Superposition with Lambdas

Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirović, Uwe Waldmann

J. Autom. Reasoning manuscript No.
(will be inserted by the editor)

Superposition with Lambdas

Alexander Bentkamp · Jasmin Blanchette · Sophie Tourret · Petar Vukmirović · Uwe Waldmann

Received: date / Accepted: date

Abstract We designed a superposition calculus for a clausal fragment of extensional polymorphic higher-order logic that includes anonymous functions but excludes Booleans. The inference rules work on $\beta\eta$-equivalence classes of λ-terms and rely on higher-order unification to achieve refutational completeness. We implemented the calculus in the Zipperposition prover and evaluated it on TPTP and Isabelle benchmarks. The results suggest that superposition is a suitable basis for higher-order reasoning.

Keywords superposition calculus · higher-order logic · refutational completeness

1 Introduction

Superposition [6] is widely regarded as the calculus par excellence for reasoning about first-order logic with equality. To increase automation in proof assistants and other verification tools based on higher-order formalisms, we propose to generalize superposition to an extensional, polymorphic, clausal version of higher-order logic (also called simple type theory). Our ambition is to achieve a *graceful* extension, which coincides with standard superposition on first-order problems and smoothly scales up to arbitrary higher-order problems.

Bentkamp, Blanchette, Cruanes, and Waldmann [12] designed a family of superposition-like calculi for a λ-free clausal fragment of higher-order logic, with currying and applied variables. We adapt their extensional nonpurifying calculus to support λ-terms (Sect. 3). Our calculus does not support interpreted Booleans. It is conceived as the penultimate milestone towards a superposition calculus for full higher-order logic. If desired, Booleans can be encoded in our logic fragment using an uninterpreted type and uninterpreted "proxy" symbols corresponding to equality, the connectives, and the quantifiers.

Designing a higher-order superposition calculus poses three main challenges:

Alexander Bentkamp (✉) · Jasmin Blanchette · Petar Vukmirović
Vrije Universiteit Amsterdam, Department of Computer Science, Section of Theoretical Computer Science, De Boelelaan 1111, 1081 HV Amsterdam, the Netherlands
E-mail: {a.bentkamp,j.c.blanchette,p.vukmirovic}@vu.nl

Jasmin Blanchette · Sophie Tourret · Uwe Waldmann
Max-Planck-Institut für Informatik, Saarland Informatics Campus E1 4, 66123 Saarbrücken, Germany
E-mail: {jblanche,stourret,uwe}@mpi-inf.mpg.de

1. Standard superposition is parameterized by a ground-total simplification order $\succ$, but such orders do not exist for λ-terms equal up to β-conversion. The relations designed for proving termination of higher-order term rewriting systems, such as HORPO [40] and CPO [22], lack many of the desired properties (e.g., transitivity, stability under grounding substitutions).

2. Higher-order unification is undecidable and may give rise to an infinite set of incomparable unifiers. For example, the constraint $\mathsf{f}\,(y\,\mathsf{a}) \stackrel{?}{=} y\,(\mathsf{f}\,\mathsf{a})$ admits infinitely many independent solutions of the form $\{y \mapsto \lambda x.\, \mathsf{f}^n\, x\}$.

3. In first-order logic, to rewrite into a term s using an oriented equation $t \approx t'$, it suffices to find a subterm of s that is unifiable with t. In higher-order logic, this is insufficient. Consider superposition from $\mathsf{f}\,\mathsf{c} \approx \mathsf{a}$ into $y\,\mathsf{c} \not\approx y\,\mathsf{b}$. The left-hand sides can obviously be unified by $\{y \mapsto \mathsf{f}\}$, but the more general unifier $\{y \mapsto \lambda x.\, z\,x\,(\mathsf{f}\,x)\}$ also gives rise to a subterm $\mathsf{f}\,\mathsf{c}$ after β-reduction. The corresponding inference generates $z\,\mathsf{c}\,\mathsf{a} \not\approx z\,\mathsf{b}\,(\mathsf{f}\,\mathsf{b})$.

To address the first challenge, we adopt the η-short β-normal form to represent $\beta\eta$-equivalence classes of λ-terms. In the spirit of Jouannaud and Rubio's early joint work [39], we state requirements on the term order only for ground terms (i.e., closed monomorphic $\beta\eta$-equivalence classes); the nonground case is connected to the ground case via stability under grounding substitutions. Even on ground terms, we cannot obtain all desirable properties. We sacrifice compatibility with arguments (the property that $s' \succ s$ implies $s'\,t \succ s\,t$), compensating with an *argument congruence* rule (ArgCong), as in Bentkamp et al. [12].

For the second challenge, we accept that there might be infinitely many incomparable unifiers and enumerate a complete set (including the notorious flex–flex pairs [37]), relying on heuristics to postpone the combinatorial explosion. The saturation loop must also be adapted to interleave this enumeration with the theorem prover's other activities (Sect. 6). Despite its reputation for explosiveness, higher-order unification is a conceptual improvement over SK combinators, because it can often *compute* the right unifier. Consider the conjecture $\exists z.\, \forall x\,y.\, z\,x\,y \approx \mathsf{f}\,y\,x$. After negation, clausification, and skolemization (which are as for first-order logic), the formula becomes $z\,(\mathsf{sk}_{\mathsf{x}}\,z)\,(\mathsf{sk}_{\mathsf{y}}\,z) \not\approx \mathsf{f}\,(\mathsf{sk}_{\mathsf{y}}\,z)\,(\mathsf{sk}_{\mathsf{x}}\,z)$. Higher-order unification quickly computes the unique unifier: $\{z \mapsto \lambda x\,y.\, \mathsf{f}\,y\,x\}$. In contrast, an encoding approach based on combinators, similar to the one implemented in Sledgehammer [52], would blindly enumerate all possible SK terms for z until the right one, $\mathsf{S}\,(\mathsf{K}\,(\mathsf{S}\,\mathsf{f}))\,\mathsf{K}$, is found. Given the definitions $\mathsf{S}\,z\,y\,x \approx z\,x\,(y\,x)$ and $\mathsf{K}\,x\,y \approx x$, the E prover [59] in *auto* mode needs to perform 3757 inferences to derive the empty clause.

For the third challenge, the idea is that, when applying $t \approx t'$ to perform rewriting inside a higher-order term s, we can encode an arbitrary context as a fresh higher-order variable z, unifying s with $z\,t$; the result is $(z\,t')\sigma$, for some unifier σ. This is performed by a dedicated *fluid subterm superposition* rule (FluidSup).

Functional extensionality is also considered a quintessential higher-order challenge [14], although similar difficulties arise with first-order sets and arrays [34]. Our approach is to add extensionality as an axiom and provide optional rules as optimizations (Sect. 5). With this axiom, our calculus is refutationally complete w.r.t. extensional Henkin semantics (Sect. 4). Our proof employs the new saturation framework by Waldmann et al. [71] to derive dynamic completeness of a given clause prover from ground static completeness.

We implemented the calculus in the Zipperposition prover [28] (Sect. 6). Our empirical evaluation includes benchmarks from the TPTP [64] and interactive verification problems exported from Isabelle/HOL [23] (Sect. 7). The results clearly demonstrate the calculus's potential. The 2020 edition of the CADE ATP System Competition (CASC) provides further

confirmation: Zipperposition finished 20 percentage points ahead of its closest rival [63]. This suggests that an implementation inside a high-performance prover such as E [59] or Vampire [48] could fulfill the promise of strong proof automation for higher-order logic (Sect. 8).

An earlier version of this article was presented at CADE-27 [11]. This article extends the conference paper with more explanations, detailed soundness and completeness proofs, including dynamic completeness, and new optional inference rules. We have also updated the empirical evaluation and extended the coverage of related work. Finally, we tightened side condition 4 of FLUIDSUP, making the rule slightly less explosive.

2 Logic

Our extensional polymorphic clausal higher-order logic is a restriction of full TPTP THF [16] to rank-1 (top-level) polymorphism, as in TH1 [41]. In keeping with standard superposition, we consider only formulas in conjunctive normal form, without explicit quantifiers or Boolean type. We use Henkin semantics [15, 31, 35], as opposed to the standard semantics that is commonly considered the foundation of the HOL systems [33]. Both semantics are compatible with the notion of provability employed by the HOL systems. But by admitting nonstandard models, Henkin semantics is not subject to Gödel's first incompleteness theorem, allowing us to claim refutational completeness of our calculus.

Syntax We fix a set Σ_{ty} of type constructors with arities and a set $\mathcal{V}_{\mathsf{ty}}$ of type variables. We require at least one nullary type constructor and a binary function type constructor $\to$ to be present in Σ_{ty}. A type τ, υ is either a type variable $\alpha \in \mathcal{V}_{\mathsf{ty}}$ or has the form $\kappa(\bar{\tau}_n)$ for an n-ary type constructor $\kappa \in \Sigma_{\mathsf{ty}}$ and types $\bar{\tau}_n$. We use the notation $\bar{a}_n$ or $\bar{a}$ to stand for the tuple $(a_1, \ldots, a_n)$ or product $a_1 \times \cdots \times a_n$, where $n \geq 0$. We write κ for $\kappa()$ and $\tau \to \upsilon$ for $\to(\tau, \upsilon)$. Type declarations have the form $\Pi\bar{\alpha}_m.\ \tau$ (or simply τ if $m = 0$), where all type variables occurring in τ belong to $\bar{\alpha}_m$.

We fix a set Σ of (function) symbols $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}, \mathsf{g}, \mathsf{h}, \ldots$, with type declarations, written as $\mathsf{f} : \Pi\bar{\alpha}_m.\ \tau$ or f, and a set $\mathcal{V}$ of term variables with associated types, written as $x : \tau$ or x. The notation $t : \tau$ will also be used to indicate the type of arbitrary terms t. We require the presence of a symbol of type $\Pi\alpha.\ \alpha$ and of a symbol $\mathsf{diff} : \Pi\alpha,\beta.\ (\alpha \to \beta) \to (\alpha \to \beta) \to \alpha$ in Σ. We use diff to express the polymorphic functional extensionality axiom. A signature is a pair $(\Sigma_{\mathsf{ty}}, \Sigma)$.

Next, we define terms on three layers of abstraction: raw λ-terms, λ-terms (as α-equivalence classes of raw λ-terms), and terms (as $\beta\eta$-equivalence classes of λ-terms).

The *raw λ-terms* over a given signature and their associated types are defined inductively as follows. Every $x : \tau \in \mathcal{V}$ is a raw λ-term of type τ. If $\mathsf{f} : \Pi\bar{\alpha}_m.\ \tau \in \Sigma$ and $\bar{\upsilon}_m$ is a tuple of types, called *type arguments*, then $\mathsf{f}\langle\bar{\upsilon}_m\rangle$ (or f if $m = 0$) is a raw λ-term of type $\tau\{\bar{\alpha}_m \mapsto \bar{\upsilon}_m\}$. If $x : \tau$ and $t : \upsilon$, then the *λ-expression* $\lambda x.\, t$ is a raw λ-term of type $\tau \to \upsilon$. If $s : \tau \to \upsilon$ and $t : \tau$, then the *application* $s\, t$ is a raw λ-term of type υ.

