# Comprehending Isabelle/HOL's Consistency

Ondřej Kunčar[1] and Andrei Popescu[2]

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

**Abstract.** Recently, we have shown that the logic of Isabelle/HOL is consistent via a semantic argument using a non-standard HOL semantics. Here we provide a less exotic proof of consistency, following the healthy intuition of *definitions as abbreviations*, realized in an extension of HOL with comprehension types.

## 1 Introduction

Isabelle/HOL [14,15] is a successful proof assistant, with hundreds of users world-wide in both academia and industry. It is being used in major verification projects, such as the seL4 operating system kernel [8]. To provide a better user experience, Isabelle/HOL takes some liberties beyond the well-understood higher-order logic kernel. Namely, its definitional mechanism allows *delayed ad hoc overloading of constant definitions*—in turn, this enables Haskell-style type classes on top of HOL [16].

In standard HOL, a polymorphic constant should either be *only declared* (and forever left uninterpreted), or *fully defined* at its most general type. By contrast, Isabelle/HOL allows first declaring a constant, and at later times "overloading" it by defining some of its instances, as in the following example:

$\texttt{consts}\ 0 : \alpha$

...

$\texttt{def (overloaded)}\ 0 : \mathsf{real} \equiv \mathsf{real\_zero}$

...

$\texttt{def (overloaded)}\ 0 : \alpha\ \mathsf{list} \equiv []$

In between the declaration and the instance definitions, arbitrary commands may occur, including type definitions ("typedef") and datatype definitions (which are derived from typedef). For example, the following is the definition of the type of polynomials in Isabelle/HOL's library, where $\forall_\infty$ is the "for all but finitely many" quantifier:

$\texttt{typedef (overloaded)}\ \alpha\ \mathsf{poly} \equiv \{f : \mathsf{nat} \to (\alpha : \mathsf{zero}) \mid \forall_\infty n.\ f\ n = 0\}$

The type $\alpha$ poly is polymorphic in a type $\alpha$ of class zero, i.e., a type $\alpha$ equipped with a constant $0 : \alpha$. When the zero type class is instantiated to concrete types, such as real and $\alpha$ list, the library theorems about arbitrary-domain polynomials are enabled for polynomials over these concrete types. Recursive overloading is also supported, as in:

$\texttt{def (overloaded)}\ 0 : \alpha\ \mathsf{list} \equiv [0 : \alpha]$

To avoid inconsistency, this overloading mechanism is regulated by syntactic checks for *orthogonality* and *termination*. Examples like the above should be allowed, whereas examples like the following encoding of Russell's paradox should be forbidden:

```
consts c : α
typedef (overloaded) T ≡ {True, c} by blast
defs (overloaded) c:bool ≡ ¬ (∀(x:T) y. x = y)
```

Because Isabelle/HOL is so popular and so relied upon, it is important that the consistency of its logic be established with a high degree of *clarity* and a high degree of *rigor*. In 2014, we started an investigation into the foundations of this logic, which has revealed a few consistency problems (including the above "paradox"). These have been addressed [1, 10, 11] and are no longer exhibited by the latest release, Isabelle 2016.

In addition to taking care of these issues, one of course needs some guarantees that similar issues are not still present in the logic. To address this, in previous work [11] we have proved that the logic is now consistent, employing a *semantic argument* in terms of a non-standard semantics for HOL. Our original proof was somewhat sketchy and lacking in rigor—full (pen-and-paper) proofs are now included in an extended report [13]. Of course, a machine-checked proof, perhaps building on a recent formalization of HOL [4, 9], would make further valuable progress on the rigor aspect.

In this paper, we hope to improve on the *clarity* aspect. As mentioned, Isabelle/HOL is richer than HOL not in the rules of deduction, but in the definitional mechanisms. A natural reluctance that comes to mind concerning our semantic proof of consistency is best expressed by Isabelle's creator initial reaction to our proof idea [19]:

> It's a bit puzzling, not to say worrying, to want a set-theoretic semantics for plain definitions. The point of definitions (and the origin of the idea that they preserve consistency) is that they are abbreviations.

This paper's contribution is a new proof of consistency for Isabelle/HOL, easy to digest by the large community of "syntacticists" who prefer to regard definitions as a form of abbreviations. For them, the main appeal of definitions is that these can be inlined without losing essential provability: any deduction in the logic augmented with definitions can be reduced to a deduction in the core logic (without definitions). Hence, consistency of the former follows from consistency of the latter. In our case, the core logic is polymorphic classic higher-order logic with Choice and Infinity: this is well-known and easily seen to be consistent. Moreover, the definition-augmented logic is Isabelle/HOL. (These logics are recalled in §2.)

The problem is that Isabelle/HOL's type definitions cannot simply be unfolded (and inlined)—a type definition is an axiom that postulates a new type and an embedding-projection pair between the new type and the original type (from where the new type is "carved out"). But the syntactic intuition persists: what if we *were* allowed to unfold type definitions as well? As it turns out, this can be achieved in a gentle extension of HOL featuring comprehension types. This extended logic, called HOL with Comprehension (HOLC), is a syntacticist's paradise, allowing for a consistency proof along their intuition. This proof is systematically developed in §3. First, HOLC is introduced (§3.1) and shown consistent by a standard argument (§3.2). Then, a translation is defined from well-formed Isabelle/HOL definitions to HOLC, which is proved sound, i.e., deduction-preserving (§3.3). The key to establish soundness is the use of a modified deduction system for HOL where type instantiation is restricted—this tames the inherent lack of uniformity brought about by ad hoc overloading. Finally, soundness of the translation together with consistency of HOLC ensures consistency of Isabelle/HOL.

## 2 The Isabelle/HOL Logic Recalled

The *logic of Isabelle/HOL* (which we shall simply refer to as *Isabelle/HOL*) consists of:

- HOL, that is, higher-order logic with rank 1 polymorphism, Hilbert choice and the Infinity axiom (recalled in §2.1)
- A definitional mechanism for introducing new types and constants *in an overloaded fashion* (recalled in §2.2)

### 2.1 HOL Syntax and Deduction

**Syntax.** All throughout this section, we fix the following:

- an infinite set $\mathsf{TVar}$, of *type variables*, ranged by $\alpha, \beta$
- an infinite set $\mathsf{VarN}$, of *(term) variables names*, ranged by $x, y, z$
- a set $K$ of symbols, ranged by $\kappa$, called *type constructors*, containing three special symbols: "$\mathsf{bool}$", "$\mathsf{ind}$" and "$\to$" (aimed at representing the type of booleans, an infinite type and the function type constructor, respectively)

We fix a function $\mathsf{arOf} : K \to \mathbb{N}$ giving arities to type constructors, such that $\mathsf{arOf}(\mathsf{bool}) = \mathsf{arOf}(\mathsf{ind}) = 0$ and $\mathsf{arOf}(\to) = 2$. Types, ranged by $\sigma, \tau$, are defined as follows:

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

Thus, a type is either a type variable or an $n$-ary type constructor $\kappa$ postfix-applied to a number of types corresponding to its arity. If $n = 1$, instead of $(\sigma)\,\kappa$ we write $\sigma\ k$.

Finally, we fix the following:

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

$$\begin{array}{ll} \mathsf{tpOf}(\longrightarrow) = \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool} & \mathsf{tpOf}(\mathsf{zero}) = \mathsf{ind} \\ \mathsf{tpOf}(=) = \alpha \to \alpha \to \mathsf{bool} & \mathsf{tpOf}(\mathsf{suc}) = \mathsf{ind} \to \mathsf{ind} \\ \mathsf{tpOf}(\varepsilon) = (\alpha \to \mathsf{bool}) \to \alpha & \end{array}$$

$\mathsf{TV}(\sigma)$ is the set of variables of a type $\sigma$. Given a function $\rho : \mathsf{TVar} \to \mathsf{Type}$, its *support* is the set of type variables where $\rho$ is not the identity: $\mathsf{supp}(\rho) = \{\alpha \mid \rho(\alpha) \neq \alpha\}$. A *type substitution* is a function $\rho : \mathsf{TVar} \to \mathsf{Type}$ with finite support. We let $\mathsf{TSubst}$ denote the set of type substitutions. Each $\rho \in \mathsf{TSubst}$ extends to a function $\overline{\rho} : \mathsf{Type} \to \mathsf{Type}$ by defining $\overline{\rho}(\alpha) = \rho(\alpha)$ and $\overline{\rho}((\sigma_1, \ldots, \sigma_n)\,\kappa) = (\overline{\rho}(\sigma_1), \ldots, \overline{\rho}(\sigma_n))\,\kappa$. We write $\sigma[\tau/\alpha]$ for $\overline{\rho}(\sigma)$, where $\rho$ is the type substitution that sends $\alpha$ to $\tau$ and each $\beta \neq \alpha$ to $\beta$. Thus, $\sigma[\tau/\alpha]$ is obtained from $\sigma$ by substituting $\tau$ for all occurrences of $\alpha$.

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

A *(typed) variable* is a pair of a variable name $x$ and a type $\sigma$, written $x_\sigma$. Let $\mathsf{Var}$ denote the set of all variables. A *constant instance* is a pair of a constant and a type, written $c_\sigma$, such that $\sigma \leq \mathsf{tpOf}(c)$. We let $\mathsf{CInst}$ denote the set of constant instances. We extend the notions of being an instance ($\leq$) and being orthogonal ($\#$) from types to constant instances:

$c_\tau \leq d_\sigma$ iff $c = d$ and $\tau \leq \sigma$ $\qquad$ $c_\tau \# d_\sigma$ iff $c \neq d$ or $\tau \# \sigma$

The tuple $(K, \mathsf{arOf}, \mathsf{Const}, \mathsf{tpOf})$, which will be fixed in what follows, is called a *signature*. This 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 typed variable, or a constant instance, or an application, or an abstraction. As usual, we identify terms modulo alpha-equivalence. Typing is defined as a binary relation between terms and types, written $t : \sigma$, inductively as follows:

$$\frac{x \in \mathsf{VarN}}{x_\sigma : \sigma} \qquad \frac{c \in \mathsf{Const} \quad \tau \leq \mathsf{tpOf}(c)}{c_\tau : \tau} \qquad \frac{t_1 : \sigma \to \tau \quad t_2 : \sigma}{t_1\ t_2 : \tau} \qquad \frac{t : \tau}{\lambda x_\sigma.\ t : \sigma \to \tau}$$

A term is a well-typed term if there exists a (necessarily unique) type $\tau$ such that $t : \tau$. We write $\mathsf{tpOf}(t)$ for this unique $\tau$. We let $\mathsf{Term}_w$ be the set of well-typed terms. We can apply a type substitution $\rho$ to a term $t$, written $\overline{\rho}(t)$, by applying $\overline{\rho}$ to the types of all variables and constant instances occurring in $t$. $\mathsf{FV}(t)$ 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 $s$ for all free occurrences of $x_\sigma$.

A *formula* is a term of type $\mathsf{bool}$. The formula connectives and quantifiers are defined in the standard way, starting from the implication and equality primitives. When writing terms, we sometimes omit the types of variables if they can be inferred.

