Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 338 (2018) 133–145

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Model-Theoretic Conservative Extension for Definitional Theories

Arve Gengelbach[a,1] Tjark Weber[a,2]

[a] *Department of Information Technology*
Uppsala University
Uppsala, Sweden

Abstract

Many logical frameworks allow extensions, i. e. the introduction of new symbols, by definitions. Different from asserting arbitrary non-logical axioms, extensions by definitions are expected to be conservative: they should entail no new theorems in the original language. The popular theorem prover Isabelle implements a variant of higher-order logic that allows *ad hoc* overloading of constants. In 2015, Kunčar and Popescu introduced *definitional theories*, which impose a non-circularity condition on constant and type definitions in this logic, and showed that this condition is sufficient for definitional extensions to preserve consistency. We strengthen and generalize this result by showing that extensions of definitional theories are model-theoretic conservative, i. e. every model of the original theory can be expanded to a model of the extended theory.

Keywords: classical higher-order logic, conservative theory extension, model-theoretic conservativity, definitional theories, ground semantics, Isabelle

1 Introduction

Among the many different mechanisms for extending theories by definitions, particularly constant and type definitions [3,12], the one used by the theorem prover Isabelle [9] had flaws. Isabelle implements polymorphic higher-order logic with *ad hoc* overloading. Users extend a theory incrementally by defining constants and types (or in the case of overloaded constants, by defining constant instances for previously defined types). A key strength of overloading is the separation of the declaration of a constant from its instance definitions. However, by combining type definitions with *ad hoc* overloading of constants, it was possible to introduce an inconsistent extension of a theory [6]. For a simple example, consider a theory that declares a polymorphic constant c_α. Define a type τ by $\tau \equiv \{\mathsf{True}, c_{\mathsf{bool}}\}$. Next, define the constant instance c_{bool} by $c_{\mathsf{bool}} \equiv \neg(\forall x_\tau, y_\tau.\ x_\tau \doteq y_\tau)$. (We use $\doteq$ rather

[1] Email: arve.gengelbach@it.uu.se
[2] Email: tjark.weber@it.uu.se

https://doi.org/10.1016/j.entcs.2018.10.009
1571-0661/© 2018 The Author(s). Published by Elsevier B.V.
This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

than $=$ to avoid confusion with equality in our meta language.) It follows that $c_{\mathsf{bool}} \doteq \mathsf{True}$ iff τ is a singleton iff $c_{\mathsf{bool}} \doteq \mathsf{False}$, a contradiction.

To address this issue, Kunčar and Popescu [6] in 2015 introduced *definitional theories* for Isabelle. In the previous example, τ (through its definition) depends on c_{bool}, and c_{bool} depends on τ. *Well-formed* definitional theories disallow circular dependencies between constant and type definitions. Using a novel semantics for higher-order logic that interprets polymorphic types as macros for families of ground types, Kunčar and Popescu showed that well-formed definitional theories preserve consistency, i. e. every well-formed definitional theory has a model. Their acyclicity check has since been integrated into Isabelle [5].

As pointed out in [6], consistency is "a crucial, but rather weak property." It merely ensures that well-formed definitional theories do not prove False. Extensions by definitions are generally expected to satisfy a much stronger property known as *conservativity* [1,13]. Conservativity comes in a proof-theoretic (syntactic) and in a model-theoretic (semantic) flavour. In this paper, we are primarily concerned with the latter. Roughly, an extension D' of a theory D is *model-theoretic conservative* if every model $\mathcal{M}$ of D can be expanded to a model $\mathcal{M}'$ of D'. (We give a more precise definition in Section 3.) Note that model-theoretic conservativity immediately implies consistency: if D has a model, then so does D'.

We observe that extensions of definitional theories are *not* model-theoretic conservative if we require that the expanded model leaves the interpretation of all constants that are defined in the original theory D unchanged. For a simple counterexample, assume constants c_{bool} and d_{bool}, let $D := \{c_{\mathsf{bool}} \equiv d_{\mathsf{bool}}\}$ and $D' := D \cup \{d_{\mathsf{bool}} \equiv \mathsf{True}\}$. Then any model of D that interprets c_{bool} as false cannot be expanded (in the above sense) to a model of D'. The issue here is again that definitions may be provided separately from declarations: c_{bool} depends on d_{bool}, but the definition of d_{bool} is only provided in D'. In general, an extension by definitions in this logic does not imply an extension of the signature, i. e. the introduction of new symbols.

This example motivates a modified (more permissive) definition of model expansion. Our main contributions in this paper are:

(i) We define a notion of model expansion that is suitable for definitional theories, where definitions may be provided separately from declarations; and

(ii) we show that extensions of well-formed definitional theories are model-theoretic conservative with respect to this notion of model expansion.

This strengthens and generalizes the consistency result previously obtained by Kunčar and Popescu [6].

The rest of the paper is structured as follows. We introduce the language of polymorphic higher-order logic and definitional theories in Section 2. Our main results, including a suitable notion of model expansion and a proof of model-theoretic conservativity, are presented in Section 3. We give an overview of related work in Section 4 and conclude with a discussion of future work, in particular with regard to proof-theoretic conservativity, in Section 5.

2 Background

This section introduces the language of polymorphic higher-order logic (HOL), definitional theories and their semantics. We follow the notation and naming conventions of [6], to which we refer the reader for additional motivation and examples. We do not describe the deductive system of HOL [10], since it is not relevant in the context of this paper.

