EPiC Series in Computing
Volume 73, 2020, Pages 498–515
LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

A Mechanised Semantics for HOL with Ad-hoc Overloading

Johannes Åman Pohjola[1,2] and Arve Gengelbach[3]

[1] CSIRO's Data61, Sydney, Australia
johannes.amanpohjola@data61.csiro.au
[2] University of New South Wales, Sydney, Australia
[3] Uppsala University, Uppsala, Sweden
arve.gengelbach@it.uu.se

Abstract

Isabelle/HOL augments classical higher-order logic with ad-hoc overloading of constant definitions— that is, one constant may have several definitions for non-overlapping types. In this paper, we present a mechanised proof that HOL with ad-hoc overloading is consistent. All our results have been formalised in the HOL4 theorem prover.

1 Introduction

The consistency of higher-order logic (HOL), and the correctness of its implementations, is the bedrock supporting the truth claims of landmark success stories in formal verification such as seL4 [16], Flyspeck [10] and CakeML [19]. And the bedrock is firm: there is something of a tradition within the HOL community, from Harrison [11] to Kumar et al. [17, 18], of eating our own dog food by using HOL provers to verify the meta-theory of HOL itself. Such self-verification efforts must always come with an asterisk, lest we run afoul of self-reference paradoxes [8]; nonetheless, the trust story is significantly strengthened.

At the time of writing, Isabelle/HOL [25] is by far the most popular HOL implementation: at the most recent ITP conference [13], traditionally the main venue for HOL theorem proving, Isabelle/HOL papers outnumbered papers about other HOL provers 13 to 3.[1] It is a shame, then, that the above self-verification efforts do not quite apply to Isabelle/HOL, because Isabelle has more expressive primitives for introducing definitions. In particular, Isabelle, unlike the other HOLs, admits *ad-hoc overloading* of polymorphic constants. This significantly complicates the trust story: the meaning of previously introduced constants may be changed by future overloads, so extra care must be taken to avoid overloads that introduce circularities or contradictions. In this paper, we strengthen the bedrock for Isabelle/HOL by presenting the first machine-checked proof of consistency for higher-order logic with ad-hoc overloading.

This continues and unifies two previous lines of work. Ad-hoc overloading in the context of Isabelle/HOL has been studied by Wenzel [31], by Obua [26], and most recently by Kunčar

[1] One paper described the emulation of HOL4 in Isabelle/HOL [15]. We count this in both columns.

and Popescu [22]. Our consistency proof is based on the pen-and-paper consistency proof by Kunčar and Popescu. Our formalisation, meanwhile, uses the HOL4 formalisation of HOL with definitions (nicknamed Candle) by Kumar et al. [17] as its starting point.

But our work is not a straightforward mechanisation of the former, nor is it a straightforward extension of the latter. From Candle we inherit most of the syntax and some generic infrastructure, but because the model construction of Kunčar and Popescu differs so much from the standard Pitts-style construction that Candle follows, we start essentially from scratch on the semantics. We find two (fixable) mistakes in Kunčar and Popescu's proof regarding the treatment of variable names and substitution, forcing us to innovate by e.g. formulating a novel style of term semantics using lazy evaluation of ground type substitutions. Furthermore, we consider signature extensions in addition to definitional extensions, and use less minimalist definitional mechanisms: constant definitions in the style of Arthan [4], and type definitions requiring proof that the new type is inhabited up-front. These introduce new complications by making the model construction depend on soundness.

The impact of this work is as follows. First, we strengthen the trust story for the Isabelle/HOL logic by formally validating (and, on rare occasions, fixing) the novel and original model construction by Kunčar and Popescu. Second, since our logic is a conservative extension of Candle, we show that the Candle consistency proof is robust to a different style of model construction. Finally, our contributions lay the groundwork for extracting a future verified CakeML implementation of an Isabelle/HOL kernel.

We delimit our scope as follows. Unlike Kumar et al., we do not currently leverage our proofs by extracting a verified CakeML implementation of our kernel. Doing so requires verifying Kunčar's algorithm for checking orthogonality and termination of dependency relations [20]; this will be the topic of future work. Unlike Kunčar and Popescu, we do not consider ad-hoc overloading of type constructors—while an intriguing concept, this is a feature that Isabelle/HOL does not have, nor are we aware of a use case that would justify its inclusion. Finally, we alluded earlier to an asterisk to avoid self-reference paradoxes. The asterisk is that our consistency proof for HOL with the axiom of infinity is predicated upon assumptions, stating the existence of a set theory, that cannot be witnessed in HOL4 without additional axioms.

All definitions and theorems in this paper are formalised in the HOL4 theorem prover, and the proof scripts are available online.[2]

2 Syntax

In this section, we present the syntax of higher-order logic that we base our work on. Much of it is inherited from Kumar et al. [17], who in turn inherited much from Harrison [11]; we will explicitly point out the places where we differ. But first, a few words on notation.

2.1 Notation

For the most part, mathematical formulas are generated from the HOL4 formalisation. Constant symbols are written in this font, variables in *this font*, and string literals are «bracketed like this». Colon means "type of", as in $\mathsf{map} : (\alpha \Rightarrow \beta) \Rightarrow \alpha\ \mathsf{list} \Rightarrow \beta\ \mathsf{list}$. Function update $f(\!|x \mapsto y|\!)$ denotes the function that is like f except it maps x to y. ++ denotes list append. We will silently convert lists to sets when convenient. The sum type $\alpha + \beta$ is the disjoint union of α and β, whose elements are tagged INL and INR, respectively. To avoid confusion between HOL4

[2] https://code.cakeml.org/tree/master/candle/overloading

notation and the deep embedding of HOL syntax within HOL4, the latter will be presented in AST form. Theorems proved in HOL4 are marked with an initial $\vdash$ before the theorem statement; when $\vdash$ is used as an infix symbol, it instead denotes provability in the (deeply embedded) HOL inference system.

2.2 Types and terms

The types and terms are those of the λ-calculus with rank 1 polymorphism:

$$\begin{aligned}\mathsf{type} &= \mathsf{Tyvar\ string} \mid \mathsf{Tyapp\ string\ (type\ list)}\\ \mathsf{term} &= \mathsf{Var\ string\ type} \mid \mathsf{Const\ string\ type} \mid \mathsf{Comb\ term\ term} \mid \mathsf{Abs\ term\ term}\end{aligned}$$

In the λ-abstraction $\mathsf{Abs}\ t_1\ t_2$, the binder t_1 is a term for uniformity. In practice we only consider well-formed terms, where $t_1 = \mathsf{Var}\ x\ ty$ for some x, ty. Note that names are just strings. We do not use any binding framework, such as nominal logic [6] or higher-order abstract syntax [27]. This is to facilitate future code extraction to CakeML. The treatment of α-conversion will not be relevant in this paper; the interested reader may consult Kumar et al. [18].

The built-in types and constants (booleans, functions and equality) are definable by the following abbreviations:

Bool	for	Tyapp «bool» []
$\mathsf{Fun}\ x\ y$	for	Tyapp «fun» $[x;\ y]$
$\mathsf{Equal}\ ty$	for	Const «=» $(\mathsf{Fun}\ ty\ (\mathsf{Fun}\ ty\ \mathsf{Bool}))$
$s === t$	for	$\mathsf{Comb}\ (\mathsf{Comb}\ (\mathsf{Equal}\ (\mathsf{typeof}\ s))\ s)\ t$

A term is welltyped if, by the following rules, it has a type:

$$\frac{}{(\mathsf{Var}\ n\ ty)\ \mathsf{has\_type}\ ty} \qquad \frac{}{(\mathsf{Const}\ n\ ty)\ \mathsf{has\_type}\ ty} \qquad \frac{s\ \mathsf{has\_type}\ (\mathsf{Fun}\ dty\ rty) \quad t\ \mathsf{has\_type}\ dty}{(\mathsf{Comb}\ s\ t)\ \mathsf{has\_type}\ rty}$$

$$\frac{t\ \mathsf{has\_type}\ rty}{(\mathsf{Abs}\ (\mathsf{Var}\ n\ dty)\ t)\ \mathsf{has\_type}\ (\mathsf{Fun}\ dty\ rty)}$$

If a term tm is welltyped, it has a unique type denoted typeof tm. A *type substitution*, ranged over by Θ, is a mapping from type variables to types. We write $\Theta\ ty$ and $\Theta\ tm$ for the result of applying Θ to all type variables of a type ty or term tm, respectively; the latter case may involve α-conversion to avoid variable capture. In the formalisation, type substitutions are sometimes encoded as functions and sometimes as lists of pairs. The presentation will abstract away from this distinction. A type is *ground* if it has no type variables. A substitution Θ is ground if, for all ty, $\Theta\ ty$ is ground. We say that ty_1 is an *instance* of ty_2 (written $ty_2 \geq ty_1$) if there exists a Θ such that $ty_1 = \Theta\ ty_2$.

