Mechanisation of Model-theoretic Conservative Extension for HOL with Ad-hoc Overloading

Arve Gengelbach
Uppsala University, Uppsala, Sweden
arve.gengelbach@it.uu.se

Johannes Åman Pohjola
CSIRO's Data61, Sydney, Australia
University of New South Wales, Sydney, Australia
johannes.amanpohjola@data61.csiro.au

Tjark Weber
Uppsala University, Uppsala, Sweden
tjark.weber@it.uu.se

Definitions of new symbols merely abbreviate expressions in logical frameworks, and no new facts (regarding previously defined symbols) should hold because of a new definition. In Isabelle/HOL, definable symbols are types and constants. The latter may be ad-hoc overloaded, i. e. have different definitions for non-overlapping types. We prove that symbols that are independent of a new definition may keep their interpretation in a model extension. This work revises our earlier notion of model-theoretic conservative extension and generalises an earlier model construction. We obtain consistency of theories of definitions in higher-order logic (HOL) with ad-hoc overloading as a corollary. Our results are mechanised in the HOL4 theorem prover.

1 Introduction

Isabelle/HOL enriches higher-order logic with ad-hoc overloading. While other theorem provers of the HOL family support overloaded syntax through enhancements of parsing and pretty printing, in Isabelle/HOL overloading is a feature of the logic. The user-defined symbols are types and constants, and in Isabelle/HOL the latter may have multiple definitions for non-overlapping types. For instance, $+_{\alpha\to\alpha\to\alpha}$ is an overloaded constant with different definitions for different type instances of commutative monoids such as the natural numbers or the integers.

Overloaded definitions need further care as the defined symbols may be used prior to their definition, which if treated improperly may lead to cyclic definitions, i. e. unfolding of definitions might not terminate.

For a logic to be useful it should have unprovable statements. A logic is *consistent* if a contradiction cannot be deduced from any of its theories. HOL with user-defined types and constants without overloading is consistent. This can be proved by an argument based on *standard semantics* [16], where Booleans, function types and the equality constant are interpreted as expected, and type variables are interpreted as elements of a fixed universe of sets. The consistency story for HOL with overloading is a long one [20, 15, 12, 17]. Åman Pohjola and Gengelbach [17] prove HOL with overloading consistent in a machine-checked proof, by constructing models of theories of definitions through a construction that originates from Kunčar and Popescu [12]. Kunčar and Popescu introduce a *dependency relation* between the symbols of a theory to track dependencies of defined symbols on their definiens. Under an additional syntactic restriction on overloading [10], they construct a model for any (finite) theory of definitions for which the dependency relation is terminating.

C. Sacerdoti Coen, A. Tiu (Eds.): 15th International Workshop on Logical Frameworks: Theory and Practice (LFMTP 2020)
EPTCS 332, 2021, pp. 1–17, doi:10.4204/EPTCS.332.1

Apart from consistency, a definitional mechanism should be *model-theoretically conservative*: any model of a theory can be extended to a model of the extended theory with new definitions, keeping interpretations of formulae that are independent of the new symbols intact. Informally, at least symbols that are independent of a theory extension may keep their interpretation in a model extension.

For HOL without overloading, model-theoretic conservativity holds unconditionally [8]. With overloading, model-theoretic conservativity holds for symbols that are independent of new definitions, as Gengelbach and Weber prove [4]. However, their proof was based on inherited wrong assumptions from Kunčar and Popescu [12], that Åman Pohjola and Gengelbach in their mechanised model construction [17] uncover and correct. Additionally, the mechanisation supports theory extension by the more expressive constant specification [2], which is a definitional mechanism also used in the theorem provers ProofPower and HOL4 to simultaneously introduce several new constants that satisfy some property.

This paper joins these two lines of work in mechanising that the definitional mechanisms of types and overloaded constants are model-theoretically conservative. The result holds for models that interpret constants introduced by constant specification equal to their witnesses and replaces the earlier monolithic model construction with an iterative one. An interpretation of this result is that the definitional mechanisms of Isabelle/HOL are semantically speaking robustly designed: at least symbols that are independent of an update may keep their interpretation in a model extension.

Even more generally, the syntactic counterpart of model-theoretic conservativity, *proof-theoretic (syntactic) conservativity* shall hold [20]. Informally, a definitional mechanism is *proof-theoretically conservative* if the definitional extension entails no new properties, except those that depend on the symbols which the extension introduces.

For Isabelle/HOL conservativity has been studied in an absolute manner, i. e. any definitional theory is a conservative extension of *initial HOL* [11], the theory of Booleans with Hilbert-choice and infinity axiom. Gengelbach and Weber [5] prove conservativity of any definitional extension above initial HOL, by translating a model-theoretic conservativity for a generalised semantics into its syntactic counterpart. As their semantics are similar to ours, this paper adds to the reliability of their result.

We describe the syntax and (lazy ground) semantics of HOL with ad-hoc overloading in Section 2. Subsequently, in Section 3 we recapitulate the *independent fragment* as the part of a theory that is independent of an extension by a new definition. This fragment is crucial in the iterative model construction, i. e. model-theoretic conservativity in Section 4. We discuss related work in Section 5. The definitions and theorems in this paper are formalised in the HOL4 theorem prover as part of the CakeML project.[1]

Contributions We make the following contributions

- We adapt and formalise the previously introduced *independent fragment* [4] (i. e. a theory's syntax fragment that is independent of a theory update) to support a more general definitional mechanism for constants: *constant specification* [2].
- We use the independent fragment to prove a notion of model-theoretic conservativity [4] in a new setting for the *lazy ground semantics* [17], which delays type variable instantiation and does not instantiate the type of term variables. To the best of our knowledge, this is the first mechanised conservativity result for a logic with overloaded definitions.
- Our work generalises and replaces the earlier monolithic model construction of Åman Pohjola and Gengelbach [17], and obtains consistency of HOL with ad-hoc overloading as a corollary.

[1] `https://code.cakeml.org/tree/master/candle/overloading/semantics/`

2 Background

In this section we introduce the syntax and semantics of HOL with ad-hoc overloading, which we inherit from the earlier work of Åman Pohjola and Gengelbach [17]. Their formalisation makes use of infrastructure from the formalisation of HOL Light (without overloading) by Kumar et al. [9], and the theoretical work on the consistency of HOL with ad-hoc overloading by Kunčar and Popescu [12].

2.1 Types and terms

Types Types, described by the grammar $\mathsf{type} = \mathsf{Tyvar\ string} \mid \mathsf{Tyapp\ string\ (type\ list)}$, are rank-1 polymorphic. Type variables Tyvar can be instantiated by a type substitution (ranged over by Θ), which extends homomorphically to type constructors Tyapp. For a type ty and a type substitution Θ, we call $\Theta\ ty$ a *(type) instance* of ty, denoted by $ty \geq \Theta\ ty$. Two types ty_1 and ty_2 are *orthogonal*, denoted by $ty_1 \# ty_2$, if they have no common instance. *Ground* types are those that contain no type variables, hence remain unchanged under any type substitution. A type substitution is *ground* if it maps every type to a ground type. As ground types only have trivial instances, any two ground types are either equal or orthogonal.