2.1 The Language of Polymorphic HOL

The syntax of polymorphic HOL is that of the simply-typed lambda calculus, enriched with a first-order language of types. We fix an infinite set TVar of *type variables*, ranged over by α, β, and an infinite set Var of *(term) variables*, ranged over by x, y.

A *signature* is a four-tuple $(K, \mathrm{arOf}, \mathsf{Const}, \mathrm{tpOf})$, where K is a countable set of symbols called *type constructors*, and Const is a countable set of symbols called *constants*. These sets are assumed to be disjoint. Each type constructor has an associated *arity* that is given by the function $\mathrm{arOf}\colon K \to \mathbb{N}$. Each constant has an associated *type* that is given by the function $\mathrm{tpOf}\colon \mathsf{Const} \to \mathsf{Type}$, where the set Type, ranged over by σ, τ, is defined inductively as the smallest set such that

- $\mathsf{TVar} \subseteq \mathsf{Type}$, and
- $(\sigma_1, \ldots, \sigma_n)k \in \mathsf{Type}$ whenever $k \in K$, $\mathrm{arOf}(k) = n$ and $\sigma_1, \ldots, \sigma_n \in \mathsf{Type}$.

When $\mathrm{arOf}(k) = 0$ we write k for the type $()k$.

For the remainder of this paper, we will assume a fixed signature. Moreover, we assume that K contains the following *built-in* type constructors:

- bool of arity 0,
- ind of arity 0,
- $\Rightarrow$ a right-associative type constructor of arity 2.

We also assume that Const contains the following *built-in* constants:

- $\to$ of type $\mathsf{bool} \Rightarrow \mathsf{bool} \Rightarrow \mathsf{bool}$,
- $\doteq$ of type $\alpha \Rightarrow \alpha \Rightarrow \mathsf{bool}$,
- some of type $(\alpha \Rightarrow \mathsf{bool}) \Rightarrow \alpha$,
- zero of type ind,
- succ of type $\mathsf{ind} \Rightarrow \mathsf{ind}$.

We say that a type σ is *built-in* if either $\sigma = \mathsf{bool}$ or $\sigma = \mathsf{ind}$ or if there exist types $\sigma_1, \sigma_2 \in \mathsf{Type}$ such that $\sigma = \sigma_1 \Rightarrow \sigma_2$.

A *type substitution* is a function $\rho\colon \mathsf{TVar} \to \mathsf{Type}$ that replaces type variables by types. We denote the set of type substitutions by TSubst. Generally we extend a type substitution ρ to a function on Type by defining, for each type constructor $k \in K$,

$$\rho\left((\sigma_1, \ldots, \sigma_{\mathrm{arOf}(k)})k\right) := \left(\rho(\sigma_1), \ldots, \rho(\sigma_{\mathrm{arOf}(k)})\right) k$$

Let σ, σ' be two types. If there exists a type substitution $\rho \in \mathsf{TSubst}$ such that $\rho(\sigma) = \sigma'$ we write $\sigma' \leq \sigma$ and say that σ' is an *instance* of σ.

The set of *constant instances* CInst is a subset of the cartesian product $\mathsf{Const} \times \mathsf{Type}$, where $(c, \sigma) \in \mathsf{CInst}$ if and only if σ is an instance of $\mathrm{tpOf}(c)$. We use c_σ as shorthand notation for the tuple (c, σ). We extend this notation to other terms, e. g. typed variables, defined below. We say that c_σ is an *instance* of c. A constant instance is *built-in* if it is an instance of a built-in constant.

The *terms* of our language are given by the following grammar, for $x \in \mathsf{Var}$, σ, $\sigma' \in \mathsf{Type}$, and $c_\sigma \in \mathsf{CInst}$:

$$t \quad ::= \quad x_\sigma \quad | \quad c_\sigma \quad | \quad (t_{\sigma' \Rightarrow \sigma}\, t'_{\sigma'})_\sigma \quad | \quad (\lambda x_{\sigma'}.t_\sigma)_{\sigma' \Rightarrow \sigma}$$

We may write t for t_σ when there is no risk of ambiguity. We require all terms to be *well-typed*, i. e. in $t\ t'$ the type of t' has to be the same as the argument type of t. Equality of terms is considered modulo α-equivalence. The set of all terms is denoted by Term. We say that a term is *closed* if it does not contain any free (term) variables. We extend tpOf to terms by defining $\mathrm{tpOf}(t_\sigma) := \sigma$.

For $u \in \mathsf{Term} \cup \mathsf{Type}$, we write $\mathrm{TV}(u)$ for the set of type variables that occur syntactically in u. We can apply a type substitution ρ to a term t, written $\rho(t)$, by applying ρ to all type variables that occur in t.

For a set S of types, constants or constant instances, we define $S^\bullet$ to mean the subset of all items that are *not* built-in. For instance, $\mathsf{Type}^\bullet$ is the set of all types that are not built-in, and $\mathsf{CInst}^\bullet$ is the set of all constant instances that are not built-in., i. e. that are not an instance of a built-in constant.