We write $x^\bullet$ for the list of outermost non-built-in types that occur in a type or term x. For example $(\mathsf{map} : (\alpha \to \mathsf{Bool}) \to \alpha\,\mathsf{list} \to \mathsf{Bool\,list})^\bullet$ returns the list containing α and $\alpha\,\mathsf{list}$. Following Kunčar and Popescu, we will often be interested in these because the model construction

gives built-in types special treatment:

$$\begin{aligned}
&\mathsf{Bool}^\bullet \stackrel{\mathsf{def}}{=} [] \\
&(\mathsf{Fun}\ dom\ rng)^\bullet \stackrel{\mathsf{def}}{=} dom^\bullet \mathbin{++} rng^\bullet \\
&ty^\bullet \stackrel{\mathsf{def}}{=} [ty] \quad \text{otherwise} \\
&(\mathsf{Var}\ v_0\ ty)^\bullet \stackrel{\mathsf{def}}{=} ty^\bullet \\
&(\mathsf{Const}\ v_1\ ty)^\bullet \stackrel{\mathsf{def}}{=} ty^\bullet \\
&(\mathsf{Comb}\ a\ b)^\bullet \stackrel{\mathsf{def}}{=} a^\bullet \mathbin{++} b^\bullet \\
&(\mathsf{Abs}\ a\ b)^\bullet \stackrel{\mathsf{def}}{=} a^\bullet \mathbin{++} b^\bullet
\end{aligned}$$

Note that Bool and $\lambda\, s\ t.\ \mathsf{Fun}\ s\ t$ are abbreviations.

2.3 Inference system

A *signature* is a mapping from constant names to their most general type, and from type constructor names to their arity. A *theory* is a pair (s,a) of a signature s and a set of axioms $a : \mathsf{term} \Rightarrow \mathsf{bool}$. These components are accessed through the self-explanatory selectors axsof, tmsof, tysof and sigof.

We write the sequent $(thy,h) \vdash p$ to denote that $p : \mathsf{term}$ can be inferred in the theory thy from the hypotheses $h : \mathsf{term\ list}$. We define $\vdash$ as an inductive relation comprised of the standard inference rules of higher-order logic, plus whatever axioms are present in the theory. We show a few rules to give the flavour:

$$\frac{\mathsf{theory\_ok}\ thy \quad p\ \mathsf{has\_type\ Bool} \quad \mathsf{term\_ok}\ (\mathsf{sigof}\ thy)\ \ p}{(thy,[p]) \vdash p}\ \mathsf{ASSUME}$$

$$\frac{\mathsf{theory\_ok}\ thy \quad c \in \mathsf{axsof}\ thy}{(thy,[]) \vdash c}\ \mathsf{AXIOM}$$

$$\frac{(thy,h_1) \vdash l_1 === r_1 \quad (thy,h_2) \vdash l_2 === r_2 \quad \mathsf{welltyped}\ (\mathsf{Comb}\ l_1\ l_2)}{(thy,\ h_1\ \cup\ h_2) \vdash \mathsf{Comb}\ l_1\ l_2 === \mathsf{Comb}\ r_1\ r_2}\ \mathsf{MK\_COMB}$$

Here term_ok and theory_ok are syntactic well-formedness conditions. For terms, all types and constants occurring in them must be part of the signature. For theories, all axioms must be term_ok and have type Bool, and the signature must contain the built-in types: functions, booleans and equality.

2.4 Theory extension

A theory is constructed incrementally by applying a sequence of *updates*:

```
update =
    ConstSpec bool ((string × term) list) term
  | TypeDefn string term string string
  | NewType string num
  | NewConst string type
  | NewAxiom term
```

NewType and NewConst introduce new type constructors and constant names into the theory signature without introducing any axioms about them. TypeDefn *name pred abs rep* defines a new type *name* by carving out from an existing type the subset that satisfies the predicate *pred*. It also introduces abstraction and representation functions for the new type, along with two axioms: $\mathsf{abs}(\mathsf{rep}\, a) = a$ and $\mathsf{pred}\, r = (\mathsf{rep}(\mathsf{abs}\, r) = r)$.

Constant specification is the first place where we depart significantly from Kumar et al. by allowing ad-hoc overloading; we will defer its discussion until we have introduced a few preliminaries.

A *context* is a list of updates. The relation updates defines what constitutes a valid update of a context. We show the rules for introducing unchecked axioms (NewAxiom) and type definitions:

$$\frac{\mathit{prop}\ \mathsf{has\_type\ Bool} \quad \mathsf{term\_ok}\ (\mathsf{sigof}\ \mathit{ctxt})\ \mathit{prop}}{\mathsf{NewAxiom}\ \mathit{prop}\ \mathsf{updates}\ \mathit{ctxt}}$$

$$\frac{\begin{array}{c}(\mathsf{thyof}\ \mathit{ctxt}, []) \vdash \mathsf{Comb}\ \mathit{pred}\ \mathit{witness} \qquad \mathsf{closed}\ \mathit{pred} \qquad \mathit{name} \notin \mathsf{domain}\ (\mathsf{tysof}\ \mathit{ctxt}) \\ \mathit{abs} \notin \mathsf{domain}\ (\mathsf{tmsof}\ \mathit{ctxt}) \qquad \mathit{rep} \notin \mathsf{domain}\ (\mathsf{tmsof}\ \mathit{ctxt}) \qquad \mathit{abs} \neq \mathit{rep}\end{array}}{\mathsf{TypeDefn}\ \mathit{name}\ \mathit{pred}\ \mathit{abs}\ \mathit{rep}\ \mathsf{updates}\ \mathit{ctxt}}$$

In words, an axiom can be introduced if it is an ok boolean term. A type definition can be introduced if it can be proved that the set carved out by *pred* is inhabited by a *witness*, if *pred* has no free term variables (closed), and the names of the type and its abstraction and representation function are fresh.

We say that ctxt_1 extends ctxt_2 if ctxt_1 can be obtained from ctxt_2 by applying a (possibly empty) sequence of valid updates. We say that ctxt_1 is a definitional_extension of ctxt_2 if, additionally, none of the updates to ctxt_2 are NewAxioms. We will be particularly interested in definitional extensions of three pre-defined initial contexts:

- init_ctxt introduces the built-ins—booleans, equality and functions—using NewType and NewConst. This is the bare minimum setup required for the inference system.
- finite_hol_ctxt extends init_ctxt with the theory of booleans, the choice constant «@», the axiom of extensionality, and the axiom of choice (the latter two with NewAxiom).
- hol_ctxt extends finite_hol_ctxt with the type of individuals and the axiom of infinity.

2.4.1 Definitional dependencies

Obua [26] noticed that ad-hoc overloading is safe if definitions do not overlap, and if unfolding of definitions terminates; we follow Kunčar and Popescu in formalising the former as *orthogonality*, and the latter as cycle-freedom of a dependency relation.

