of $\blacktriangleright\!\!\!\blacktriangleright_i$:

- In the first case (given by recursive clause (U2)), all depth_{k_j} remain the same or decrease, and depth_k decreases by 1
- In the second case (given by recursive clause (U3)), we know from the well-foundedness of D that σ only contains type constructors l with $k \succ l$. Therefore, we have:

  $\mathsf{depth}_k((\sigma_1,\ldots,\sigma_m)\ k) =$
  $1+\max\{\mathsf{depth}_k(\sigma_j) \mid j\in\{1,\ldots,m\}\} >$
  $\max\{\mathsf{depth}_l(\sigma_j) \mid j\in\{1,\ldots,m\} \wedge \alpha_i \in \mathsf{TV}(\sigma)\} =$
  $\mathsf{depth}_k(\sigma[\sigma_1/\alpha_1,\ldots,\sigma_m/\alpha_m])$

  Moreover, for any l such that $l \succ k$, we have:

  $\mathsf{depth}_l((\sigma_1,\ldots,\sigma_m)\ k) =$
  $\max\{\mathsf{depth}_l(\sigma_j) \mid j\in\{1,\ldots,m\}\} \geq$
  $\max\{\mathsf{depth}_l(\sigma_j) \mid j\in\{1,\ldots,m\} \wedge \alpha_i \in \mathsf{TV}(\sigma)\} =$
  $\mathsf{depth}_l(\sigma[\sigma_1/\alpha_1,\ldots,\sigma_m/\alpha_m])$

  Thus, depth_k decreases strictly and, for $l \succ k$, depth_l remains the same or decreases; this ensures that meas decreases. ☐

Proof of Lemma 9. We proceed similarly to the proof of termination for the call graph of HOST_i, but considering Σ_i-constants in addition to Σ_i-type constructors. Similarly to there, for each $e \in \mathsf{K}_i \cup \mathsf{Const}_i$, we define $\mathsf{depth}_e : \mathsf{Type}_{\Sigma_i} \cup \mathsf{Term}_{\Sigma_i} \Rightarrow \mathbb{N}$, the u-depth of a type or term, to the length of the longest nesting of u's appearing in it. We similarly order the items in $\mathsf{K}_i \cup \mathsf{Const}_i$ by a relation $\succ$ asking that all defined items are greater than all non-defined ones and a later defined item is greater than an earlier defined one. Assuming $\mathsf{K}_i \cup \mathsf{Const}_i$ has the form $\{e_1,\ldots,e_p\}$ with $e_1 \succ \ldots \succ e_p$, we define the measure $\mathsf{meas} : \mathsf{Type}_{\Sigma_i} \cup \mathsf{Term}_{\Sigma_i} \to \mathbb{N}^p$ by $\mathsf{meas}(v) = (\mathsf{depth}_{e_1}(v),\ldots,\mathsf{depth}_{e_p}(v))$.

We show that meas decreases with $\rightsquigarrow_i^{\downarrow}$ w.r.t. the lexicographic order on $\mathbb{N}^p$ (which makes $\rightsquigarrow_i^{\downarrow}$ terminating). Assume $u \rightsquigarrow_i^{\downarrow} v$. Then there exists u', v', ρ such that $u = u'[\rho]$, $v = v'[\rho]$ and $u' \rightsquigarrow_i v'$. Then u' is either a type of the form $(\alpha_1,\ldots,\alpha_m)k$ with $k \in \mathsf{K}_i$, or a constant instance c_σ; meaning u is either $(\rho(\alpha_1),\ldots,\rho(\alpha_m))k$ or $c_{\sigma[\rho]}$. We let e denote either k or c. In both cases, we have $v' \in \mathsf{types}^\bullet(t) \cup \mathsf{cinsts}^\bullet(t)$ for some $t \in \mathsf{Term}_{\Sigma_i}$. Hence $v \in \mathsf{types}^\bullet(t[\rho]) \cup \mathsf{cinsts}^\bullet(t[\rho])$. By the well-formedness of D, e is greater than all the type constructors and constants in t (w.r.t. $\succ$). Then $\mathsf{depth}_e(u) > \mathsf{depth}_e(v)$ and, for all e' such that $e' \succ e$, $\mathsf{depth}_{e'}(u) \geq \mathsf{depth}_{e'}(v)$. This ensures $\mathsf{meas}(u) > \mathsf{meas}(v)$. ☐