**Deduction.** A *theory* is a set of formulas. The HOL axioms, forming a set denoted by $\mathsf{Ax}$, are the standard ones, containing axioms for equality, infinity, choice, and excluded middle. A *context* $\Gamma$ is a finite set of formulas. The notation $\alpha \notin \Gamma$ (or $x_\sigma \notin \Gamma$) means that the variable $\alpha$ (or $x_\sigma$) is not free in any of the formulas in $\Gamma$. We define *deduction* as a ternary relation $\vdash$ between theories $D$, contexts $\Gamma$ and formulas $\varphi$, written $D; \Gamma \vdash \varphi$.

$$\frac{}{D; \Gamma \vdash \varphi}\ [\varphi \in \mathsf{Ax} \cup D]\ (\textsc{Fact}) \qquad \frac{}{D; \Gamma \vdash \varphi}\ [\varphi \in \Gamma]\ (\textsc{Assum})$$

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

$$\frac{}{D; \Gamma \vdash (\lambda x_\sigma.\ t)\ s = t[s/x_\sigma]}\ (\textsc{Beta}) \qquad \frac{D; \Gamma \vdash \varphi \longrightarrow \chi \qquad D; \Gamma \vdash \varphi}{D; \Gamma \vdash \chi}\ (\text{MP})$$

$$\frac{D; \Gamma \cup \{\varphi\} \vdash \chi}{D; \Gamma \vdash \varphi \longrightarrow \chi}\ (\textsc{ImpI}) \qquad \frac{D; \Gamma \vdash f\ x_\sigma = g\ x_\sigma}{D; \Gamma \vdash f = g}\ [x_\sigma \notin \Gamma]\ (\textsc{Ext})$$

These rules are a variant the standard ones for HOL (as in, e.g., [6, 7]). A theory $D$ is called *consistent* if $D; \emptyset \nvdash \mathsf{False}$.

**Built-Ins and Non-Built-Ins.** A *built-in type* is any type of the form $\mathsf{bool}$, $\mathsf{ind}$, or $\sigma \to \tau$ for $\sigma, \tau \in \mathsf{Type}$. We let $\mathsf{Type}^\bullet$ denote the set of types that are *not* built-in. Note that a non-built-in type can have a built-in type as a subtype, and vice versa; e.g., if $\mathsf{list}$ is a type constructor, then $\mathsf{bool\ list}$ and $(\alpha \to \beta)\ \mathsf{list}$ are non-built-in types, whereas $\alpha \to \beta\ \mathsf{list}$ is a built-in type.

Given a type $\sigma$, we define $\mathsf{types}^\bullet(\sigma)$, the *set of non-built-in types* of $\sigma$, as follows:

$$\mathsf{types}^\bullet(\alpha) = \mathsf{types}^\bullet(\mathsf{bool}) = \mathsf{types}^\bullet(\mathsf{ind}) = \emptyset$$
$$\mathsf{types}^\bullet((\sigma_1, \ldots, \sigma_n)\,\kappa) = \{(\sigma_1, \ldots, \sigma_n)\,\kappa\},\ \text{if } \kappa \text{ is different from } \to$$
$$\mathsf{types}^\bullet(\sigma_1 \to \sigma_2) = \mathsf{types}^\bullet(\sigma_1) \cup \mathsf{types}^\bullet(\sigma_2)$$

Thus, $\mathsf{types}^\bullet(\sigma)$ is the smallest set of non-built-in types that can produce $\sigma$ by repeated application of the built-in type constructors. E.g., if the type constructors real (nullary) and list (unary) are in the signature and if $\sigma$ is $(\mathsf{bool} \to \alpha\ \mathsf{list}) \to \mathsf{real} \to (\mathsf{bool} \to \mathsf{ind})\ \mathsf{list}$, then $\mathsf{types}^\bullet(\sigma)$ has three elements: $\alpha\ \mathsf{list}$, real and $(\mathsf{bool} \to \mathsf{ind})\ \mathsf{list}$.

A built-in constant is a constant of the form $\longrightarrow$, $=$, $\varepsilon$, zero or suc. We let $\mathsf{CInst}^\bullet$ be the set of constant instances that are *not* instances of built-in constants.

As a general notation rule, the superscript $\bullet$ indicates non-built-in items, where an item can be either a type or a constant instance.

Given a term $t$, we let $\mathsf{consts}^\bullet(t) \subseteq \mathsf{CInst}^\bullet$ be the set of all non-built-in constant instances occurring in $t$ and $\mathsf{types}^\bullet(t) \subseteq \mathsf{Type}^\bullet$ be the set of all non-built-in types that compose the types of non-built-in constants and (free or bound) variables occurring in $t$. Note that the $\mathsf{types}^\bullet$ operator is overloaded for types and terms.

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

### 2.2 The Isabelle/HOL Definitional Mechanisms

**Constant Definitions.** Given $c_\sigma \in \mathsf{CInst}^\bullet$ 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 $\mathsf{TV}(t) \subseteq \mathsf{TV}(c_\sigma)$ (i.e., $\mathsf{TV}(t) \subseteq \mathsf{TV}(\sigma)$).

**Type Definitions.** Given the types $\tau \in \mathsf{Type}^\bullet$ and $\sigma \in \mathsf{Type}$ and the closed term $t$ whose type is $\sigma \to \mathsf{bool}$, we let $\tau \equiv t$ denote the formula

$$\begin{array}{l} (\exists x_\sigma.\ t\ x) \longrightarrow \\ \quad \exists \mathit{rep}_{\tau\to\sigma}.\ \exists \mathit{abs}_{\sigma\to\tau}. \\ \qquad (\forall x_\tau.\ t\ (\mathit{rep}\ x)) \wedge (\forall x_\tau.\ \mathit{abs}\ (\mathit{rep}\ x) = x) \wedge (\forall y_\sigma.\ t\ y \longrightarrow \mathit{rep}\ (\mathit{abs}\ y) = y). \end{array} \tag{1}$$

We call $\tau \equiv t$ a *type definition*, provided $\tau$ has the form $(\alpha_1, \ldots, \alpha_n)\,\kappa$ such that $\alpha_i$ are all distinct type variables and $\mathsf{TV}(t) \subseteq \{\alpha_1, \ldots, \alpha_n\}$. (Hence, we have $\mathsf{TV}(t) \subseteq \mathsf{TV}(\tau)$, which also implies $\mathsf{TV}(\sigma) \subseteq \mathsf{TV}(\tau)$.)

Thus, $\tau \equiv t$ means: provided $t$ represents a non-empty subset of $\sigma$, the new type $\tau$ is isomorphic to this subset via *abs* and *rep*. Note that this is a *conditional* type definition, which distinguishes Isabelle/HOL from other HOL-based provers where an *unconditional* version is postulated (but only after the user proves nonemptiness). We shall see that this conditional approach, known among the Isabelle developers as the Makarius Wenzel trick, is useful in the overall scheme of proving consistency.

However, as far as user interaction is concerned, Isabelle/HOL proceeds like the other HOL provers. When the user issues a command to define $\tau$ via $t : \sigma \to \mathsf{bool}$, the

system asks the user to prove $\exists x_\sigma.\ t\ x$, after which the new type $\tau$ and the unconditional version of formula (1) are produced, the later by Modus Ponens. Finally, the system automatically defines the morphisms *abs* and *rep* using the Hilbert choice operator $\epsilon$.

An Isabelle/HOL development proceeds by declaring types and constants, issuing constant instance and type definitions, and proving theorems about them via HOL deduction. Therefore, at any point in the development, there is a finite set $D$ of registered constant instance and/or type definitions (over a HOL signature $\Sigma$)—we call such a set a *definitional theory*. We are interested in proving the consistency of definitional theories, under the syntactic well-formedness restrictions imposed by the system.

**Well-Formed Definitional Theories.** Given any binary relation $R$ on $\mathsf{Type}^\bullet \cup \mathsf{CInst}^\bullet$, we write $R^\downarrow$ for its (type-)substitutive closure, defined as follows: $p\ R^\downarrow\ q$ iff there exist $p'$, $q'$ and a type substitution $\rho$ such that $p = \overline{\rho}(p')$, $q = \overline{\rho}(q')$ and $p'\ R\ q'$. We say that a relation $R$ is *terminating* if there exists no sequence $(p_i)_{i\in\mathbb{N}}$ such that $p_i\ R\ p_{i+1}$ for all $i$. We shall write $R^+$ and $R^*$ for the transitive and the reflexive-transitive closure of $R$.

Let us fix a definitional theory $D$. We say $D$ is *orthogonal* if the following hold for any two distinct definitions $def_1, def_2 \in D$:

- either one of them is a type definition and the other is a constant instance definition
- or both are type definitions with orthogonal left-hand sides, i.e., $def_1$ has the form $\tau_1 \equiv \ldots$, $def_2$ has the form $\tau_2 \equiv \ldots$, and $\tau_1 \,\#\, \tau_2$
- or both are constant instance definitions with orthogonal left-hand sides, i.e., $def_1$ has the form $c_{\tau_1} \equiv \ldots$, $def_2$ has the form $d_{\tau_2} \equiv \ldots$, and $c_{\tau_1} \# d_{\tau_2}$

We define the binary relation $\rightsquigarrow$ on $\mathsf{Type}^\bullet \cup \mathsf{CInst}^\bullet$ by setting $u \rightsquigarrow v$ iff one of the following holds:

1. there exists in $D$ a definition of the form $u \equiv t$ such that $v \in \mathsf{consts}^\bullet(t) \cup \mathsf{types}^\bullet(t)$
2. $u \in \mathsf{CInst}^\bullet$ such that $u$ has the form $c_{\mathsf{tpOf}(c)}$, and $v \in \mathsf{types}^\bullet(\mathsf{tpOf}(c))$

We call $\rightsquigarrow$ the *dependency relation* associated to $D$: it shows how the types and constant instances depend on each other through definitions in $D$. The fact that built-in items do not participate at this relation (as shown by the bullets which restrict to non-built-in items) is justified by the built-in items having a pre-determined interpretation, which prevents them from both "depending" and "being depended upon" [11].

We call the definitional theory $D$ *well-formed* if it is orthogonal and the substitutive closure of its dependency relation, $\rightsquigarrow^\downarrow$, is terminating. Orthogonality prevents inconsistency arising from overlapping left-hand sides of definitions: defining $c_{\alpha\times\mathsf{ind}\to\mathsf{bool}}$ to be $\lambda x_{\alpha\times\mathsf{ind}}.\mathsf{False}$ and $c_{\mathsf{ind}\times\alpha\to\mathsf{bool}}$ to be $\lambda x_{\mathsf{ind}\times\alpha}.\mathsf{True}$ yields $\lambda x_{\mathsf{ind}\times\mathsf{ind}}.\mathsf{False} = c_{\mathsf{ind}\times\mathsf{ind}\to\mathsf{bool}} = \lambda x_{\mathsf{ind}\times\mathsf{ind}}.\mathsf{True}$ and hence $\mathsf{False} = \mathsf{True}$. Termination prevents inconsistency arising from circularity, as in the encoding of Russel's paradox in the introduction.

In previous work [11, 13], we proved that these prevention measures are sufficient:

**Theorem 1.** If $D$ is well-formed, then $D$ is consistent.

Let us briefly recall the difficulties arising in proving the consistency theorem. A main problem problem rests in the fact that (recursive) overloading does not interact well with a set-theoretic semantics. This makes it difficult to give a meaning to the overloaded definitions, in spite of the fact that their syntactic dependency terminates.