Orthogonality A context is *orthogonal* if definitions do not overlap, that is, if all constants and types have at most one definition at every type instance. Two types ty_1, ty_2 are orthogonal, written $ty_1 \# ty_2$, if they do not have any common type instance.

$$ty_1 \# ty_2 \stackrel{\mathsf{def}}{=} \neg\exists\, ty.\ ty_1 \geq ty \wedge ty_2 \geq ty$$

Orthogonality extends to constants by looking at the types if the constant names are alike:

$$\mathsf{Const}\ c\ ty_1 \# \mathsf{Const}\ d\ ty_2 \stackrel{\mathsf{def}}{=} c \neq d \vee ty_1 \# ty_2$$

A context is orthogonal if all defined types and constants are pairwise orthogonal.

Cycle freedom Every context induces a dependency relation $\rightsquigarrow$ on type + term. The intent is that $s \rightsquigarrow_{ctxt} t$ means that the definition of s depends directly on the definition of t in the definitional theory *ctxt*.

The *substitutive closure* subst_clos $\mathcal{R}$ of a relation $\mathcal{R}$ relates $\Theta\ t_1$ and $\Theta\ t_2$ whenever $t_1\ \mathcal{R}\ t_2$ for all Θ, t_1, t_2. We say that a relation $\mathcal{R}$ is terminating if there is no sequence x such that $x_i\ \mathcal{R}\ x_{i+1}$ for all $i \in \mathbb{N}$. Note that if a relation is terminating, its inverse is well-founded. Cycle freedom holds whenever the substitutive closure of the dependency relation is terminating.

In the definition of the dependency relation, we consider constants occuring within terms. The function $\cdot^\circ$ gives the argument term's non-built-in constants.

$$
\begin{aligned}
(\mathsf{Var}\ x\ ty)^\circ &\stackrel{\mathsf{def}}{=} [] \\
(\mathsf{Comb}\ a\ b)^\circ &\stackrel{\mathsf{def}}{=} a^\circ +\!\!+ b^\circ \\
(\mathsf{Abs}\ \_\ a)^\circ &\stackrel{\mathsf{def}}{=} a^\circ \\
(\mathsf{Equal}\ ty)^\circ &\stackrel{\mathsf{def}}{=} [] \\
(\mathsf{Const}\ c\ ty)^\circ &\stackrel{\mathsf{def}}{=} [\mathsf{Const}\ c\ ty] \quad \text{otherwise}
\end{aligned}
$$

The dependency relation of Kunčar and Popescu is inductively defined via two rules, where $u \equiv t$ means a definition with definiendum u defined by a term t and either u is a constant (introduced through ConstSpec) or a type (introduced by TypeDefn). In the following cases $u \rightsquigarrow_{ctxt} v$ holds:

1. There is a constant or type definition $u \equiv t$ in *ctxt* such that $v \in t^\bullet$, or $v \in t^\circ$; otherwise,
2. u is a constant-instance of type σ and $v \in \sigma^\bullet$

In our mechanisation, the definition of the dependency relation has seven cases. The first five encode 1 and 2. Case 2 is very close to its mechanisation. Whenever a constant is declared, it introduces dependencies on its type's non-built-in subtypes:

$$
\frac{t_1 \in t_2^\bullet \quad \mathsf{NewConst}\ name\ t_2 \in ctxt}{\mathsf{INR}\ (\mathsf{Const}\ name\ t_2) \rightsquigarrow_{ctxt} \mathsf{INL}\ t_1}
$$

The other two rules are absent in Kunčar and Popescu's dependency relation. The first states that a type declaration NewType *name arity* (without a definition) introduces dependencies on all of the—possibly distinct—arguments to the constructor.

$$
\frac{\begin{array}{c}\mathsf{NewType}\ name\ arity \in ctxt \\ tyname \in tynames \qquad tynames = \mathsf{genlist}\ (\lambda x.\ \mathsf{implode}\ (\mathsf{replicate}\ (\mathsf{Suc}\ x)\ \#\text{“a”}))\ arity\end{array}}{\mathsf{INL}\ (\mathsf{Tyapp}\ name\ (\mathsf{map}\ \mathsf{Tyvar}\ tynames)) \rightsquigarrow_{ctxt} \mathsf{INL}\ (\mathsf{Tyvar}\ tyname)}
$$

That is, a constructor Tyapp *name* (map Tyvar *tynames*) depends on each of its distinctly named type variables. As we shall see in Section 5, it is crucial that this clause applie to the built-in types: since built-ins such as Fun are introduced into the initial context with NewType, this rule gives us the ability to track dependencies that occur as type parameters of the built-ins.

Second, a type definition TypeDefn *name pred abs rep* introduces the constants *abs* and *rep*; the final rule states that these constants depend on their types.

2.4.2 Constant specification

Following Candle, we admit constant specifications in the style of Arthan [4], to whom we refer for motivation, and extend this mechanism to admit ad-hoc overloading. In a constant specification

ConstSpec *ov eqs prop*, *ov* is a flag indicating whether the introduced specifications are overloads. Each element (c_i, t_i) of the list *eqs* denotes the defining equation Const c_i (typeof t_i) === t_i of the constant c_i. These equations are *not* added as axioms to the context; instead, ConstSpec allows more abstraction by adding the user's choice of any *prop* as an axiom, provided the user can prove that *prop* follows from the defining equations.

Formally, the rule for introducing new specifications is:

$$\frac{\begin{array}{c}(\mathsf{thyof}\ \mathit{ctxt},\ \mathsf{map}\ (\lambda\,(s,t).\ \mathsf{Var}\ s\ (\mathsf{typeof}\ t) === t)\ \mathit{eqs}) \vdash \mathit{prop} \\ \forall\,(c,t) \in \mathit{eqs}.\ \mathsf{closed}\ t \wedge \forall\, v.\ v \in \mathsf{tvars}\ t \Rightarrow v \in \mathsf{tyvars}\ (\mathsf{typeof}\ t) \\ \forall\, x\ \mathit{ty}.\ \mathsf{Var}\ x\ \mathit{ty} \in \mathsf{fv}\ \mathit{prop} \Rightarrow (x,\mathit{ty}) \in \mathsf{map}\ (\lambda\,(s,t).\ (s,\mathsf{typeof}\ t))\ \mathit{eqs} \\ \mathsf{constspec\_ok}\ \mathit{ov}\ \mathit{eqs}\ \mathit{prop}\ \mathit{ctxt}\end{array}}{\mathsf{ConstSpec}\ \mathit{ov}\ \mathit{eqs}\ \mathit{prop}\ \mathsf{updates}\ \mathit{ctxt}}$$

Three conditions must hold regardless of whether we are introducing overloads or not. First, the *prop* must follow from the equations, as described above. Second, for each defining equation the RHS must have no free term variables, and may not mention type variables except those on the LHS. Third, the *prop* must have no free term variables except those on the LHS of *eqs*.[3] The requisites that differ depending on whether we are introducing overloads or not are gathered in constspec_ok:

$$\begin{array}{l}\mathsf{constspec\_ok}\ \mathit{ov}\ \mathit{eqs}\ \mathit{prop}\ \mathit{ctxt} \stackrel{\mathsf{def}}{=} \\ \quad \mathsf{if}\ \mathit{ov}\ \mathsf{then} \\ \quad\quad \mathsf{terminating}\ (\mathsf{subst\_clos}\ (\rightsquigarrow_{\mathsf{ConstSpec}\ \mathit{ov}\ \mathit{eqs}\ \mathit{prop}::\mathit{ctxt}})) \wedge \\ \quad\quad \mathsf{orth\_ctxt}\ (\mathsf{ConstSpec}\ \mathit{ov}\ \mathit{eqs}\ \mathit{prop}::\mathit{ctxt}) \wedge \\ \quad\quad \forall\, \mathit{name}\ \mathit{trm}. \\ \quad\quad\quad (\mathit{name},\mathit{trm}) \in \mathit{eqs} \Rightarrow \\ \quad\quad\quad\quad \exists\, \mathit{ty}'. \\ \quad\quad\quad\quad\quad \mathsf{NewConst}\ \mathit{name}\ \mathit{ty}' \in \mathit{ctxt} \wedge \mathit{ty}' \geq \mathsf{typeof}\ \mathit{trm} \wedge \\ \quad\quad\quad\quad\quad \mathsf{alookup}\ (\mathsf{const\_list}\ \mathit{ctxt})\ \mathit{name} = \mathsf{Some}\ \mathit{ty}' \wedge \neg \mathsf{is\_reserved\_name}\ \mathit{name} \\ \quad \mathsf{else}\ \mathsf{all\_distinct}\ (\mathsf{map}\ \mathsf{fst}\ \mathit{eqs}) \wedge \forall\, s.\ s \in \mathsf{map}\ \mathsf{fst}\ \mathit{eqs} \Rightarrow s \notin \mathsf{domain}\ (\mathsf{tmsof}\ \mathit{ctxt})\end{array}$$