We define a function $\mathrm{types}^\bullet : \mathsf{Term} \cup \mathsf{Type} \to \mathcal{P}(\mathsf{Type}^\bullet)$ to collect all non-built-in types that occur in a given term or type. For k not a built-in type constructor, we define $\mathrm{types}^\bullet((\sigma_1, \ldots, \sigma_n)k) := \{(\sigma_1, \ldots, \sigma_n)k\}$, i. e. $\mathrm{types}^\bullet$ does *not* recurse into the argument types of non-built-in type constructors. For instance, if K contains a type constructor list of arity 1 (written in postfix notation), $\mathrm{types}^\bullet(\alpha\ \mathsf{list} \Rightarrow \mathsf{bool}) = \{\alpha\ \mathsf{list}\}$. Likewise, we define a function $\mathrm{consts}^\bullet : \mathsf{Term} \to \mathcal{P}(\mathsf{CInst}^\bullet)$ to collect all non-built-in constant instances that occur in a given term.

We say that a type is *ground* if it contains no type variables. The set of ground types is denoted by GType. We denote the set of type substitutions that map type variables to ground types only by $\mathsf{GTSubst}$. A constant instance $c_\sigma \in \mathsf{CInst}$ is *ground* if its type σ is ground. We denote the set of ground constant instances by GCInst.

2.2 Definitional Theories

Definitional theories are theories that consist of definitions $u \equiv t$, with the constant instance or type u that is being defined on the left-hand side, and the defining term t on the right-hand side. Constant instances are defined by a term, and types are defined by a predicate.

Definition 2.1 [Definitional Theory]

- A *constant instance definition* is of the form $c_\sigma \equiv t$ with $c_\sigma \in \mathsf{CInst}^\bullet$, $t \in \mathsf{Term}$ a

closed term of type σ, and $\mathrm{TV}(t) \subseteq \mathrm{TV}(\sigma)$.

- A *type definition* is of the form $\tau \equiv t$ with $\tau \in \mathsf{Type}^\bullet \setminus \mathsf{TVar}$, $t \in \mathsf{Term}$ a closed term of type $\sigma \Rightarrow \mathsf{bool}$ (for some $\sigma \in \mathsf{Type}$), and $\mathrm{TV}(t) \subseteq \mathrm{TV}(\tau)$.
- A *definitional theory* is a finite set of (constant instance or type) definitions.

The semantics of $\equiv$ is described further below.

First introduced in [6], certain definitional theories guarantee consistency, i. e. the existence of a model. These are the so-called *well-formed* definitional theories. Well-formedness requires *orthogonality* of definitions, which we define next.

Definition 2.2 [Orthogonality] We say two types σ and τ are *orthogonal*, written $\sigma \# \tau$, if they have no common instance.

We extend this notion to constant instances by defining $c_\sigma \# d_\tau$ to mean $c \neq d$ or $\sigma \# \tau$.

We say a definitional theory D is *orthogonal* if for all distinct elements $u \equiv t$, $u' \equiv t' \in D$ either one is a type definition and the other is a constant instance definition, or both are of the same kind and u and u' are orthogonal.

Moreover, as motivated by the example in Section 1, well-formedness requires that there are no circular dependencies between definitions. We introduce a binary relation, written $\rightsquigarrow$, on non-built-in types and non-built-in constant instances, to record the types and constant instances on which a definition depends, i. e. the types and constant instances that occur within the right-hand side of a definition.

Definition 2.3 [Dependency Relation] Let D be a definitional theory. For u, $v \in \mathsf{CInst}^\bullet \cup \mathsf{Type}^\bullet$ non-built-in constant instances or types, we define $u \rightsquigarrow_D v$ if one of the following two conditions holds:

- $\exists t \in \mathsf{Term}, u \equiv t \in D$ such that $v \in \mathrm{types}^\bullet(t) \cup \mathrm{consts}^\bullet(t)$, or
- $\exists c \in \mathsf{Const}^\bullet$ such that $u = c_{\mathrm{tpOf}(c)}$ and $v \in \mathrm{types}^\bullet(\mathrm{tpOf}(c))$.

In this case, we say that u *depends on* v.

The relation $\rightsquigarrow_D$ is determined by the definitional theory D, which we may omit when it is clear from the context. We give simple examples for the dependency relation: trivially $c_{\mathrm{tpOf}(c)} \rightsquigarrow \mathrm{tpOf}(c)$ holds for any constant c whose type is not built-in. (Note that all built-in constants have a built-in type.) Assume a theory that defines a constant $\mathsf{map}_{(\alpha\Rightarrow\beta)\Rightarrow\alpha\ \mathsf{list}\Rightarrow\beta\ \mathsf{list}}$ in terms of another constant fold_σ. Both $\mathsf{map} \rightsquigarrow \alpha\ \mathsf{list}$ and $\mathsf{map} \rightsquigarrow \mathsf{fold}_\sigma$ hold.

Definition 2.4 [Type-Substitutive Closure] When R is a binary relation on $\mathsf{CInst}^\bullet \cup \mathsf{Type}^\bullet$, we write $R^\downarrow$ for the *type-substitutive closure* of R, defined as follows: for s', $t' \in \mathsf{CInst}^\bullet \cup \mathsf{Type}^\bullet$,

$$(s', t') \in R^\downarrow \quad \text{if} \quad \exists \rho \in \mathsf{TSubst}, (s, t) \in R : \ s' = \rho(s) \text{ and } t' = \rho(t)$$

Thus, the type-substitutive closure extends R to its image under arbitrary type substitutions.

We say that a binary relation R is *terminating* if there exists no infinite sequence $(a_i)_{i\in\mathbb{N}}$ such that $a_i \; R \; a_{i+1}$ for all $i \in \mathbb{N}$.

The previous definitions allow us to introduce well-formed definitional theories.