The function type constructor $\to$ is right-associative; application is left-associative. Using the spine notation [26], raw λ-terms can be decomposed in a unique way as a nonapplication *head* t applied to zero or more arguments: $t\, s_1 \ldots s_n$ or $t\, \bar{s}_n$ (abusing notation).

A raw λ-term s is a *subterm* of a raw λ-term t, written $t = t[s]$, if $t = s$, if $t = (\lambda x.\, u[s])$, if $t = (u[s])\, v$, or if $t = u\, (v[s])$ for some raw λ-terms u and v. A *proper* subterm of a raw λ-term t is any subterm of t that is distinct from t itself.

A variable occurrence is *free* in a raw λ-term if it is not bound by a λ-expression. A raw λ-term is *ground* if it is built without using type variables and contains no free term variables.

The α-renaming rule is defined as $(\lambda x.\, t) \longrightarrow_\alpha (\lambda y.\, t\{x \mapsto y\})$, where y does not occur free in t and is not captured by a λ-binder in t. Raw λ-terms form equivalence classes modulo α-renaming, called *λ-terms*. We lift the above notions on raw λ-terms to λ-terms.

A substitution ρ is a function from type variables to types and from term variables to λ-terms such that it maps all but finitely many variables to themselves. We also require that it is type-correct—i.e., for each $x : \tau \in \mathcal{V}$, $x\rho$ is of type $\tau\rho$. The letters θ,π,ρ,σ are reserved for substitutions. Substitutions implicitly α-rename λ-terms to avoid capture; for example, $(\lambda x.\, y)\{y \mapsto x\} = (\lambda x'.\, x)$. The composition $\rho\sigma$ applies ρ first: $t\rho\sigma = (t\rho)\sigma$. The notation $\sigma[\bar{x}_n \mapsto \bar{s}_n]$ stands for the substitution that replaces each x_i by s_i and that otherwise coincides with σ.

The β- and η-reduction rules are specified on λ-terms as $(\lambda x.\, t)\, u \longrightarrow_\beta t\{x \mapsto u\}$ and $(\lambda x.\, t\, x) \longrightarrow_\eta t$. For β, bound variables in t are implicitly renamed to avoid capture; for η, the variable x must not occur free in t. The λ-terms form equivalence classes modulo $\beta\eta$-reduction, called *$\beta\eta$-equivalence classes* or simply *terms*.

Convention 1 When defining operations that need to analyze the structure of terms, we will use the η-short β-normal form $t{\downarrow_{\beta\eta}}$, obtained by applying $\longrightarrow_\beta$ and $\longrightarrow_\eta$ exhaustively, as a representative of the equivalence class t. In particular, we lift the notions of subterms and occurrences of variables to $\beta\eta$-equivalence classes via their η-short β-normal representative.

Many authors prefer the η-long β-normal form [37, 39, 51], but in a polymorphic setting it has the drawback that instantiating a type variable with a functional type can lead to η-expansion. We reserve the letters s,t,u,v for terms and x,y,z for variables.

An equation $s \approx t$ is formally an unordered pair of terms s and t. A literal is an equation or a negated equation, written $\neg\, s \approx t$ or $s \not\approx t$. A clause $L_1 \vee \cdots \vee L_n$ is a finite multiset of literals L_j. The empty clause is written as $\bot$.

A *complete set of unifiers* on a set X of variables for two terms s and t is a set U of unifiers of s and t such that for every unifier θ of s and t there exists a member $\sigma \in U$ and a substitution ρ such that $x\sigma\rho = x\theta$ for all $x \in X$. We let $\mathrm{CSU}_X(s,t)$ denote an arbitrary (preferably minimal) complete set of unifiers on X for s and t. We assume that all $\sigma \in \mathrm{CSU}_X(s,t)$ are idempotent on X—i.e., $x\sigma\sigma = x\sigma$ for all $x \in X$. The set X will consist of the free variables of the clauses in which s and t occur and will be left implicit.

Given a substitution σ, the σ-instance of a term t or clause C is the term $t\sigma$ or the clause $C\sigma$, respectively. If $t\sigma$ or $C\sigma$ is ground, we call it a σ-ground instance.

Semantics A *type interpretation* $\mathcal{I}_{\mathsf{ty}} = (\mathcal{U}, \mathcal{J}_{\mathsf{ty}})$ is defined as follows. The *universe* $\mathcal{U}$ is a nonempty collection of nonempty sets, called *domains*. The function $\mathcal{J}_{\mathsf{ty}}$ associates a function $\mathcal{J}_{\mathsf{ty}}(\kappa) : \mathcal{U}^n \to \mathcal{U}$ with each n-ary type constructor κ, such that for all domains $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}$, the set $\mathcal{J}_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ is a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$. The semantics is *standard* if $\mathcal{J}_{\mathsf{ty}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ is the entire function space for all $\mathcal{D}_1, \mathcal{D}_2$.

A *type valuation* ξ is a function that maps every type variable to a domain. The *denotation* of a type for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and a type valuation ξ is defined by $[\![\alpha]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}} = \xi(\alpha)$ and $[\![\kappa(\bar{\tau})]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}} = \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\tau}]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}})$. We abuse notation by applying an operation on a tuple when it must be applied elementwise; thus, $[\![\bar{\tau}_n]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$ stands for $[\![\tau_1]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}, \ldots, [\![\tau_n]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$. A type valuation ξ can be extended to be a *valuation* by additionally assigning an element $\xi(x) \in [\![\tau]\!]^\xi_{\mathcal{I}_{\mathsf{ty}}}$ to each variable $x : \tau$. An *interpretation function* $\mathcal{J}$ for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ associates with

each symbol $\mathsf{f} : \Pi\bar{\alpha}_m.\ \tau$ and domain tuple $\bar{\mathcal{D}}_m \in \mathcal{U}^m$ a value $\mathcal{J}(\mathsf{f}, \bar{\mathcal{D}}_m) \in [\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$, where ξ is the type valuation that maps each α_i to $\mathcal{D}_i$.

The comprehension principle states that every function designated by a λ-expression is contained in the corresponding domain. Loosely following Fitting [31, Sect. 2.4], we initially allow λ-expressions to designate arbitrary elements of the domain, to be able to define the denotation of a term. We impose restrictions afterwards using the notion of a proper interpretation. A *λ-designation function* $\mathcal{L}$ for a type interpretation $\mathcal{I}_{\mathsf{ty}}$ is a function that maps a valuation ξ and a λ-expression of type τ to elements of $[\![\tau]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$. A type interpretation, an interpretation function, and a λ-designation function form an (*extensional*) *interpretation* $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$. For an interpretation $\mathcal{I}$ and a valuation ξ, the denotation of a term is defined as $[\![x]\!]^{\xi}_{\mathcal{I}} = \xi(x)$, $[\![\mathsf{f}\langle\bar{\tau}_m\rangle]\!]^{\xi}_{\mathcal{I}} = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}_m]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}})$, $[\![s\,t]\!]^{\xi}_{\mathcal{I}} = [\![s]\!]^{\xi}_{\mathcal{I}}([\![t]\!]^{\xi}_{\mathcal{I}})$, and $[\![\lambda x.\ t]\!]^{\xi}_{\mathcal{I}} = \mathcal{L}(\xi, \lambda x.\ t)$. For ground terms t, the denotation does not depend on the choice of the valuation ξ, which is why we sometimes write $[\![t]\!]_{\mathcal{I}}$ for $[\![t]\!]^{\xi}_{\mathcal{I}}$.

An interpretation $\mathcal{I}$ is *proper* if $[\![\lambda x.\ t]\!]^{\xi}_{\mathcal{I}}(a) = [\![t]\!]^{\xi[x\mapsto a]}_{\mathcal{I}}$ for all λ-expressions $\lambda x.\ t$, all valuations ξ, and all a. If a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and an interpretation function $\mathcal{J}$ can be extended by a λ-designation function $\mathcal{L}$ to a proper interpretation $(\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$, then this $\mathcal{L}$ is unique [31, Proposition 2.18]. Given an interpretation $\mathcal{I}$ and a valuation ξ, an equation $s \approx t$ is true if $[\![s]\!]^{\xi}_{\mathcal{I}}$ and $[\![t]\!]^{\xi}_{\mathcal{I}}$ are equal and it is false otherwise. A disequation $s \not\approx t$ is true if $s \approx t$ is false. A clause is true if at least one of its literals is true. A clause set is true if all its clauses are true. A proper interpretation $\mathcal{I}$ is a *model* of a clause set N, written $\mathcal{I} \models N$, if N is true in $\mathcal{I}$ for all valuations ξ.

Axiomatization of Booleans Our clausal logic lacks a Boolean type, but it can easily be axiomatized as follows. We extend the signature with a nullary type constructor $\mathit{bool} \in \Sigma_{\mathsf{ty}}$ equipped with the proxy constants $\mathsf{t}, \mathsf{f} : \mathit{bool}$, $\mathsf{not} : \mathit{bool} \to \mathit{bool}$, $\mathsf{and}, \mathsf{or}, \mathsf{impl}, \mathsf{equiv} : \mathit{bool} \to \mathit{bool} \to \mathit{bool}$, $\mathsf{forall}, \mathsf{exists} : \Pi\alpha.\ (\alpha \to \mathit{bool}) \to \mathit{bool}$, $\mathsf{eq} : \Pi\alpha.\ \alpha \to \alpha \to \mathit{bool}$, and $\mathsf{choice} : \Pi\alpha.\ (\alpha \to \mathit{bool}) \to \alpha$, characterized by the axioms

$$\begin{array}{ccc}
\mathsf{t} \not\approx \mathsf{f} & \mathsf{or}\,\mathsf{t}\,x \approx \mathsf{t} & \mathsf{equiv}\,x\,y \approx \mathsf{and}\,(\mathsf{impl}\,x\,y)\,(\mathsf{impl}\,y\,x) \\
x \approx \mathsf{t} \vee x \approx \mathsf{f} & \mathsf{or}\,\mathsf{f}\,x \approx x & \mathsf{forall}\langle\alpha\rangle\,(\lambda x.\ \mathsf{t}) \approx \mathsf{t} \\
\mathsf{not}\,\mathsf{t} \approx \mathsf{f} & \mathsf{impl}\,\mathsf{t}\,x \approx x & y \approx (\lambda x.\ \mathsf{t}) \vee \mathsf{forall}\langle\alpha\rangle\,y \approx \mathsf{f} \\
\mathsf{not}\,\mathsf{f} \approx \mathsf{t} & \mathsf{impl}\,\mathsf{f}\,x \approx \mathsf{t} & \mathsf{exists}\langle\alpha\rangle\,y \approx \mathsf{not}\,(\mathsf{forall}\langle\alpha\rangle\,(\lambda x.\,\mathsf{not}\,(y\,x))) \\
\mathsf{and}\,\mathsf{t}\,x \approx x & x \not\approx y \vee \mathsf{eq}\langle\alpha\rangle\,x\,y \approx \mathsf{t} & y\,x \approx \mathsf{f} \vee y\,(\mathsf{choice}\langle\alpha\rangle\,y) \approx \mathsf{t} \\
\mathsf{and}\,\mathsf{f}\,x \approx \mathsf{f} & x \approx y \vee \mathsf{eq}\langle\alpha\rangle\,x\,y \approx \mathsf{f} &
\end{array}$$

This axiomatization of Booleans can be used in a prover to support full higher-order logic with or without Hilbert choice, corresponding to the TPTP THF format variants TH0 (monomorphic) [66] and TH1 (polymorphic) [41]. The prover's clausifier would transform the outer first-order skeleton of a formula into a clause and use the axiomatized Booleans within the terms. It would also add the proxy axioms to the clausal problem. As an alternative to this complete axiomatization, Vukmirović and Nummelin [70] present a possibly refutationally incomplete calculus extension with dedicated rules to support Booleans. This approach works better in practice and contributed to Zipperposition's victory at CASC 2020.