**Example 2.** $\texttt{consts c} : \alpha \to \texttt{bool} \quad \texttt{consts d} : \alpha$
$\texttt{typedef (overloaded)}\ \alpha\ \texttt{k} \equiv \{\texttt{x} : \alpha \mid \texttt{x} = \texttt{c (d} : \alpha\texttt{)}\}$
$\texttt{def (overloaded) c} : \alpha\ \texttt{k} \to \texttt{bool} \equiv \lambda\ \texttt{x} : \alpha\ \texttt{k}.\ \neg\ \texttt{c (d} : \alpha\texttt{)}$

Writing $k^n$ for the $n$-times repeated application of the type constructor $k$, we have that $\alpha\, k^{n+1}$ depends on $\alpha\, k^n$. And such dependencies can of course happen indirectly, and involve any number of intermediate types and constants—here, the dependency involves $c_{\alpha\, k^n \to \mathsf{bool}}$, namely: $\alpha\, k^{n+1} \rightsquigarrow c_{\alpha\, k^n \to \mathsf{bool}} \rightsquigarrow \alpha\, k^n$

Well-formedness ensures that each dependency chain of instances terminates, e.g.:

$\mathsf{bool}\, k \rightsquigarrow^{\downarrow} c_{\mathsf{bool} \to \mathsf{bool}}$
$\mathsf{bool}\, k^2 \rightsquigarrow^{\downarrow} c_{\mathsf{bool}\, k \to \mathsf{bool}} \rightsquigarrow^{\downarrow} \mathsf{bool}\, k \rightsquigarrow^{\downarrow} c_{\mathsf{bool} \to \mathsf{bool}}$
$\mathsf{bool}\, k^3 \rightsquigarrow^{\downarrow} c_{\mathsf{bool}\, k^2 \to \mathsf{bool}} \rightsquigarrow^{\downarrow} \mathsf{bool}\, k^2 \rightsquigarrow^{\downarrow} c_{\mathsf{bool}\, k \to \mathsf{bool}} \rightsquigarrow^{\downarrow} \mathsf{bool}\, k \rightsquigarrow^{\downarrow} c_{\mathsf{bool} \to \mathsf{bool}}$

However, since the type constructor $k$ behaves differently depending on what it is applied to (e.g., bool, bool $k$, bool $k^2$), it is not clear how to interpret it as a set operator, as required by standard HOL semantics. Indeed, we would need a custom set theory where certain systems of equations admit fixed points. In addition, we would need to decorate the type constructor interpretation with syntactic annotations to ensure that, e.g., the interpretations of bool $k$ and bool do not overlap—since overlapping would yield an inconsistency in the interpretations of $k$ applied to these.

In [11, 13], we resolved into acknowledging that ad hoc overloading regards different instances of the same type as completely unrelated types. Instead of interpreting type constructors as operators on sets, we interpret each (non-built-in) ground type separately, in the order prescribed by the terminating dependency relation. Here, for example, bool $k$ is interpreted first, then bool $k^2$, then bool $k^3$. (But note that termination does not necessarily come from structural descent on types: definitions such as $d_{\mathsf{nat}} \equiv \mathsf{head}(d_{\mathsf{nat\ list}})$ are also acceptable.) Finally, polymorphic formulas are interpreted as the conjunction of the interpretation of all their ground instances: for example, $c_{\alpha \to \mathsf{bool}}\, d_\alpha$ is true iff $c_{\sigma \to \mathsf{bool}}\, d_\sigma$ is true for all ground types $\sigma$. This way, we were able to construct a ground model for the definitional theory. And after showing that the deduction rules for (polymorphic!) HOL are sound for ground models, we inferred consistency. In summary, our solution was a mixture of syntax and semantics: interpret type variables by universally quantifying over all ground instances, and interpret ground types disregarding their structure.

Such a hybrid approach, involving a non-standard semantics, may seem excessive. There is a more common-sense alternative for accommodating the observation that standard semantics and ad hoc overloading do not mesh well: view overloaded definitions as mere textual abbreviations. The "semantics" of an overloaded constant will then be the result of unfolding the definitions. This is the alternative taken by our new proof.

## 3 New Proof of Consistency

The HOL logical infrastructure allows unfolding constant definitions, but not type definitions. To amend this limitation, we take an approach common in mathematics. The reals were introduced by closing the rationals under Cauchy convergence, the complex numbers were introduced by closing the reals under roots of polynomials. Similarly,

we introduce a logic, HOL with Comprehension (HOLC), by closing HOL under type comprehension—that is, adding to HOL comprehension types to express subsets of the form $\{x : \sigma \mid t\ x\}$ (§3.1). While there is some tension between these subsets being possibly empty and the HOLC types having to be nonempty due to the Hilbert choice operator, this is resolved thanks to the HOLC comprehension axioms being conditioned by nonemptiness. With this proviso, HOLC admits standard set-theoretical models, making it manifestly consistent (§3.2). In turn, Isabelle/HOL-style overloaded constants and types can be normalized in HOLC by unfolding their definitions (§3.3). The normalization process provides an intuition and a justification for the syntactic checks involving non-built-in types and constants. Finally, consistency of Isabelle/HOL is inferable from consistency of HOLC.

### 3.1 HOL with Comprehension (HOLC)

**Syntax.** Just like for HOL, we fix the sets $\mathsf{TVar}$ (of type variables) and $\mathsf{VarN}$ (of term variable names), as well as the following:

- a set $K$ of type constructors including the built-in ones $\mathsf{bool}$, $\mathsf{ind}$, $\to$
- a function $\mathsf{arOf} : K \to \mathbb{N}$ assigning an arity to each type constructor.
- a set $\mathsf{Const}$ of constants, including the built-in ones $\longrightarrow$, $=$, $\varepsilon$, $\mathsf{zero}$ and $\mathsf{suc}$

The HOLC types and terms, which we call *ctypes* and *cterms*, are defined as follows:

$$\sigma = \alpha \mid (\sigma_1, \ldots, \sigma_{\mathsf{arOf}(\kappa)})\,\kappa \mid \{\!|t|\!\} \qquad\qquad t = x_\sigma \mid c_\sigma \mid t_1\ t_2 \mid \lambda x_\sigma.\, t$$

Emphasized is the only difference from the HOL types and terms: the comprehension types, whose presence make the ctypes and cterms mutually recursive. Indeed, $\{\!|t|\!\}$ contains the term $t$, whereas a typed variable $x_\sigma$ and a constant instance $c_\sigma$ contain the type $\sigma$. We think of a comprehension type $\{\!|t|\!\}$ with $t : \sigma \to \mathsf{bool}$ as representing a set of elements which in standard mathematical notation would be written $\{x : \sigma \mid t\ x\}$, that is, the set of all elements of $\sigma$ satisfying $t$. $\mathsf{Var}$ denotes the set of (typed) variables, $x_\sigma$. $\mathsf{CType}$ and $\mathsf{CTerm}$ denotes the sets of ctypes and cterms.

We also fix a function $\mathsf{tpOf} : \mathsf{Const} \to \mathsf{CType}$, assigning ctypes to constants. Similarly to the case of HOL, we call the tuple $(K, \mathsf{arOf}, \mathsf{Const}, \mathsf{tpOf})$, which shall be fixed in what follows, a *HOLC signature*. Since ctypes contain cterms, we define typing mutually recursively together with the notion of a ctype being well-formed (i.e., only containing well-typed terms):

$$\frac{\alpha \in \mathsf{TVar}}{\mathsf{wf}(\alpha)}\ (\mathrm{W}_1) \qquad \frac{\mathsf{wf}(\sigma_1)\ \ldots\ \mathsf{wf}(\sigma_{\mathsf{arOf}(\kappa)})}{\mathsf{wf}((\sigma_1, \ldots, \sigma_{\mathsf{arOf}(\kappa)})\,\kappa)}\ (\mathrm{W}_2) \qquad \frac{t :: \sigma \to \mathsf{bool}}{\mathsf{wf}(\{\!|t|\!\})}\ (\mathrm{W}_3) \qquad \frac{t :: \tau \quad \mathsf{wf}(\sigma)}{\lambda x_\sigma.\, t :: \sigma \to \tau}\ (\textsc{Abs})$$

$$\frac{x \in \mathsf{VarN} \quad \mathsf{wf}(\sigma)}{x_\sigma :: \sigma}\ (\textsc{Var}) \qquad \frac{c \in \mathsf{Const} \quad \mathsf{wf}(\tau) \quad \tau \leq \mathsf{tpOf}(c)}{c_\tau :: \tau}\ (\textsc{Const}) \qquad \frac{t_1 :: \sigma \to \tau \quad t_2 :: \sigma}{t_1\ t_2 :: \tau}\ (\textsc{App})$$

We let $\mathsf{CType}_w$ and $\mathsf{CTerm}_w$ be the sets of well-formed ctypes and well-typed cterms. Also, we let $\mathsf{Var}_w$ be the set of variables $x_\sigma$ that are well-typed as cterms, i.e., have their ctype $\sigma$ well-formed.

The notions of type substitution, a type or a constant instance being an instance of ($\leq$) or being orthogonal with (#) another type or constant instance, are defined similarly to those for HOL. Since in HOLC types may contain terms, we naturally lift term

concepts to types. Thus, the free (cterm) variables of a ctype $\sigma$, written $\mathsf{FV}(\sigma)$, are all the free variables occurring in the cterms that $\sigma$ contains. A type is called closed if it has no free variables.

**A note on declaration circularity.** In HOLC we allow $\mathsf{tpOf}$ to produce declarations circles—for example, the type of a constant may contain instances of that constant, as in $\mathsf{tpOf}(c) = \{\!|c_{\mathsf{bool}}|\!\}$. However, the typing system will ensure that no such cyclic entity will be well-typed. For example, to type an instance $c_\sigma$, we need to apply the rule (CONST), requiring that $\{\!|c_{\mathsf{bool}}|\!\}$ be well-formed. For the latter, we need the rule ($\mathsf{W}_3$), requiring that $c_{\mathsf{bool}}$ be well-typed. Finally, to type $c_{\mathsf{bool}}$, we again need the rule (CONST), requiring that $\{\!|c_{\mathsf{bool}}|\!\}$ be well-formed. So $c_\sigma$ can never be typed. It may seem strange to allow constant declarations whose instances cannot be typed (hence cannot belong to well-typed terms and well-formed types)—however, this is harmless, since HOLC deduction only deals with well-typed and well-formed items. Moreover, all the constants translated from HOL will be shown to be well-typed.

**Deduction.** The notion of formulas and all the related notions are defined similarly to HOL, so that HOL formulas are particular cases of HOLC formulas. In addition to the axioms of HOL (the set $\mathsf{Ax}$), HOLC shall include the following type comprehension axiom type_comp:

$$\begin{array}{l}\forall t_{\alpha\to\mathsf{bool}}.\ (\exists x_\alpha.\ t\ x) \longrightarrow \\ \quad \exists rep_{\{\!|t|\!\}\to\alpha}.\ \exists abs_{\alpha\to\{\!|t|\!\}}. \\ \quad\quad (\forall x_{\{\!|t|\!\}}.\ t\ (rep\ x)) \wedge (\forall x_{\{\!|t|\!\}}.\ abs\ (rep\ x) = x) \wedge (\forall y_\alpha.\ t\ y \longrightarrow rep\ (abs\ y) = y)\end{array}$$

This axiom is nothing but a generalization of the HOL type definition $\tau \equiv t$, taking advantage of the fact that in HOLC we have a way to write the expression defining $\tau$ as the type $\{\!|t|\!\}$. Note that $\alpha$ is a type variable standing for an arbitrary type, previously denoted by $\sigma$. Thus, HOLC allows us to express what used to be in HOL a schema (i.e., an infinite set of formulas, one for each type $\sigma$) by a single axiom.