Definition 2.5 [Well-formed Definitional Theory] A definitional theory D is *well-formed* if it is orthogonal and the type-substitutive closure of its dependency relation $\rightsquigarrow_D$ is terminating.

A main result of [6] is that each well-formed definitional theory has a model.

2.3 Models of Definitional Theories

We now define the semantics of constant instance and type definitions, as well as a semantics of polymorphic HOL (called *ground, fragment-localized semantics* in [6]) that interprets ground types as sets, and ground terms as elements of these sets.

A *formula* is a term of type bool. We let Fmla, ranged over by φ, denote the set of formulas. We assume that the usual formula connectives and quantifiers are present in the signature, and that they are defined in the standard way (cf. [10, §2.4.2]), starting from the built-in constants implication and equality.

Definition 2.6 [Semantics of $\equiv$]

- A constant instance definition $c_\sigma \equiv t$ stands for the formula $c_\sigma \doteq t$.
- A type definition $\tau \equiv t$ stands for the formula

$$
\begin{array}{l}
(\exists x_\sigma.\ t\ x) \rightarrow \\
\quad \exists rep_{\tau\Rightarrow\sigma}.\ \exists abs_{\sigma\Rightarrow\tau}. \\
\qquad (\forall y_\tau.\ t\ (rep\ y)) \wedge (\forall y_\tau.\ abs\ (rep\ y) \doteq y) \wedge (\forall x_\sigma.\ t\ x \rightarrow rep\ (abs\ x) \doteq x).
\end{array}
$$

As expected, a constant instance definition equates the constant instance with the defining term. A type definition asserts the existence of an isomorphism between the type τ and the subset of σ that is given by the predicate t, provided this subset is non-empty.

For a set of types $T \subseteq \mathsf{Type}$ let $\mathsf{Cl}(T)$ denote the *built-in closure* of T, defined inductively as the smallest set such that

- $T \cup \{\mathsf{bool}, \mathsf{ind}\} \subseteq \mathsf{Cl}(T)$, and
- $\sigma_1 \Rightarrow \sigma_2 \in \mathsf{Cl}(T)$ whenever $\sigma_1, \sigma_2 \in \mathsf{Cl}(T)$.

Thus, $\mathsf{Cl}(T)$ contains those types that can be constructed from T by repeated application of built-in type constructors.

Recall that $\mathsf{GType}^\bullet$ is the set of ground non-built-in types, and $\mathsf{GCInst}^\bullet$ is the set of ground non-built-in constant instances. We first interpret (fragments of) these sets, before extending the semantics to built-in types and terms as well as polymorphic (i. e. non-ground) formulas.

Definition 2.7 [Signature Fragment] A *(signature) fragment* is a pair (T, C), with $T \subseteq \mathsf{GType}^\bullet$ and $C \subseteq \mathsf{GCInst}^\bullet$, such that $\sigma \in \mathsf{Cl}(T)$ for all $c_\sigma \in C$.

When $F = (T, C)$ is a fragment, we write $\mathsf{Type}^F := \mathsf{Cl}(T)$ for the types generated by this fragment, $\mathsf{Term}^F := \{t \in \mathsf{Term} \mid \mathrm{types}^\bullet(t) \subseteq T \wedge \mathrm{consts}^\bullet(t) \subseteq C\}$ for the terms that fall within this fragment, and $\mathsf{Fmla}^F := \mathsf{Fmla} \cap \mathsf{Term}^F$ for its formulas.

The significance of fragments lies in the fact that Term^F is closed under taking subterms, and that $\mathrm{tpOf}(t) \in \mathsf{Type}^F$ for each $t \in \mathsf{Term}^F$. These properties allow us to define semantic interpretations for fragments, independent of the interpretation of terms or types outside the fragment.

We fix a two-element set $\mathbb{B} := \{\mathsf{true}, \mathsf{false}\}$, and we assume a global choice function, choice, such that $\mathsf{choice}(A) \in A$ for each non-empty set A. (We will use these to define the semantics of the built-in type bool and the built-in constant some, respectively.) We also fix a singleton set $\{\star\}$.

Definition 2.8 [Fragment Interpretation] Let $F = (T, C)$ be a fragment. We define an *F-interpretation* as a pair of families $\mathcal{I} = ((\![\tau]\!])_{\tau \in T}, ([c_\sigma])_{c_\sigma \in C})$ that satisfy two conditions:

(i) For all $\tau \in T$, the set $[\tau]$ is non-empty.

We extend the family $(\![\tau]\!])_{\tau \in T}$ to a family $(\![\tau]\!])_{\tau \in \mathsf{Cl}(T)}$ by interpreting the built-in type constructors as expected: $[\![\mathsf{bool}]\!] := \mathbb{B}$, $[\![\mathsf{ind}]\!] := \mathbb{N}$, [3] and $[\![\sigma \Rightarrow \tau]\!] := [\![\sigma]\!] \to [\![\tau]\!]$, i. e. the set of functions from $[\![\sigma]\!]$ to $[\![\tau]\!]$.

(ii) For all $c_\sigma \in C$, we have $[c_\sigma] \in [\![\sigma]\!]$.

Let GBI^F denote the set of ground built-in constant instances whose type is in Type^F. We extend the family $([c_\sigma])_{c_\sigma \in C}$ to a family $([c_\sigma])_{c_\sigma \in C \cup \mathsf{GBI}^F}$ by interpreting the built-in constants as expected (cf. [6, §4.4]): e. g. $[\![\to_{\mathsf{bool} \Rightarrow \mathsf{bool} \Rightarrow \mathsf{bool}}]\!]$ as logical implication, $[\![\doteq_{\tau \Rightarrow \tau \Rightarrow \mathsf{bool}}]\!]$ as the equality relation on $[\![\tau]\!]$, etc.