If we are not introducing overloads (the else branch), all specified constants must be fresh in the context and have pairwise distinct names. Otherwise, if we are introducing overloads: the substitutive closure of its dependency relation must be terminating, the resulting context must be orthogonal, and for each new defining equation, the constant name must not be reserved, and a more general type instance of all constants must already be present in the signature via a NewConst. The reserved names are for equality «=» and Hilbert choice «@». Since these are axiomatised, we cannot allow overloaded definitions of them because such definitions might contradict the axioms. Note that we only need the terminating check when we introduce overloads, because we prove that no other theory extensions can create cycles.

This separate treatment of overloaded and non-overloaded constant specification could perhaps be unified, but the separation is motivated by practical concerns. First, we want to give users the freedom to specify through $ov = \mathsf{F}$ that certain polymorphic constants cannot be overloaded. For example, the axiom of infinity is expressed in terms of two non-builtin polymorphic constants, namely existential and universal quantification. Overloads on these would change the meaning of the axiom of infinity at certain type instances, which means all bets are off in terms of consistency. Second, it simplifies definitions and proofs if we know whether an update extends the signature or not independently of which context it updates.

[3] After the ConstSpec is complete, these variables will be promoted to constants.

3 Semantics

In this section, we introduce the semantic domain in which we model HOL, and define what it means for a sequent to be true in such a model.

3.1 Semantic domain

Following Pitts [28] and subsequent work, we use a universe of sets satisfying Zermelo's axioms as our semantic domain. By Gödel's second incompleteness theorem we cannot construct such a semantic domain within HOL. Harrison obtains such a construction by augmenting HOL with axioms that grant access to large cardinals [11]. We follow an alternative approach due to Arthan [3] (to whom we refer for more details), where rather than making an explicit construction of a semantic domain, we define what properties it must have. That is, definitions and theorems that make reference to set theory are parameterised on a type variable $\mathcal{U}$ denoting the universe of sets, and a term variable $mem : \mathcal{U} \Rightarrow \mathcal{U} \Rightarrow \mathsf{bool}$ denoting its membership relation. The predicate $\mathsf{is\_set\_theory}\ mem$ asserts that mem obeys the Zermelo axioms except choice and infinity, and the predicate $\mathsf{is\_infinite}\ mem\ indset$ asserts that $indset$ is an element of $\mathcal{U}$ with infinitely many members. We do not need to assume anything about the structure of $indset$. It is convenient to separate out the axiom of infinity because without it, $\mathsf{is\_set\_theory}$ can be witnessed in HOL4. We do not need the axiom of choice in our set theory because the meta-language (HOL4) has it already.

The advantage of this approach is that definitions and proofs are independent of any particular universe construction, that it is clear which theorems require such a construction, and that there is no need to pollute the global HOL environment with axiomatic extensions. Note that this approach has implications for the trusted computing base of our mechanisation. We trust that HOL4 faithfully implements classical higher-order logic with choice, infinity and extensionality as described by Pitts [28]. Additionally, we trust that the Zermelo axioms as expressed by $\mathsf{is\_infinite}$ and $\mathsf{is\_set\_theory}$ are consistent. If desired, it is possible to introduce sufficient axioms into HOL4 so that $\mathsf{is\_infinite}$ can be witnessed at a later point; we do not do so in this paper, but see Kumar et al. [17] for a discussion.

Notation We use the following notation for certain standard constructions in set theory. $x \in\!: y$ abbreviates $mem\ x\ y$. One is a singleton set, and $\mathsf{Boolset}$ is a set with exactly two distinct elements, called True and False. $\mathsf{Boolean} : \mathsf{bool} \Rightarrow \mathcal{U}$ maps HOL4 booleans into $\mathsf{Boolset}$ in the obvious way.

$x\ \mathsf{suchthat}\ P$, where $x : \mathcal{U}$ and $P : \mathcal{U} \Rightarrow \mathsf{bool}$, is the set of all elements of x that satisfy P. $\mathsf{Funspace}\ x\ y$, where $x, y : \mathcal{U}$, is the set of all function graphs with domain x and codomain y. $\mathsf{Abstract}\ x\ y\ f$, where $f : \mathcal{U} \Rightarrow \mathcal{U}$, is the subset of $x \times y$ such that all its elememts have the form $(a,f\ a)$ for some a. If for all such a it holds that $f\ a \in\!: y$, this construction is the function graph of f and an element of $\mathsf{Funspace}\ x\ y$. We write $b\ '\ a$ for function application: if b is a function graph $\mathsf{Abstract}\ x\ y\ f$ and $a \in\!: x$, then $b\ '\ a = f\ a$.

3.2 Fragments and fragment-localised semantics

Kunčar and Popescu's style of semantics departs from standard HOL semantics in two main ways. First, it does not interpret type variables. Hence, for the most part, we only need to consider interpretations of ground terms and ground types. Later, we will see that type variables are interpreted in a point-wise manner for every possible ground instantiation.

Second, the semantics is *fragment-localised*: since overloading may introduce dependencies that do not follow the order in which terms and types were introduced, it is sometimes necessary to give semantics to terms on the RHS of defining equations before all types and constants in the signature have an interpretation, or in other words, to give semantics to fragments of the signature. A *signature fragment* (*tys*,*consts*) of a signature *sig* is comprised of subsets of the (non-built-in) ground types (*tys*) and constants (*consts*) of *sig*. We require that the fragment is self-contained: for every $(c,ty) \in consts$, *ty* must be constructed using only *tys* and the built-in type constructors Fun and Bool:

$$\begin{aligned}&\mathsf{is\_sig\_fragment}\ sig\ (tys,consts) \stackrel{\mathsf{def}}{=}\\ &\quad tys \subseteq \mathsf{ground\_types}\ sig \wedge tys \subseteq \mathsf{nonbuiltin\_types} \wedge\\ &\quad consts \subseteq \mathsf{ground\_consts}\ sig \wedge consts \subseteq \mathsf{nonbuiltin\_constinsts} \wedge\\ &\quad \forall s\ c.\ (s,c) \in consts \Rightarrow c \in \mathsf{builtin\_closure}\ tys\end{aligned}$$

The *total fragment* of a signature is the fragment comprised of all its non-built-in types and constants:

$$\begin{aligned}&\mathsf{total\_fragment}\ sig \stackrel{\mathsf{def}}{=}\\ &\quad (\mathsf{ground\_types}\ sig \cap \mathsf{nonbuiltin\_types}, \mathsf{ground\_consts}\ sig \cap \mathsf{nonbuiltin\_constinsts})\end{aligned}$$