3 The Calculus

The *Boolean-free λ-superposition calculus* presented here is inspired by the extensional non-purifying Boolean-free λ-free higher-order superposition calculus described by Bentkamp et

al. [12]. The text of this and the next section is partly based on that paper and the associated journal submission [10] (with Cruanes's permission). The central idea is that superposition inferences are restricted to *unapplied* subterms occurring in the first-order outer skeleton of clauses—that is, outside λ-expressions and outside the arguments of applied variables. We call these "green subterms." Thus, $\mathsf{g} \approx (\lambda x.\, \mathsf{f}\, x\, x)$ cannot be used directly to rewrite $\mathsf{g}\,\mathsf{a}$ to $\mathsf{f}\,\mathsf{a}\,\mathsf{a}$, because g is applied in $\mathsf{g}\,\mathsf{a}$. A separate inference rule, ArgCong, takes care of deriving $\mathsf{g}\, x \approx \mathsf{f}\, x\, x$, which can be oriented independently of its parent clause and used to rewrite $\mathsf{g}\,\mathsf{a}$ or $\mathsf{f}\,\mathsf{a}\,\mathsf{a}$.

Definition 2 (Green positions and subterms) A *green position* of a term (i.e., of a $\beta\eta$-equivalence class) is a finite sequence of natural numbers defined inductively as follows. For any term t, the empty tuple ε is a green position of t. For all symbols $\mathsf{f} \in \Sigma$, types $\bar{\tau}$, and terms $\bar{u}$, if p is a green position of u_i for some i, then $i.p$ is a green position of $\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}$.

The *green subterm* of a term at a given green position is defined inductively as follows. For any term t, t itself is the green subterm of t at green position ε. For all symbols $\mathsf{f} \in \Sigma$, types $\bar{\tau}$, and terms $\bar{u}$, if t is a green subterm of u_i at some green position p for some i, then t is the green subterm of $\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}$ at green position $i.p$. We denote the green subterm of s at the green position p by $s|_p$.

In $\mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,(y\,\mathsf{b})\,(\lambda x.\, \mathsf{h}\,\mathsf{c}\,(\mathsf{g}\, x))$, the proper green subterms are a, $\mathsf{g}\,\mathsf{a}$, $y\,\mathsf{b}$, and $\lambda x.\, \mathsf{h}\,\mathsf{c}\,(\mathsf{g}\, x)$. The last two of these do not look like first-order terms, and hence their subterms are not green.

Definition 3 (Green contexts) We write $t = s\langle u\rangle_p$ to express that u is a green subterm of t at the green position p and call $s\langle\,\rangle_p$ a *green context*. We omit the subscript p if there are no ambiguities.

In a $\beta\eta$-normal representative of a green context, the hole never occurs applied. Therefore, inserting a $\beta\eta$-normal term into the context produces another $\beta\eta$-normal term.

Another key notion is that of a fluid term. Fluid terms are certain variable-headed terms and λ-expressions into which the calculus must rewrite to be refutationally complete. Fluid terms trigger the FluidSup rule, which complements the familiar superposition rule Sup.

Definition 4 (Fluid terms) A term t is called *fluid* if (1) $t{\downarrow_{\beta\eta}}$ is of the form $y\,\bar{u}_n$ where $n \geq 1$, or (2) $t{\downarrow_{\beta\eta}}$ is a λ-expression and there exists a substitution σ such that $t\sigma{\downarrow_{\beta\eta}}$ is not a λ-expression (due to η-reduction).

Case (2) can arise only if t contains an applied variable. Intuitively, fluid terms are terms whose η-short β-normal form can change radically as a result of instantiation. For example, $\lambda x.\, y\,\mathsf{a}\,(z\, x)$ is fluid because applying $\{z \mapsto \lambda x.\, x\}$ makes the λ vanish: $(\lambda x.\, y\,\mathsf{a}\, x) = y\,\mathsf{a}$. Similarly, $\lambda x.\, \mathsf{f}\,(y\, x)\, x$ is fluid because $(\lambda x.\, \mathsf{f}\,(y\, x)\, x)\{y \mapsto \lambda x.\, \mathsf{a}\} = (\lambda x.\, \mathsf{f}\,\mathsf{a}\, x) = \mathsf{f}\,\mathsf{a}$.

3.1 The Core Inference Rules

The calculus is parameterized by a strict ground term order, a strict term order, a nonstrict term order, and a selection function. A *strict ground term order* $\succ$ needs to enjoy certain properties for our completeness proof to work. A *strict term order* $\succ$ underapproximates the lifting of a strict ground term order to the nonground level. To gain some precision in the side conditions of our rules, we introduce a *nonstrict term order* $\succsim$ that can compare more terms than the reflexive closure $\succeq$ of a strict term order $\succ$. The *selection function* resembles the selection function of first-order superposition, with a minor restriction concerning literals containing applied variables.

Definition 5 (Strict ground term order) A *strict ground term order* is a well-founded strict total order $\succ$ on ground terms satisfying the following criteria, where $\succeq$ denotes the reflexive closure of $\succ$:

- *green subterm property*: $t\langle s\rangle \succeq s$;
- *compatibility with green contexts*: $s' \succ s$ implies $t\langle s'\rangle \succ t\langle s\rangle$.

Given a strict ground term order, we extend it to literals and clauses via the multiset extensions in the standard way [6, Sect. 2.4].

Two properties that are not required are *compatibility with λ-expressions* ($s' \succ s$ implies $(\lambda x.\, s') \succ (\lambda x.\, s)$) and *compatibility with arguments* ($s' \succ s$ implies $s'\, t \succ s\, t$). The latter would even be inconsistent with totality. To see why, consider the symbols $\mathsf{c} \succ \mathsf{b} \succ \mathsf{a}$ and the terms $\lambda x.\, \mathsf{b}$ and $\lambda x.\, x$. Owing to totality, one of the terms must be larger than the other, say, $(\lambda x.\, \mathsf{b}) \succ (\lambda x.\, x)$. By compatibility with arguments, we get $(\lambda x.\, \mathsf{b})\, \mathsf{c} \succ (\lambda x.\, x)\, \mathsf{c}$, i.e., $\mathsf{b} \succ \mathsf{c}$, a contradiction. A similar line of reasoning applies if $(\lambda x.\, \mathsf{b}) \prec (\lambda x.\, x)$, using a instead of c.

Definition 6 (Strict term order) A *strict term order* is a relation $\succ$ on terms, literals, and clauses such that the restriction to ground entities is a strict ground term order and such that it is stable under grounding substitutions (i.e., $t \succ s$ implies $t\theta \succ s\theta$ for all substitutions θ grounding the entities t and s).

Definition 7 (Nonstrict term order) Given the reflexive closure $\succeq$ of a strict ground term order $\succ$, a *nonstrict term order* is a relation $\succsim$ on terms, literals, and clauses such that $t \succsim s$ implies $t\theta \succeq s\theta$ for all θ grounding the entities t and s.

Although we call them orders, a strict term order $\succ$ is not required to be transitive on non-ground entities, and a nonstrict term order $\succsim$ does not need to be transitive at all. Normally, $t \succeq s$ should imply $t \succsim s$, but this is not required either. A nonstrict term order $\succsim$ allows us to be more precise than the reflexive closure $\succeq$ of $\succ$. For example, we cannot have $y\,\mathsf{b} \succeq y\,\mathsf{a}$, because $y\,\mathsf{b} \neq y\,\mathsf{a}$ and $y\,\mathsf{b} \not\succ y\,\mathsf{a}$ by stability under grounding substitutions (with $\{y \mapsto \lambda x.\, \mathsf{c}\}$). But we can have $y\,\mathsf{b} \succsim y\,\mathsf{a}$ if $\mathsf{b} \succ \mathsf{a}$. In practice, $\succ$ and $\succsim$ should be chosen so that they can compare as many terms as possible while being computable and reasonably efficient.

Definition 8 (Maximality) An element x of a multiset M is $\trianglerighteq$*-maximal* for some relation $\trianglerighteq$ if for all $y \in M$ with $y \trianglerighteq x$, we have $y \trianglelefteq x$. It is *strictly* $\trianglerighteq$*-maximal* if it is $\trianglerighteq$-maximal and occurs only once in M.

Definition 9 (Selection function) A *selection function* is a function that maps each clause to a subclause consisting of negative literals, which we call the *selected* literals of that clause. A literal $L\langle y\rangle$ must not be selected if $y\,\bar{u}_n$, with $n > 0$, is a $\succeq$-maximal term of the clause.

The restriction on the selection function is needed for our proof, but it is an open question whether it is actually necessary for refutational completeness.

Our calculus is parameterized by a strict term order $\succ$, a nonstrict term order $\succsim$, and a selection function *HSel*. The term orders $\succ$ and $\succsim$ must be based on the same strict ground term order $\succ$. The calculus rules depend on the following auxiliary notions.

Definition 10 (Eligibility) A literal L is (*strictly*) $\trianglerighteq$*-eligible* w.r.t. a substitution σ in C for some relation $\trianglerighteq$ if it is selected in C or there are no selected literals in C and $L\sigma$ is (strictly) $\trianglerighteq$-maximal in $C\sigma$. If σ is the identity substitution, we leave it implicit.

Definition 11 (Deep occurrence) A variable *occurs deeply* in a clause C if it occurs inside a λ-expression or inside an argument of an applied variable.

For example, x and z occur deeply in $\mathsf{f}\,x\,y \approx y\,x \vee z \not\approx (\lambda w.\, z\,\mathsf{a})$, whereas y does not occur deeply. In particular, a variable occurring as a nongreen subterm is not necessarily a deeply occurring variable, as exemplified by y. This definition aims to capture all variables with an occurrence under a λ-expression in some ground instances of C.

The first rule of our calculus is the superposition rule. We regard positive and negative superposition as two cases of a single rule

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \quad \overbrace{C' \vee s\langle u\rangle \mathrel{\dot{\approx}} s'}^{C}}{(D' \vee C' \vee s\langle t'\rangle \mathrel{\dot{\approx}} s')\sigma}\ \textsc{Sup}$$

where $\dot{\approx}$ denotes either $\approx$ or $\not\approx$. The following side conditions apply:

1. u is not fluid;
2. u is not a variable deeply occurring in C;
3. *variable condition*: if u is a variable y, there must exist a grounding substitution θ such that $t\sigma\theta \succ t'\sigma\theta$ and $C\sigma\theta \prec C''\sigma\theta$, where $C'' = C\{y \mapsto t'\}$;
4. $\sigma \in \mathrm{CSU}(t,u)$;
5. $t\sigma \not\trianglelefteq t'\sigma$;
6. $s\langle u\rangle\sigma \not\trianglelefteq s'\sigma$;
7. $C\sigma \not\trianglelefteq D\sigma$;
8. $t \approx t'$ is strictly $\succsim$-eligible in D w.r.t. σ;
9. $s\langle u\rangle \mathrel{\dot{\approx}} s'$ is $\succsim$-eligible in C w.r.t. σ, and strictly $\succsim$-eligible if it is positive.