In other words, by interpreting the built-in type constructors and constants as expected, each F-interpretation $\mathcal{I} = ((\![\tau]\!])_{\tau \in T}, ([c_\sigma])_{c_\sigma \in C})$ gives rise to a pair of extended families $\big((\![\tau]\!]_{\tau \in \mathsf{Cl}(T)}), ([c_\sigma])_{c_\sigma \in C \cup \mathsf{GBI}^F}\big)$. We may write $[\![\tau]\!]^{\mathcal{I}}$ or $[\![\tau]\!]^{F,\mathcal{I}}$ (and similarly for constant instances) to indicate the interpretation $\mathcal{I}$ and the fragment F.

Given an F-interpretation $\mathcal{I}$, we call a function $\xi \colon \mathsf{Var} \times \mathsf{Type}^F \to \bigcup_{\sigma \in \mathsf{Type}^F} [\![\sigma]\!]^{\mathcal{I}}$ a *valuation for $\mathcal{I}$* if each variable is mapped to a value of its type, i. e. if $\xi(x_\sigma) \in [\![\sigma]\!]^{\mathcal{I}}$ for each $x_\sigma \in \mathsf{Var} \times \mathsf{Type}^F$. Let $\mathsf{Val}^{\mathcal{I}}$ denote the set of valuations for $\mathcal{I}$.

With this setup we can interpret the terms in Term^F according to an F-interpretation $\mathcal{I}$. Let $t \in \mathsf{Term}^F$ be a term that falls within the fragment. We recursively define the function $[t] \colon \mathsf{Val}^{\mathcal{I}} \to [\![\mathrm{tpOf}(t)]\!]$ in the expected way over the structure of terms:

$$
\begin{aligned}
[x_\sigma](\xi) &:= \xi(x_\sigma) \\
[c_\sigma](\xi) &:= [c_\sigma] \\
[t_1\ t_2](\xi) &:= [t_1](\xi)([t_2](\xi)) \\
[\lambda x_\sigma.\ t](\xi) &\colon [\![\sigma]\!] \to [\![\mathrm{tpOf}(t)]\!], a \mapsto [t](\xi(x_\sigma \leftarrow a))
\end{aligned}
$$

[3] As noted in [6], any infinite set would do here.

In the last clause, $\xi(x_\sigma \leftarrow a)$ denotes a function that takes the value a at x_σ and otherwise agrees with ξ. Kunčar and Popescu motivate the correctness of this recursive definition [6, Lemma 8.(5)].

For closed terms t, the interpretation does not depend on the particular valuation: the function $[t]$ is constant. We now restrict our attention to closed formulas.

For $\varphi \in \mathsf{Fmla}^F$ a ground formula, we say that $\mathcal{I}$ is a *model* of φ, written $\mathcal{I} \models \varphi$, if $[\varphi]^{\mathcal{I}} = \mathsf{true}$. We call the fragment $\top := (\mathsf{GType}^\bullet, \mathsf{GCInst}^\bullet)$ the *total fragment*. If $\mathcal{I}$ is a $\top$-interpretation and φ is a polymorphic formula, we say that $\mathcal{I}$ is a *model* of φ, also written $\mathcal{I} \models \varphi$, if $\mathcal{I} \models \rho(\varphi)$ for all ground type substitutions $\rho \in \mathsf{GTSubst}$. We extend this notion to sets of formulas, by defining $\mathcal{I} \models S$ as $\mathcal{I} \models \varphi$ for all $\varphi \in S$.

3 Results

A main result of Kunčar and Popescu is that each well-formed definitional theory has a model; more precisely, each well-formed definitional theory D has a $\top$-interpretation $\mathcal{I}$ such that $\mathcal{I} \models D$ [6, Theorem 11]. Consequently, each such theory is consistent [6, Theorem 6]. We strengthen and generalize these results.

3.1 Model-theoretic Conservativity

To establish model-theoretic conservativity for definitional theories, we need to show that whenever a well-formed definitional theory D is extended to another well-formed definitional theory $D' := D \cup \{u \equiv t\}$, then any model $\mathcal{M}$ of D can be expanded to a model $\mathcal{M}'$ of D'.

Note that we assume D and D' to be theories over the same signature. Therefore, a priori it is not entirely clear what it means here to expand $\mathcal{M}$. In particular, $\mathcal{M}$ already interprets u and t. Unless $\mathcal{M} \models u \equiv t$, the model $\mathcal{M}'$ will need to interpret at least some type or constant instance differently.

Moreover, the counterexample given in Section 1 demonstrates that it is in general not sufficient to alter the interpretation of u in $\mathcal{M}'$. There may be other types or constant instances that depend on u (or on one of its type instances), and their interpretations may need to be adjusted as well. Our main result is that these are the *only* types and constant instances whose interpretation may need to change.

When R is a binary relation, we write R^* for the reflexive-transitive closure of R.