We let δ range over type interpretations $\mathsf{type} \Rightarrow \mathcal{U}$, and γ range over constant interpretations $\mathsf{string} \times \mathsf{type} \Rightarrow \mathcal{U}$. The built-in types and constants have a standard, immutable interpretation; hence, it is convenient to consider the extension ext of an interpretation δ to the built-ins (written $\mathsf{ext}\ \delta$), and likewise for constant interpretations:

$$\begin{aligned}&\mathsf{ext}\ \delta\ \mathsf{Bool} \stackrel{\mathsf{def}}{=} \mathsf{Boolset}\\ &\mathsf{ext}\ \delta\ (\mathsf{Fun}\ ty_1\ ty_2) \stackrel{\mathsf{def}}{=} \mathsf{Funspace}\ (\mathsf{ext}\ \delta\ ty_1)\ (\mathsf{ext}\ \delta\ ty_2)\\ &\mathsf{ext}\ \delta\ ty \stackrel{\mathsf{def}}{=} \delta\ ty \quad \text{otherwise}\end{aligned}$$

$$\begin{aligned}&\mathsf{ext}\ \delta\ \gamma\ (\text{«=»}, \mathsf{Fun}\ ty\ (\mathsf{Fun}\ ty\ \mathsf{Bool})) \stackrel{\mathsf{def}}{=}\\ &\quad \mathsf{Abstract}\ (\delta\ ty)\ (\mathsf{Funspace}\ (\delta\ ty)\ \mathsf{Boolset})\ (\lambda x.\ \mathsf{Abstract}\ (\delta\ ty)\ \mathsf{Boolset}\ (\lambda y.\ \mathsf{Boolean}\ (x = y)))\\ &\mathsf{ext}\ \delta\ \gamma\ ty \stackrel{\mathsf{def}}{=} \gamma\ ty \quad \text{otherwise}\end{aligned}$$

We say that (δ,γ) is a *fragment interpretation* of a fragment (*tys*,*tms*) if δ maps its types to non-empty sets, and γ maps its constants to elements of their types' interpretation:

$$\mathsf{is\_type\_frag\_interpretation}\ tys\ \delta \stackrel{\mathsf{def}}{=} \forall ty.\ ty \in tys \Rightarrow \mathsf{inhabited}\ (\delta\ ty)$$

$$\begin{aligned}&\mathsf{is\_frag\_interpretation}\ (tys,tms)\ \delta\ \gamma \stackrel{\mathsf{def}}{=}\\ &\quad \mathsf{is\_type\_frag\_interpretation}\ tys\ \delta \wedge\\ &\quad \forall (c,ty).\ (c,ty) \in tms \Rightarrow \gamma\ (c,ty) \in: \mathsf{ext}\ \delta\ ty\end{aligned}$$

A *fragment valuation*, ranged over by v, assigns meaning to all variables whose types are contained in the fragment:

$$\begin{aligned}&\mathsf{valuates\_frag}\ frag\ \delta\ v\ \Theta \stackrel{\mathsf{def}}{=}\\ &\quad \forall x\ ty.\ \Theta\ ty \in \mathsf{types\_of\_frag}\ frag \Rightarrow v\ (x,ty) \in: \mathsf{ext}\ \delta\ (\Theta\ ty)\end{aligned}$$

Here we depart slightly from Kunčar and Popescu by considering valuations of polymorphic terms relative to a ground type substitution Θ. Note that $v\ (x,ty)$ here is *not* an element of

$\delta\ ty$, but of $\delta\ (\Theta\ ty)$. We make this adaptation to avoid a variable capture issue. HOL has Church-style atoms, in the sense that variables with the same name but distinct types are considered to be distinct variables, so e.g. Var «x» (Tyvar «a») and Var «x» Bool may coexist in terms and theorem statements. Because the semantics of type variables are always taken relative to a ground substitution Θ, Kunčar and Popescu's definition of a fragment valuation (called an *$\mathcal{I}$-compatible valuation* in [22]) would force these two distinct «x» variables to have the same interpretation whenever Θ (Tyvar «a») = Bool.

The same issue must be considered for the term semantics, where we lift the interpretation of a fragment's types, constants and variables to all terms that can be built from the fragment:

$$\begin{aligned}
&\mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ (\mathsf{Var}\ x\ ty) \stackrel{\mathsf{def}}{=} v\ (x,ty)\\
&\mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ (\mathsf{Const}\ name\ ty) \stackrel{\mathsf{def}}{=} \gamma\ (name,\Theta\ ty)\\
&\mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ (\mathsf{Comb}\ t_1\ t_2) \stackrel{\mathsf{def}}{=}\\
&\quad \mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ t_1\ '\ (\mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ t_2)\\
&\mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ (\mathsf{Abs}\ (\mathsf{Var}\ x\ ty)\ b) \stackrel{\mathsf{def}}{=}\\
&\quad \mathsf{Abstract}\ (\delta\ (\Theta\ ty))\ (\delta\ (\Theta\ (\mathsf{typeof}\ b)))\\
&\qquad (\lambda\, m.\ \mathsf{termsem}\ \delta\ \gamma\ v(\!|(x,ty)\ \mapsto\ m|\!)\ \Theta\ b)
\end{aligned}$$

Where Kunčar and Popescu apply the ground type substitution Θ eagerly, i.e. before taking the term semantics, we instead parameterise our term semantics on Θ, applying it only at the latest possible moment. Crucially, we do not apply Θ at all to term variables.

We are now ready to define the meaning of sequents. An interpretation (δ,γ) of the fragment *frag* satisfies the sequent (h,c) relative to the ground substitution Θ if, whenever all the hypotheses h have semantics True under a fragment valuation v, then so does the conclusion c:

$$\begin{aligned}
&\mathsf{satisfies}\ frag\ \delta\ \gamma\ \Theta\ (h,c) \stackrel{\mathsf{def}}{=}\\
&\ \forall\, v.\\
&\quad \mathsf{valuates_frag}\ frag\ \delta\ v\ \Theta \wedge c \in \mathsf{terms_of_frag_uninst}\ frag\ \Theta \wedge\\
&\quad \mathsf{every}\ (\lambda\, t.\ t \in \mathsf{terms_of_frag_uninst}\ frag\ \Theta)\ h \wedge\\
&\quad \mathsf{every}\ (\lambda\, t.\ \mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ t = \mathsf{True})\ h \Rightarrow \mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ c = \mathsf{True}
\end{aligned}$$

As promised, we then give semantics to all sequents of a signature by closing satisfies under all ground type substitutions:

$$\begin{aligned}
&\mathsf{sat}\ sig\ \delta\ \gamma\ (h,c) \stackrel{\mathsf{def}}{=}\\
&\ \forall\, \Theta.\\
&\quad (\forall\, ty.\ \mathsf{tyvars}\ (\Theta\ ty) = []) \wedge (\forall\, ty.\ \mathsf{type_ok}\ (\mathsf{tysof}\ sig)\ (\Theta\ ty)) \wedge\\
&\quad \mathsf{every}\ (\lambda\, tm.\ tm \in \mathsf{ground_terms_uninst}\ sig\ \Theta)\ h \wedge\\
&\quad c \in \mathsf{ground_terms_uninst}\ sig\ \Theta \Rightarrow\\
&\qquad \mathsf{satisfies}\ (\mathsf{total_fragment}\ sig)\ \delta\ \gamma\ \Theta\ (h,c)
\end{aligned}$$

We say that an interpretation (δ,γ) *models* a theory if it interprets its total fragment and satisfies all of its axioms:

$$\begin{aligned}
&\mathsf{models}\ \delta\ \gamma\ thy \stackrel{\mathsf{def}}{=}\\
&\quad \mathsf{is_frag_interpretation}\ (\mathsf{total_fragment}\ (\mathsf{sigof}\ thy))\ \delta\ \gamma \wedge\\
&\quad \forall\, p.\ p \in \mathsf{axsof}\ thy \Rightarrow \mathsf{sat}\ (\mathsf{sigof}\ thy)\ (\mathsf{ext}\ \delta)\ (\mathsf{ext}\ (\mathsf{ext}\ \delta)\ \gamma)\ ([],p)
\end{aligned}$$

Finally, we define the entailment relation $\models$, the semantic counterpart to $\vdash$. In words, (thy,h) $\models$ c

holds if all its arguments are well-formed and if (h,c) is satisfied in every model of thy.