Terms Terms are simply typed λ-expressions, described by the grammar

$$\mathsf{term} = \mathsf{Var\ string\ type} \mid \mathsf{Const\ string\ type} \mid \mathsf{Comb\ term\ term} \mid \mathsf{Abs\ term\ term}$$

We only consider well-formed terms, that is, λ-abstractions must be of the form $\mathsf{Abs}\ (\mathsf{Var}\ x\ ty)\ t$, i. e. have a term variable as first argument representing the binder. A closed term t, denoted $\mathsf{closed}\ t$, contains only bound term variables. A term is welltyped if it has a type by the following rules (wherein $\to$ abbreviates the later introduced function type):

$$\frac{}{\mathsf{Var}\ n\ ty\ \mathsf{has\_type}\ ty} \qquad \frac{}{\mathsf{Const}\ n\ ty\ \mathsf{has\_type}\ ty}$$

$$\frac{s\ \mathsf{has\_type}\ (dty \to rty) \quad t\ \mathsf{has\_type}\ dty}{\mathsf{Comb}\ s\ t\ \mathsf{has\_type}\ rty} \qquad \frac{t\ \mathsf{has\_type}\ rty}{\mathsf{Abs}\ (\mathsf{Var}\ n\ dty)\ t\ \mathsf{has\_type}\ (dty \to rty)}$$

A well-typed term tm has a unique type which we denote $\mathsf{typeof}\ tm$. Applying a type substitution Θ to a term means to apply Θ to the types within, e. g. $\Theta\ (\mathsf{Const}\ c\ ty) = \mathsf{Const}\ c\ (\Theta\ ty)$ (which we call *constant instance*) for a constant $\mathsf{Const}\ c\ ty$. Orthogonality extends from types to constant instances:

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

A user may introduce (non-built-in) types and constants by theory extension, as described in Section 2.3. For types and constants we generally say *symbols*.

Built-ins We abbreviate

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

Any type with any of these type constructors at the top level is *built-in*, as is the constant Equal. These are the only symbols which are not user-defined. A *formula* is a term of type Bool.

For a set of types *tys* we consider its *built-in closure*, written builtin_closure *tys*:

$$\frac{}{\mathsf{Bool} \in \mathsf{builtin\_closure}\ tys} \qquad \frac{ty \in tys}{ty \in \mathsf{builtin\_closure}\ tys} \qquad \frac{ty_1 \in \mathsf{builtin\_closure}\ tys \quad ty_2 \in \mathsf{builtin\_closure}\ tys}{(ty_1 \to ty_2) \in \mathsf{builtin\_closure}\ tys}$$

Non-built-ins We define operators to collect the non-built-in types of terms and types, and also the non-built-in constants of terms. The list $x^\bullet$ consists of the outermost non-built-in types of a type or term x.

$$\begin{aligned}
&\mathsf{Bool}^\bullet \stackrel{\mathsf{def}}{=} [] \\
&(dom \to rng)^\bullet \stackrel{\mathsf{def}}{=} dom^\bullet \mathbin{+\!\!+} rng^\bullet \\
&ty^\bullet \stackrel{\mathsf{def}}{=} [ty] \quad \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
&(\mathsf{Var}\ v_0\ ty)^\bullet \stackrel{\mathsf{def}}{=} ty^\bullet \\
&(\mathsf{Const}\ v_1\ ty)^\bullet \stackrel{\mathsf{def}}{=} ty^\bullet \\
&(\mathsf{Comb}\ a\ b)^\bullet \stackrel{\mathsf{def}}{=} a^\bullet \mathbin{+\!\!+} b^\bullet \\
&(\mathsf{Abs}\ a\ b)^\bullet \stackrel{\mathsf{def}}{=} a^\bullet \mathbin{+\!\!+} b^\bullet
\end{aligned}$$

As an example, the outermost non-built-in types of $map_{(\alpha \to \mathsf{Bool}) \to \alpha\ \mathsf{list} \to \mathsf{Bool\ list}}$ over a polymorphic unary list type α list are:

$$\begin{aligned}
&(\mathsf{Const}\ \text{«map»}\ ((\mathsf{Tyvar}\ \alpha \to \mathsf{Bool}) \to (\mathsf{Tyvar}\ \alpha)\ \mathsf{list} \to \mathsf{Bool\ list}))^\bullet = \\
&\ [\mathsf{Tyvar}\ \alpha;\ (\mathsf{Tyvar}\ \alpha)\ \mathsf{list};\ \mathsf{Bool\ list}]
\end{aligned}$$

Any type *ty* can be recovered from built-in types and the type's outermost non-built-in types:

$$\forall ty.\ ty \in \mathsf{builtin\_closure}\ (ty^\bullet)$$

For terms t, we define the list t° to contain all non-built-in constants of t:

$$\begin{aligned}
&(\mathsf{Comb}\ a\ b)^\circ \stackrel{\mathsf{def}}{=} a^\circ \mathbin{+\!\!+} b^\circ \\
&(\mathsf{Abs}\ _\ a)^\circ \stackrel{\mathsf{def}}{=} a^\circ
\end{aligned}$$

$$\begin{aligned}
&(\mathsf{Var}\ x\ ty)^\circ \stackrel{\mathsf{def}}{=} [] \\
&(\mathsf{Equal}\ ty)^\circ \stackrel{\mathsf{def}}{=} [] \\
&(\mathsf{Const}\ c\ ty)^\circ \stackrel{\mathsf{def}}{=} [\mathsf{Const}\ c\ ty] \quad \text{otherwise}
\end{aligned}$$

2.2 Inference system

A *signature* is a pair of functions that assign type constructor names their corresponding arity and constant names their corresponding type. A *theory* is a pair (s,a) of a signature s and a set of terms (axioms) a. Gengelbach and Weber [4] consider a fixed signature, that is all symbols are initially declared, and a fixed set of axioms. Here instead, both the signature and the (possibly non-definitional) axioms may be extended (see Section 2.3). The functions axsof, tysof and tmsof return the respective components of a theory or signature.

Derivability of *sequents* is defined inductively as a ternary relation $(thy,hyps) \vdash p$ between a theory *thy*, a list of terms (hypotheses) *hyps* and a term (conclusion) p. We display three of the standard inference rules of higher-order logic, with their syntactic well-formedness constraints. The condition type_ok (tysof *ctxt*) *ty* requires that *ty* is either a type variable or a type constructor applied to the correct number of arguments, as indicated by its arity in the signature, and that these arguments are also type_ok. Similarly, term_ok (sigof *thy*) p requires that p is a well-typed term, and that its types and constants are instances from the given signature. Finally, theory_ok *ctxt* requires that in the context *ctxt* all axioms are well-formed formulae, the theory has well-typed types and contains at least the built-in symbols.

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

$$\frac{\mathsf{theory\_ok}\ thy \quad \mathsf{type\_ok}\ (\mathsf{tysof}\ thy)\ ty \quad \mathsf{term\_ok}\ (\mathsf{sigof}\ thy)\ t}{(thy,[]) \vdash \mathsf{Comb}\ (\mathsf{Abs}\ (\mathsf{Var}\ x\ ty)\ t)\ (\mathsf{Var}\ x\ ty) === t}\ \mathsf{ABS}$$

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

2.3 Theory extensions

A theory is obtained from the empty theory by incremental *updates*. A list of updates is a *context*, and the function thyof returns the context's theory.

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

NewAxiom adds its argument formula to the theory's set of axioms. NewType and NewConst are type and constant *declarations*; they extend the theory's signature. The remaining TypeDefn and ConstSpec are *definitions* of a type and of constants, respectively. Definitions may extend both the signature and the set of axioms, and we defer their discussion to Sections 2.4 and 2.5.

The updates relation specifies when an update is a valid extension of a context:

$$\frac{prop\ \mathsf{has\_type\ Bool} \quad \mathsf{term\_ok}\ (\mathsf{sigof}\ ctxt)\ prop}{\mathsf{NewAxiom}\ prop\ \mathsf{updates}\ ctxt} \qquad \frac{name \notin \mathsf{domain}\ (\mathsf{tmsof}\ ctxt) \quad \mathsf{type\_ok}\ (\mathsf{tysof}\ ctxt)\ ty}{\mathsf{NewConst}\ name\ ty\ \mathsf{updates}\ ctxt} \qquad \frac{name \notin \mathsf{domain}\ (\mathsf{tysof}\ ctxt)}{\mathsf{NewType}\ name\ arity\ \mathsf{updates}\ ctxt}$$

The rule for NewAxiom requires that an axiom is a formula over the context's signature. The rule for NewConst requires that the constant's name is new for the context and that its type is from the context's signature. Similarly, the rule for NewType requires that the type name is new for the context.