There are four main differences with the statement of the standard superposition rule: Contexts $s[\,]$ are replaced by green contexts $s\langle\,\rangle$. The standard condition $u \notin \mathcal{V}$ is generalized by conditions 2 and 3. Most general unifiers are replaced by complete sets of unifiers. And $\not\trianglelefteq$ is replaced by the more precise $\not\trianglelefteq$.

The second rule is a variant of Sup that focuses on fluid green subterms:

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \quad \overbrace{C' \vee s\langle u\rangle \mathrel{\dot{\approx}} s'}^{C}}{(D' \vee C' \vee s\langle z\,t'\rangle \mathrel{\dot{\approx}} s')\sigma}\ \textsc{FluidSup}$$

with the following side conditions, in addition to Sup's conditions 5 to 9:

1. u is either a fluid term or a variable deeply occurring in C;
2. z is a fresh variable;
3. $\sigma \in \mathrm{CSU}(z\,t, u)$;
4. $(z\,t')\sigma \neq (z\,t)\sigma$.

The equality resolution and equality factoring rules are almost identical to their standard counterparts:

$$\frac{\overbrace{C' \vee u \not\approx u'}^{C}}{C'\sigma}\ \textsc{ERes} \qquad \frac{\overbrace{C' \vee u' \approx v' \vee u \approx v}^{C}}{(C' \vee v \not\approx v' \vee u \approx v')\sigma}\ \textsc{EFact}$$

For ERes: $\sigma \in \mathrm{CSU}(u,u')$ and $u \not\approx u'$ is $\succsim$-eligible in C w.r.t. σ. For EFact: $\sigma \in \mathrm{CSU}(u,u')$, $u\sigma \not\trianglelefteq v\sigma$, and $u \approx v$ is $\succsim$-eligible in C w.r.t. σ.

Argument congruence, a higher-order concern, is embodied by the rule

$$\frac{\overbrace{C' \vee s \approx s'}^{C}}{C'\sigma \vee s\sigma\,\bar{x}_n \approx s'\sigma\,\bar{x}_n}\ \textsc{ArgCong}$$

where $n > 0$ and σ is the most general type substitution that ensures well-typedness of the conclusion. In particular, if s accepts k arguments, then ArgCong yields k conclusions—one for each $n \in \{1,\ldots,k\}$—where σ is the identity substitution. If the result type of s is a type variable, ArgCong yields infinitely many additional conclusions—one for each $n > k$—where σ instantiates the result type of s with $\alpha_1 \to \cdots \to \alpha_{n-k} \to \beta$ for fresh $\bar{\alpha}_{n-k}$ and β. Moreover, the literal $s \approx s'$ must be strictly $\succsim$-eligible in C w.r.t. σ, and $\bar{x}_n$ is a tuple of distinct fresh variables.

The rules are complemented by the polymorphic functional extensionality axiom:

$$y\,(\mathsf{diff}\langle\alpha,\beta\rangle\,y\,z) \not\approx z\,(\mathsf{diff}\langle\alpha,\beta\rangle\,y\,z) \vee y \approx z \tag{Ext}$$

From now on, we will omit the type arguments to diff since they can be inferred from the term arguments.

3.2 Rationale for the Rules

The calculus realizes the following division of labor: Sup and FluidSup are responsible for green subterms, which are outside λs, ArgCong effectively gives access to the remaining positions outside λs, and the extensionality axiom takes care of subterms inside λs. The following examples illustrate these mechanisms. The unifiers below were chosen to keep the clauses reasonably small.

Example 12 The clause $\mathsf{g} \approx \mathsf{f}$ cannot superpose into $\mathsf{g\,a\,b} \not\approx \mathsf{f\,a\,b}$ because g occurs in a nongreen context. Instead, we refute these two clauses as follows:

$$\textsc{ERes}\,\dfrac{\textsc{Sup}\,\dfrac{\textsc{ArgCong}\,\dfrac{\mathsf{g} \approx \mathsf{f}}{\mathsf{g}\,x_1\,x_2 \approx \mathsf{f}\,x_1\,x_2} \qquad \mathsf{g\,a\,b} \not\approx \mathsf{f\,a\,b}}{\mathsf{f\,a\,b} \not\approx \mathsf{f\,a\,b}}}{\bot}$$

The ArgCong inference adds two arguments to g, yielding the term $\mathsf{g}\,x_1\,x_2$, which is unifiable with the green subterm $\mathsf{g\,a\,b}$. Thus we can apply Sup to the resulting clause.

Example 13 Applied variables give rise to subtle situations with no counterparts in first-order logic. Consider the clauses $\mathsf{f\,a} \approx \mathsf{c}$ and $\mathsf{h}\,(y\,\mathsf{b})\,(y\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f\,b}))\,(\mathsf{g\,c})$, where $\mathsf{f\,a} \succ \mathsf{c}$. It is easy to see that the clause set is unsatisfiable, by grounding the second clause with $\theta = \{y \mapsto \lambda x.\,\mathsf{g}\,(\mathsf{f}\,x)\}$. However, to mimic the superposition inference that can be performed at the ground level, it is necessary to superpose at an imaginary position *below* the applied variable y and yet *above* its argument a, namely, into the subterm $\mathsf{f\,a}$ of $\mathsf{g}\,(\mathsf{f\,a}) = (\lambda x.\,\mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (y\,\mathsf{a})\theta$. We need FluidSup:

$$\textsc{ERes}\,\dfrac{\textsc{FluidSup}\,\dfrac{\mathsf{f\,a} \approx \mathsf{c} \quad \mathsf{h}\,(y\,\mathsf{b})\,(y\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f\,b}))\,(\mathsf{g\,c})}{\mathsf{h}\,(z\,(\mathsf{f\,b}))\,(z\,\mathsf{c}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f\,b}))\,(\mathsf{g\,c})}}{\bot}$$

FluidSup's variable z effectively transforms $\mathsf{f\,a} \approx \mathsf{c}$ into $z\,(\mathsf{f\,a}) \approx z\,\mathsf{c}$, whose left-hand side can be unified with $y\,\mathsf{a}$ by taking $\{y \mapsto \lambda x.\,z\,(\mathsf{f}\,x)\}$.

Example 14 The clause set consisting of $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$, $\mathsf{f}\,\mathsf{b} \approx \mathsf{d}$, and $\mathsf{g}\,\mathsf{c} \not\approx y\,\mathsf{a} \vee \mathsf{g}\,\mathsf{d} \not\approx y\,\mathsf{b}$ has a similar flavor. ERES applies on either literal of the third clause, but the computed unifier, $\{y \mapsto \lambda x.\, \mathsf{g}\,\mathsf{c}\}$ or $\{y \mapsto \lambda x.\, \mathsf{g}\,\mathsf{d}\}$, is not the right one. Again, we need FLUIDSUP:

$$\dfrac{\mathsf{f}\,\mathsf{b} \approx \mathsf{d} \qquad \dfrac{\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{g}\,\mathsf{c} \not\approx y\,\mathsf{a} \vee \mathsf{g}\,\mathsf{d} \not\approx y\,\mathsf{b}}{\mathsf{g}\,\mathsf{c} \not\approx z\,\mathsf{c} \vee \mathsf{g}\,\mathsf{d} \not\approx z\,(\mathsf{f}\,\mathsf{b})}\,\textsc{FluidSup}}{\mathsf{g}\,\mathsf{d} \not\approx \mathsf{g}\,(\mathsf{f}\,\mathsf{b})}\,\textsc{Eres}}{\dfrac{\mathsf{g}\,\mathsf{d} \not\approx \mathsf{g}\,\mathsf{d}}{\bot}\,\textsc{Sup}}\,\textsc{Eres}$$

Again, the FLUIDSUP inference uses the unifier $\{y \mapsto \lambda x.\, z\,(\mathsf{f}\,x)\} \in \mathrm{CSU}(z\,(\mathsf{f}\,\mathsf{a}), y\,\mathsf{a})$.

Example 15 Third-order clauses containing subterms of the form $y\,(\lambda x.\, t)$ can be even more stupefying. The clause set consisting of $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$ and $\mathsf{h}\,(y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\,y \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x)$ is unsatisfiable. To see why, apply $\theta = \{y \mapsto \lambda w\,x.\, w\,x\}$ to the second clause, yielding $\mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{a}))\,(\lambda w\,x.\, w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x)$. Let $\mathsf{f}\,\mathsf{a} \succ \mathsf{c}$. A SUP inference is possible between the first clause and this ground instance of the second one:

$$\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{a}))\,(\lambda w\,x.\, w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x)}{\mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x)}\,\textsc{Sup}$$

But at the nonground level, the subterm $\mathsf{f}\,\mathsf{a}$ is not clearly localized: $\mathsf{g}\,(\mathsf{f}\,\mathsf{a}) = (\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (\lambda w\,x.\, w\,x)\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\theta$. The FLUIDSUP rule can cope with this using the unifier $\{y \mapsto \lambda w\,x.\, w\,x, z \mapsto \mathsf{g}\} \in \mathrm{CSU}(z\,(\mathsf{f}\,\mathsf{a}), y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})$:

$$\dfrac{\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{h}\,(y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\,y \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x)}{\mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x) \not\approx \mathsf{h}\,(\mathsf{g}\,\mathsf{c})\,(\lambda w\,x.\, w\,x)}\,\textsc{FluidSup}}{\bot}\,\textsc{Eres}$$

Example 16 The FLUIDSUP rule is concerned not only with applied variables but also with λ-expressions that, after substitution, may be η-reduced to reveal new applied variables or green subterms. Consider the clause set consisting of $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$ and $\mathsf{h}\,(\lambda u.\, y\,u\,\mathsf{b})\,(\lambda u.\, y\,u\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})$, where $\mathsf{f}\,\mathsf{a} \succ \mathsf{c}$. Applying the substitution $\{y \mapsto \lambda u'\,v.\, \mathsf{g}\,(\mathsf{f}\,v)\,u'\}$ to the second clause yields $\mathsf{h}\,(\lambda u.\, \mathsf{g}\,(\mathsf{f}\,\mathsf{b})\,u)\,(\lambda u.\, \mathsf{g}\,(\mathsf{f}\,\mathsf{a})\,u) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})$ after β-reduction and $\mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{a})) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})$ after $\beta\eta$-reduction. A SUP inference is possible between the first clause and this new ground clause:

$$\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{a})) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}{\mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}\,\textsc{Sup}$$

Because it also considers λ-expressions, the FLUIDSUP rule applies at the nonground level to derive a corresponding nonground clause using $\{y \mapsto \lambda u'\,v.\, z\,(\mathsf{f}\,v)\,u'\} \in \mathrm{CSU}(z\,(\mathsf{f}\,\mathsf{a}), \lambda u.\, y\,u\,\mathsf{a})$:

$$\dfrac{\dfrac{\mathsf{f}\,\mathsf{a} \approx \mathsf{c} \quad \mathsf{h}\,(\lambda u.\, y\,u\,\mathsf{b})\,(\lambda u.\, y\,u\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}{\mathsf{h}\,(z\,(\mathsf{f}\,\mathsf{b}))\,(z\,\mathsf{c}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})}\,\textsc{FluidSup}}{\bot}\,\textsc{Eres}$$