$$
\begin{array}{l}
(thy,h) \vDash c \stackrel{\mathsf{def}}{=} \\
\quad \mathsf{theory\_ok}\ thy \wedge \mathsf{every}\ (\mathsf{term\_ok}\ (\mathsf{sigof}\ thy))\ (c::h) \wedge \\
\quad \mathsf{every}\ (\lambda\, p.\ p\ \mathsf{has\_type\ Bool})\ (c::h) \wedge \mathsf{hypset\_ok}\ h \wedge \\
\quad \forall\, \delta\ \gamma.\ \mathsf{models}\ \delta\ \gamma\ thy \Rightarrow \mathsf{sat}\ (\mathsf{sigof}\ thy)\ (\mathsf{ext}\ \delta)\ (\mathsf{ext}\ (\mathsf{ext}\ \delta)\ \gamma)\ (h,c)
\end{array}
$$

4 Soundness

Our first main result is *soundness*:

$$\vdash \mathsf{is\_set\_theory}\ mem \Rightarrow \forall\, thy\ h\ c.\ (thy,h) \vdash c \Rightarrow (thy,h) \vDash c$$

In words, the provable sequents of a theory are satisfied in all models of the theory. The proof comprises around 700 lines of HOL4, and is a mostly straightforward induction on the derivation of the $\vdash$ judgement. The most notable aspect of this proof is that with the satisfiability relation of Kunčar and Popescu [22], which features eager application of ground substitutions instead of the lazy term semantics we introduce in Section 3, the ABS case of the induction does not go through.[4] The proof obligation in the ABS case is:

$$
\begin{array}{l}
\mathsf{is\_set\_theory}\ mem \Rightarrow \\
\quad \forall\, thy\ x\ ty\ h\ l\ r. \\
\qquad \mathsf{Var}\ x\ ty \notin \mathsf{fv}\ h \wedge \mathsf{type\_ok}\ (\mathsf{tysof}\ thy)\ ty \wedge (thy,h) \vDash l === r \Rightarrow \\
\qquad\quad (thy,h) \vDash \mathsf{Abs}\ (\mathsf{Var}\ x\ ty)\ l === \mathsf{Abs}\ (\mathsf{Var}\ x\ ty)\ r
\end{array}
$$

The issue here is that, after unfolding $\vDash$ and sat to obtain a ground type substitution Θ, the freshness side condition no longer applies: it does not follow that $\mathsf{Var}\ x\ (\Theta\ ty)$ is fresh in h. If h already contains such a variable, its valuation becomes captured by the valuation of the bound variable x, rendering the induction hypothesis inapplicable. In our lazy semantics, Θ is not applied before the valuation, and hence the variable capture problem does not arise.

In personal communication, Popescu acknowledges this issue and proposes an alternative fix which involves adding additional freshness side conditions to the inference rules. That would be less invasive in terms of semantics, but has the disadvantage of burdening the user of the inference system with more side conditions. Our solution leaves the inference system unchanged.

5 Model construction

In the previous section, we showed that all provable sequents of a theory are satisfied in all models of the theory. This section tackles the missing puzzle piece before we can consider consistency: to show that if the theory is constructed by definitional extension of the pre-defined initial contexts (see Section 2.4), then a model exists.

This is the most challenging part of the work, and the part where we differ most from Kumar et al. In the absence of overloading, the model can be constructed incrementally: when theory extension cannot change the meaning of previously introduced types and constants, any model (δ,γ) of the old theory can be updated to create a model of the new theory, e.g., $(\delta(\!|ty \mapsto x|\!),\gamma(\!|c \mapsto y|\!))$ where x, y model the new types and constants ty, c. This has the very

[4] ABS is a derived rule in Kunčar and Popescu, but the same problem surfaces for their EXT rule. The proof step that fails is the first invocation of Lemma 11(4) in the proof of Lemma 12 [22, p. 550], where it does not follow that the valuations ξ and ξ' coincide on $\theta(\Gamma)$. They coincide on Γ, but this does not suffice.

pleasant consequence that, when augmenting a model to accommodate a theory update, the details of how exactly the previous model was constructed can be ignored. Hence there is no need to explicitly write down a model for the whole theory: it suffices to prove its existence by induction on the context.

Here, we have no such luxury: overloading can change the meaning of previously defined terms, and even types if the overloaded constant is used to define the types characteristic predicate. Moreover, since we only interpret ground terms and types, the old model can become useless even for non-overloading extensions: signature extension causes the set of ground terms and types to grow, and the new ones will not be covered by the model construction for the previous, smaller signature.

With incremental construction not an option, we define instead a pair of mutually recursive functions, type_interpretation and term_interpretation. Their definition is 140 lines of HOL4 script, excluding auxiliary functions, and hence not feasible to include in full here. The main ideas of the interpretation are standard:

- A constant with no definition is interpreted as an arbitrary (Hilbert-chosen) element of its type's interpretation.
- Types with no definition are interpreted as One.
- A constant Const c (Θ ty) with defining equation Const c ty === t is interpreted as the term semantics of t relative to Θ.
- A type Θ ty, whose outermost type constructor matches TypeDefn s $pred$ abs rep, is interpreted as x suchthat ($\lambda\, tm.\ y$ ' tm = True), where x is the type semantics of Θ (domain (typeof $pred$)) and y is the term semantics of $pred$ relative to Θ.
- If a constant matches the abstraction function of a type definition, and if the abstract and concrete types have (respectively) interpretations δ_1, δ_2, then its interpretation is

  $$\mathsf{Abstract}\ \delta_2\ \delta_1\ \ (\lambda\, v.\ \mathsf{if}\ v \in\!: \delta_1\ \mathsf{then}\ v\ \mathsf{else}\ \varepsilon v.\ v \in\!: \delta_1)$$

  Similarly, a constant matching the representation function is interpreted by Abstract δ_1 δ_2 I, where I is identity.
- «ind» is interpreted as an infinite set if the axiom of infinity is present in the theory (otherwise, any inhabited set suffices), and «@» is interpreted as a choice function if the axiom of choice is present.

The definition is bogged down by the bookkeeping necessary to realise the above: at every step, the context must be scanned for matching definitions, and ground substitutions witnessing any such match must be constructed and juggled. The most tedious bookkeeping involves the termination argument. First, notice that because (non-definitional) contexts may have cyclic definitions, this model construction does not terminate in general. We circumvent this with a trick, similar to the definition of HOL's WHILE combinator: recursive calls are guarded by an if statement checking if the dependency relation is terminating, so that the recursion is short-circuited precisely when it is not well-founded:

$$\mathsf{type\_interpretation}\ \mathit{ind}\ \mathit{ctxt}\ \mathit{ty} = \mathsf{if}\ \neg\mathsf{terminating}\ (\mathsf{subst\_clos}\ (\rightsquigarrow\ \mathit{ctxt}))\ \mathsf{then}\ \mathsf{One}\ \mathsf{else}\ \ldots$$

The second issue is that the function must be carefully written to make sure all its recursive calls stay within the (transitive closure of the substitutive closure of the) dependency relation. E.g.,

whenever we use the term semantics in the informal explanation above, we must first construct a theory fragment from its dependents and take the term semantics wrt. a fragment model of that interpretation. This termination proof is quite laborious, at 495 lines of proof script. Through our effort to prove termination, we noticed that the dependency relation as defined by Kunčar and Popescu [22] is not sufficient to show that the model construction is well-founded: their dependency relation is only defined on non-built-in types, and hence fails to capture dependencies in type arguments guarded by function arrows. This seems to have gone unnoticed because in the precise spot where our proof needs this extended dependency relation, their proof relies on an innocent-looking but faulty lemma stating that type instantiation commutes with $^\bullet$:

$$\mathsf{set}\ (\Theta\ ty)^\bullet = \{\ \Theta\ ty' \mid ty' \in ty^\bullet\ \}$$

For a counterexample, choose $ty = \mathsf{Tyvar}$ «a» and Θ such that Θ (Tyvar «a») = Fun ty' ty' where ty' is a non-built-in ground type. The LHS evaluates to $\{\ ty'\ \}$, but the RHS to $\{\ \mathsf{Fun}\ ty'\ ty'\ \}$.