The reflexive relation $ctxt_2$ extends $ctxt_1$ expresses that a context $ctxt_2$ is obtained from a context $ctxt_1$ by a sequence of updates. The context init_ctxt contains the built-ins, i. e. the types Bool and Fun and the equality constant. Its extension hol_ctxt also contains a type of individuals, the theory of Booleans, a Hilbert-choice constant with its characteristic axiom, and the axioms of extensionality and infinity.

2.4 Type definitions

A type definition TypeDefn *name pred abs rep* introduces a new type constructor *name* defined by its characteristic, closed predicate *pred* as a subset of a host type. It makes available the type Tyapp *name l* where the argument list *l* corresponds to the distinct type variables of *pred*. A proof that the predicate is satisfiable is a prerequisite, as in HOL types are non-empty. Additionally, abstraction and representation bijections between the new type and the subset of the host type are axiomatically introduced.

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

2.5 Constant specification

Constant specification ConstSpec *ov eqs prop* defines possibly several constants by one axiom *prop*. For $(c_i,t_i) \in eqs$, each of the constants c_i is introduced by a closed witness term t_i, that is, the predicate *prop* holds assuming all equalities

$$(thy, [\mathsf{Var}\ c_1\ (\mathsf{typeof}\ t_1) === t_1; \ldots; \mathsf{Var}\ c_n\ (\mathsf{typeof}\ t_n) === t_n]) \vdash prop.$$

Each of the variables Var c_i serves as a placeholder for Const c_i.

If the constant specification is marked as overloading, i. e. if *ov* is true, the mechanism allows to introduce instances of already declared constants. Non-overloading constant specifications need to introduce constants with fresh names.

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

Here VFREE_IN *x tm* denotes that *x* is a free term variable in *tm*. The predicate constspec_ok imposes two important restrictions on constant specifications: the context resulting from the update needs to be orthogonal (no two defined symbols have a common type instance), and any introduced overloading of previously declared constants must not allow cycles through the definitions. We discuss how the latter is avoided with a dependency relation and define orthogonality of contexts in Section 2.6.

Constant specification generalises the introduction of new constants via equational axioms, as considered in [4], by allowing implicit definitions.[2] For further discussion of its advantages we refer to [2].

2.6 Non-cyclic theories

Cycles in theories with overloaded symbols can be avoided by restricting possible definitions in two ways that we define in this section. First, dependencies introduced by definitions and declarations need to be terminating, which is achieved by Kunčar and Popescu through a dependency relation that Åman Pohjola and Gengelbach [17] extend to its present form. Secondly, declared or defined symbols need to be orthogonal [15], that is any pair of constants or any pair of types that originates from distinct definitions is orthogonal.

[2] For instance, Euler's number *e* can be implicitly defined as the real-valued solution of a particular differential equation.

We write $u \equiv t$ for definitional updates, to mean that either u is introduced by a type definition with predicate t or otherwise u is one of the constants introduced by a constant specification with the witness t. For a context *ctxt* and types or terms u and v the dependency relation $u \rightsquigarrow_{ctxt} v$ holds whenever:

1. There is a definition $u \equiv t$ in the context *ctxt* and $v \in t^\bullet \cup t^\circ$, or
2. $u = \mathsf{Const}\ \_\ ty$ is a constant of type ty and $v \in ty^\bullet$, or
3. $u = \mathsf{Tyapp}\ \_\ l$ is a type and $v \in l$.

The first rule applies only to symbols defined by TypeDefn or ConstSpec, whereas the other rules apply also to symbols declared with NewType and NewConst. Formally $\rightsquigarrow$ is a relation on type + term, a disjoint union with canonical injections INL and INR.

The *(type-)substitutive closure* $\mathscr{R}^\downarrow$ of a binary relation $\mathscr{R}$ relates $\Theta\ t_1$ and $\Theta\ t_2$ if $t_1\ \mathscr{R}\ t_2$. A relation $\mathscr{R}$ is *terminating* if there is no sequence $(x_i)_{i\in\mathbb{N}}$ such that $x_i\ \mathscr{R}\ x_{i+1}$ for all $i \in \mathbb{N}$. If a binary relation $\mathscr{R}$ is terminating, its inverse $(\lambda x\, y.\, y \mathscr{R} x)$ is well-founded.

A context is *orthogonal* if any two distinct type definitions and any two distinct constant definitions are orthogonal. Orthogonality ensures that definitional theories have at most one definition for each ground symbol (recall *ground* means type-variable free).

Åman Pohjola and Gengelbach prove that a model exists for each orthogonal context with overloaded definitions whose substitutive closure of the dependency relation is terminating.

2.7 Semantics

In this section we introduce the semantics, which we inherit from Åman Pohjola and Gengelbach [17].

Zermelo-Fraenkel set theory The semantics is parametrised on a universe where the axioms of Zermelo-Fraenkel set theory (ZF) hold. A model of ZF is not constructible within HOL by Gödel's incompleteness argument. This setup is not new [17]. The existence of a set-theoretic universe is also an assumption in the mechanised proof of soundness of HOL Light (without overloading) [8], and it originates with Arthan [1].

Although this parametrisation appears as the assumption $\mathsf{is\_set\_theory}\ mem$ in some theorem statements, in the pretty-printed definitions we often omit the additional argument $mem : \mathscr{U} \Rightarrow \mathscr{U} \Rightarrow \mathsf{bool}$. Herein, the type variable $\mathscr{U}$ is the universe of sets. We also assume $\mathsf{is\_infinite}\ mem\ indset$, which states that $indset : \mathscr{U}$ is an infinite set.

For set membership $mem\ x\ s$ we write $x \in: s$. One is a singleton set, Boolset is the set of two distinct elements True and False, and $\mathsf{Boolean} : \mathsf{bool} \Rightarrow \mathscr{U}$ injects Booleans from HOL into $\mathscr{U}$ in the expected way. $\mathsf{Funspace}\ s\ r$ contains as elements all functions with domain $s : \mathscr{U}$ and co-domain $r : \mathscr{U}$. $\mathsf{Abstract}\ s\ r\ f$ is the intersection of the graph of $f : \mathscr{U} \Rightarrow \mathscr{U}$ with $s \times r$. In the special case that for any $x \in: s$ we have $(x, f\ x) \in: r$, then $\mathsf{Abstract}\ s\ r\ f \in: \mathsf{Funspace}\ s\ r$. For $x \in: s$ and $g = \mathsf{Abstract}\ s\ r\ f$, we write $g\ '\ x$ for $f\ x$, namely the second component of $(x, f\ x)$ from g.