Example 17 Consider the clause set consisting of the facts $C_{\mathrm{succ}} = \mathsf{succ}\, x \not\approx \mathsf{zero}$, $C_{\mathrm{div}} = n \approx \mathsf{zero} \vee \mathsf{div}\, n\, n \approx \mathsf{one}$, $C_{\mathrm{prod}} = \mathsf{prod}\, K\, (\lambda k.\, \mathsf{one}) \approx \mathsf{one}$, and the negated conjecture $C_{\mathrm{nc}} = \mathsf{prod}\, K\, (\lambda k.\, \mathsf{div}\, (\mathsf{succ}\, k)\, (\mathsf{succ}\, k)) \not\approx \mathsf{one}$. Intuitively, the term $\mathsf{prod}\, K\, (\lambda k.\, u)$ is intended to denote the product $\prod_{k \in K} u$, where k ranges over a finite set K of natural numbers. The calculus derives the empty clause as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y\,(\mathsf{diff}\langle\alpha,\beta\rangle\, y\, z) \not\approx z\,(\mathsf{diff}\langle\alpha,\beta\rangle\, y\, z) \vee y \approx z}\,\textsc{Ext} \qquad C_{\mathrm{div}}}{w\,(\mathsf{diff}\langle\alpha,\iota\rangle\,(\lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k))\, z) \approx \mathsf{zero} \vee \mathsf{one} \not\approx z\,(\mathsf{diff}\langle\alpha,\iota\rangle\,(\lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k))\, z) \vee (\lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k)) \approx z}\,\textsc{FluidSup}}{w\,(\mathsf{diff}\langle\alpha,\iota\rangle\,(\lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k))\,(\lambda k.\,\mathsf{one})) \approx \mathsf{zero} \vee (\lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k)) \approx (\lambda k.\,\mathsf{one})}\,\textsc{Eres} \qquad C_{\mathrm{succ}}}{\mathsf{zero} \not\approx \mathsf{zero} \vee (\lambda k.\,\mathsf{div}\,(\mathsf{succ}\,k)\,(\mathsf{succ}\,k)) \approx (\lambda k.\,\mathsf{one})}\,\textsc{Sup}}{(\lambda k.\,\mathsf{div}\,(\mathsf{succ}\,k)\,(\mathsf{succ}\,k)) \approx (\lambda k.\,\mathsf{one})}\,\textsc{Eres} \qquad C_{\mathrm{nc}}}{\mathsf{prod}\, K\, (\lambda k.\,\mathsf{one}) \not\approx \mathsf{one}}\,\textsc{Sup} \qquad C_{\mathrm{prod}}}{\mathsf{one} \not\approx \mathsf{one}}\,\textsc{Sup}}{\bot}\,\textsc{Eres}$$

Since the calculus does not superpose into λ-expressions, we need to use the extensionality axiom to refute this clause set. We perform a FLUIDSUP inference into the extensionality axiom with the unifier $\{\beta \mapsto \iota,\, z' \mapsto \lambda x.\, x,\, n \mapsto w\,(\mathsf{diff}\langle\alpha,\iota\rangle\,(\lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k))\, z),\, y \mapsto \lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k)\} \in \mathrm{CSU}(z'\,(\mathsf{div}\, n\, n),\, y\,(\mathsf{diff}\langle\alpha,\beta\rangle\, y\, z))$. Then we apply ERES with the unifier $\{z \mapsto \lambda k.\,\mathsf{one}\} \in \mathrm{CSU}(\mathsf{one},\, z\,(\mathsf{diff}\langle\alpha,\iota\rangle\,(\lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k))\, z))$ to eliminate the negative literal. Next, we superpose into C_{succ} with the unifier $\{\alpha \mapsto \iota,\, w \mapsto \mathsf{succ},\, x \mapsto \mathsf{diff}\langle\alpha,\iota\rangle\,(\lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k))\,(\lambda k.\,\mathsf{one})\} \in \mathrm{CSU}(w\,(\mathsf{diff}\langle\alpha,\iota\rangle\,(\lambda k.\,\mathsf{div}\,(w\,k)\,(w\,k))\,(\lambda k.\,\mathsf{one})),\, \mathsf{succ}\, x)$. To eliminate the trivial literal, we apply ERES. We then apply a SUP inference into C_{nc} and superpose into the resulting clause from C_{prod}. Finally we derive the empty clause by ERES.

Because it gives rise to flex–flex pairs—unification constraints where both sides are variable-headed—FLUIDSUP can be very prolific. With variable-headed terms on both sides of its maximal literal, the extensionality axiom is another prime source of flex–flex pairs. Flex–flex pairs can also arise in the other rules (SUP, ERES, and EFACT). Due to order restrictions and fairness, we cannot postpone solving flex–flex pairs indefinitely. Thus, we cannot use Huet's pre-unification procedure [37] and must instead choose a full unification procedure such as Jensen and Pietrzykowski's [38], Snyder and Gallier's [61], or the procedure that has recently been developed by Vukmirović, Bentkamp, and Nummelin [68]. On the positive side, optional inference rules can efficiently cover many cases where FLUIDSUP or the extensionality axiom would otherwise be needed (Sect. 5), and heuristics can help postpone the explosion. Moreover, flex–flex pairs are not always as bad as their reputation; for example, $y\,\mathsf{a}\,\mathsf{b} \stackrel{?}{=} z\,\mathsf{c}\,\mathsf{d}$ admits a most general unifier: $\{y \mapsto \lambda w\, x.\, y'\, w\, x\, \mathsf{c}\, \mathsf{d},\, z \mapsto y'\, \mathsf{a}\, \mathsf{b}\}$.

The calculus is a graceful generalization of standard superposition, except for the extensionality axiom. From simple first-order clauses, the axiom can be used to derive clauses containing λ-expressions, which are useless if the problem is first-order. For instance, the clause $\mathsf{g}\, x \approx \mathsf{f}\, x\, x$ can be used for a FLUIDSUP inference into the axiom (EXT) yielding the clause $w\, t\, (\mathsf{f}\, t\, t) \not\approx z\, t \vee (\lambda u.\, w\, u\, (\mathsf{g}\, u)) \approx z$ via the unifier $\{\alpha \mapsto \iota,\, \beta \mapsto \iota,\, x \mapsto t,\, v \mapsto \lambda u.\, w\, t\, u,\, y \mapsto \lambda u.\, w\, u\, (\mathsf{g}\, u)\} \in \mathrm{CSU}(v\,(\mathsf{g}\, x),\, y\,(\mathsf{diff}\langle\alpha,\beta\rangle\, y\, z))$ where $t = \mathsf{diff}\langle\iota,\iota\rangle\,(\lambda u.\, w\, u\, (\mathsf{g}\, u))\, z$, the variable w is freshly introduced by unification, and v is the fresh variable introduced by FLUIDSUP (named z in the definition of the rule). By ERES, with the unifier $\{z \mapsto \lambda u.\, w\, u\, (\mathsf{f}\, u\, u)\} \in$

CSU$(w\,t\,(\mathsf{f}\,t\,t), z\,t)$, we can then derive $(\lambda u.\, w\,u\,(\mathsf{g}\,u)) \approx (\lambda u.\, w\,u\,(\mathsf{f}\,u\,u))$, an equality of two λ-expressions, although we started with a simple first-order clause. This could be avoided if we could find a way to make the positive literal $y \approx z$ of (EXT) larger than the other literal, or to select $y \approx z$ without losing refutational completeness. The literal $y \approx z$ interacts only with green subterms of functional type, which do not arise in first-order clauses.

3.3 Soundness

To show soundness of the inferences, we need the substitution lemma for our logic:

Lemma 18 (Substitution lemma) *Let $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$ be a proper interpretation. Then*

$$[\![\tau\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}} \quad \text{and} \quad [\![t\rho]\!]^{\xi}_{\mathcal{J}} = [\![t]\!]^{\xi'}_{\mathcal{J}}$$

for all terms t, all types τ, and all substitutions ρ, where $\xi'(\alpha) = [\![\alpha\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$ for all type variables α and $\xi'(x) = [\![x\rho]\!]^{\xi}_{\mathcal{J}}$ for all term variables x.

Proof First, we prove that $[\![\tau\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}$ by induction on the structure of τ. If $\tau = \alpha$ is a type variable,

$$[\![\alpha\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = \xi'(\alpha) = [\![\alpha]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}$$

If $\tau = \kappa(\bar{\upsilon})$ for some type constructor κ and types $\bar{\upsilon}$,

$$[\![\kappa(\bar{\upsilon})\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}} = \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\upsilon}\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}) \overset{\text{IH}}{=} \mathcal{J}_{\mathsf{ty}}(\kappa)([\![\bar{\upsilon}]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}) = [\![\kappa(\bar{\upsilon})]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}$$

Next, we prove $[\![t\rho]\!]^{\xi}_{\mathcal{J}} = [\![t]\!]^{\xi'}_{\mathcal{J}}$ by induction on the structure of a λ-term representative of t, allowing arbitrary substitutions ρ in the induction hypothesis. If $t = y$, then by the definition of the denotation of a variable

$$[\![y\rho]\!]^{\xi}_{\mathcal{J}} = \xi'(y) = [\![y]\!]^{\xi'}_{\mathcal{J}}$$

If $t = \mathsf{f}\langle\bar{\tau}\rangle$, then by the definition of the term denotation

$$[\![\mathsf{f}\langle\bar{\tau}\rangle\rho]\!]^{\xi}_{\mathcal{J}} = \mathcal{J}(\mathsf{f}, [\![\bar{\tau}\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}) \overset{\text{IH}}{=} \mathcal{J}(\mathsf{f}, [\![\bar{\tau}]\!]^{\xi'}_{\mathcal{I}_{\mathsf{ty}}}) = [\![\mathsf{f}\langle\bar{\tau}\rangle]\!]^{\xi'}_{\mathcal{J}}$$

If $t = u\,v$, then by the definition of the term denotation

$$[\![(u\,v)\rho]\!]^{\xi}_{\mathcal{J}} = [\![u\rho]\!]^{\xi}_{\mathcal{J}}([\![v\rho]\!]^{\xi}_{\mathcal{J}}) \overset{\text{IH}}{=} [\![u]\!]^{\xi'}_{\mathcal{J}}([\![v]\!]^{\xi'}_{\mathcal{J}}) = [\![u\,v]\!]^{\xi'}_{\mathcal{J}}$$

If $t = \lambda z.\, u$, let $\rho'(z) = z$ and $\rho'(x) = \rho(x)$ for $x \neq z$. Using properness of $\mathcal{I}$ in the second and the last step, we have

$$[\![(\lambda z.\, u)\rho]\!]^{\xi}_{\mathcal{J}}(a) = [\![(\lambda z.\, u\rho')]\!]^{\xi}_{\mathcal{J}}(a) = [\![u\rho']\!]^{\xi[z\mapsto a]}_{\mathcal{J}} \overset{\text{IH}}{=} [\![u]\!]^{\xi'[z\mapsto a]}_{\mathcal{J}} = [\![\lambda z.\, u]\!]^{\xi'}_{\mathcal{J}}(a) \qquad \square$$

Lemma 19 *If $\mathcal{I} \models C$ for some interpretation $\mathcal{I}$ and some clause C, then $\mathcal{I} \models C\rho$ for all substitutions ρ.*

Proof We have to show that $C\rho$ is true in $\mathcal{I}$ for all valuations ξ. Given a valuation ξ, define ξ' as in Lemma 18. Then, by Lemma 18, a literal in $C\rho$ is true in $\mathcal{I}$ for ξ if and only if the corresponding literal in C is true in $\mathcal{I}$ for ξ'. There must be at least one such literal because $\mathcal{I} \models C$ and hence C is in particular true in $\mathcal{I}$ for ξ'. Therefore, $C\rho$ is true in $\mathcal{I}$ for ξ. $\square$

Theorem 20 (Soundness) *The inference rules* SUP, FLUIDSUP, ERES, EFACT, *and* ARGCONG *are sound (even without the variable condition and the side conditions on fluidity, deeply occurring variables, order, and eligibility).*

Proof We fix an inference and an interpretation $\mathcal{I}$ that is a model of the premises. We need to show that it is also a model of the conclusion.