Lemma 3.1 *Let D be a definitional theory, and let $u \in \mathsf{CInst}^\bullet \cup \mathsf{Type}^\bullet$. We write V_u for the pre-image of type instances of u under the reflexive-transitive, type-substitutive closure of the dependency relation $\rightsquigarrow_D$, i. e.*

$$V_u := \left\{ v \in \mathsf{CInst}^\bullet \cup \mathsf{Type}^\bullet \mid \exists \rho \in \mathsf{TSubst}.\ v \left(\rightsquigarrow^{\downarrow}\right)^* \rho(u) \right\}.$$

Then $F_u := (\mathsf{GType}^\bullet \setminus V_u, \mathsf{GCInst}^\bullet \setminus V_u)$ is a fragment.

Proof. Let $c_\sigma \in \mathsf{GCInst}^\bullet \setminus V_u$ be a ground constant instance of type σ. Assume that $\sigma \notin \mathsf{Cl}(\mathsf{GType}^\bullet \setminus V_u)$. Then there must be a type $\tau \in \mathrm{types}^\bullet(\sigma)$ such that $\tau \in V_u$.

But then $c_\sigma \rightsquigarrow^{\downarrow} \tau$ (cf. [6, Lemma 4.(2)]) and $\tau\ (\rightsquigarrow^{\downarrow})^*\ \rho(u)$ for some $\rho \in \mathsf{TSubst}$. Hence $c_\sigma\ (\rightsquigarrow^{\downarrow})^*\ \rho(u)$, contradicting $c_\sigma \notin V_u$. □

Definition 3.2 [Independent Fragment] In the situation of the preceding Lemma 3.1 we call F_u the *u-independent fragment.*

Theorem 3.3 (Model-theoretic Conservativity) *Let $\mathcal{M}$ be a model of a well-formed definitional theory D, i. e. $\mathcal{M} \models D$. Moreover, let $D' := D \cup \{u \equiv t\}$ be a well-formed extension of D. There exists a model $\mathcal{M}'$ of the extended theory D' with the following property: the models $\mathcal{M}$ and $\mathcal{M}'$ agree on the interpretation of all types and terms in $\mathsf{Type}^{F_u} \cup \mathsf{Term}^{F_u}$.*

In other words, $\mathcal{M}$ and $\mathcal{M}'$, as obtained from Theorem 3.3, agree on the interpretation of all (ground) types and terms that fall within the u-independent fragment.

Proof. Since all models interpret the built-in type constructors and constants in the same way, it suffices to construct a model $\mathcal{M}'$ of D' that agrees with $\mathcal{M}$ on the interpretation of (non-built-in, ground) types and constant instances in the fragment F_u. As before, let $V_u := \left\{ v \in \mathsf{CInst}^\bullet \cup \mathsf{Type}^\bullet \mid \exists \rho \in \mathsf{TSubst}.\ v\ (\rightsquigarrow^{\downarrow})^*\ \rho(u) \right\}$.

Hence for $v \in (\mathsf{GType}^\bullet \setminus V_u) \cup (\mathsf{GCInst}^\bullet \setminus V_u)$, we define $[v]^{\mathcal{M}'} := [v]^{\mathcal{M}}$. This yields an interpretation for the fragment F_u where $\mathcal{M}'$ agrees with $\mathcal{M}$ by definition.

It remains to define $[v]^{\mathcal{M}'}$ for $v \in (\mathsf{GType}^\bullet \cup \mathsf{GCInst}^\bullet) \cap V_u$. We proceed by well-founded recursion over $(\rightsquigarrow^{\downarrow})^+$, the transitive closure of $\rightsquigarrow^{\downarrow}$, restricted to the set $(\mathsf{GType}^\bullet \cup \mathsf{GCInst}^\bullet) \cap V_u$. (Note that $(\rightsquigarrow^{\downarrow})^+$ is a terminating relation because the theory is well-formed.) This is analogous to the model construction in [6]. Assume the interpretation $[w]^{\mathcal{M}'}$ has been defined for all $w \in \mathsf{GType}^\bullet \cup \mathsf{GCInst}^\bullet$ with $v\ (\rightsquigarrow^{\downarrow})^+\ w$.

We consider two cases. We say a definition $(s \equiv r) \in D'$ *matches* v if there is a type substitution $\rho \in \mathsf{TSubst}$ such that $v = \rho(s)$, meaning v is a type instance of s. In this case, we note that the matching definition is uniquely determined (due to orthogonality), and that $[\rho(r)]^{\mathcal{M}'}$ is defined by recursion. (Strictly speaking, only $[\rho(r)]^{F,I}$ is defined, where F is a sufficiently large fragment with $\rho(r) \in \mathsf{Term}^F$, and I is the restriction of $\mathcal{M}'$ to F. However, the two values coincide [6, Lemma 9].) Following the model construction in [6], we define $[v]^{\mathcal{M}'}$ such that $\mathcal{M}' \models v \equiv \rho(r)$.

On the other hand, if there is no matching definition for v in D', then $v \in V_u$ implies that v must be a constant instance, i. e. $v = c_\sigma$ for some $c_\sigma \in \mathsf{GCInst}^\bullet$. (Note that undefined types do not depend on other types or constant instances.) Again following the model construction in [6], we define $[v]^{\mathcal{M}'} := \mathsf{choice}([\sigma]^{\mathcal{M}'})$.