Our solution, as discussed in Section 2, is to extend the dependency relation to account for dependencies on built-in types. From personal correspondence with Popescu, we learn that he has independently discovered the same issue since publication. Popescu proposes a similar fix, except that it keeps the dependency relation unchanged, instead proving that taking the closure of the dependency relation under destruction of built-in type constructors preserves well-foundedness. Our fix has minimal impact on our semantics at the cost of kicking a can down the road: future verification of a cyclicity checker will need to consider a larger dependency relation.

We write axioms_admissible *ctxt* if the unchecked axioms of *ctxt* are at most those hol_ctxt. The main result of this section is that for such contexts, the interpretation defined above is indeed a model:

$$\begin{array}{l}\vdash \mathsf{is\_set\_theory}\ mem \wedge \mathsf{inhabited}\ ind \Rightarrow \\ \quad \forall\, ctxt. \\ \qquad ctxt\ \mathsf{extends}\ \mathsf{init\_ctxt} \wedge \mathsf{axioms\_admissible}\ mem\ ind\ ctxt \Rightarrow \\ \qquad\ \mathsf{models}\ (\mathsf{type\_interpretation}\ ind\ ctxt)\ (\mathsf{term\_interpretation}\ ind\ ctxt)\ (\mathsf{thyof}\ ctxt)\end{array}$$

The proof is tedious: the script file dedicated to the model construction is around 5500 lines. Recall that models has two conjuncts: the candidate model must be a total fragment interpretation, and all axioms introduced by the context must be satisfied in the model. The first conjunct is by well-founded induction on the definition of the dependency relation. The second conjunct is by induction on $ctxt_2$ for contexts of the form $ctxt_1$ ++ $ctxt_2$. The prefix $ctxt_1$ must be carried through the induction because, as discussed above, the model construction applies to the entire context only. Most interesting are the axioms arising from ConstSpec and TypeDefn; let a denote such an axiom, and let (δ,γ) abbreviate the model under consideration. The definition of updates yields (thyof $ctxt_2$,[]) $\vdash a$, but soundness is not immediately useful because (δ,γ) models $ctxt_1$ ++ $ctxt_2$, not $ctxt_2$. The solution is to notice that the model construction is valid for subtheories of thyof ($ctxt_1$ ++ $ctxt_2$) that have the same signature but fewer axioms. Since $\vdash$ is closed under theory extension we get ((sigof ($ctxt_1$ ++ $ctxt_2$),axsof $ctxt_2$),[]) $\vdash a$, and by the induction hypothesis, models δ γ (sigof ($ctxt_1$ ++ $ctxt_2$),axsof $ctxt_2$); by soundness we get that a is satisfied in (δ,γ).

6 Consistency

We say that a theory is *consistent* if it has a provable sequent and an unprovable sequent.[5] The sequents are fixed for convenience:

$$\begin{aligned}&\mathsf{consistent\_theory}\ thy \stackrel{\mathsf{def}}{=}\\ &\quad (thy,[]) \vdash \mathsf{Var}\ «\mathrm{x}»\ \mathsf{Bool} === \mathsf{Var}\ «\mathrm{x}»\ \mathsf{Bool} \wedge\\ &\quad \neg((thy,[]) \vdash \mathsf{Var}\ «\mathrm{x}»\ \mathsf{Bool} === \mathsf{Var}\ «\mathrm{y}»\ \mathsf{Bool})\end{aligned}$$

Our main result, the consistency of higher-order logic with ad-hoc overloading, is a straightforward consequence of soundness and the existence of a model. It is stated as follows:

$$\begin{aligned}&\vdash \mathsf{is\_set\_theory}\ mem \wedge \mathsf{is\_infinite}\ mem\ ind \Rightarrow\\ &\quad \forall\, ctxt.\ \mathsf{definitional\_extension}\ ctxt\ \mathsf{hol\_ctxt} \Rightarrow \mathsf{consistent\_theory}\ (\mathsf{thyof}\ ctxt)\end{aligned}$$

7 Related Work

Our mechanisation extends the work of Kumar et al. [17], who prove soundness of higher-order logic to arrive at a—down to the machine code—verified HOL theorem prover. Earlier in Section 3 we describe the approach to defining a model for a definitional theory in set theory which they follow. Their consistency proof is via a reduction to stateless HOL [32], where the context is not updated as definitions are annotated with definitions for each constant and type. The journal version [18] drops stateless HOL in favour of a direct consistency proof. We build upon much of their infrastructure; for example, definitions and properties of type substitutions. More recently, Abrahamsson used Candle to build a verified OpenTheory article checker [1].

Wenzel [31] aims to close foundational gaps that arise in Isabelle through allowing type classes and overloading for HOL. Definitional extensions are claimed to be *meta-safe*, i.e. conservative and *realisable*. The latter means that new constants in provable terms of an extension can be replaced such that the resulting term is provable in a smaller theory. A restriction is that constant and type definitions cannot arbitrarily be mixed. Obua [26] states that (mixed) type definitions and overloading are safe if any two definitions with the same name are non-overlapping and unfolding definitions terminates. He also gives an algorithm, but misses that additional type definitions may lead to inconsistencies.

These issues are discovered and solved by Kunčar and Popescu [20, 22] whose ideas we utilise. They introduce a dependency relation to track types and constants that a definition depends on, and ultimately prove consistency of definitional theories with overloading by constructing a model with ground, fragment-localised semantics. The soundness proof goes by recursion over a hull of the dependency relation, which takes type instances and the transitive closure into account. Their semantics are wrt. ground terms, i.e., a formula ϕ holds in a model $\mathcal{I}$ if for all ground type substitutions θ and all valuations ξ the term $[\theta(\phi)]^{\mathcal{I}}(\xi)$ evaluates to true in $\mathcal{I}$. In subsequent work, Kunčar and Popescu [21] show proof-theoretic conservativity (and meta-safety) of HOL with overloading over initial HOL, which we call the initial context.

Gengelbach and Weber [7] generalise the model construction of Kunčar and Popescu to give a model-theoretic conservativity result. A definitional theory extension with overloading may affect the interpretation of terms in a model prior to extension compared with a model for the extended theory. They observe that a model for the extensions can be constructed keeping the (by the theory extension) unaffected parts of the smaller model. When updating only necessary

[5] As the logic is classical, the existence of an unprovable sequent is equivalent to the usual definition of consistency: for all φ, at most one of $(thy, []) \vdash \varphi$ and $(thy, []) \vdash \neg\varphi$ are sequents.

interpretations, both models have some equal fragments. The problems that we found in Kunčar and Popescu's work impact their reasoning too, but should be resolvable in the same way.

By constraining definitions to *composable* dependencies, Kunčar makes the dependency relation decidable [20]. The dependency relation of a theory D is composable if a type substitutive dependency chain from a definiendum u to another definiendum u' of D exists only with constraints on the last step: only if from $u \rightsquigarrow_D^{\downarrow+} v$ either $v \geq u'$ or $u' \geq v$ holds. In the paper the author gives a pen-and-paper proof for decidability and correctness to check acyclicity of a composable dependency relation of a definitional theory.

Carneiro [5] agrees that correctness of definitional axioms is critical for the soundness of a logic. In his approach to bootstrapping trust into the Metamath Zero (MM0) proof system, definitions are conservative: The definiendum can be replaced by the definiens, hence soundness without definitions implies soundness of the system with definitions. As every expression in MM0 has a unique sort and there is no hierarchy of sorts, there is no support for overloading in MM0. Contrary to ours, at present the soundness proof is not using the implementation but an abstraction. The MM0 verification system consists of a verifier that checks proofs against a specification, and a compiler that compiles a high level proof into a low-level representation for the verifier. Ultimately, MM0 aims for correctness of the verifier only, which is specified in its own language and not yet formally proven correct.