From the definition of the denotation of a term, it is obvious that congruence holds in our logic, at least for subterms that are not inside a λ-expression. In particular, it holds for green subterms and for the left subterm t of an application $t\,s$.

By Lemma 19, $\mathcal{I}$ is a model of the σ-instances of the premises as well, where σ is the substitution used for the inference. Let ξ be a valuation. By making case distinctions on the truth under $\mathcal{I},\xi$ of the literals of the σ-instances of the premises, using the conditions that σ is a unifier, and applying congruence, it follows that the conclusion is true under $\mathcal{I},\xi$. Hence, $\mathcal{I}$ is a model of the conclusion. □

As in the λ-free higher-order logic of Bentkamp et al. [10], skolemization is unsound in our logic. As a consequence, axiom (EXT) does not hold in all interpretations, but the axiom is consistent with our logic, i.e., there exist models of (EXT).

3.4 The Redundancy Criterion

A redundant clause is usually defined as a clause whose ground instances are entailed by smaller ($\prec$) ground instances of existing clauses. This would be too strong for our calculus, as it would make most clauses produced by ARGCONG redundant. The solution is to base the redundancy criterion on a weaker ground logic—ground monomorphic first-order logic—in which argument congruence and extensionality do not hold. The resulting notion of redundancy gracefully generalizes the standard first-order notion.

We employ an encoding $\mathcal{F}$ to translate ground higher-order terms into ground first-order terms. $\mathcal{F}$ indexes each symbol occurrence with the type arguments and the number of term arguments. For example, $\mathcal{F}(\mathsf{f}\,\mathsf{a}) = \mathsf{f}_1(\mathsf{a}_0)$ and $\mathcal{F}(\mathsf{g}\langle\kappa\rangle) = \mathsf{g}_0^\kappa$. In addition, $\mathcal{F}$ conceals λ-expressions by replacing them with fresh symbols. These measures effectively disable argument congruence and extensionality. For example, the clause sets $\{\mathsf{g} \approx \mathsf{f},\, \mathsf{g}\,\mathsf{a} \not\approx \mathsf{f}\,\mathsf{a}\}$ and $\{\mathsf{b} \approx \mathsf{a},\, (\lambda x.\, \mathsf{b}) \not\approx (\lambda x.\, \mathsf{a})\}$ are unsatisfiable in higher-order logic, but the encoded clause sets $\{\mathsf{g}_0 \approx \mathsf{f}_0,\, \mathsf{g}_1(\mathsf{a}_0) \not\approx \mathsf{f}_1(\mathsf{a}_0)\}$ and $\{\mathsf{b}_0 \approx \mathsf{a}_0,\, \mathsf{lam}_{\lambda x.\,\mathsf{b}} \not\approx \mathsf{lam}_{\lambda x.\,\mathsf{a}}\}$ are satisfiable in first-order logic, where $\mathsf{lam}_{\lambda x.\,t}$ is a family of fresh symbols.

Given a higher-order signature $(\Sigma_{\mathsf{ty}}, \Sigma)$, we define a ground first-order signature $(\Sigma_{\mathsf{ty}}, \Sigma_{\mathrm{GF}})$ as follows. The type constructors Σ_{ty} are the same in both signatures, but $\to$ is uninterpreted in first-order logic. For each ground instance $\mathsf{f}\langle\bar{\upsilon}\rangle : \tau_1 \to \cdots \to \tau_n \to \tau$ of a symbol $\mathsf{f} \in \Sigma$, we introduce a first-order symbol $\mathsf{f}_j^{\bar{\upsilon}} \in \Sigma_{\mathrm{GF}}$ with argument types $\bar{\tau}_j$ and return type $\tau_{j+1} \to \cdots \to \tau_n \to \tau$, for each j. Moreover, for each ground term $\lambda x.\, t$, we introduce a symbol $\mathsf{lam}_{\lambda x.\,t} \in \Sigma_{\mathrm{GF}}$ of the same type.

Thus, we consider three levels of logics: the higher-order level H over a given signature $(\Sigma_{\mathsf{ty}}, \Sigma)$, the ground higher-order level GH, which is the ground fragment of H, and the ground monomorphic first-order level GF over the signature $(\Sigma_{\mathsf{ty}}, \Sigma_{\mathrm{GF}})$ defined above. We use $\mathcal{T}_{\mathrm{H}}$, $\mathcal{T}_{\mathrm{GH}}$, and $\mathcal{T}_{\mathrm{GF}}$ to denote the respective sets of terms, $\mathcal{T}\!\mathit{y}_{\mathrm{H}}$, $\mathcal{T}\!\mathit{y}_{\mathrm{GH}}$, and $\mathcal{T}\!\mathit{y}_{\mathrm{GF}}$ to denote the respective sets of types, and $\mathcal{C}_{\mathrm{H}}$, $\mathcal{C}_{\mathrm{GH}}$, and $\mathcal{C}_{\mathrm{GF}}$ to denote the respective sets of clauses. Each of the three levels has an entailment relation $\models$. A clause set N_1 entails a clause set N_2, denoted $N_1 \models N_2$, if every model of N_1 is also a model of N_2. For H and GH, we use higher-order models; for GF, we use first-order models. This machinery may seem excessive, but it is essential to define redundancy of clauses and inferences properly, and it will play an important role in the refutational completeness proof (Sect. 4).

The three levels are connected by two functions $\mathcal{G}$ and $\mathcal{F}$:

Definition 21 (Grounding function $\mathcal{G}$ on terms and clauses) The grounding function $\mathcal{G}$ maps terms $t \in \mathcal{T}_{\mathrm{H}}$ to the set of their ground instances—i.e., the set of all $t\theta \in \mathcal{T}_{\mathrm{GH}}$ where θ is a substitution. It also maps clauses $C \in \mathcal{C}_{\mathrm{H}}$ to the set of their ground instances—i.e., the set of all $C\theta \in \mathcal{C}_{\mathrm{GH}}$ where θ is a substitution.

Definition 22 (Encoding $\mathcal{F}$ on terms and clauses) The encoding $\mathcal{F} : \mathcal{T}_{\mathrm{GH}} \to \mathcal{T}_{\mathrm{GF}}$ is defined recursively as

$$\mathcal{F}(\lambda x.\, t) = \mathsf{lam}_{\lambda x.\, t} \qquad\qquad \mathcal{F}(\mathsf{f}\langle \bar{v} \rangle\, \bar{s}_j) = \mathsf{f}_j^{\bar{v}}(\mathcal{F}(\bar{s}_j))$$

using η-short β-normal representatives of terms. The encoding $\mathcal{F}$ is extended to map from $\mathcal{C}_{\mathrm{GH}}$ to $\mathcal{C}_{\mathrm{GF}}$ by mapping each literal and each side of a literal individually.

Schematically, the three levels are connected as follows:

$$\underset{\text{higher-order}}{\mathrm{H}} \xrightarrow{\quad\mathcal{G}\quad} \underset{\text{ground higher-order}}{\mathrm{GH}} \xrightarrow{\quad\mathcal{F}\quad} \underset{\text{ground first-order}}{\mathrm{GF}}$$

The mapping $\mathcal{F}$ is clearly bijective. Using the inverse mapping, the order $\succ$ can be transferred from $\mathcal{T}_{\mathrm{GH}}$ to $\mathcal{T}_{\mathrm{GF}}$ and from $\mathcal{C}_{\mathrm{GH}}$ to $\mathcal{C}_{\mathrm{GF}}$ by defining $t \succ s$ as $\mathcal{F}^{-1}(t) \succ \mathcal{F}^{-1}(s)$ and $C \succ D$ as $\mathcal{F}^{-1}(C) \succ \mathcal{F}^{-1}(D)$. As with standard superposition, $\succ$ on clauses is the multiset extension of $\succ$ on literals, which in turn is the multiset extension of $\succ$ on terms, because $\mathcal{F}^{-1}$ maps the multiset representations elementwise.

For example, let $C = y\,\mathsf{b} \approx y\,\mathsf{a} \vee y \not\approx \mathsf{f}\,\mathsf{a} \in \mathcal{C}_{\mathrm{H}}$. Then $\mathcal{G}(C)$ contains, among many other clauses, $C\theta = \mathsf{f}\,\mathsf{b}\,\mathsf{b} \approx \mathsf{f}\,\mathsf{a}\,\mathsf{a} \vee (\lambda x.\, \mathsf{f}\,x\,x) \not\approx \mathsf{f}\,\mathsf{a} \in \mathcal{C}_{\mathrm{GH}}$, where $\theta = \{y \mapsto \lambda x.\, \mathsf{f}\,x\,x\}$. On the GF level, this clause corresponds to $\mathcal{F}(C\theta) = \mathsf{f}_2(\mathsf{b}_0, \mathsf{b}_0) \approx \mathsf{f}_2(\mathsf{a}_0, \mathsf{a}_0) \vee \mathsf{lam}_{\lambda x.\, \mathsf{f}\,x\,x} \not\approx \mathsf{f}_1(\mathsf{a}_0) \in \mathcal{C}_{\mathrm{GF}}$.

A key property of $\mathcal{F}$ is that green subterms in $\mathcal{T}_{\mathrm{GH}}$ correspond to subterms in $\mathcal{T}_{\mathrm{GF}}$. This allows us to show that well-foundedness, totality on ground terms, compatibility with contexts, and the subterm property hold for $\succ$ on $\mathcal{T}_{\mathrm{GF}}$.