Proof of Lemma 10. By routine structural induction on t. ☐

Proof of Lemma 11. Let us assume by absurd that $\blacktriangleright_i$ does not terminate. Then there exists an infinite sequence $(w_p)_{p\in\mathbb{N}}$ such that $w_p \blacktriangleright_i w_{p+1}$ for all p. Since $\blacktriangleright_i$ is defined as $\equiv_i^{\downarrow} \cup \rhd$ and $\rhd$ clearly terminates, there must exist an infinite subsequence $(w_{p_j})_{j\in\mathbb{N}}$ such that $w_{p_j} \equiv_i^{\downarrow} w_{p_{j+1}} \rhd^* w_{p_{j+1}}$ for all j. Since from the definition of $\equiv_i^{\downarrow}$ we have $w_{p_j} \in \mathsf{Type}^\bullet_{\Sigma_i} \cup \mathsf{CInst}^\bullet_{\Sigma_i}$, we obtain from Lemma 10 that $w_{p_j} \rightsquigarrow_i^{\downarrow} w_{p_{j+1}}$ for all p. This contradicts the termination of $\rightsquigarrow_i^{\downarrow}$. ☐

Proof of Lemma 12. (1): By an easy well-founded induction on σ w.r.t. $\blacktriangleright\!\!\!\blacktriangleright_i$, distinguishing between the different cases in the definition of HOST_i and HOST_{i+1}. The definitions are identical for the two functions, and for the defined type case (clause (H3)), we know that k is in Σ_i, ensuring that $(\sigma_1,\ldots,\sigma_m)k \equiv t$ is in D_i.
(2) and (3): Similar to (1), by an easy well-founded induction on σ and t w.r.t. $\blacktriangleright_i$. ☐

Proof of Lemma 13. By well-founded induction on σ and t, distinguishing between the different cases in the definitions of REL_i and UNF_i. The proof is routine. We only show the two slightly less obvious cases, where we employ the local notations used in the definitions (e.g., σ', t'):
The defined type case for REL_i (clause (P4)): We know that $(\sigma_1,\ldots,\sigma_m)k \blacktriangleright_i \sigma'$ and $(\sigma_1,\ldots,\sigma_m)k \blacktriangleright_i t'$. Moreover, from $t : \sigma$ we obtain $t' : \sigma'$. Hence, by (IH), we have $\mathsf{REL}_i(\sigma') : \mathsf{HOST}(\sigma') \Rightarrow \mathsf{bool}$ and $\mathsf{UNF}_i(t') : \mathsf{HOST}(\sigma')$. From this, the definition of REL_i and the HOL typing rules, we obtain $\mathsf{REL}_i((\sigma_1,\ldots,\sigma_m)k) : \mathsf{HOST}_i(\sigma') \Rightarrow \mathsf{bool}$. Finally, from the definition of HOST_i we have $\mathsf{HOST}_i(\sigma') = \mathsf{HOST}_i((\sigma_1,\ldots,\sigma_m)k)$, hence $\mathsf{REL}_i((\sigma_1,\ldots,\sigma_m)k) : \mathsf{HOST}_i((\sigma_1,\ldots,\sigma_m)k) \Rightarrow \mathsf{bool}$, as desired.
The defined constant case for UNF_i (clause (U3)): We know that $c_\sigma \blacktriangleright_i t[\rho]$. Moreover, since $\sigma = \tau[\rho]$ and $t : \tau$, by Lemma 28 we have that $t[\rho] : \sigma$. By (IH), we have $\mathsf{UNF}(t[\rho]) : \mathsf{HOST}(\sigma)$; and since $\mathsf{UNF}_i(c_\sigma) = \mathsf{UNF}_i(t[\rho])$, we obtain $\mathsf{UNF}_i(c_\sigma) : \mathsf{HOST}(\sigma)$, as desired. ☐

Proof of Lemma 14. (1) and (2): Immediate by structural induction on σ.
(3): Immediate by structural induction on t. For the variable

case, we use point (2) to obtain $\vdash_{\Sigma_0} \mathsf{UNF}_i(x_\sigma) = x_\sigma$ from the behavior of if-then-else.