HOLC's deduction $\Vdash$ is defined by the same rules as HOL's deduction $\vdash$, but applied to ctypes and cterms instead of types and terms and using the additional axiom type_comp. Another difference from HOL is that HOLC deduction does not factor in a theory $D$—this is because we do not include any definitional principles in HOLC.

$$\frac{}{\Gamma \Vdash \varphi}\ [\varphi \in \mathsf{Ax} \cup \{\mathsf{type\_comp}\}]\ (\textsc{Fact}) \qquad \frac{}{\Gamma \Vdash \varphi}\ [\varphi \in \Gamma]\ (\textsc{Assum})$$

$$\frac{\Gamma \Vdash \varphi}{\Gamma \Vdash \varphi[\sigma/\alpha]}\ [\alpha \notin \Gamma]\ (\text{T-}\textsc{Inst}) \qquad \frac{\Gamma \Vdash \varphi}{\Gamma \Vdash \varphi[t/x_\sigma]}\ [x_\sigma \notin \Gamma]\ (\textsc{Inst})$$

$$\frac{}{\Gamma \Vdash (\lambda x_\sigma.\ t)\ s = t[s/x_\sigma]}\ (\textsc{Beta}) \qquad \frac{\Gamma \Vdash \varphi \longrightarrow \chi \qquad \Gamma \Vdash \varphi}{\Gamma \Vdash \chi}\ (\textsc{MP})$$

$$\frac{\Gamma \cup \{\varphi\} \Vdash \chi}{\Gamma \Vdash \varphi \longrightarrow \chi}\ (\textsc{ImpI}) \qquad \frac{\Gamma \Vdash f\ x_\sigma = g\ x_\sigma}{\Gamma \Vdash f = g}\ [x \notin \Gamma]\ (\textsc{Ext})$$

### 3.2 Consistency of HOLC

In a nutshell, HOLC is consistent for a similar reason why HOL (without definitions) is consistent: the types have a straightforward set-theoretic interpretation and the deduction rules are manifestly sound w.r.t. this interpretation. Similar logics, employing mutual dependency between types and terms, have been shown to be consistent for the foundations of Coq [2] and PVS [18].

Compared to these logics, the only twist of HOLC is that all types have to be nonempty. Indeed, the Hilbert choice operator, $\varepsilon : (\alpha \to \mathsf{bool}) \to \alpha$, is central in the HOL infrastructure, as it allows Skolemizing any existentially quantified formula into a new constant. Therefore, if HOLC is to host an argument for HOL consistency, it has to inherit $\varepsilon$. And the very existence of this built-in constant forces all types $\sigma$ to be inhabited by $\varepsilon\ (\lambda x_\sigma.\ \mathsf{True})$.

From a technical point of view, this nonemptiness requirement is easy to satisfy. The only types that are "in danger" of being empty are the comprehension types $\{\!|t|\!\}$. We will interpret them according to their expected semantics, namely, as the subset of $\sigma$ for which $t$ holds, *only if this subset turns out to be empty*; otherwise we will interpret them as a fixed singleton set $\{*\}$. This is consistent with the HOLC comprehension axiom, type_comp, which only requires that $\{\!|t|\!\}$ have the expected semantics if $\exists x_\alpha.\ t\ x$ holds. Notice how the Makarius Wenzel trick of introducing type definitions as conditional statements in Isabelle/HOL, which has inspired a similar condition for type_comp, turns out to be very useful in our journey. Of all the HOL-based provers, this "trick" is only used by Isabelle/HOL, as if anticipating the need for a more involved argument for consistency.

**A full-frame model for HOLC.** We fix a Grothendieck universe $\mathcal{V}$ and let $\mathcal{U} = \mathcal{V} \setminus \{\emptyset\}$ (since all types will have nonempty interpretations). We fix the following items in $\mathcal{U}$ and operators on $\mathcal{U}$:

- a two-element set $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\} \in \mathcal{U}$
- a singleton set $\{*\} \in \mathcal{U}$
- for each $k \in K$, a function $\boxed{\mathrm{k}} : \mathcal{U}^{\mathsf{arOf}(k)} \to \mathcal{U}$
- a global choice function, choice, that assigns to each nonempty set $A \in \mathcal{U}$ an element $\mathsf{choice}(a) \in A$

We wish to interpret well-formed ctypes and well-typed cterms, $u$, as sets and elements $[u]$ in $\mathcal{U}$. Because $u$ may contain term and type variables, the interpretation $[u]$ will be parametrized by valuation functions $\xi : \mathsf{TVar} \cup \mathsf{Var}_w \to \mathcal{U}$—namely, $[u]$ will be a function from valuations to $\mathcal{U}$. However, not all such functions should be allowed, but only those that properly interpret variables $x_\sigma$ as elements of $[\sigma](\xi)$. But at the time when we define $[u]$ we cannot have $[\sigma]$ for all $x_\sigma$, since, e.g., $\sigma$ may contain $u$. We resolve this circularity in a standard way by restricting the valuation functions to the free (term) variables of $u$, namely, taking $\xi : \mathsf{TVar} \cup \mathsf{FV}(u) \to \mathcal{U}$. This works because $\sigma$ is structurally smaller than $u$ for all $x_\sigma \in \mathsf{FV}(u)$.

Given $u \in \mathsf{CType}_w \cup \mathsf{CTerm}_w$, we let $\mathsf{types}(u)$ be the set of $u$'s subtypes, i.e., the set of all types appearing in $u$. Given a ctype interpretation function $I : \mathsf{types}(u) \to \mathcal{U}$, an $(u, I)$-*valuation* is a function $\xi : \mathsf{TVar} \cup \mathsf{FV}(u) \to \mathcal{U}$ such that $\xi(x_\sigma) \in I(\sigma)$ for all $x_\sigma \in \mathsf{FV}(u)$. We let $\mathsf{Val}_{u,I}$ be the set of $(u, I)$-valuations. We now define $[\_] :$

$\prod_{u \in \mathsf{CType}_w \cup \mathsf{CTerm}_w} \mathsf{Val}_{u,I_u} \to \mathcal{U}$, where $I_u$ is the restriction of $[\_]$ to $\mathsf{types}(u)$. The definition proceeds by structural recursion on $u$. First, the equations for type interpretations:

$$[\alpha](\xi) = \xi(\alpha) \tag{2}$$
$$[\mathsf{bool}](\xi) = \mathbb{B} \tag{3}$$
$$[\mathsf{ind}](\xi) = \mathbb{N} \quad \text{(the set of natural numbers)} \tag{4}$$
$$[\sigma \to \tau](\xi) = [\sigma](\xi) \to [\tau](\xi) \quad \text{(the set of functions from } [\sigma](\xi) \text{ to } [\tau](\xi)) \tag{5}$$
$$[(\sigma_1, \ldots, \sigma_n)\ \kappa](\xi) = \boxed{\mathrm{k}}\,([\sigma_1](\xi), \ldots, [\sigma_n](\xi)) \text{ where } \overline{\sigma}\ \kappa \in \mathsf{Type}^\bullet \tag{6}$$
$$[\{\!|t|\!\}](\xi) = \begin{cases} \{x \in [\sigma](\xi) \mid [t](\xi)\ x = \mathsf{true}\} & \text{if set non-empty and } t : \sigma \to \mathsf{bool} \\ \{*\} & \text{otherwise} \end{cases} \tag{7}$$

The equation (7) shows how we interpret comprehension types with no inhabitants (e.g., $\{\!|\lambda x_{\mathsf{ind}}.\ \mathsf{False}|\!\}$)—we chose the singleton set $\{*\}$ (in fact, any non-empty set would do the job). As previously discussed, this conforms to the type_comp axiom, which only prescribes the meaning of inhabited comprehension types.

Next, the equations for term interpretations:

$$[\longrightarrow_{\mathsf{bool}\to\mathsf{bool}\to\mathsf{bool}}](\xi) \text{ as the logical implication on } \mathbb{B} \tag{8}$$
$$[=_{\tau\to\tau\to\mathsf{bool}}](\xi) \text{ as the equality predicate in } [\tau](\xi) \to [\tau](\xi) \to \mathbb{B} \tag{9}$$
$$[\varepsilon_{(\tau\to\mathsf{bool})\to\tau}](\xi)(f) = \begin{cases} \mathsf{choice}(A_f) & \text{if } A_f \text{ is non-empty} \\ \mathsf{choice}([\tau](\xi)) & \text{otherwise,} \end{cases} \tag{10}$$
$$\text{where } A_f = \{a \in [\tau](\xi) \mid f(a) = \mathsf{true}\} \text{ for each } f : [\tau](\xi) \to \mathbb{B}$$
$$[\mathsf{zero}_{\mathsf{ind}}](\xi) = 0 \text{ and } [\mathsf{suc}_{\mathsf{ind}\to\mathsf{ind}}](\xi) \text{ as the successor function for } \mathbb{N} \tag{11}$$
$$[x_\sigma](\xi) = \xi(x_\sigma) \tag{12}$$
$$[c_\sigma](\xi) = \mathsf{choice}\,([\sigma](\xi)) \tag{13}$$
$$[t_1\ t_2](\xi) = [t_1](\xi_1)\ [t_2](\xi_2) \text{ where } \xi_i \text{ is the restriction of } \xi \text{ to } \mathsf{TVar} \cup \mathsf{FV}(t_i) \tag{14}$$
$$[\lambda x_\sigma.\ t](\xi) = \Lambda_{a \in [\sigma](\xi)}[t](\xi[x_\sigma \leftarrow a]) \tag{15}$$

Since the logic has no definitions, we are free to choose any interpretation for non-built-in constant instances—as seen in (13), we do this using the global choice operator $\mathsf{choice}$. In (15), we use $\Lambda$ for meta-level lambda-abstraction.

Given a context $\Gamma$ and a formula $\varphi$ we say that $(\Gamma, \varphi)$ *is true under the valuation* $\xi \in \mathsf{Val}_{\varphi,I_\varphi}$, written $\Gamma \models_\xi \varphi$, if $[\varphi](\xi) = \mathsf{true}$ whenever $[\psi](\xi) = \mathsf{true}$ for all $\psi \in \Gamma$. We say that $(\Gamma, \varphi)$ *is true*, written $\Gamma \models \varphi$, if $\Gamma \models_\xi \varphi$ for all valuations $\xi \in \mathsf{Val}_{\varphi,I_\varphi}$.

**Theorem 3.** HOLC is consistent, in that $\emptyset \nvdash \mathsf{False}$.

*Proof.* It is routine to verify that HOLC's deduction is sound w.r.t. to its semantics: for every HOLC deduction rule of the form

$$\frac{\Gamma_1 \vdash \varphi_1 \qquad \ldots \qquad \Gamma_n \vdash \varphi_n}{\Gamma \vdash \varphi}$$

it holds that $\Gamma \models \varphi$ if $\Gamma_i \models \varphi_i$ for all $i \leq n$. Then $\emptyset \nvdash \mathsf{False}$ follows from $\emptyset \not\models \mathsf{False}$. □

### 3.3 Translation of Isabelle/HOL to HOLC

We fix a HOL signature $\Sigma = (K, \mathsf{arOf}, \mathsf{Const}, \mathsf{tpOf})$ and an Isabelle/HOL well-formed definitional theory $D$ over $\Sigma$. We will produce a translation of the types and well-typed terms of $\Sigma$ into well-formed ctypes and well-typed cterms of the HOLC signature $\Sigma_D = (K, \mathsf{arOf}, \mathsf{Const}, \mathsf{tpOf}_D)$ (having the same type constructors and constants as $\Sigma$). The typing function $\mathsf{tpOf}_D : \mathsf{Const} \to \mathsf{CType}$ will be defined later. For $\Sigma_D$, we use all the notations from §3.1—we write $\mathsf{CType}$ and $\mathsf{CTerm}$ for the sets of cterms and ctypes, etc.