Lemma 23 *Let $s,t \in \mathcal{T}_{\mathrm{GH}}$. We have $\mathcal{F}(t\langle s \rangle_p) = \mathcal{F}(t)[\mathcal{F}(s)]_p$. In other words, s is a green subterm of t at green position p if and only if $\mathcal{F}(s)$ is a subterm of $\mathcal{F}(t)$ at position p.*

Proof Analogous to Lemma 3.17 of Bentkamp et al. [10]. □

Lemma 24 *Well-foundedness, totality, compatibility with contexts, and the subterm property hold for $\succ$ in $\mathcal{T}_{\mathrm{GF}}$.*

Proof Analogous to Lemma 3.19 of Bentkamp et al. [10], using Lemma 23. □

The saturation procedures of superposition provers delete clauses that are strictly subsumed by other clauses. A clause C *subsumes* D if there exists a substitution σ such that $C\sigma \subseteq D$. A clause C *strictly subsumes* D if C subsumes D but D does not subsume C. For example, $x \approx \mathsf{c}$ strictly subsumes both $\mathsf{a} \approx \mathsf{c}$ and $\mathsf{b} \not\approx \mathsf{a} \vee x \approx \mathsf{c}$. The proof of refutational completeness of resolution and superposition provers relies on the well-foundedness of the strict subsumption relation. Unfortunately, this property does not hold for higher-order logic, where $\mathsf{f}\,x\,x \approx \mathsf{c}$ is strictly subsumed by $\mathsf{f}\,(x\,\mathsf{a})\,(x\,\mathsf{b}) \approx \mathsf{c}$, which is strictly subsumed by $\mathsf{f}\,(x\,\mathsf{a}\,\mathsf{a}')\,(x\,\mathsf{b}\,\mathsf{b}') \approx \mathsf{c}$, and so on. To prevent such infinite chains, we use a well-founded partial order $\sqsupset$ on $\mathcal{C}_{\mathrm{H}}$. We can define $\sqsupset$ as $\gtrsim \cap >_{\mathrm{size}}$, where $\gtrsim$ stands for "subsumed by" and $D >_{\mathrm{size}} C$ if either $\mathit{size}(D) > \mathit{size}(C)$ or $\mathit{size}(D) = \mathit{size}(C)$ and D contains fewer distinct variables than C; the *size* function is some notion of syntactic size, such as the number of constants and variables

contained in a clause. This yields for instance $\mathsf{a} \approx \mathsf{c} \sqsupset x \approx \mathsf{c}$ and $\mathsf{f}\,(x\,\mathsf{a}\,\mathsf{a}) \approx \mathsf{c} \sqsupset \mathsf{f}\,(y\,\mathsf{a}) \approx \mathsf{c}$. To justify the deletion of subsumed clauses, we set up our redundancy criterion to cover subsumption, following Waldmann et al. [71].

We define the sets of redundant clauses w.r.t. a given clause set as follows:

- Given $C \in \mathcal{C}_{\mathrm{GF}}$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$, let $C \in \mathit{GFRed}_{\mathrm{C}}(N)$ if $\{D \in N \mid D \prec C\} \models C$.
- Given $C \in \mathcal{C}_{\mathrm{GH}}$ and $N \subseteq \mathcal{C}_{\mathrm{GH}}$, let $C \in \mathit{GHRed}_{\mathrm{C}}(N)$ if $\mathcal{F}(C) \in \mathit{GFRed}_{\mathrm{C}}(\mathcal{F}(N))$.
- Given $C \in \mathcal{C}_{\mathrm{H}}$ and $N \subseteq \mathcal{C}_{\mathrm{H}}$, let $C \in \mathit{HRed}_{\mathrm{C}}(N)$ if for every $D \in \mathcal{G}(C)$, we have $D \in \mathit{GHRed}_{\mathrm{C}}(\mathcal{G}(N))$ or there exists $C' \in N$ such that $C \sqsupset C'$ and $D \in \mathcal{G}(C')$.

For example, $(\mathsf{h}\,\mathsf{g})\,x \approx (\mathsf{h}\,\mathsf{f})\,x$ is redundant w.r.t. $\mathsf{g} \approx \mathsf{f}$, but $\mathsf{g}\,x \approx \mathsf{f}\,x$ and $(\lambda x.\,\mathsf{g}) \approx (\lambda x.\,\mathsf{f})$ are not, because $\mathcal{F}$ translates an unapplied g to g_0, whereas an applied g is translated to g_1 and the expression $\lambda x.\,\mathsf{g}$ is translated to $\mathsf{lam}_{\lambda x.\,\mathsf{g}}$. These different translations prevent entailment on the GF level. For an example of subsumption, we assume that $\mathsf{a} \approx \mathsf{c} \sqsupset x \approx \mathsf{c}$ holds, for instance using the above definition of $\sqsupset$. Then $\mathsf{a} \approx \mathsf{c}$ is redundant w.r.t. $x \approx \mathsf{c}$.

Along with the three levels of logics, we consider three inference systems: *HInf*, *GHInf*, and *GFInf*. *HInf* is the inference system described in Sect. 3.1. For uniformity, we regard the extensionality axiom as a premise-free inference rule EXT whose conclusion is axiom (EXT). The rules of *GHInf* include SUP, ERES, and EFACT from *HInf*, but with the restriction that premises and conclusion are ground and with all references to $\succsim$ replaced by $\succeq$. In addition, *GHInf* contains a premise-free rule GEXT whose infinitely many conclusions are the ground instances of (EXT), and the following ground variant of ARGCONG:

$$\frac{C' \vee s \approx s'}{C' \vee s\,\bar{u}_n \approx s'\,\bar{u}_n}\ \textsc{GArgCong}$$

where $s \approx s'$ is strictly $\succeq$-eligible in $C' \vee s \approx s'$ and $\bar{u}_n$ is a nonempty tuple of ground terms.

GFInf contains all SUP, ERES, and EFACT inferences from *GHInf* translated by $\mathcal{F}$. It coincides with standard first-order superposition.

Each of the three inference systems is parameterized by a selection function. For *HInf*, we globally fix one selection function *HSel*. For *GHInf* and *GFInf*, we need to consider different selection functions. We write $\mathit{GHInf}^{\mathit{GHSel}}$ for *GHInf* and $\mathit{GFInf}^{\mathit{GFSel}}$ for *GFInf* to make the dependency on the respective selection functions *GHSel* and *GFSel* explicit. Let $\mathcal{G}(\mathit{HSel})$ denote the set of all selection functions on $\mathcal{C}_{\mathrm{GH}}$ such that for each clause in $C \in \mathcal{C}_{\mathrm{GH}}$, there exists a clause $D \in \mathcal{C}_{\mathrm{H}}$ with $C \in \mathcal{G}(D)$ and corresponding selected literals. For each selection function *GHSel* on $\mathcal{C}_{\mathrm{GH}}$, via the bijection $\mathcal{F}$, we obtain a corresponding selection function on $\mathcal{C}_{\mathrm{GF}}$, which we denote by $\mathcal{F}(\mathit{GHSel})$.

We extend the functions $\mathcal{F}$ and $\mathcal{G}$ to inferences:

Notation 25 Given an inference ι, we write $\mathit{prems}(\iota)$ for the tuple of premises, $\mathit{mprem}(\iota)$ for the main (i.e., rightmost) premise, and $\mathit{concl}(\iota)$ for the conclusion.

Definition 26 (Encoding $\mathcal{F}$ on inferences) Given a SUP, ERES, or EFACT inference $\iota \in \mathit{GHInf}$, let $\mathcal{F}(\iota) \in \mathit{GFInf}$ denote the inference defined by $\mathit{prems}(\mathcal{F}(\iota)) = \mathcal{F}(\mathit{prems}(\iota))$ and $\mathit{concl}(\mathcal{F}(\iota)) = \mathcal{F}(\mathit{concl}(\iota))$.

Definition 27 (Grounding function $\mathcal{G}$ on inferences) Given an inference $\iota \in \mathit{HInf}$, and a selection function $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$, we define the set $\mathcal{G}^{\mathit{GHSel}}(\iota)$ of ground instances of ι to be all inferences $\iota' \in \mathit{GHInf}^{\mathit{GHSel}}$ such that $\mathit{prems}(\iota') = \mathit{prems}(\iota)\theta$ and $\mathit{concl}(\iota') = \mathit{concl}(\iota)\theta$ for some grounding substitution θ.

This will map SUP and FLUIDSUP to SUP, EFACT to EFACT, ERES to ERES, EXT to GEXT, and ARGCONG to GARGCONG inferences, but it is also possible that $\mathcal{G}^{GHSel}(\iota)$ is the empty set for some inferences ι.

We define the sets of redundant inferences w.r.t. a given clause set as follows:

- Given $\iota \in GFInf^{GFSel}$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$, let $\iota \in GFRed_{\mathrm{I}}^{GFSel}(N)$ if $prems(\iota) \cap GFRed_{\mathrm{C}}(N) \neq \varnothing$ or $\{D \in N \mid D \prec mprem(\iota)\} \models concl(\iota)$.
- Given $\iota \in GHInf^{GHSel}$ and $N \subseteq \mathcal{C}_{\mathrm{GH}}$, let $\iota \in GHRed_{\mathrm{I}}^{GHSel}(N)$ if
  - ι is not a GARGCONG or GEXT inference and $\mathcal{F}(\iota) \in GFRed_{\mathrm{I}}^{\mathcal{F}(GHSel)}(\mathcal{F}(N))$; or
  - ι is a GARGCONG or GEXT inference and $concl(\iota) \in N \cup GHRed_{\mathrm{C}}(N)$.
- Given $\iota \in HInf$ and $N \subseteq \mathcal{C}_{\mathrm{H}}$, let $\iota \in HRed_{\mathrm{I}}(N)$ if $\mathcal{G}^{GHSel}(\iota) \subseteq GHRed_{\mathrm{I}}(\mathcal{G}(N))$ for all $GHSel \in \mathcal{G}(HSel)$.

Occasionally, we omit the selection function in the notation when it is irrelevant. A clause set N is *saturated* w.r.t. an inference system and the inference component Red_{I} of a redundancy criterion if every inference from clauses in N is in $Red_{\mathrm{I}}(N)$.

3.5 Simplification Rules

The redundancy criterion $(HRed_{\mathrm{I}}, HRed_{\mathrm{C}})$ is strong enough to support most of the simplification rules implemented in Schulz's first-order prover E [57, Sections 2.3.1 and 2.3.2], some only with minor adaptions. Deletion of duplicated literals, deletion of resolved literals, syntactic tautology deletion, negative simplify-reflect, and clause subsumption adhere to our redundancy criterion.

Positive simplify-reflect and equality subsumption are supported by our criterion if they are applied in green contexts $u\langle\,\rangle$ instead of arbitrary contexts $u[\,]$. Semantic tautology deletion can be applied as well, but we must use the entailment relation of the GF level—i.e., only rewriting in green contexts can be used to establish the entailment. Similarly, rewriting of positive and negative literals (demodulation) can only be applied in green contexts. Moreover, for positive literals, the rewriting clause must be smaller than the rewritten clause—a condition that is also necessary with the standard first-order redundancy criterion but not always fulfilled by Schulz's rule. As for destructive equality resolution, even in first-order logic the rule cannot be justified with the standard redundancy criterion, and it is unclear whether it preserves refutational completeness.

As a representative example, we show how demodulation into green contexts can be justified. The justification for the other simplification rules is similar.

Lemma 28 *Demodulation into green contexts is a simplification:*

$$\frac{t \approx t' \quad \overbrace{s\langle t\sigma\rangle \mathrel{\dot{\approx}} s' \vee C'}^{C}}{t \approx t' \quad s\langle t'\sigma\rangle \mathrel{\dot{\approx}} s' \vee C'}\ \textsc{Demod}$$

where $t\sigma \succ t'\sigma$ and $C \succ (t \approx t')\sigma$. It adheres to the redundancy criterion $HRed_{\mathrm{C}}$—i.e., the deleted premise C is redundant w.r.t. the conclusions.