Lazy ground semantics A pillar of the semantics is a *(signature) fragment*, which is a tuple $(tys,consts)$ from a signature *sig* satisfying:

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

The types *tys* are ground, non-built-in types from the signature *sig*. Each constant from *consts* is non-built-in and has a ground type from the fragment, where types_of_frag (*tys*,*consts*) is defined as the built-in type closure builtin_closure *tys*. The *total fragment* is the largest fragment of a signature *sig*.

$$\mathsf{total\_fragment}\ \mathit{sig} \stackrel{\mathsf{def}}{=} (\mathsf{ground\_types}\ \mathit{sig} \cap \mathsf{nonbuiltin\_types}, \mathsf{ground\_consts}\ \mathit{sig} \cap \mathsf{nonbuiltin\_constinsts})$$

The function δ: type $\Rightarrow$ $\mathscr{U}$ assigns to each non-built-in type of a fragment a value in the universe. ext δ extends this to built-in types in a standard manner. Similarly, ext γ extends an interpretation of non-built-in constants γ to the built-in constants. A *(fragment) interpretation* is a tuple (δ,γ) such that

$$\begin{array}{l}\mathsf{is\_type\_frag\_interpretation}\ \mathit{tys}\ \delta \stackrel{\mathsf{def}}{=} \forall \mathit{ty}.\ \mathit{ty} \in \mathit{tys} \Rightarrow \mathsf{inhabited}\ (\delta\ \mathit{ty}) \\ \mathsf{is\_frag\_interpretation}\ (\mathit{tys},\mathit{consts})\ \delta\ \gamma \stackrel{\mathsf{def}}{=} \\ \quad \mathsf{is\_type\_frag\_interpretation}\ \mathit{tys}\ \delta \wedge \forall (c,\mathit{ty}).\ (c,\mathit{ty}) \in \mathit{consts} \Rightarrow \gamma\ (c,\mathit{ty}) \in: \mathsf{ext}\ \delta\ \mathit{ty}\end{array}$$

Ground semantics means that only ground instances of types and constants are interpreted. A *fragment valuation* v assigns to each Var x ty, with Θ ty a (ground) type of the fragment, a value that lies in the interpretation of Θ ty.

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

The term semantics is defined as a continuation of a fragment interpretation, parametrised by a fragment valuation v and a type instantiation Θ.

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

Herein $f(\!|x\ \mapsto\ y|\!)$ is the function that at x takes the value y and elsewhere equals f.

The semantics applies type substitutions *lazily*, i. e. as late as possible and never to the type of term variables. This avoids a problem [17] with the eager semantics of Kunčar and Popescu: in HOL's Church-style atoms, variables Var x (Tyvar a) and Var x Bool are distinct and hence should be allowed to have different valuations under all type substitutions. With the lazy ground semantics, for Θ (Tyvar a) = Bool we just have v (x,Tyvar a) $\in$: ext δ (Θ (Tyvar a))= Boolset and v (x,Bool) $\in$: ext δ (Θ Bool)= Boolset. In contrast, eager ground semantics erroneously identifies v (x,Θ (Tyvar a)) = v (x,Θ Bool).

We define the satisfaction relation of a fragment interpretation (δ,γ), hypotheses *hyps* and a term p w. r. t. a fragment *frag* and a type substitution Θ. Every fragment valuation v that satisfies all instantiated hypotheses must satisfy the instantiated term Θ p.

$$\begin{array}{l}\mathsf{satisfies}\ \mathit{frag}\ \delta\ \gamma\ \Theta\ (\mathit{hyps},p) \stackrel{\mathsf{def}}{=} \\ \quad \forall v. \\ \qquad \mathsf{valuates\_frag}\ \mathit{frag}\ \delta\ v\ \Theta \wedge p \in \mathsf{terms\_of\_frag\_uninst}\ \mathit{frag}\ \Theta \wedge \\ \qquad \mathsf{every}\ (\lambda t.\ t \in \mathsf{terms\_of\_frag\_uninst}\ \mathit{frag}\ \Theta)\ \mathit{hyps} \wedge \mathsf{every}\ (\lambda t.\ \mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ t = \mathsf{True})\ \mathit{hyps} \Rightarrow \\ \qquad\ \mathsf{termsem}\ \delta\ \gamma\ v\ \Theta\ p = \mathsf{True}\end{array}$$

Satisfaction of hypotheses *hyps* and a conclusion p w. r. t. a fragment interpretation (δ,γ) and a signature *sig* is quantified over all ground type substitutions of the signature.

$$\begin{array}{l}\mathsf{sat}\ sig\ \delta\ \gamma\ (hyps,p) \stackrel{\mathsf{def}}{=}\\ \quad \forall\Theta.\\ \quad\quad (\forall ty.\ \mathsf{tyvars}\ (\Theta\ ty) = [\,]) \wedge (\forall ty.\ \mathsf{type\_ok}\ (\mathsf{tysof}\ sig)\ (\Theta\ ty)) \wedge\\ \quad\quad \mathsf{every}\ (\lambda\ tm.\ tm \in \mathsf{ground\_terms\_uninst}\ sig\ \Theta)\ hyps \wedge p \in \mathsf{ground\_terms\_uninst}\ sig\ \Theta \Rightarrow\\ \quad\quad\ \mathsf{satisfies}\ (\mathsf{total\_fragment}\ sig)\ \delta\ \gamma\ \Theta\ (hyps,p)\end{array}$$

A total fragment interpretation (δ,γ) is a model of a theory *thy* if all of the theory's axioms are satisfied.

$$\begin{array}{l}\mathsf{models}\ \delta\ \gamma\ thy \stackrel{\mathsf{def}}{=}\\ \quad \mathsf{is\_frag\_interpretation}\ (\mathsf{total\_fragment}\ (\mathsf{sigof}\ thy))\ \delta\ \gamma \wedge\\ \quad \forall p.\ p \in \mathsf{axsof}\ thy \Rightarrow \mathsf{sat}\ (\mathsf{sigof}\ thy)\ (\mathsf{ext}\ \delta)\ (\mathsf{ext}\ (\mathsf{ext}\ \delta)\ \gamma)\ ([\,],p)\end{array}$$

As the semantic counterpart of derivability (Section 2.2), we define semantic entailment $(thy,hyps) \models p$.

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

The inference system is sound w. r. t. this semantics [17].

3 Symbol-independent fragment

After recapitulating the syntax and semantics in the previous section, we are set to discuss our contribution. The convenience that constants may be used prior to their definition comes at the price that interpretations of previously introduced symbols may change in extensions that define previously undefined symbols. For instance, the interpretation may change for defined orderings on lists, lexicographically defined as $\leq_{\alpha\ \mathsf{list}\to\alpha\ \mathsf{list}\to\mathsf{Bool}} === \mathsf{lex}(\leq_{\alpha\to\alpha\to\mathsf{Bool}})$, if an update defines any previously undefined instance of $\leq$. In this section we carve out the fragment of all symbols that are unaffected by a theory update.

An *independent fragment* collects constants and types of a host fragment *frag* whose definitions within a theory context *ctxt* are independent of any of the symbols from a set U.

$$\begin{array}{l}\mathsf{indep\_frag}\ ctxt\ U\ frag \stackrel{\mathsf{def}}{=}\\ \quad \mathsf{let}\ V = \{\ x \mid \exists\Theta\ u.\ u \in U \wedge x\ (\rightsquigarrow_{ctxt}{}^{\downarrow})^*\ \Theta\ u\ \}\ ;\\ \quad\quad V_2 = \{\ (x,ty) \mid \mathsf{INR}\ (\mathsf{Const}\ x\ ty) \in V\ \}\ ;\ V_1 = \{\ x \mid \mathsf{INL}\ x \in V\ \}\ \mathsf{in}\\ \quad (\mathsf{fst}\ frag \setminus V_1, \mathsf{snd}\ frag \setminus V_2)\end{array}$$

The set U contains the symbols introduced by a theory extension. In contrast to [4], where U is a singleton set, we allow the introduction of several symbols at once, e. g. via constant specification. The set V is the pre-image of type instances $\Theta\ u$ of elements u from U (with Θ a ground type substitution) under the reflexive-transitive, type-substitutive closure of the dependency relation $\rightsquigarrow_{ctxt}$. As host fragment *frag*, we only consider total fragments (over different signatures). An independent fragment of a total fragment is indeed a signature fragment, since constants depend on their types.

$$\begin{array}{l}\vdash ctxt\ \mathsf{extends\ init\_ctxt} \Rightarrow\\ \quad \mathsf{is\_sig\_fragment}\ (\mathsf{sigof}\ ctxt)\ (\mathsf{indep\_frag}\ ctxt\ U\ (\mathsf{total\_fragment}\ (\mathsf{sigof}\ ctxt)))\end{array}$$

We prove this claim in script, to give a flavour of the reasoning involved in the mechanisation. Thereby we amend the earlier proof [4] for the case where a type substitution ρ and $^\bullet$ do not commute on a type ς, i. e. $\rho(\varsigma^\bullet) \neq \rho(\varsigma)^\bullet$. (This case had been excluded by a faulty lemma inherited from Kunčar and Popescu.)