Since $\mathcal{M}$ is a $\top$-interpretation, it is immediate that $\mathcal{M}'$ is a $\top$-interpretation. It remains to show that $\mathcal{M}' \models D'$. Let $(s \equiv r) \in D'$, and let $\rho \in \mathsf{GTSubst}$ be a ground type substitution. Clearly $s \equiv r$ is a (and due to orthogonality, the unique) matching definition for $\rho(s)$. Hence if $\rho(s) \in V_u$, then by the first case above, $[\rho(s)]^{\mathcal{M}'}$ is defined such that $\mathcal{M}' \models \rho(s) \equiv \rho(r)$. On the other hand, if $\rho(s) \notin V_u$, then $\mathsf{types}^\bullet(\rho(r)) \subseteq \mathsf{GType}^\bullet \setminus V_u$ and $\mathsf{consts}^\bullet(\rho(r)) \subseteq \mathsf{GCInst}^\bullet \setminus V_u$, hence $\rho(r) \in \mathsf{Term}^{F_u}$. Therefore $[\rho(s)]^{\mathcal{M}'} = [\rho(s)]^{\mathcal{M}}$ and $[\rho(r)]^{\mathcal{M}'} = [\rho(r)]^{\mathcal{M}}$. More-

over, $\rho(s) \notin V_u$ implies $s \neq u$. Hence $(s \equiv r) \in D$ in this case, and $\mathcal{M}' \models \rho(s) \equiv \rho(r)$ follows from $\mathcal{M} \models D$. □

Our approach expands any existing model $\mathcal{M}$ of a well-formed definitional theory D to a model of the well-formed theory extension D'. A model of the extended theory is constructed by recursion over part of the $\rightsquigarrow^{\downarrow}$ relation, based on the model $\mathcal{M}$. In contrast, the model construction in [6] obtains a model from the ground up, by recursion over the entire $\rightsquigarrow^{\downarrow}$ relation, without reference to any previous interpretation.

As a consequence of the previous theorem we obtain the result [6, Theorem 11]:

Corollary 3.4 *Any well-formed definitional theory has a model.*

Proof. Let D be a well-formed definitional theory. Clearly, the empty theory has a model. (This can be obtained, e. g. by interpreting non-built-in types as $\{\star\}$, and non-built-in constant instances using choice, cf. the model construction given in [6].) Moreover, every subset of D is a well-formed definitional theory. The corollary immediately follows by induction over $|D|$, which is finite. □

Since the deduction rules of polymorphic HOL are sound w. r. t. our semantics [6, Theorem 10], the existence of a model immediately implies the (syntactic) consistency of well-formed definitional theories [6, Theorem 6].

Corollary 3.5 *Any well-formed definitional theory is consistent.*

4 Related Work

Extensions by definitions have a long history. In 1967, Shoenfield [12, §4.6] discussed two definitional mechanisms for extensions of theories in (untyped) first-order logic by predicate and function symbols. A new symbol is defined by an equivalence or equality whose right-hand side must not contain the new symbol. For function symbols, the definition must be accompanied by a proof that this equality describes a function in the mathematical sense. Both mechanisms are shown to be proof-theoretic conservative. Moreover, each model of the original theory has a *unique* expansion that is a model of the extended theory; this immediately implies model-theoretic conservativity.

In 1997, Wenzel [13] discussed the theoretical foundation for overloaded definitions and type classes in higher-order logic. He "consider[s] syntactic conservativity as a minimum requirement for well-behaved extension mechanisms within purely deductive logical frameworks." Additionally Wenzel introduces *realisability*, which formalises the intuition that constant definitions can be unfolded, and shows that overloaded constant definitions are both conservative and realisable. However, he assumes that all instances of an overloaded constant are defined at once, and he does not consider the interplay of overloading with type definitions (cf. the example in Section 1).

In 2006, Obua [11] noted that to avoid inconsistencies, the process of unfolding definitions must terminate. He shows that for overloaded definitions that recurse

through types, termination is not semi-decidable in general. Obua considers both type and constant definitions and gives a proof sketch that overloading in Isabelle is conservative, but he misses that dependencies through types may introduce inconsistencies.

Most closely related is the already mentioned work by Kunčar and Popescu [6], who in 2015 introduced definitional theories for Isabelle and show that they preserve consistency, i. e. every well-formed definitional theory has a model. In [7], the same authors prove syntactic consistency (i. e. False is not derivable) of definitional theories by a proof-theoretic argument. They introduce a richer logic, HOL with comprehension types (HOLC), into which they encode formulas of HOL by unfolding type and constant definitions. This encoding preserves derivability. The consistency of definitional theories then follows from the consistency of HOLC.

In a recent work [8], Kunčar and Popescu show a much stronger result: using a different unfolding approach that relativises formulas involving defined types to a predicate on the host type (and thus stays within the language of HOL), they establish proof-theoretic conservativity of definitional theories over minimal HOL, i. e. relative to an empty theory that contains no axioms. In contrast, the present paper proves model-theoretic conservativity relative to arbitrary (well-formed) definitional theories.

Other theorem provers for higher-order logic, e. g. HOL4 [10, §2.5.2], implement a more general mechanism for *constant specification*. This mechanism, which in its current form was suggested by Arthan [2], allows implicit definitions. It takes as input a theorem of the form $v_1 \doteq t_1, \ldots, v_n \doteq t_n \vdash P$ (where the v_i are variables) and introduces new constants $c_1, \ldots, c_n$ with $P[c_1/v_1, \ldots, c_n/v_n]$ as their defining axiom. Conventional constant definitions $c \equiv t$ are recovered as a special case when P is of the form $v \doteq t$. Constant specification is proof- and model-theoretic conservative [3], and has been formalised and verified using HOL4 by Kumar et al. [4]. However, in contrast to Isabelle, which supports *ad hoc* overloading natively in its logic, other theorem provers for higher-order logic offer support for overloading only as syntactic sugar, through extensions of parsing and pretty-printing.