The translation will consist of two homonymous "normal form" functions $\mathsf{NF} : \mathsf{Type} \to \mathsf{CType}_w$ and $\mathsf{NF} : \mathsf{Term}_w \to \mathsf{CTerm}_w$. However, since we have not yet defined $\mathsf{tpOf}_D$, the sets $\mathsf{CType}_w$ and $\mathsf{CTerm}_w$ (of well-formed ctypes and well-typed cterms) are not yet defined either. To bootstrap the definitions, we first define $\mathsf{NF} : \mathsf{Type} \to \mathsf{CType}$ and $\mathsf{NF} : \mathsf{Term}_w \to \mathsf{CTerm}$, then define $\mathsf{tpOf}_D$, and finally show that the images of the $\mathsf{NF}$ functions are included in $\mathsf{CType}_w$ and $\mathsf{CTerm}_w$.

The $\mathsf{NF}$ functions are defined mutually recursively by two kinds of equations. First, there are equations for recursive descent in the structure of terms and types:

$$\mathsf{NF}(t_1\ t_2) = \mathsf{NF}(t_1)\ \mathsf{NF}(t_2) \tag{16}$$
$$\mathsf{NF}(\lambda x_\sigma.\ t) = \lambda x_{\mathsf{NF}(\sigma)}.\ \mathsf{NF}(t) \tag{17}$$
$$\mathsf{NF}(x_\sigma) = x_{\mathsf{NF}(\sigma)} \tag{18}$$
$$\mathsf{NF}(c_\sigma) = c_{\mathsf{NF}(\sigma)} \text{ if } c_\sigma \notin \mathsf{CInst}^\bullet \tag{19}$$
$$\mathsf{NF}(\sigma \to \tau) = \mathsf{NF}(\sigma) \to \mathsf{NF}(\tau) \tag{20}$$
$$\mathsf{NF}(\mathsf{bool}) = \mathsf{bool} \tag{21}$$
$$\mathsf{NF}(\mathsf{ind}) = \mathsf{ind} \tag{22}$$
$$\mathsf{NF}(\alpha) = \alpha \tag{23}$$

Second, there are equations for unfolding the definitions in $D$. But before listing these, we need some notation. Given $u, v \in \mathsf{Type} \cup \mathsf{Term}_w$, we write $u \equiv^\downarrow v$ to mean that there exists a definition $u' \equiv v'$ in $D$ and a type substitution $\rho$ such that $u = \rho(u')$ and $v = \rho(v')$. This notation is intuitively consistent (although slightly abusively so) with the notation for the substitutive closure of a relation, where we would pretend that $\equiv$ is a relation on $\mathsf{Type} \cup \mathsf{Term}_w$, with $u' \equiv v'$ meaning $(u' \equiv v') \in D$. By Orthogonality, we have that, for all $u \in \mathsf{Type}^\bullet \cup \mathsf{CInst}^\bullet$, there exists at most one $v \in \mathsf{Type} \cup \mathsf{Term}_w$ such that $u \equiv^\downarrow v$. Here are the equations for unfolding:

$$\mathsf{NF}(c_\sigma) = \begin{cases} c_{\mathsf{NF}(\sigma)} & \text{if there is no matching definition for } c_\sigma \text{ in } D \\ \mathsf{NF}(t) & \text{if there exists } t \text{ such that } c_\sigma \equiv^\downarrow t \end{cases} \tag{24}$$

$$\mathsf{NF}(\sigma) = \begin{cases} \sigma & \text{if there is no matching definition for } \sigma \text{ in } D \\ \{\!|\mathsf{NF}(t)|\!\} & \text{if there exists } t : \tau \to \mathsf{bool} \text{ such that } \sigma \equiv^\downarrow t \end{cases} \tag{25}$$

Thus, the functions $\mathsf{NF}$ first traverse the terms and types "vertically," delving into the built-in structure (function types, $\lambda$-abstractions, applications, etc.). When a non-built-in item is being reached that is matched by a definition in $D$, $\mathsf{NF}$ proceed "horizontally" by unfolding this definition. Since the right-hand side of the definition can be any term, $\mathsf{NF}$ switch again to vertical mode. Hence, $\mathsf{NF}$ repeatedly unfold the definitions when a definitional match in a subexpression is being found, following a topmost-first strategy (with the exception that proper subexpressions of non-built-in types are not investigated). For example, if a constant $c_\sigma$ is matched by a definition, as in $c_\sigma \equiv^\downarrow t$, then $c_\sigma$ is eagerly unfolded to $t$, as opposed to unfolding the items occurring in $\sigma$. This seems to be a reasonable strategy, given that after unfolding $c_\sigma$ the possibility to process $\sigma$ is not lost: since $t : \sigma$, we have that $\sigma$ occurs in $t$.

Let us show the results of $\mathsf{NF}$ on some of Example 2's constant instances and types. To keep the expressions simple, we omit the types of the bound occurrences of variables.

$$\begin{aligned}
&\mathsf{NF}(\alpha\, k) \;=\; \{\!|\lambda x_\alpha.\, x_\alpha = c_{\alpha\to\mathsf{bool}}\, d_\alpha|\!\}\\
&\mathsf{NF}(c_{\mathsf{bool}\, k\to\mathsf{bool}}) \;=\; \lambda x_{\{\!|\lambda x_{\mathsf{bool}}.\; x=c_{\mathsf{bool}\to\mathsf{bool}}\, d_{\mathsf{bool}}|\!\}}.\, c_{\mathsf{bool}\to\mathsf{bool}}\, d_{\mathsf{bool}}\\
&\mathsf{NF}(\mathsf{bool}\, k^2) \;=\; \{\!|\lambda x_{\{\!|\lambda x_{\mathsf{bool}}.\; x=c_{\mathsf{bool}\to\mathsf{bool}}\, d_{\mathsf{bool}}|\!\}}.\, x = (\lambda x_{\{\!|\lambda x_{\mathsf{bool}}.\; x=c_{\mathsf{bool}\to\mathsf{bool}}\, d_{\mathsf{bool}}|\!\}}.\, c_{\mathsf{bool}\to\mathsf{bool}}\\
&\qquad\qquad d_{\mathsf{bool}})\; d_{\{\!|\lambda x_{\mathsf{bool}}.\; x=c_{\mathsf{bool}\to\mathsf{bool}}\, d_{\mathsf{bool}}|\!\}}\,|\!\}
\end{aligned}$$

The evaluation of $\mathsf{NF}$ on $(\mathsf{bool}\, k^n)$ terminates in a number of steps depending on $n$, and the result contains comprehension types nested $n$ levels.

The first fact that we need to show is that $\mathsf{NF}$ is well-defined, i.e., its recursive calls terminate. For this, we take the relation $\blacktriangleright$ to be $\equiv^{\downarrow} \cup\; \rhd$, where $\equiv^{\downarrow}$ was defined above and $\rhd$ simply contains the structural recursive calls of $\mathsf{NF}$:

$$\begin{array}{llll}
t_1 t_2 \rhd t_1 & \lambda x_\sigma.\, t \rhd \sigma & x_\sigma \rhd \sigma & \sigma_1 \to \sigma_2 \rhd \sigma_1\\
t_1 t_2 \rhd t_2 & \lambda x_\sigma.\, t \rhd t & c_\sigma \rhd \sigma & \sigma_1 \to \sigma_2 \rhd \sigma_2
\end{array}$$

It is immediate to see that $\blacktriangleright$ captures the recursive calls of $\mathsf{NF}$: the structural calls via $\rhd$ and the unfolding calls via $\equiv^{\downarrow}$. So the well-definedness of $\mathsf{NF}$ is reduced to the termination of $\equiv^{\downarrow}$.

**Lemma 4.** The relation $\blacktriangleright$ is terminating (hence the functions $\mathsf{NF}$ are well-defined).

*Proof.* We shall use the following crucial fact, which follows by induction using the definitions of $\rhd$ and $\rightsquigarrow^{\downarrow}$: If $u, v \in \mathsf{Type}^{\bullet} \cup \mathsf{CInst}^{\bullet}$ and $u \equiv^{\downarrow} t \rhd^{*} v$, then $u \rightsquigarrow^{\downarrow} v$. (*)

Let us assume by absurd that $\blacktriangleright$ does not terminate. Then there exists an infinite sequence $(w_i)_{i\in\mathbb{N}}$ such that $w_i \blacktriangleright w_{i+1}$ for all $i$. Since $\blacktriangleright$ is defined as $\equiv^{\downarrow} \cup \rhd$ and $\rhd$ clearly terminates, there must exist an infinite subsequence $(w_{i_j})_{j\in\mathbb{N}}$ such that $w_{i_j} \equiv^{\downarrow} w_{i_j+1} \rhd^{*} w_{i_{j+1}}$ for all $j$. Since from the definition of $\equiv$ we have $w_{i_j} \in \mathsf{Type}^{\bullet} \cup \mathsf{CInst}^{\bullet}$, we obtain from (*) that $w_{i_j} \rightsquigarrow^{\downarrow} w_{i_{j+1}}$ for all $j$. This contradicts the termination of $\rightsquigarrow^{\downarrow}$. □

With $\mathsf{NF}$ in place, we can define the missing piece of the target HOLC signature: we take $\mathsf{tpOf}_D$ to be the normalized version of $\mathsf{tpOf}$, i.e. $\mathsf{tpOf}_D(c) = \mathsf{NF}(\mathsf{tpOf}(c))$.

**Lemma 5.** $\mathsf{NF}$ preserves typing, in the following sense:

- $\mathsf{NF}(\sigma)$ is well-formed in HOLC.
- If $t : \tau$, then $\mathsf{NF}(t) :: \mathsf{NF}(\tau)$.

Our main theorem about the translation will be its soundness:

**Theorem 6.** If $D; \emptyset \vdash \varphi$ in HOL, then $\emptyset \Vdash \mathsf{NF}(\varphi)$ in HOLC.

Let us focus on proving this theorem. If we define $\mathsf{NF}(\Gamma)$ as $\{\mathsf{NF}(\varphi) \mid \varphi \in \Gamma\}$, the proof that $D; \Gamma \vdash \varphi$ implies $\mathsf{NF}(\Gamma) \Vdash \mathsf{NF}(\varphi)$ should proceed by induction on the definition of $D; \Gamma \vdash \varphi$. Due to the similarity of $\vdash$ and $\Vdash$, most of the cases go smoothly.

For the HOL rule (BETA), we need to prove $\mathsf{NF}(\Gamma) \Vdash \mathsf{NF}((\lambda x_\sigma.\, t)\; s = t[s/x_\sigma])$, that is, $\mathsf{NF}(\Gamma) \Vdash (\lambda x_{\mathsf{NF}(\sigma)}.\, \mathsf{NF}(t))\; \mathsf{NF}(s) = \mathsf{NF}(t[s/x_\sigma])$. Hence, in order to conclude the proof for this case using the HOLC rule (BETA), we need that $\mathsf{NF}$ commutes with term substitution—this is not hard to show, since substituting terms for variables does not influence the matching of definitions, i.e., the behavior of $\mathsf{NF}$:

**Lemma 7.** $\mathsf{NF}(t[s/x_\sigma]) = \mathsf{NF}(t)\; [\mathsf{NF}(s)/x_{\mathsf{NF}(\sigma)}]$

However, our proof (of Theorem 6) gets stuck when handing the (T-INST) case. It is worth looking at this difficulty, since it is revealing about the nature of our encoding. We assume that in HOL we inferred $D; \Gamma \vdash \varphi[\sigma/\alpha]$ from $D; \Gamma \vdash \varphi$, where $\alpha \notin \Gamma$. By the induction hypothesis, we have $\mathsf{NF}(\Gamma) \Vdash \mathsf{NF}(\varphi)$. Hence, by applying (T-INST) in HOLC, we obtain $\mathsf{NF}(\Gamma) \Vdash \mathsf{NF}(\varphi)[\mathsf{NF}(\sigma)/\mathsf{NF}(\alpha)]$. Therefore, to prove the desired fact, we would need that $\mathsf{NF}$ commutes with type substitutions in formulas, and therefore also in arbitrary terms (which may be contained in formulas):

$$\mathsf{NF}(t[\sigma/\alpha]) = \mathsf{NF}(t)[\mathsf{NF}(\sigma)/\alpha]$$

But this is not true, as seen, e.g., when $\mathsf{tpOf}(c) = \alpha$ and $c_{\mathsf{bool}} \equiv \mathsf{True}$ is in $D$:

$$\mathsf{NF}(c_\alpha[\mathsf{bool}/\alpha]) = \mathsf{NF}(c_{\mathsf{bool}}) = \mathsf{True} \neq c_{\mathsf{bool}} = c_\alpha[\mathsf{bool}/\alpha] = \mathsf{NF}(c_\alpha)\ [\mathsf{NF}(\mathsf{bool})/\alpha]$$

The problem rests in the very essence of overloading: a constant $c$ is declared at a type $\sigma$ ($\alpha$ in the above example) and defined at a less general type $\tau$ (bool in the example). Our translation reflects this: it leaves $c_\sigma$ as it is, whereas it compiles away $c_\tau$ by unfolding its definition. So then how can such a translation be sound? Essentially, it is sound because in HOL nothing interesting can be deduced about the undefined $c_\sigma$ that may affect what is being deduced about $c_\tau$—hence it is OK to decouple the two when moving to HOLC.