For a fixed context, F_U denotes the fragment independent of symbols U, and $\mathsf{GType}^\bullet$ and GInst° are all types and non-built-in constants of the total fragment, respectively.

Proof. For a ground constant instance $c_\sigma \in \mathsf{GInst}^\circ \setminus V$, we show that also its type σ is from the types of F_U. Assume that $\sigma \notin \mathsf{builtin\_closure}(\mathsf{GType}^\bullet \setminus V)$. Thus $\sigma^\bullet \not\subseteq \mathsf{GType}^\bullet \setminus V$ and there is a type $\tau \in \sigma^\bullet \cap V$. Assuming the dependency $c_\sigma \rightsquigarrow^{\downarrow+} \tau$ the contradiction $c_\sigma \in V$ follows. We now show $c_\sigma \rightsquigarrow^{\downarrow+} \tau$ for $\tau \in \sigma^\bullet$:
Let c_ς be a (defined or declared) constant. It holds $c_\varsigma \rightsquigarrow t$ for $t \in \varsigma^\bullet$ and thus for any instance $c_{\rho(\varsigma)} \rightsquigarrow^\downarrow \rho(t)$ for $t \in \varsigma^\bullet$. Generally, $\rho(\varsigma)^\bullet \neq \rho(\varsigma^\bullet)$ as Åman Pohjola and Gengelbach notice [17]. If ς is a type variable or a non-built-in type, $\varsigma^\bullet = \{\varsigma\}$, then $c_{\rho(\varsigma)} \rightsquigarrow^\downarrow \rho(\varsigma)$ and $\rho(\varsigma) \rightsquigarrow t$ for $t \in \rho(\varsigma)^\bullet$. If on the other hand $\varsigma = a \to b$ is the built-in function type, thus σ is a function type and let ρ be such that $\rho(a \to b) = \sigma$. Any type below σ and above $\tau \in \sigma^\bullet$ is a function type (as $\tau \in \sigma^\bullet \neq \{\sigma\}$). If τ is introduced by a type instantiation, then within $a \to b$ there is a type variable α such that $\rho(\alpha)$ syntactically contains τ. Thus $c_{a\to b} \rightsquigarrow \alpha$ by $\alpha \in (a \to b)^\bullet$ and $\rho(\alpha) \rightsquigarrow^+ \tau$ (as in $\rho(\alpha)$ there are only function types above τ). If τ was not introduced by a type instantiation and τ' is the type within $a \to b$ such that $\rho(\tau') = \tau$, then $c_{a\to b} \rightsquigarrow \tau'$ and consequently $c_{\rho(a\to b)} \rightsquigarrow^\downarrow \rho(\tau') = \tau$. □

Symbols introduced by a theory extension Until now, the independent fragment has been defined without regard to the theory extension mechanism, to contain all symbols that are independent of the symbols from an arbitrary set U. The relevant independent fragments are those that are independent of a theory extension, i. e. those for which U contains the constant instances and types that are introduced by a theory update. For an update *upd*, we set $U = \mathsf{upd\_introduces}\ upd$ as the apex of the independent fragment cone.

$$\begin{aligned}
&\mathsf{upd_introduces}\ (\mathsf{ConstSpec}\ ov\ eqs\ prop) \stackrel{\mathsf{def}}{=} \mathsf{map}\ (\lambda\ (s,t).\ \mathsf{INR}\ (\mathsf{Const}\ s\ (\mathsf{typeof}\ t)))\ eqs\\
&\mathsf{upd_introduces}\ (\mathsf{TypeDefn}\ name\ pred\ abs\ rep) \stackrel{\mathsf{def}}{=}\\
&\ [\mathsf{INL}\ (\mathsf{Tyapp}\ name\ (\mathsf{map}\ \mathsf{Tyvar}\ (\mathsf{mlstring_sort}\ (\mathsf{tvars}\ pred))))]\\
&\mathsf{upd_introduces}\ (\mathsf{NewType}\ name\ arity) \stackrel{\mathsf{def}}{=}\\
&\ [\mathsf{INL}\ (\mathsf{Tyapp}\ name\ (\mathsf{map}\ \mathsf{Tyvar}\ (\mathsf{genlist}\ (\lambda x.\ \mathsf{implode}\ (\mathsf{replicate}\ (\mathsf{SUC}\ x)\ \#\text{“a”}))\ arity))))]\\
&\mathsf{upd_introduces}\ (\mathsf{NewConst}\ name\ ty) \stackrel{\mathsf{def}}{=} [\mathsf{INR}\ (\mathsf{Const}\ name\ ty)]\\
&\mathsf{upd_introduces}\ (\mathsf{NewAxiom}\ prop) \stackrel{\mathsf{def}}{=} [\,]
\end{aligned}$$

For constant specifications and declarations, upd_introduces returns the constants available for use after the theory update. For type definitions, the introduced type constructor has as arguments all type variables of the defining predicate sorted by name. Type declarations introduce a type constructor whose arguments are *arity* many distinct type variables.

In the definition of upd_introduces we make two choices:

- The independent fragment of an update defining a type τ by a predicate $t_{\sigma\to\mathsf{Bool}}$ defines $U = \{\tau\}$. For a type substitution ρ either all instances $\rho(\tau)$, $\mathsf{rep}_{\rho(\sigma\to\tau)}$ and $\mathsf{abs}_{\rho(\tau\to\sigma)}$ are in F_U or otherwise in its complement (that we earlier denoted V). Although the proof of said property is non-trivial, the choice of defining $U = \{\tau\}$ instead of $U = \{\tau, \mathsf{rep}_{\sigma\to\tau}, \mathsf{abs}_{\tau\to\sigma}\}$ adds the convenience (for case

analysis in some proofs) that any constant introduced by an update (w. r. t. upd_introduces) does not come from a type definition.

- As non-definitional axioms generally are not conservative, any symbol's interpretation may be affected by such an update, hence we define upd_introduces (NewAxiom *prop*) $\stackrel{\text{def}}{=}$ [].

We henceforth only regard independent fragments related to theory updates.

$$\text{indep\_frag\_upd}\ \mathit{ctxt}\ \mathit{upd}\ \mathit{frag} \stackrel{\text{def}}{=} \text{indep\_frag}\ \mathit{ctxt}\ (\text{upd\_introduces}\ \mathit{upd})\ \mathit{frag}$$

The independent fragment of a theory *ctxt* extended by *upd* is carved out from the total fragment over the extended signature, but factually any symbols introduced by the update are not within the fragment:

$$\begin{array}{l}\vdash \text{let}\ \mathit{idf} = \text{indep\_frag\_upd}\ (\mathit{upd}::\mathit{ctxt})\ \mathit{upd}\ (\text{total\_fragment}\ (\text{sigof}\ (\mathit{upd}::\mathit{ctxt})))\ \text{in}\\ \quad \mathit{upd}::\mathit{ctxt}\ \text{extends init\_ctxt} \Rightarrow\\ \quad\quad \text{fst}\ \mathit{idf} \subseteq \text{fst}\ (\text{total\_fragment}\ (\text{sigof}\ \mathit{ctxt})) \wedge \text{snd}\ \mathit{idf} \subseteq \text{snd}\ (\text{total\_fragment}\ (\text{sigof}\ \mathit{ctxt}))\end{array}$$

Hereby, the *upd*-independent fragments over the signatures *ctxt* and *upd*::*ctxt* are equal, as each symbol introduced by the extension by *upd* depends on a symbol in upd_introduces *upd*.