5 Conclusion

We defined a notion of model expansion that is suitable for definitional theories, and we showed that extensions of definitional theories are model-theoretic conservative with respect to this notion. This strengthens and generalizes an earlier consistency result for definitional theories [6]. We have thereby established an important property of the definitional mechanisms that are implemented in the theorem prover Isabelle.

Model-theoretic conservativity has a proof-theoretic (syntactic) counterpart. Roughly, an extension is *proof-theoretic conservative* if it entails no new theorems in the original language. In other words, every formula of the original language that is a theorem in the extension is already provable in the original theory. Adapting this notion to definitional theories, we conjecture that if D' is an extension of D such

that $D' \vdash \varphi$, where φ is a formula that does not contain any constant instance or type that depends on definitions in $D' \setminus D$, then $D \vdash \varphi$.

For logics that have a sound and complete deductive system, model-theoretic conservativity implies proof-theoretic conservativity: suppose $D' \vdash \varphi$. By completeness it suffices to show that φ holds in all models of D. Let $\mathcal{M}$ be a model of D. By model-theoretic conservativity, $\mathcal{M}$ can be expanded to a model $\mathcal{M}'$ of D' that agrees with $\mathcal{M}$ on the interpretation of φ. Since $D' \vdash \varphi$, soundness implies that $\mathcal{M}'$ is a model of φ. Hence $\mathcal{M}$ is a model of φ.

Unfortunately, this argument does not immediately apply to higher-order logic, which is not complete with respect to its standard semantics [10, §2.4.5]. However, higher-order logic *is* complete with respect to non-standard (Henkin) semantics [1, §55]. By adapting the completeness proof to the variant of higher-order logic implemented in Isabelle and to the novel (ground, fragment-localized) semantics of polymorphic types suggested in [6], it may be possible to derive proof-theoretic conservativity for extensions of definitional theories from their model-theoretic conservativity. It would also be interesting to study the connection between the ground, fragment-localized semantics and the traditional set-theoretic semantics of higher-order logic in more detail. We leave this to future work.

References

[1] Andrews, P. B., "An Introduction to Mathematical Logic and Type Theory: To Truth through Proof," Number 27 in Applied logic series, Kluwer Academic Publishers, Dordrecht ; Boston, 2002, 2nd ed edition.

[2] Arthan, R., *HOL Constant Definition Done Right*, in: *Interactive Theorem Proving* (2014), pp. 531–536.
URL http://dx.doi.org/10.1007/978-3-319-08970-6_34

[3] Arthan, R., *On Definitions of Constants and Types in HOL*, Journal of Automated Reasoning **56** (2016), pp. 205–219.
URL http://dx.doi.org/10.1007/s10817-016-9366-4

[4] Kumar, R., R. Arthan, M. O. Myreen and S. Owens, *HOL with Definitions: Semantics, Soundness, and a Verified Implementation*, in: *Interactive Theorem Proving* (2014), pp. 308–324.
URL http://dx.doi.org/10.1007/978-3-319-08970-6_20

[5] Kunčar, O., *Correctness of Isabelle's Cyclicity Checker: Implementability of Overloading in Proof Assistants*, in: *Proceedings of the 2015 Conference on Certified Programs and Proofs*, CPP '15 (2015), pp. 85–94.
URL http://doi.acm.org/10.1145/2676724.2693175

[6] Kunčar, O. and A. Popescu, *A Consistent Foundation for Isabelle/HOL*, in: C. Urban and X. Zhang, editors, *Interactive Theorem Proving*, number 9236 in Lecture Notes in Computer Science, Springer International Publishing, 2015 pp. 234–252.
URL http://dx.doi.org/10.1007/978-3-319-22102-1_16

[7] Kunčar, O. and A. Popescu, *Comprehending Isabelle/HOL's Consistency*, in: H. 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*, Lecture Notes in Computer Science **10201** (2017), pp. 724–749.
URL https://doi.org/10.1007/978-3-662-54434-1_27

[8] Kunčar, O. and A. Popescu, *Safety and Conservativity of Definitions in HOL and Isabelle/HOL*, Proc. ACM Program. Lang. **2** (2017), pp. 24:1–24:26.
URL http://doi.acm.org/10.1145/3158112

[9] Nipkow, T., L. C. Paulson and M. Wenzel, "Isabelle/HOL - A Proof Assistant for Higher-Order Logic," Number 2283 in Lecture Notes in Computer Science, Springer, 2002.
URL https://doi.org/10.1007/3-540-45949-9

[10] Norrish, M. and K. Slind, *The HOL System LOGIC* (2014).
URL http://downloads.sourceforge.net/project/hol/hol/kananaskis-10/kananaskis-10-logic.pdf

[11] Obua, S., *Checking Conservativity of Overloaded Definitions in Higher-Order Logic*, in: *Term Rewriting and Applications* (2006), pp. 212–226.
URL http://dx.doi.org/10.1007/11805618_16

[12] Shoenfield, J. R., "Mathematical Logic," A.K. Peters, Natick, Mass, 1967.

[13] Wenzel, M., *Type classes and overloading in higher-order logic*, in: E. L. Gunter and A. Felty, editors, *Theorem Proving in Higher Order Logics*, number 1275 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1997 pp. 307–322.
URL http://dx.doi.org/10.1007/BFb0028402