To capture this notion, of an undefined $c_\sigma$ not affecting a defined instance $c_\tau$ in HOL, we introduce a variant of HOL deduction that restricts type instantiation—in particular, it will not allow arbitrary statements about $c_\sigma$ to be instantiated to statements about $c_\tau$. Concretely, we define $\vdash'$ by modifying $\vdash$ as follows. We remove (T-INST) and strengthen (FACT) to a rule that combines the use of axioms with type instantiation, where $\rho$ is a type substitution:

$$\frac{}{D; \Gamma \vdash' \overline{\rho}(\varphi)}\ [\varphi \in \mathsf{Ax} \cup D,\ \forall \alpha \in \mathsf{supp}(\rho).\ \alpha \notin \Gamma]\ \text{(FACT-T-INST)}$$

Note the difference of (FACT-T-INST) from the combination of (FACT) and (T-INST): in the former, only axioms and elements of $D$ are allowed to be type-instantiated, whereas in the latter instantiation can occur at any moment in the proof. It is immediate to see that $\vdash$ is at least as powerful as $\vdash'$, since (FACT-T-INST) can de simulated by (FACT) and (T-INST). Conversely, it is routine to show that $\vdash'$ is closed under type substitution, and a fortiori under (T-INST); and (FACT-T-INST) is stronger than (FACT).

Using $\vdash'$ instead of $\vdash$, we can prove Theorem 6. All the cases that were easy with $\vdash$ are also easy with $\vdash'$. In addition, for the case (FACT-T-INST) where one infers $D; \Gamma \vdash \rho(\varphi)$ with $\varphi \in \mathsf{Ax}$, we need a less general lemma than commutation of $\mathsf{NF}$ in an arbitrary term. Namely, *noticing that the HOL axioms do not contain non-built-in constants or types*, we need the following lemma, which can be proved by routine induction over $t$:

**Lemma 8.** $\mathsf{NF}(\overline{\rho}(t)) = \overline{\mathsf{NF} \circ \rho}(t)$ whenever $\mathsf{types}^\bullet(t) \cup \mathsf{consts}^\bullet(t) = \emptyset$.

Now, assume (FACT-T-INST) is being used to derive $D; \Gamma \vdash' \overline{\rho}(\varphi)$ for $\varphi \in \mathsf{Ax}$. We need to prove $\Gamma \Vdash \mathsf{NF}(\overline{\rho}(\varphi))$, that is, $\Gamma \Vdash \overline{\mathsf{NF} \circ \rho}(\varphi)$. But this follows from $n$ applications of the (T-INST) rule (in HOLC), where $n$ is the size of $\overline{\mathsf{NF} \circ \rho}$'s support (as any finite-support simultaneous substitution can be reduced to a sequence of unary substitutions).

It remains to handle the case when (FACT-T-INST) is being used to derive $D; \Gamma \vdash' \overline{\rho}(\varphi)$ for $\varphi \in D$. Here, Lemma 8 does not apply, since of course the definitions in $D$

contain non-built-in items. However, we can take advantage of the particular shape of the definitions. The formula $\varphi$ necessarily has the form $u \equiv t$. By Orthogonality, it follows that $\overline{\rho}(t)$ is the unique term such that $\overline{\rho}(u) \equiv^{\downarrow} \overline{\rho}(t)$. We have two cases:

- If $u$ is a constant instance $c_\sigma$, then by the definition of $\mathsf{NF}$ we have $\mathsf{NF}(\overline{\rho}(u)) = \mathsf{NF}(\overline{\rho}(t))$. But then $\Gamma \Vdash \mathsf{NF}(\rho(\varphi))$, that is, $\Gamma \Vdash \mathsf{NF}(\overline{\rho}(u)) = \mathsf{NF}(\overline{\rho}(t))$, follows from (FACT) applied with the reflexivity axiom.
- If $u$ is a type $\sigma$ and $t : \tau \to \mathsf{bool}$, then $\overline{\rho}(\varphi)$ is $\overline{\rho}(\sigma) \equiv^{\downarrow} \overline{\rho(t)}$. In other words, $\overline{\rho}(\varphi)$ has the format of a HOL type definition, just like $\varphi$. Hence, $\mathsf{NF}(\overline{\rho}(\varphi))$ is seen to be an instance of type_comp, namely, type_comp$[\mathsf{NF}(\overline{\rho}(\sigma))/\alpha]$ together with $\mathsf{NF}(\overline{\rho}(t))$ substituted for the first quantifier. Hence $\Gamma \Vdash \mathsf{NF}(\overline{\rho}(\varphi))$ follows from (FACT) applied with type_comp, followed by $\forall$-instantiation (the latter being the standardly derived rule for $\forall$).

In summary, our HOLC translation of overloading emulates overloading itself in that it treats the defined constant instances $c_\tau$ as being disconnected from their "mother" instances $c_{\mathsf{tpOf}(c)}$. The translation is sound thanks to the fact that the considered theory has no axioms about these constants besides the overloaded definitions. This sound translation immediately allows us to reach our overall goal:

**Proof of Theorem 1 (Consistency of Isabelle/HOL).** By contradiction. Let $D; \emptyset \vdash \mathsf{False}$. Then by Theorem 6, we obtain $\emptyset \Vdash \mathsf{NF}(\mathsf{False})$ and since $\mathsf{NF}(\mathsf{False}) = \mathsf{False}$, we derive contradiction with Theorem 3. □

## 4 Conclusions and Related Work

It took the Isabelle/HOL community almost twenty years to reach a definitional mechanism that is indeed definitional, i.e., consistent by construction, w.r.t. both types and constants.[3] This paper, which presents what we claim to be a clean syntactic argument for consistency, is a culmination of previous efforts by Wenzel [22], Obua [17], ourselves [10, 11], and many other Isabelle designers and developers.

The key ingredients of our proof are a type-instantiation restricted version of HOL deduction and a gentle extension of HOL with comprehension types. This logic, called HOLC, is similar to a restriction of Coq's Calculus of Inductive Constructions (CiC) [3], where: (a) proof irrelevance and excluded middle axioms are enabled; (b) polymorphism is restricted to rank 1; (c) the formation of (truly) dependent product types is suppressed. However, unlike CiC, HOLC sticks to the HOL tradition of *avoiding empty types* (hence enabling unconditional Hilbert choice which plays a central role in HOL). HOLC also bears some similarities to HOL with predicate subtyping [20] as featured by PVS [21]. Yet, HOLC does not have real subtyping: from $t : \sigma \to \mathsf{bool}$ and $s :: \{\!|t|\!\}$ we cannot infer $s :: \sigma$. Instead, HOLC retains HOL's separation between a type defined by comprehension and the original type: the former is not included, but merely embedded in the latter. Finally, comprehension types are also known in the programming language literature as refinement types [5].

[3] Isabelle is by no means the only prover with longstanding foundational issues—see our "inconsistency club" discussion from [11].

The soundness of our translation from Isabelle/HOL to HOLC (Theorem 6), is useful beyond the goal of this paper, of establishing consistency. In [12], we use it to prove the soundness of new rules proposed for Isabelle/HOL.

**Acknowledgments.** We thank Tobias Nipkow, Larry Paulson, Makarius Wenzel, and the members of the Isabelle mailing list for inspiring and occasionally intriguing opinions and suggestions concerning the foundations of Isabelle/HOL.

## References

1. Isabelle Foundation & Certification (2015), archived at `https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2015-September/thread.html`
2. Barras, B.: Sets in Coq, Coq in Sets. Journal of Formalized Reasoning 3(1) (2010)
3. Bertot, Y., Casteran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Springer (2004)
4. Fallenstein, B., Kumar, R.: Proof-producing reflection for HOL - with an application to model polymorphism. In: ITP. pp. 170–186 (2015)
5. Freeman, T., Pfenning, F.: Refinement types for ML. In: PLDI. pp. 268–277 (1991)
6. Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)
7. Harrison, J.: Towards self-verification of HOL Light. In: IJCAR 2006. Springer (2006)
8. 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)
9. Kumar, R., Arthan, R., Myreen, M., Owens, S.: HOL with Definitions: Semantics, Soundness, and a Verified Implementation. In: ITP. Springer (2014)
10. Kunčar, O.: Correctness of Isabelle's cyclicity checker: Implementability of overloading in proof assistants. In: CPP. pp. 85–94 (2015)
11. Kunčar, O., Popescu, A.: A Consistent Foundation for Isabelle/HOL. In: ITP. pp. 234–252 (2015)
12. Kunčar, O., Popescu, A.: From Types To Sets By Local Type Definitions in Higher-Order Logic (2016), submitted to ITP 2016. Available at `http://www21.in.tum.de/~kuncar/kuncar-popescu-t2s2016.pdf`
13. Kunčar, O., Popescu, A.: A consistent foundation for Isabelle/HOL (2016), submitted. Available at `http://www21.in.tum.de/~kuncar/kuncar-popescu-jar2016.pdf`
14. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. vol. 2283 of LNCS, Springer (2002)
15. Nipkow, T., Klein, G.: Concrete Semantics - With Isabelle/HOL. Springer (2014)
16. Nipkow, T., Snelting, G.: Type Classes and Overloading Resolution via Order-Sorted Unification. In: Functional Programming Languages and Computer Architecture (1991)
17. Obua, S.: Checking Conservativity of Overloaded Definitions in Higher-Order Logic. In: RTA. Springer (2006)
18. Owre, S., Shankar, N.: The formal semantics of PVS (March 1999), sRI technical report. `http://www.csl.sri.com/papers/csl-97-2/`
19. Paulson, L.: Personal communication (2014)
20. Rushby, J.M., Owre, S., Shankar, N.: Subtypes for specifications: Predicate subtyping in PVS. IEEE Trans. Software Eng. 24(9), 709–720 (1998)
21. Shankar, N., Owre, S., Rushby, J.M.: PVS Tutorial. Computer Science Laboratory, SRI International (1993)
22. Wenzel, M.: Type Classes and Overloading in Higher-Order Logic. In: TPHOLS '97

# APPENDIX

## A More details on HOL

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 $\lambda x.\ (y_{\sigma\to\tau}\ x)$ instead of $\lambda x_\sigma.\ (y_{\sigma\to\tau}\ x)$ or $\varepsilon(\lambda x_\sigma.\ P\ x)$ instead of $\varepsilon_{(\sigma\to\mathsf{bool})\to\sigma}(\lambda x_\sigma.\ P_{\sigma\to\mathsf{bool}}\ x)$.
- 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 $\varepsilon$ 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 in the standard way, starting from the implication and equality primitives:

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

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

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

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 formulas:

Equality:

$$\begin{array}{lcl}
\mathsf{refl} & = & x_\alpha = x \\
\mathsf{subst} & = & x_\alpha = y \longrightarrow P\ x \longrightarrow P\ y \\
\mathsf{iff} & = & (p \longrightarrow q) \longrightarrow (q \longrightarrow p) \longrightarrow (p = q)
\end{array}$$

Infinity:

$$\begin{array}{lcl}
\mathsf{suc_inj} & = & \mathsf{suc}\ x = \mathsf{suc}\ y \longrightarrow x = y
\end{array}$$

$$\begin{array}{rcl} \mathsf{suc\_not\_zero} & = & \neg\,(\mathsf{suc}\ x = \mathsf{zero}) \\ \text{Choice:} & & \\ \mathsf{some\_intro} & = & p_{\alpha\to\mathsf{bool}}\ x \longrightarrow p\ (\varepsilon\ p) \\ \text{Excluded Middle:} & & \\ \mathsf{True\_or\_False} & = & (b = \mathsf{True}) \vee (b = \mathsf{False}) \end{array}$$

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 axiomatizes Hilbert choice. Finally, True_or_False makes the logic classical.