4 Model-theoretic Conservativity

In this section we discuss how we construct a model of an extended theory while keeping parts of a model from the theory prior to extension. With the properties of the construction and an extra assumption on the given models we prove model-theoretic conservativity.

4.1 Model construction

From a model (Δ,Γ) of a theory *ctxt* we construct a model (δ,γ) of the extension *upd*::*ctxt*. Supported theory extensions are either extensions by definition or declaration of constants or a type, or otherwise admissible non-definitional axioms. A model of the extended theory is constructed by recursion over part of the $\rightsquigarrow^{\downarrow}$ relation, based on the model (Δ,Γ). In contrast, the model construction in [12] obtains a model from the ground up, by recursion over the entire $\rightsquigarrow^{\downarrow}$ relation, without reference to any previous interpretation.

A model is constructed by two mutually recursive functions type_interpretation_ext *ind ctxt upd* Δ Γ *ty* and term_interpretation_ext *ind ctxt upd* Δ Γ *c ty* that return the interpretation of a type or constant instance, respectively. As arguments these functions take the model (Δ,Γ) of the theory *ctxt*, the update *upd* and an infinite type *ind*. The model construction for a definitional theory extension *upd*::*ctxt* is guarded with a check: if the symbol to interpret lies in the independent fragment

$$\text{indep\_frag\_upd}\ (\mathit{upd}::\mathit{ctxt})\ \mathit{upd}\ (\text{total\_fragment}\ (\text{sigof}\ \mathit{ctxt}))$$

of a definitional update *upd*, the symbol may be interpreted w. r. t. the model (Δ,Γ). Otherwise the symbol's interpretation is constructed as discussed in earlier work [17, 12, 4].

Our amendments to the model construction are a few lines each (here the four lines of the second if branch) in type_interpretation_ext and term_interpretation_ext. The inherited tedious parts are elided.

$$
\begin{array}{l}
\mathsf{type\_interpretation\_ext}\ \mathit{ind}\ \mathit{upd}\ \mathit{ctxt}\ \Delta\ \Gamma\ \mathit{ty} \stackrel{\mathsf{def}}{=} \\
\quad \mathsf{if}\ \neg\mathsf{wellformed}\ (\mathit{upd}::\mathit{ctxt}) \\
\quad \mathsf{then}\ \mathsf{One} \\
\quad \mathsf{else\ if} \\
\quad\quad (\forall \mathit{tm}.\ \mathit{upd} \neq \mathsf{NewAxiom}\ \mathit{tm}) \wedge \\
\quad\quad \mathit{ty} \in \mathsf{fst}\ (\mathsf{indep\_frag\_upd}\ (\mathit{upd}::\mathit{ctxt})\ \mathit{upd}\ (\mathsf{total\_fragment}\ (\mathsf{sigof}\ \mathit{ctxt}))) \\
\quad \mathsf{then}\ \Delta\ \mathit{ty} \\
\quad \mathsf{else} \ldots
\end{array}
$$

Requirements for Constant Specification The differing constant definition mechanism entails that the model construction yields no model, but only a fragment interpretation of the theory's total fragment.

For a theory *ctxt* that has a model (Δ,Γ) we need to prove that any axiom from *ctxt* holds in a model (δ,γ) of a valid theory extension $\mathit{upd}::\mathit{ctxt}$. In its proof we are presented with a sub-case that occurs due to the different definitional mechanism for constants, as compared to Gengelbach and Weber. We illustrate the problem by an example theory:

Let *ctxt* be a theory where by constant specification two constants d_{Bool} and e_{Bool} are defined to be distinct by the axiom $d_{\mathsf{Bool}} \neq e_{\mathsf{Bool}}$, that holds for the witnesses $d_{\mathsf{Bool}} = \mathsf{False}$ and $e_{\mathsf{Bool}} = (c_{\mathsf{Bool}} \Rightarrow \mathsf{True})$ for a declared-only constant c_{Bool}. Let an update *upd* define $c_{\mathsf{Bool}} = \mathsf{True}$. For the fragment of *ctxt* that is independent of this update of c_{Bool} we write F. For a model (Δ,Γ) of *ctxt* we have to show that the axiom $d_{\mathsf{Bool}} \neq e_{\mathsf{Bool}}$ holds in the model extension (δ,γ) for $\mathit{upd}::\mathit{ctxt}$ as obtained from the above model construction. With $d_{\mathsf{Bool}} \in F$ and $e_{\mathsf{Bool}} \notin F$, it is impossible to prove that $\gamma(d_{\mathsf{Bool}}) \neq \gamma(e_{\mathsf{Bool}})$ as we only know $\gamma(d_{\mathsf{Bool}}) = \Gamma(d_{\mathsf{Bool}})$ and $\gamma(e_{\mathsf{Bool}}) = \mathsf{true}$.

In the example two constants are simultaneously introduced and defined in terms of another, and only one lies in the independent fragment. In the iterative model construction, information is lost on how constants are interpreted that are dependencies of a symbol. We choose to only extend models where each defined constant is interpreted as its witness.

We require that all constants, defined by constant specifications in a context *ctxt*, are interpreted equal to their witness in a model (Δ,Γ).

$$
\begin{array}{l}
\mathsf{models\_witnesses}\ \Delta\ \Gamma\ \mathit{ctxt} \stackrel{\mathsf{def}}{=} \\
\quad \forall \mathit{ov}\ \mathit{cl}\ \mathit{prop}\ c\ \mathit{cdefn}\ \mathit{ty}\ \Theta. \\
\quad\quad \mathsf{ConstSpec}\ \mathit{ov}\ \mathit{cl}\ \mathit{prop} \in \mathit{ctxt} \wedge (c,\mathit{cdefn}) \in \mathit{cl} \wedge \mathit{ty} = \mathsf{typeof}\ \mathit{cdefn} \wedge \\
\quad\quad (c,\Theta\ \mathit{ty}) \in \mathsf{ground\_consts}\ (\mathsf{sigof}\ \mathit{ctxt}) \wedge (c,\Theta\ \mathit{ty}) \in \mathsf{nonbuiltin\_constinsts} \Rightarrow \\
\quad\quad \Gamma\ (c,\Theta\ \mathit{ty}) = \mathsf{termsem}\ (\mathsf{ext}\ \Delta)\ (\mathsf{ext}\ (\mathsf{ext}\ \Delta)\ \Gamma)\ \mathsf{empty\_valuation}\ \Theta\ \mathit{cdefn}
\end{array}
$$

This added requirement is preserved by the model construction.

$$
\begin{array}{l}
\vdash \mathsf{is\_set\_theory}\ \mathit{mem} \Rightarrow \\
\quad \forall \mathit{upd}\ \mathit{ctxt}\ \Delta\ \Gamma. \\
\quad\quad \mathit{upd}::\mathit{ctxt}\ \mathsf{extends\ init\_ctxt} \wedge \mathsf{inhabited}\ \mathit{ind} \wedge \\
\quad\quad \mathsf{is\_frag\_interpretation}\ (\mathsf{total\_fragment}\ (\mathsf{sigof}\ \mathit{ctxt}))\ \Delta\ \Gamma \wedge \\
\quad\quad \mathsf{models\_witnesses}\ \Delta\ \Gamma\ \mathit{ctxt} \Rightarrow \\
\quad\quad\quad \mathsf{models\_witnesses}\ (\mathsf{type\_interpretation\_ext}\ \mathit{ind}\ \mathit{upd}\ \mathit{ctxt}\ \Delta\ \Gamma) \\
\quad\quad\quad\quad (\mathsf{term\_interpretation\_ext}\ \mathit{ind}\ \mathit{upd}\ \mathit{ctxt}\ \Delta\ \Gamma)\ (\mathit{upd}::\mathit{ctxt})
\end{array}
$$

The restriction to models of theories that satisfy models_witnesses keeps the expressivity of Arthan's constant specification and is conservative w. r. t. constant definition.