Another non-HOL proof assistant is Nuprl, which uses the language of untyped λ-calculus in a variant of Martin Löf's dependent type theory. The meta-theory of Nuprl is consistent relative to the consistency of the Coq theorem prover [2], which amounts to showing soundness of 70 proof rules where both types and terms are expressed in λ-calculus. On this result, Rahli and Bickford [29] base their verified Nuprl implementation. Constants and types, called *abstractions* in Nuprl, may syntactically occur prior to their definition, which enables implicit definitions as for example fix-points. Their semantics similarly treats sequents as true, if they are true for all possible extensions of the yet undefined syntax. That corresponds to Kunčar and Popescu's approach of ground semantics. The verified implementation translates a proof from Nuprl into the embedding in Coq.

Sozeau et al. develop a correct verifier for the Coq kernel within Coq [30], and as in Coq's logic a verification task is a type check, they prove the implementation of a type checker correct. The type checker will return a proof that a term has a certain type. By Coq's extraction mechanism they obtain a verified executable type checker for the kernel of Coq.

In addition to proving consistency through a model-theoretic argument and soundness of the derivation rules for HOL, Kunčar and Popescu [23] arrive at a proof-theoretic/syntactic result for HOL relative to HOL with unfolded definitions, called HOLC. By translating type definitions as well as constant definitions and restricting the use of the type instantiation rule (a complication due to overloading), they achieve conservativity of HOLC relative to HOL with definitions. Consistency of HOL with ad-hoc overloading thus translates to consistency of HOLC, which in turn holds as HOL without definitions is consistent.

8 Conclusion

We have presented a mechanised proof of consistency for HOL with ad-hoc overloading of constant definitions. In doing so, we take an important first step towards a verified implementation of an Isabelle/HOL kernel. Given executable HOL4 definitions, extraction of such an implementation is almost routine thanks to the rich ecosystem around CakeML [24, 14, 9]. Our termination criterion on contexts, however, is not executable as expressed. The main missing puzzle piece is to implement and verify Kunčar's algorithm for checking orthogonality and termination of

dependency relations [20]. This will be the topic of future work.

Our formalisation can be also be used as a staging ground for exploring other meta-theoretical properties of Isabelle/HOL, such as safety and conservativity. Another interesting idea is to extend our formalisation to account for other ways that Isabelle/HOL's logic differs from the other HOLs, such as axiomatic type classes, schematic variables and the meta logic/object logic distinction. Presently, our setting is essentially HOL Light [12] with ad-hoc overloading.

In the meantime, our work significantly strengthens our assurance in the foundational argument for why Isabelle/HOL is consistent. It is almost cliché to motivate a verification effort, *ex post facto*, by reference to the bugs it uncovers. But for bugs in the consistency argument for a widely used proof assistant, this motivation seems doubly strong.

9 Acknowledgments

We are grateful to Andrei Popescu for insightful technical discussions, and to Oskar Abrahamsson, Yong Kiam Tan, Konrad Slind and Tjark Weber for their comments on drafts. We thank the anonymous reviewers for their remarks. We also thank C.F. Liljewalchs stipendiestiftelse for supporting the second author during his visit to UNSW and CSIRO's Data61 in Sydney, Australia.

References

[1] Oskar Abrahamsson. A verified proof checker for higher-order logic. *Journal of Logical and Algebraic Methods in Programming*, page 100530, 2020.

[2] Abhishek Anand and Vincent Rahli. Towards a Formally Verified Proof Assistant. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving*, Lecture Notes in Computer Science, pages 27–44, Cham, 2014. Springer International Publishing.

[3] Rob Arthan. HOL formalised: Semantics. http://lemma-one.com/ProofPower/specs/spc002.pdf.

[4] Rob Arthan. HOL constant definition done right. In *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 531–536. Springer, 2014.

[5] Mario Carneiro. Metamath Zero: The Cartesian Theorem Prover. *arXiv:1910.10703 [cs, math]*, October 2019. arXiv: 1910.10703.

[6] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.

[7] Arve Gengelbach and Tjark Weber. Model-theoretic Conservative Extension of Definitional Theories. In *Proceedings of 12th Workshop on Logical and Semantic Frameworks with Applications (LSFA 2017)*, pages 4–16, Brasília, Brasil, September 2017. Elsevier.

[8] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, pages 173–198, 1931.

[9] Armaël Guéneau, Magnus O. Myreen, Ramana Kumar, and Michael Norrish. Verified characteristic formulae for CakeML. In Hongseok Yang, editor, *European Symposium on Programming (ESOP)*, volume 10201 of *LNCS*. Springer, 2017.

[10] Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5:e2, 2017.

[11] John Harrison. Towards self-verification of HOL Light. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2006.

[12] John Harrison. HOL Light: An overview. In *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany. Proceedings*, pages 60–66. Springer, 2009.

[13] John Harrison, John O'Leary, and Andrew Tolmach, editors. *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[14] Son Ho, Oskar Abrahamsson, Ramana Kumar, Magnus O. Myreen, Yong Kiam Tan, and Michael Norrish. Proof-producing synthesis of CakeML with I/O and local state from monadic HOL functions. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 646–662. Springer, 2018.

[15] Fabian Immler, Jonas Rädle, and Makarius Wenzel. Virtualization of HOL4 in Isabelle. In Harrison et al. [13], pages 21:1–21:18.

[16] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009*, pages 207–220. ACM, 2009.

[17] Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. HOL with definitions: Semantics, soundness, and a verified implementation. In *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 308–324. Springer, 2014.

[18] Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. Self-formalisation of higher-order logic - semantics, soundness, and a verified implementation. *J. Autom. Reasoning*, 56(3), 2016.

[19] Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. CakeML: a verified implementation of ML. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 179–192. ACM, 2014.

[20] Ondřej Kunčar. Correctness of Isabelle's cyclicity checker: Implementability of overloading in proof assistants. In *Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, Mumbai, India, January 15-17, 2015*, pages 85–94. ACM, 2015.

[21] Ondřej Kunčar and Andrei Popescu. Safety and Conservativity of Definitions in HOL and Isabelle/HOL. *Proc. ACM Program. Lang.*, 2(POPL):24:1–24:26, December 2017.

[22] Ondřej Kunčar and Andrei Popescu. A consistent foundation for Isabelle/HOL. *Journal of Automated Reasoning*, 62(4):531–555, Apr 2019.

[23] Ondřej Kunčar and Andrei Popescu. Comprehending Isabelle/HOL's Consistency. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 724–749. Springer, 2017.

[24] Magnus Myreen and Scott Owens. Proof-producing translation of higher-order logic into pure and stateful ML. *J. Funct. Program.*, 24(2–3):284–315, 2014.

[25] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: a Proof Assistant for Higher-Order Logic*, volume 2283. Springer, 2002.

[26] Steven Obua. Checking conservativity of overloaded definitions in higher-order logic. In *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings*, pages 212–226. Springer, 2006.

[27] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In Richard L. Wexelblat, editor, *PLDI*, pages 199–208. ACM, 1988.

[28] Andrew M. Pitts. The HOL logic. In M.J.C. Gordon and Tom Melham, editors, *Introduction to HOL: A Theorem-Proving Environment for Higher-Order Logic*, pages 191–232. Cambridge University Press, 1993.

[29] Vincent Rahli, Liron Cohen, and Mark Bickford. A Verified Theorem Prover Backend Supported by a Monotonic Library. In *EPiC Series in Computing*, volume 57, pages 564–582. EasyChair, October 2018.

[30] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq Coq correct! verification of type checking and erasure for Coq, in Coq. *Proceedings of the ACM on Programming Languages*, 4(POPL):8:1–8:28, December 2019.

[31] Markus Wenzel. Type classes and overloading in higher-order logic. In *Theorem Proving in Higher Order Logics, 10th International Conference, TPHOLs'97, Murray Hill, NJ, USA, August 19-22, 1997, Proceedings*, pages 307–322, 1997.

[32] Freek Wiedijk. Stateless HOL. *Electronic Proceedings in Theoretical Computer Science*, 53:47–61, March 2011. arXiv: 1103.3322 version: 1.