## B More details on HOLC

**Lemma 9.** 1. $c_{\mathsf{tpOf}(c)} \in \mathsf{CTerm}_w$.
2. $c_\sigma \in \mathsf{CTerm}_w$ if and only if $\exists \rho \in \mathsf{CTSubst}_w.\ \sigma \leq_\rho \mathsf{tpOf}(c)$.
3. If $t :: \sigma$, then $\mathsf{wf}(\sigma)$.

*Proof.* Immediate from the definitions of wf and ::. □

## C More details and proofs concerning the translation from Isabelle/HOL to HOLC

**Lemma 10.** The relation $\rhd$ is terminating.

*Proof.* The right-hand sides of the defining clauses for $\rhd$ are structurally smaller than the left-hand sides. □

**Lemma 11.** Let $t : \sigma_1 \to \ldots \to \sigma_n$ where $\sigma_n$ is not the function type, then $t \rhd^+ \sigma_i$ for all $i \leq n$.

*Proof.* By structural induction on $t$. First, let us observe that if $t \rhd \tau_1 \to \ldots \to \tau_k$ for some $\tau_i$, then $t \rhd^+ \tau_i$ for all $i \leq k$. Let $\sigma$ denote $\sigma_1 \to \ldots \to \sigma_n$. If $t$ is a constant instance or a variable, then $t = c_\sigma \rhd \sigma$ or $t = x_\sigma \rhd \sigma$. If $t$ is a lambda application, i.e., $t = t_1 t_2$ where $t_1 : \tau \to \sigma$ for some $\tau$, we can derive $t \rhd t_1 \rhd^+ \sigma_i$ from the induction hypothesis for $t_1$. Finally, if $t$ is a lambda abstraction, then $t = \lambda x_{\sigma_1}.\ t_2$, where $t_2 : \sigma_2 \to \ldots \to \sigma_n$. Then $t \rhd \sigma_1$ and $t \rhd t_2$. The fact $t \rhd^+ \sigma_i$ for all $2 \leq i \leq n$ follows from the induction hypothesis for $t_2$. □

**Lemma 12.** Let $w \in \mathsf{Type} \cup \mathsf{Term}$ and $v \in \mathsf{Type}^\bullet \cup \mathsf{CInst}^\bullet$. If $w \rhd^* v$, then $v \in \mathsf{types}^\bullet(w) \cup \mathsf{consts}^\bullet(w)$.

*Proof.* By induction on derivation of $w \rhd^* v$. Base case: $w = v$. If $w \in \mathsf{Type}^\bullet$, then $\mathsf{types}^\bullet(w) = \{w\}$. If $w \in \mathsf{CInst}^\bullet$, then $\mathsf{consts}^\bullet(w) = \{w\}$. Induction step: let us assume that $w_i \rhd^* v$ implies $v \in \mathsf{types}^\bullet(w_i) \cup \mathsf{consts}^\bullet(w_i)$. Let $w_{i+1} \rhd w_i \rhd^* v$. The obligation follows from the monotonicity of $\mathsf{types}^\bullet$ and $\mathsf{consts}^\bullet$ wrt $\rhd$; i.e., $x \rhd y$ implies $\mathsf{types}^\bullet(x) \supseteq \mathsf{types}^\bullet(y)$ and $\mathsf{consts}^\bullet(x) \supseteq \mathsf{consts}^\bullet(y)$. □

**Corollary 13.** Let $u, v \in \mathsf{Type}^\bullet \cup \mathsf{CInst}^\bullet$. If $u \equiv^\downarrow t \rhd^* v$ for some term $t$, then $u \rightsquigarrow^\downarrow v$.

**More detailed proof of Lemma 4.** By contradiction: let us assume that $\blacktriangleright$ does not terminate. That means there exists an infinite sequence $(w_i)_{i\in\mathbb{N}}$ such that $w_i \blacktriangleright w_{i+1}$ for all $i$. Since $\blacktriangleright$ is defined as $\equiv^\downarrow \cup \rhd$ and $\rhd$ is terminating (Lemma 10), there must exist an infinite subsequence $(w_{i_j})_{j\in\mathbb{N}}$ such that $w_{i_j} \equiv^\downarrow w_{i_j+1} \rhd^* w_{i_{j+1}}$ for all $j$. Notice that $w_{i_j} \in \mathsf{Type}^\bullet \cup \mathsf{CInst}^\bullet$, which follows from definition of $\equiv$ for all $j$. Finally we obtain $w_{i_j} \rightsquigarrow^\downarrow w_{i_{j+1}}$ for all $j$ from Corollary 13, which is a contradiction to the termination of $\rightsquigarrow^\downarrow$. □

**Proof of Lemma 5.** We will prove well-formedness and well-typedness by mutual induction on the structure of the type $\sigma$ and the term $t$. Let $t : \tau$.

Well-formedness of ctypes: The types $\mathsf{NF}(\mathsf{bool})$, $\mathsf{NF}(\mathsf{ind})$ and $\mathsf{NF}(\alpha)$ are trivially well-formed.

Assume $\sigma$ has the form $\sigma_1 \to \sigma_2$. By the induction hypothesis, $\mathsf{NF}(\sigma_1)$ and $\mathsf{NF}(\sigma_2)$ are well-formed. Thus $\mathsf{NF}(\sigma) = \mathsf{NF}(\sigma_1) \to \mathsf{NF}(\sigma_2)$ is well-formed by the typing rule ($\mathrm{W}_2$).

Assume $\sigma \notin \mathsf{Type}^\bullet$ and there does not exist any matching definition. Then $\mathsf{NF}(\sigma) = \sigma$. And every HOL type $\sigma$ is a well-formed HOLC type (as can be easily proved by induction on the structure of $\sigma$).

Finally, assume $\sigma \notin \mathsf{Type}^\bullet$ and there exists a matching definition for $\sigma$. Then we obtain $\mathsf{NF}(\sigma)$ is $\{\!|\mathsf{NF}(t)|\!\}$ for some $t : \tau \to \mathsf{bool}$. We obtain $\mathsf{NF}(t) :: \mathsf{NF}(\tau \to \mathsf{bool}) = \mathsf{NF}(\tau) \to \mathsf{bool}$ from the induction hypothesis and therefore $\{\!|\mathsf{NF}(t)|\!\}$ is well-formed by the typing rule ($\mathrm{W}_3$).

Well-typedness of cterms:

Assume $t$ has the form $t_1\ t_2$ with $t_1 : \tau' \to \tau$ and $t_2 : \tau'$ for some $\tau'$. By induction hypothesis $\mathsf{NF}(t_1) :: \mathsf{NF}(\tau' \to \tau) = \mathsf{NF}(\tau') \to \mathsf{NF}(\tau)$ and $\mathsf{NF}(t_2) :: \mathsf{NF}(\tau')$. Thus $\mathsf{NF}(t) = \mathsf{NF}(t_1)\ \mathsf{NF}(t_2) :: \mathsf{NF}(\tau)$ by the typing rule (APP).

Assume $t$ has the form $\lambda x_{\tau_1}.\ t_2$ with $t_2 : \tau_2$ and $\tau = \tau_1 \to \tau_2$. By the induction hypothesis $\mathsf{NF}(\tau_1)$ is well-formed and $\mathsf{NF}(t_2) :: \mathsf{NF}(\tau_2)$. Therefore $\mathsf{NF}(t) = \lambda x_{\mathsf{NF}(\tau_1)}.\ \mathsf{NF}(t_2) :: \mathsf{NF}(\tau_1) \to \mathsf{NF}(\tau_2) = \mathsf{NF}(\tau_1 \to \tau_2)$ by the typing rule (ABS).

Assume $t$ has the form $c_\tau$ with $c \in \mathsf{Const}$ and $\tau \leq \mathsf{tpOf}(c)$ and there exists no matching definition for $c_\tau$. By the induction hypothesis $\mathsf{NF}(\tau)$ is well-formed and thus $c_{\mathsf{NF}(\tau)} :: \mathsf{NF}(\tau)$ by the typing rule (CONST). Similarly for $t = x_\tau$.

Assume $t$ has the form $c_\tau$ and there exists a matching definition for $c_\tau$. Then $c_\tau \equiv^\downarrow t$ for some $t$ such that $t : \tau$. From the induction hypothesis, $\mathsf{NF}(t) :: \mathsf{NF}(\tau)$. □

**Proof of Lemma 7.** We actually prove two facts, by mutual structural induction on $\tau$ and $t$:

1. $\mathsf{NF}(\tau[s/x_\sigma]) = \mathsf{NF}(\tau)[\mathsf{NF}(s)/x_{\mathsf{NF}(\sigma)}]$
2. $\mathsf{NF}(t[s/x_\sigma]) = \mathsf{NF}(t)[\mathsf{NF}(s)/x_{\mathsf{NF}(\sigma)}]$

1: The cases when $\tau$ is built-in or does not have a matching definition follow immediately from the induction hypothesis and the definition of $\mathsf{NF}$.

Assume that there exists a matching definition. Then $\tau \equiv^{\downarrow} t$ with $t : \tau' \to \mathsf{bool}$, and therefore $\tau[s/x_\sigma] \equiv^{\downarrow} t[s/x_\sigma]$. Hence, since also $t[s/x_\sigma] : \tau'[s/x_\sigma] \to \mathsf{bool}$, by the definition of $\mathsf{NF}$ and the induction hypothesis, we have $\mathsf{NF}(\tau[s/x_\sigma]) = \mathsf{NF}(\{\!|t[s/x_\sigma]|\!\}) = \{\!|\mathsf{NF}(t)[\mathsf{NF}(s)/x_{\mathsf{NF}(\sigma)}]|\!\} = \mathsf{NF}(c_\tau)[\mathsf{NF}(s)/x_{\mathsf{NF}(\sigma)}]$, as desired.

2: The cases when $t$ has the form $t_1\ t_2$ or $\lambda x_\sigma.\ t'$ or $x_\tau$ or [$c_\tau$ with $c \notin \mathsf{Const}^\bullet$ or there is no matching definition] follow immediately from the induction hypothesis and the definition of $\mathsf{NF}$.