Alternatively, the problem as depicted in the example can be circumvented through extending the dependency relation with *cross-dependencies*. For simultaneously introduced constants d and e, any dependency x of e (i. e. $e \rightsquigarrow x$) also becomes a dependency of d (i. e. $d \rightsquigarrow x$) and likewise with d and e swapped. Any constants that are introduced together would thereby be assumed to be related, which reduces expressivity. For two declared constants f_α and g_{Bool} the otherwise legitimate simultaneous definition of $f_\alpha = g_\alpha$ and $g_{\mathsf{Bool}} = \mathsf{True}$ becomes impossible, as it is cyclic: $g_{\mathsf{Bool}} \rightsquigarrow^{\downarrow} g_{\mathsf{Bool}}$. Instead each conjunct would need to be a theory extension on its own.

4.2 Model-theoretic conservativity

In this subsection we introduce our main result. The mechanism to extend theories by definitions or declarations is model-theoretically conservative if for any theory *ctxt* with a model (Δ, Γ) that interprets any constant witness pair from constant specification equal, any theory extension *upd*::*ctxt* (where *upd* is a definition or declaration) has a model (δ, γ) that also satisfies the property:

$$\begin{array}{c}\mathsf{let}\ idf\ =\ \mathsf{indep\_frag\_upd}\ (upd{::}ctxt)\ upd\ (\mathsf{total\_fragment}\ (\mathsf{sigof}\ ctxt))\ \mathsf{in}\\ (\forall ty.\ ty \in \mathsf{fst}\ idf \Rightarrow \delta\ ty = \Delta\ ty) \wedge \forall c\ ty.\ (c,ty) \in \mathsf{snd}\ idf \Rightarrow \gamma\ (c,ty) = \Gamma\ (c,ty)\end{array}$$

If this property holds, it naturally extends to any ground term that is built from built-in types and symbols from the independent fragment *idf*. Hence any such term's interpretation in the new model (δ, γ) equals its interpretation in the old model (Δ, Γ). With the restriction to models that interpret constants as their witnesses we derive that the construction in Section 4.1 yields a model.

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

This constructed model trivially satisfies the given property that interpretations from the *upd*-independent fragment are kept if the *upd* is a declaration or a definition.

At different stages in the proof of model-theoretic conservativity, case analysis occurs of how an update *upd* may extend a theory *ctxt* by *upd* updates *ctxt*. As an example, proof obligations similar to the following reoccur frequently in the formalisation. To show that a symbol x keeps its interpretation in a model extension w. r. t. an update *upd*, one has to show that x is independent of the update *upd* by proving that all dependencies of x are on symbols from the *upd*-independent fragment.

Future work could investigate if the model construction may be conservative even w. r. t. NewAxiom updates of admissible axioms from hol_ctxt.

4.3 Consistency

As a consequence of the model construction from the previous section, we obtain consistency of *definitional* extensions of hol_ctxt, that is extensions that do not contain NewAxiom.

A theory is *consistent* if there is a provable and an unprovable sequent. We inherit the following definition from Kumar et al. [9].

$$\mathsf{consistent\_theory}\ \mathit{thy} \stackrel{\mathsf{def}}{=} (\mathit{thy},[]) \vdash \mathsf{Var}\ \text{«x»}\ \mathsf{Bool} === \mathsf{Var}\ \text{«x»}\ \mathsf{Bool} \wedge \neg((\mathit{thy},[]) \vdash \mathsf{Var}\ \text{«x»}\ \mathsf{Bool} === \mathsf{Var}\ \text{«y»}\ \mathsf{Bool})$$

As a corollary of our work, the existence of a model of init_ctxt combined with the incremental model construction yields consistency of definitional extensions of hol_ctxt [12, 17]. The restriction on the interpretations of constants as their witnesses trivially holds in init_ctxt and is an invariant in the induction.

$$\vdash \mathsf{is\_set\_theory}\ \mathit{mem} \wedge \mathsf{is\_infinite}\ \mathit{mem}\ \mathit{ind} \Rightarrow \forall \mathit{ctxt}.\ \mathsf{definitional\_extension}\ \mathit{ctxt}\ \mathsf{hol\_ctxt} \Rightarrow \mathsf{consistent\_theory}\ (\mathsf{thyof}\ \mathit{ctxt})$$

This work thus generalises and replaces the earlier non-incremental model construction [17].

5 Related Work

For untyped first-order logic, extension by definition of predicate and function symbols is discussed by Shoenfield [19, § 4.6]. A definitions by a predicate extends a theory with an equivalence that contains the predicate only on the left-hand side; a definition by a function symbol requires the proof that the function symbol indeed is a mathematical function. These mechanisms are proof-theoretically conservative, and each model of the original theory has one unique corresponding model of the extended theory. In consequence, both definitional mechanisms are model-theoretically conservative.

Farmer [3] defines an extension of a theory to be a super-set that is a model-theoretic conservative extension, hence keeps model interpretation and consistency. By example of simply-typed first-order logic with extension by algebraic datatypes and constant definitions, the author discusses also weaker notions of semantic conservativity and its properties w. r. t. theory embeddings, so called theory instantiation.

In their formalisation of HOL Light without overloading [9], Kumar et al. also make model-theoretic conservativity a requirement for theory extension by definitions or declarations. They denote this property sound_update *ctxt upd* of each such extension of *ctxt* by *upd*, and prove consistency by an inductive argument. As the definition mechanism for constants they use constant specification that allows to introduce multiple constants at once, given witnesses for which the defining axiom is derivable. Constant specification was first introduced by Arthan [2], and is is implemented in HOL4 [14] and ProofPower.

The study of theoretical foundations of overloaded definitions (together with type classes in higher-order logic) dates back to Wenzel [20]. For Wenzel an extension mechanism for deductive logics needs to be syntactically conservative, which he proves for constant definition where all instances are defined at once. In addition, the considered constant definitions can be unfolded, which is called *realisability*. In this discussion the interplay of overloaded constants and type definitions is not considered.

To avoid inconsistencies Obua [15] remarks that the unfolding of definitions needs to terminate for both type and constant definitions. Further Obua discusses that termination is not semi-decidable for overloaded definitions that recurse through types. The proof sketch of conservativity of overloading in Isabelle, he misses that inconsistencies may be introduced by dependencies through types.

For the Isabelle framework with its Haskell-style type classes, Wenzel and Haftmann [7] state requirements on overloading definitions without discussing if these suffice for acyclic dependencies.

Kunčar and Popescu [12] aim to close the consistency gap for definitional theories in Isabelle, in showing that every definitional theory has a model, by a model construction that recurses into the dependencies of definitions. Fixable gaps in their result are closed in the mechanisation of the model

construction by Åman Pohjola and Gengelbach [17]. Instead of constant definition their mechanisation considers Arthan's constant specification, and gives the above discussed *lazy fragment-ground* semantics. We base on their implementation work and generalise their monolithic model construction.

In two works, Kunčar and Popescu study consistency of definitional theories by syntactic arguments. They encode formulas through an unfolding of definitions into a richer logic HOL with comprehension types (HOLC) and prove that provability is preserved [13]. Ultimately, definitional theories are consistent by the consistency of HOLC.