(Since Σ_{min} has no defined or declared items, the recursive cases that deal with such items do not occur when applying the translations, and in particular REL_i does not depend recursively of UNF_i. This is why structural induction does the job, so there is no need for the more powerful well-founded induction.) $\square$

Proof of Lemma 15. Immediate well-founded induction, using the property that definitions do not introduce free term variables or type variables. $\square$

Proof of Lemma 16. By routine well-founded induction, using the properties of type substitution. For example: In the defined type cases for HOST_i and REL_i (clauses (H3) and (P4)), we use that, if $\mathsf{TV}(\sigma) \subseteq \{\alpha_1, \ldots, \alpha_m\}$ (as guaranteed by Def. 1), $\sigma[\sigma_1/\alpha_1, \ldots, \sigma_m/\alpha_m][\tau/\alpha] = \sigma[(\sigma_1[\tau/\alpha])/\alpha_1, \ldots, (\sigma_m[\tau/\alpha])/\alpha_m]$; in the defined constant case for UNF_i (clause (U3)), we use that $t[\rho][\tau/\alpha] = t[\rho \cdot (\tau/\alpha)]$. (Recall that $\cdot$ is the composition of substitutions.) $\square$

Remaining cases in the proof of Lemma 17.
$(1)_1$: By the well-formedness of D (Def. 2), we have that $t \in \mathsf{Term}_{\Delta^i}$ and $\sigma \in \mathsf{Term}_{\Delta^i}$, hence $\mathsf{HOST}_0(\sigma) = \sigma$, $\vdash_{\Delta^i} \mathsf{REL}_0(\sigma) = \lambda x_\sigma.\, \mathsf{True}$ and $\vdash_{\Delta^i} \mathsf{UNF}_0(t) = t$. From this, we obtain that the fact to be proved is equivalent to $\vdash_{\Delta^i} \exists x_\sigma.\, t\, x$, which is again true by the well-formedness of D.

Next, we fix $i \in \{1, \ldots, n\}$.

$(2)_i$ implies $(3)_i$: Assume $(2)_i$. Then $(3)_i$ follows by rule induction on the definition of typing. For the variable case, we use $(2)_i$ and the Choice axiom, which ensure us that $\vdash_{\Delta^i} \mathsf{REL}_i(\sigma)(\epsilon\, \mathsf{REL}_i(\sigma))$ holds, hence $\vdash_{\Delta^i} \mathsf{REL}_i(\sigma)(\mathsf{UNF}_i(x_\sigma))$ holds.

$(3)_i$ implies $(4)_i$: Assume $(3)_i$. Then $(4)_i$ follows by well-founded induction on t. The only interesting case is in the variable case (clause (U1)), when the variable coincides with the to-be substituted variable x_σ. Thus, $t = x_\tau$. Here, we need to show $\vdash_{\Delta^i} \mathsf{UNF}_i(t') =$ $\mathsf{if\_t\_e}\ (\mathsf{REL}_i(\sigma)\, \mathsf{UNF}_i(t'))\ (\mathsf{UNF}_i(t'))\ (\varepsilon\, \mathsf{REL}_i(\sigma))$. This follows from the fact that, thanks to $(3)_i$ and $t' : \sigma$, we have $\vdash_{\Delta^i} \mathsf{REL}_i(\sigma)\, \mathsf{UNF}_i(t')$.

Next, we fix $i \in \{1, \ldots, n-1\}$.
$(5)_i$ implies $(1)_{i+1}$: Assume $(5)_i$ and let σ, t be as in the formulation of $(1)_{i+1}$, namely, $def_{i+1} = \sigma \equiv t$. By the well-formedness of D (Def. 2), we have $D_i \vdash_{\Sigma_i} \exists x_\sigma.\, t\, x$. Applying $(5)_i$, we obtain $\vdash_{\Delta^i} \mathsf{UNF}_i(\exists x_\sigma.\, t\, x)$. By the definition of the $\exists$ quantifier and the definition of UNF_i, the above is equivalent to $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma)}.\, \mathsf{REL}_i(\sigma)\, x_{\mathsf{HOST}_i(\sigma)} \wedge \mathsf{UNF}_i(t)\, t'$, where t' is the following term:

$$\mathsf{if\_t\_e}\ (\mathsf{REL}_i(\sigma)\ x_{\mathsf{HOST}_i(\sigma)})\ x\ (\varepsilon\, \mathsf{REL}_i(\sigma))$$

By the definition of the if-then-else operator, we can replace t' by x. So the above is further equivalent to $\vdash_{\Delta^i} \exists x_{\mathsf{HOST}_i(\sigma)}.\, \mathsf{REL}_i(\sigma)\, x \wedge \mathsf{UNF}_i(t)\, x$. By Lemma 12 and the fact that $\Delta^i \subseteq \Delta^{i+1}$, the above implies $\vdash_{\Delta^{i+1}}$ $\exists x_{\mathsf{HOST}_{i+1}(\sigma)}.\, \mathsf{REL}_{i+1}(\sigma)\, x \wedge \mathsf{UNF}_{i+1}(t)\, x$, as desired. $\square$