Assume that $t$ has the form $c_\tau$ and there exists a matching definition. Then $c_\tau \equiv^{\downarrow} t$, and therefore $c_{\tau[s/x_\sigma]} \equiv^{\downarrow} t[s/x_\sigma]$. Hence, by the definition of $\mathsf{NF}$ and the induction hypothesis, $\mathsf{NF}(c_\tau[s/x_\sigma]) = \mathsf{NF}(t[s/x_\sigma]) = \mathsf{NF}(t)[\mathsf{NF}(s)/x_{\mathsf{NF}(\sigma)}] = \mathsf{NF}(c_\tau)[\mathsf{NF}(s)/x_{\mathsf{NF}(\sigma)}]$, as desired. □

**Proof of Lemma 8.** We actually prove two facts, by mutual structural induction on $\tau$ and $t$:

1. If $\mathsf{types}^\bullet(\tau) = \emptyset$, then $\mathsf{NF}(\overline{\rho}(\tau)) = \overline{\mathsf{NF} \circ \rho}(\tau)$.
2. If $\mathsf{types}^\bullet(t) \cup \mathsf{consts}^\bullet(t) = \emptyset$, then $\mathsf{NF}(\overline{\rho}(t)) = \overline{\mathsf{NF} \circ \rho}(t)$.

1: Since $\mathsf{types}^\bullet(\tau) = \emptyset$, we know that $\tau \notin \mathsf{Type}^\bullet$.

Assume $\tau$ is a type variable $\alpha$, then $\mathsf{NF}(\rho(\alpha)) = \overline{\mathsf{NF} \circ \rho}(\alpha)$ from the definition of $\overline{\mathsf{NF} \circ \rho}$. Let $\tau = \mathsf{bool}$, then $\mathsf{NF}(\rho(\mathsf{bool})) = \mathsf{NF}(\mathsf{bool}) = \mathsf{bool} = \overline{\mathsf{NF} \circ \rho}(\mathsf{bool})$. The same for $\mathsf{ind}$.

Assume $\tau$ has the form $\tau_1 \to \tau_2$, then $\mathsf{NF}(\rho(\tau_1 \to \tau_2)) = \mathsf{NF}(\rho(\tau_1)) \to \mathsf{NF}(\rho(\tau_2))$. By using the induction hypothesis we continue as $\ldots = \overline{\mathsf{NF} \circ \rho}(\tau_1) \to \overline{\mathsf{NF} \circ \rho}(\tau_2) = \overline{\mathsf{NF} \circ \rho}(\tau_1 \to \tau_2)$.

2: Assume $t$ has the form $t_1\ t_2$. This case is analgous to the case of $\tau$ having the form $\tau_1 \to \tau_2$.

Assume $t$ has the form $\lambda x_\tau.\ t_2$. Then $\mathsf{NF}(\rho(\lambda x_\tau.\ t_2)) = \lambda x_{\mathsf{NF}(\rho(\tau))}.\ \mathsf{NF}(\rho(t_2))$. By the induction hypothesis, point 1 and the fact that $\mathsf{types}^\bullet(\tau) = \emptyset$, we obtain $\ldots = \lambda x_{\overline{\mathsf{NF} \circ \rho}(\tau)}.\ \overline{\mathsf{NF} \circ \rho}(t_2) = \overline{\mathsf{NF} \circ \rho}(\lambda x_\tau.\ t_2)$.

Assume $t$ has the form $c_\sigma$. Since $\mathsf{types}^\bullet(t) \cup \mathsf{consts}^\bullet(t) = \emptyset$, it must hold that $c_\sigma \notin \mathsf{Const}^\bullet$. Thus $\mathsf{NF}(\rho(c_\tau)) = c_{\mathsf{NF}(\rho(\tau))} = c_{\overline{\mathsf{NF} \circ \rho}(\tau)} = \overline{\mathsf{NF} \circ \rho}(c_\tau)$ by using poing 1 and the fact that $\mathsf{types}^\bullet(\tau) = \emptyset$.

Assume $t$ has the form $x_\sigma$. This case is analogous to the previous case. □

**Recallection of the rules for $\vdash'$, the modified system for HOL**

$$\frac{}{D; \Gamma \vdash' \overline{\rho}(\varphi)}\ [\varphi \in \mathsf{Ax} \cup D,\ \forall \alpha \in \mathsf{supp}(\rho).\ \alpha \notin \Gamma]\ (\textsc{Fact-T-Inst})$$

$$\frac{}{D; \Gamma \vdash' (\lambda x_\sigma.\ t)\ s = t[s/x_\sigma]}\ (\textsc{Beta}) \qquad \frac{D; \Gamma \vdash' \varphi \longrightarrow \chi \qquad D; \Gamma \vdash' \varphi}{D; \Gamma \vdash' \chi}\ (\textsc{MP})$$

$$\frac{D;\Gamma\cup\{\varphi\}\vdash'\chi}{D;\Gamma\vdash'\varphi\longrightarrow\chi}\ (\textsc{Impi}) \qquad \frac{D;\Gamma\vdash' f\ x_\sigma = g\ x_\sigma}{D;\Gamma\vdash' f=g}\ [x_\sigma\notin\Gamma]\ (\textsc{Ext})$$

**Lemma 14.** The HOL deductionsystems $\vdash$ and $\vdash'$ are equivalent.

*Proof.* As mentioned in the main paper, it is immediate to see that $\vdash$ is at least as powerful as $\vdash'$, since (FACT-T-INST) can de simulated by (FACT) and (T-INST). Conversely, (FACT-T-INST) is stronger than (FACT). It remains to show that (T-INST) is admissible in $\vdash'$. We prove something stronger, namely,

$$D;\Gamma\vdash'\varphi \text{ implies } D;\Gamma[\sigma/\alpha]\vdash'\varphi[\sigma/\alpha]$$

(Then admissibility of (T-INST) follows, since under the side-condition of (T-INST), $\alpha\notin\Gamma$, we have that $\Gamma[\sigma/\alpha]=\Gamma$.)

The proof goes induction on the definition of $\vdash'$. All the cases except for (BETA) and (FACT-T-INST) are straightforward.

(BETA): We need to prove $D;\Gamma[\sigma/\alpha]\vdash'((\lambda x_\tau.\ t)\ s = t[s/x_\tau])[\sigma/\alpha]$, which (by standard properties of substitution) is the same as $D;\Gamma[\sigma/\alpha]\vdash'(\lambda x_{\tau[\sigma/\alpha]}.\ t[\sigma/\alpha])\ (s[\sigma/\alpha]) = (t[\sigma/\alpha])[s[\sigma/\alpha]/x_{\tau[\sigma/\alpha]}]$. But this follows from (BETA).

(FACT-T-INST): We assume that no variable in $\mathsf{supp}(\rho)$ is free in $\Gamma$. (*)

We need to prove that $D;\Gamma[\sigma/\alpha]\vdash'(\overline{\rho}(\varphi))[\sigma/\alpha]$, which (by standard properties of substitution) is the same as $D;\Gamma[\sigma/\alpha]\vdash'\overline{\rho[\sigma/\alpha]}(\varphi[\sigma/\alpha])$ (**), where $\rho[\sigma/\alpha]$ is the composition of the substitutions $\rho$ and $[\sigma/\alpha]$, sending each $\beta$ to $\rho(\beta)[\sigma/\alpha]$. From (*), it follows that no variable in $\mathsf{supp}(\rho[\sigma/\alpha])$ appears free in $\Gamma[\sigma/\alpha]$, and therefore (**) follows from (FACT-T-INST) □

**Proof of Theorem 6.** Thanks to Lemma 14, we can use $\vdash'$ instead of $\vdash$.

By induction on the derivation of $D;\Gamma\vdash'\varphi$, for every HOL inference rule

$$\frac{\Gamma_1\vdash'\varphi_1 \qquad \dots \qquad \Gamma_n\vdash'\varphi_n}{\Gamma\vdash'\varphi}$$

we construct a HOLC proof of $\mathsf{NF}(\Gamma)\Vdash\mathsf{NF}(\varphi)$, assuming $\mathsf{NF}(\Gamma_1)\Vdash\mathsf{NF}(\varphi_1)\dots\mathsf{NF}(\Gamma_n)\Vdash\mathsf{NF}(\varphi_n)$.

(FACT-T-INST): Assume $D;\Gamma\vdash'\rho(\varphi)$ with $\varphi\in\mathsf{Ax}\cup D$ and $\forall\alpha\in\mathsf{supp}(\rho).\ \alpha\notin\Gamma$. We distinguish two cases:

1. $\varphi\in\mathsf{Ax}$. We note that $\mathsf{types}^\bullet(\varphi)\cup\mathsf{consts}^\bullet(\varphi)=\emptyset$. Thus by using Lemma 8, we obtain $\mathsf{NF}(\rho(\varphi))=\overline{\mathsf{NF}\circ\rho}(\varphi)$. Moreover, we have $\mathsf{NF}(\Gamma)\Vdash\varphi$ by (FACT). From this, we obtain $\mathsf{NF}(\Gamma)\Vdash\overline{\mathsf{NF}\circ\rho}(\varphi)$ by applying (T-INST)a number of times equal to the domain of $\overline{\mathsf{NF}\circ\rho}$ (which is finite). But the last is the same as $\mathsf{NF}(\Gamma)\Vdash\mathsf{NF}(\rho(\varphi))$, as desired.
2. $\varphi\in D$. That means that $\varphi$ is a definitional axiom for either a constant or a type:

- Assume $\varphi$ has the form $c_\tau \equiv t$. Recall that $c_\tau \equiv t$ denotes $c_\tau = t$, and thus $\mathsf{NF}(\varphi) = (\mathsf{NF}(t) = \mathsf{NF}(t))$ by the definition of $\mathsf{NF}$. Therefore we can derive $\mathsf{NF}(\varphi)$ as an instance of the reflexivity axiom $\mathsf{refl}$.
- Assume $\varphi$ has the form $\tau \equiv t$ where $t : \sigma \to \mathsf{bool}$. Recall that $\tau \equiv t$ denotes HOL's type definition axiom (1) on page 5. Thus, $\mathsf{NF}(\varphi)$ is an instance of the HOLC's axiom $\mathsf{type\_comp}$, since $\mathsf{NF}(\tau) = \{\!|\mathsf{NF}(t)|\!\}$ (Equation (25)). More precisely, $\mathsf{NF}(\varphi) = (\mathsf{type\_comp}[\sigma/\mathsf{NF}(\sigma)])[t/\mathsf{NF}(t)]$.

(ASSUM): Assume $D; \Gamma \vdash' \varphi$ with $\varphi \in \Gamma$. Then $\{\mathsf{NF}(\varphi)\} \Vdash \mathsf{NF}(\varphi)$ also by (ASSUM).

(BETA): Assume $D; \Gamma \vdash' (\lambda x_\sigma.\, t)\ s = t[s/x_\sigma]$. Using (BETA)in HOLCF, we have $\mathsf{NF}(\Gamma) \Vdash (\lambda x_{\mathsf{NF}(\sigma)}.\, \mathsf{NF}(t))\ \mathsf{NF}(s) = \mathsf{NF}(t)[s/x_{\mathsf{NF}(\sigma)}]$. But using Lemma 7 and the definition of $\mathsf{NF}$ for $\lambda$-abstraction, application and equality, this is the same as $\mathsf{NF}(\Gamma) \Vdash \mathsf{NF}((\lambda x_\sigma.\, t)\ s = t[s/x_\sigma])$, as desired.

(MP), (IMPI)and (EXT): Immediate from the induction hypothesis and an application of the corresponding rule in HOLC. □