In another paper, they use an unfolding that stays in the logic of HOL [11] by relativising defined types in formulas to a predicate on the defined type's host type. The proof-theoretic conservativity result holds for any definitional theory unfolded into initial HOL, and motivates a dual model-theoretic conservativity result where any model of initial HOL can be extended to a model of a given definitional theory. Our paper proves model-theoretic conservativity of two arbitrary definitional theories.

In recent work Gengelbach and Weber [5] prove model-theoretic conservativity of definitional theories [4] for semantics that do not require full function spaces in order to derive their syntactic counterparts. A definitional extension of a theory is proof-theoretically conservative, that is, if a formula's types and constants are unchanged by a theory update, and the formula is derivable after the update, then it is also derivable from the theory before the update. Their proof-theoretical result holds for constant definition and it is unclear how that result is transferable to constant specification with regard to the additional restriction on models models_witnesses in our proof.

Mizar is a theorem prover that supports overloading of symbols even for overlapping sub-types [6], where either the interpretation w. r. t. a definition may be specified or the most recently introduced definition is chosen for interpretation. Despite mentions of consistency of this sophisticated mechanism (e. g. [18]) there is no proof for consistency or conservativity of Mizar.

6 Conclusion

We established that type definitions and constant specifications in HOL with ad-hoc overloading of arbitrary theories above init_ctxt with fixed admissible axioms from hol_ctxt are model-theoretically conservative. The result holds for models that interpret each constant introduced by constant specification equal to the constant's witness. An interpretation of this result is that the definitional mechanisms of Isabelle/HOL are semantically speaking robustly designed: at least symbols that are independent of an update may keep their interpretation in a model extension.

Model-theoretic conservativity has a proof-theoretic (syntactic) counterpart. Roughly, an extension is *proof-theoretically 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.

In earlier work, Kunčar and Popescu [11] show that any definitional theory is a proof-theoretically conservative extension of *initial HOL*, i. e. hol_ctxt. The semantic counterpart is that any definitional theory is model-theoretically conservative above initial HOL. Comparably, our semantic conservativity is stronger as it holds for arbitrary theories above hol_ctxt, which we achieved by utilising the independent fragment, i. e. a subset of the signature that is independent of a theory extension.

We conjecture that the syntactic counterpart of our result holds: if D' is an extension of D such that $D' \vdash \varphi$, where φ is a formula whose non-built-in constant instances and types are independent of symbols defined in $D' \setminus D$, then $D \vdash \varphi$. Gengelbach and Weber recently proved this conjecture for constant definition through equality axioms [5]. We leave its study for the more general constant specification

mechanism [2] to future work.

References

[1] Rob Arthan: *HOL Formalised: Semantics*. Available at `http://www.lemma-one.com/ProofPower/specs/spc002.pdf`.

[2] Rob Arthan (2014): *HOL Constant Definition Done Right*. In: *Interactive Theorem Proving*, Springer International Publishing, pp. 531–536, doi:10.1007/978-3-319-08970-6_34.

[3] William M. Farmer: *A General Method for Safely Overwriting Theories in Mechanized Mathematics Systems*. Available at `http://imps.mcmaster.ca/doc/overwriting-theories.pdf`.

[4] Arve Gengelbach & Tjark Weber (2017): *Model-Theoretic Conservative Extension for Definitional Theories*. In Sandra Alves & Renata Wasserman, editors: *12th Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2017, Brasília, Brazil, September 23-24, 2017*, *Electronic Notes in Theoretical Computer Science* 338, Elsevier, pp. 133–145, doi:10.1016/j.entcs.2018.10.009.

[5] Arve Gengelbach & Tjark Weber (2020): *Proof-theoretic Conservativity for HOL with Ad-hoc Overloading*. In Violet Ka I Pun, Volker Stolz & Adenilso da Silva Simão, editors: *Theoretical Aspects of Computing - ICTAC 2020 - 17th International Colloquium, Macau, China, November 30 - December 4, 2020, Proceedings*, *Lecture Notes in Computer Science* 12545, Springer, pp. 23–42, doi:10.1007/978-3-030-64276-1_2.

[6] Adam Grabowski, Artur Kornilowicz & Adam Naumowicz (2010): *Mizar in a Nutshell*. *J. Formalized Reasoning* 3(2), pp. 153–245, doi:10.6092/issn.1972-5787/1980.

[7] Florian Haftmann & Makarius Wenzel (2006): *Constructive Type Classes in Isabelle*. In Thorsten Altenkirch & Conor McBride, editors: *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, *Lecture Notes in Computer Science* 4502, Springer, pp. 160–174, doi:10.1007/978-3-540-74464-1_11.

[8] Ramana Kumar, Rob Arthan, Magnus O. Myreen & Scott Owens (2014): *HOL with Definitions: Semantics, Soundness, and a Verified Implementation*. In: *Interactive Theorem Proving*, Springer, Cham, pp. 308–324, doi:10.1007/978-3-319-08970-6_20.

[9] Ramana Kumar, Rob Arthan, Magnus O. Myreen & Scott Owens (2016): *Self-Formalisation of Higher-Order Logic - Semantics, Soundness, and a Verified Implementation*. *J. Autom. Reasoning* 56(3), doi:10.1007/s10817-015-9357-x.

[10] Ondrej Kuncar (2015): *Correctness of Isabelle's Cyclicity Checker: Implementability of Overloading in Proof Assistants*. In Xavier Leroy & Alwen Tiu, editors: *Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, Mumbai, India, January 15-17, 2015*, ACM, pp. 85–94, doi:10.1145/2676724.2693175.

[11] Ondrej Kuncar & Andrei Popescu (2018): *Safety and conservativity of definitions in HOL and Isabelle/HOL*. *PACMPL* 2(POPL), pp. 24:1–24:26, doi:10.1145/3158112.

[12] Ondrej Kuncar & Andrei Popescu (2019): *A Consistent Foundation for Isabelle/HOL*. *J. Autom. Reasoning* 62(4), pp. 531–555, doi:10.1007/s10817-018-9454-8.

[13] Ondřej Kunčar & Andrei Popescu (2017): *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*, *Lecture Notes in Computer Science* 10201, Springer, pp. 724–749, doi:10.1007/978-3-662-54434-1_27.

[14] Michael Norrish & Konrad Slind (2014): *The HOL System LOGIC*. Available at `http://downloads.sourceforge.net/project/hol/hol/kananaskis-10/kananaskis-10-logic.pdf`.

[15] Steven Obua (2006): *Checking Conservativity of Overloaded Definitions in Higher-Order Logic*. In Frank Pfenning, editor: *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA,*

USA, August 12-14, 2006, Proceedings, *Lecture Notes in Computer Science* 4098, Springer, pp. 212–226, doi:10.1007/11805618_16.

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

[17] Johannes Åman Pohjola & Arve Gengelbach (2020): *A Mechanised Semantics for HOL with Ad-hoc Overloading*. In Elvira Albert & Laura Kovács, editors: *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, *EPiC Series in Computing* 73, EasyChair, pp. 498–515, doi:10.29007/413d. Available at https://easychair.org/publications/paper/9Hcd.

[18] Piotr Rudnicki (1992): *An Overview of the Mizar Project*. In Bengt Nordström, Kent Petersson & Gordon Plotkin, editors: *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, pp. 311–332. Available at http://mizar.org/project/MizarOverview.pdf.

[19] Joseph R. Shoenfield (1967): *Mathematical Logic*. A.K. Peters, Natick, Mass.

[20] Markus Wenzel (1997): *Type Classes and Overloading in Higher-Order Logic*. In Elsa L. Gunter & Amy P. Felty, editors: *Theorem Proving in Higher Order Logics, 10th International Conference, TPHOLs'97, Murray Hill, NJ, USA, August 19-22, 1997, Proceedings*, *Lecture Notes in Computer Science* 1275, Springer, pp. 307–322, doi:10.1007/BFb0028402.