Proof Let N be the set consisting of the two conclusions. We must show that $C \in HRed_{\mathrm{C}}(N)$. Let $C\theta$ be a ground instance of C. By the definition of $HRed_{\mathrm{C}}$, it suffices to show that $C\theta \in GHRed_{\mathrm{C}}(\mathcal{G}(N))$. By the definition of $GHRed_{\mathrm{C}}$, we must therefore show that $\mathcal{F}(C\theta) \in$

$\mathit{GFRed}_{\mathrm{C}}(\mathcal{F}(\mathcal{G}(N)))$. By the definition of $\mathit{GFRed}_{\mathrm{C}}$, this is equivalent to proving that the clauses in $\mathcal{F}(\mathcal{G}(N))$ that are smaller than $\mathcal{F}(C\theta)$ entail $\mathcal{F}(C\theta)$.

By compatibility with green contexts and stability under grounding substitutions of $\succ$, the condition $t\sigma \succ t'\sigma$ implies that $D = \mathcal{F}((s\langle t'\sigma\rangle \mathrel{\dot{\approx}} s' \vee C')\theta)$ is a clause in $\mathcal{F}(\mathcal{G}(N))$ that is smaller than $\mathcal{F}(C\theta)$. By stability under grounding substitutions, $C \succ (t \approx t')\sigma$ implies that $E = \mathcal{F}((t \approx t')\sigma\theta)$ is another clause in $\mathcal{F}(\mathcal{G}(N))$ that is smaller than $\mathcal{F}(C\theta)$. By Lemma 23, green subterms on the GH level correspond to subterms on the GF level. Thus, $\{D, E\} \models \mathcal{F}(C\theta)$ by congruence.

3.6 A Derived Term Order

We stated some requirements on the term orders $\succ$ and $\succsim$ in Sect. 3.1 but have not shown how to fulfill them. To derive a suitable strict term order $\succ$, we propose to encode η-short β-normal forms into untyped first-order terms and apply an order $\succ_{\mathsf{fo}}$ of first-order terms such as the Knuth–Bendix order [45] or the lexicographic path order [43].

The encoding, denoted by $\mathcal{O}$, indexes symbols with their number of term arguments, similarly to the $\mathcal{F}$ encoding. Unlike the $\mathcal{F}$ encoding, $\mathcal{O}$ translates $\lambda x\colon\tau.\, t$ to $\mathsf{lam}(\mathcal{O}(\tau), \mathcal{O}(t))$ and uses De Bruijn [25] symbols to represent bound variables. The $\mathcal{O}$ encoding replaces fluid terms t by fresh variables z_t and maps type arguments to term arguments, while erasing any other type information. For example, $\mathcal{O}(\lambda x\colon\kappa.\, \mathsf{f}\,(\mathsf{f}\,(\mathsf{a}\langle\kappa\rangle))\,(y\,\mathsf{b})) = \mathsf{lam}(\kappa, \mathsf{f}_2(\mathsf{f}_1(\mathsf{a}_0(\kappa)), z_{y\mathsf{b}}))$. The use of De Bruijn indices and the monolithic encoding of fluid terms ensure stability under both α-renaming and substitution.

Definition 29 (Encoding $\mathcal{O}$) Given a signature $(\Sigma_{\mathsf{ty}}, \Sigma)$, $\mathcal{O}$ encodes types and terms as terms over the untyped first-order signature $\Sigma_{\mathsf{ty}} \uplus \{\mathsf{f}_k \mid \mathsf{f} \in \Sigma, k \in \mathbb{N}\} \uplus \{\mathsf{lam}\} \uplus \{\mathsf{db}^i_k \mid i, k \in \mathbb{N}\}$. We reuse higher-order type variables as term variables in the target untyped first-order logic. Moreover, let z_t be an untyped first-order variable for each higher-order term t. The auxiliary function $\mathcal{B}_x(t)$ replaces each free occurrence of the variable x by a symbol db^i, where i is the number of λ-expressions surrounding the variable occurrence. The type-to-term version of $\mathcal{O}$ is defined by $\mathcal{O}(\alpha) = \alpha$ and $\mathcal{O}(\kappa(\bar{\tau})) = \kappa(\mathcal{O}(\bar{\tau}))$. The term-to-term version is defined by

$$\mathcal{O}(t) = \begin{cases} z_t & \text{if } t = x \text{ or } t \text{ is fluid} \\ \mathsf{lam}(\mathcal{O}(\tau), \mathcal{O}(\mathcal{B}_x(u))) & \text{if } t = (\lambda x\colon\tau.\, u) \text{ and } t \text{ is not fluid} \\ \mathsf{f}_k(\mathcal{O}(\bar{\tau}), \mathcal{O}(\bar{u}_k)) & \text{if } t = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}_k \end{cases}$$

For example, let $s = \lambda y.\, \mathsf{f}\,y\,(\lambda w.\, \mathsf{g}\,(y\,w))$ where y has type $\kappa \to \kappa$ and w has type κ. We have $\mathcal{B}_y(\mathsf{f}\,y\,(\lambda w.\, \mathsf{g}\,(y\,w))) = \mathsf{f}\,\mathsf{db}^0(\lambda w.\, \mathsf{g}\,(\mathsf{db}^1\,w))$ and $\mathcal{B}_w(\mathsf{g}\,(\mathsf{db}^1\,w)) = \mathsf{g}\,(\mathsf{db}^1\,\mathsf{db}^0)$. Neither s nor $\lambda w.\, \mathsf{g}\,(y\,w)$ are fluid. Hence, we have $\mathcal{O}(s) = \mathsf{lam}(\to(\kappa,\kappa), \mathsf{f}_2(\mathsf{db}^0_0, \mathsf{lam}(\kappa, \mathsf{g}_1(\mathsf{db}^1_1(\mathsf{db}^0_0)))))$.

Definition 30 (Derived strict term order) Let the strict term order derived from $\succ_{\mathsf{fo}}$ be $\succ_\lambda$ where $t \succ_\lambda s$ if $\mathcal{O}(t) \succ_{\mathsf{fo}} \mathcal{O}(s)$.

We will show that the derived $\succ_\lambda$ fulfills all properties of a strict term order (Definition 6) if $\succ_{\mathsf{fo}}$ fulfills the corresponding properties on first-order terms. For the nonstrict term order $\succsim$, we can use the reflexive closure $\succeq_\lambda$ of $\succ_\lambda$.

Lemma 31 *Let $\succ_{\mathsf{fo}}$ be a strict partial order on first-order terms and $\succ_\lambda$ the derived term order on $\beta\eta$-equivalence classes. If the restriction of $\succ_{\mathsf{fo}}$ to ground terms enjoys well-foundedness, totality, the subterm property, and compatibility with contexts (w.r.t. first-order terms), the restriction of $\succ_\lambda$ to ground terms enjoys well-foundedness, totality, the green subterm property, and compatibility with green contexts (w.r.t. $\beta\eta$-equivalence classes).*

Proof Transitivity and irreflexivity of $\succ_{\mathsf{fo}}$ imply transitivity and irreflexivity of $\succ_\lambda$.

WELL-FOUNDEDNESS: If there existed an infinite chain $t_1 \succ_\lambda t_2 \succ_\lambda \cdots$ of ground terms, there would also be the chain $O(t_1) \succ_{\mathsf{fo}} O(t_2) \succ_{\mathsf{fo}} \cdots$, contradicting the well-foundedness of $\succ_{\mathsf{fo}}$ on ground λ-free terms.

TOTALITY: By ground totality of $\succ_{\mathsf{fo}}$, for any ground terms t and s we have $O(t) \succ_{\mathsf{fo}} O(s)$, $O(t) \prec_{\mathsf{fo}} O(s)$, or $O(t) = O(s)$. In the first two cases, it follows that $t \succ_\lambda s$ or $t \prec_\lambda s$. In the last case, it follows that $t = s$ because O is clearly injective.

GREEN SUBTERM PROPERTY: Let s be a term. We show that $s \succeq_\lambda s|_p$ by induction on p, where $s|_p$ denotes the green subterm at a green position p. If $p = \varepsilon$, this is trivial. If $p = p'.i$, we have $s \succeq_\lambda s|_{p'}$ by the induction hypothesis. Hence, it suffices to show that $s|_{p'} \succeq_\lambda s|_{p'.i}$. From the existence of the green position $p'.i$, we know that $s|_{p'}$ must be of the form $s|_{p'} = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}_k$. Then $s|_{p'.i} = u_i$. The encoding yields $O(s|_{p'}) = \mathsf{f}_k(O(\bar{\tau}), O(\bar{u}_k))$ and hence $O(s|_{p'}) \succeq_{\mathsf{fo}} O(s|_{p'.i})$ by the ground subterm property of $\succ_{\mathsf{fo}}$. Hence, $s|_{p'} \succeq_\lambda s|_{p'.i}$ and thus $s \succeq_\lambda s|_p$.

COMPATIBILITY WITH GREEN CONTEXTS: By induction on the depth of the context, it suffices to show that $t \succ_\lambda s$ implies $\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,t\,\bar{v} \succ_\lambda \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,s\,\bar{v}$ for all t, s, f, $\bar{\tau}$, $\bar{u}$, and $\bar{v}$. This amounts to showing that $O(t) \succ_{\mathsf{fo}} O(s)$ implies $O(\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,t\,\bar{v}) = \mathsf{f}_k(O(\bar{\tau}), O(\bar{u}), O(t), O(\bar{v})) \succ_{\mathsf{fo}} \mathsf{f}_k(O(\bar{\tau}), O(\bar{u}), O(s), O(\bar{v})) = O(\mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}\,s\,\bar{v})$, which follows directly from ground compatibility of $\succ_{\mathsf{fo}}$ with contexts and the induction hypothesis. □

Lemma 32 *Let $\succ_{\mathsf{fo}}$ be a strict partial order on first-order terms. If $\succ_{\mathsf{fo}}$ is stable under grounding substitutions (w.r.t. first-order terms), the derived term order $\succ_\lambda$ is stable under grounding substitutions (w.r.t. $\beta\eta$-equivalence classes).*

Proof Assume $s \succ_\lambda s'$ for some terms s and s'. Let θ be a higher-order substitution grounding s and s'. We must show $s\theta \succ_\lambda s'\theta$. We will define a first-order substitution ρ grounding $O(s)$ and $O(s')$ such that $O(s)\rho = O(s\theta)$ and $O(s')\rho = O(s'\theta)$. Since $s \succ_\lambda s'$, we have $O(s) \succ_{\mathsf{fo}} O(s')$. By stability of $\succ_{\mathsf{fo}}$ under grounding substitutions, $O(s)\rho \succ_{\mathsf{fo}} O(s')\rho$. It follows that $O(s\theta) \succ_{\mathsf{fo}} O(s'\theta)$ and hence $s\theta \succ_\lambda s'\theta$.

We define the first-order substitution ρ as $\alpha\rho = \alpha\theta$ for type variables α and $z_u\rho = O(u\theta)$ for terms u. Strictly speaking, the domain of a substitution must be finite, so we restrict this definition of ρ to the finitely many variables that occur in the computation of $O(s)$ and $O(s')$.

Clearly $O(\tau)\rho = O(\tau\theta)$ for all types τ occurring in the computation of $O(s)$ and $O(s')$. Moreover, $O(t)\rho = O(t\theta)$ for all t occurring in the computation of $O(s)$ and $O(s')$, which we show by induction on the definition of the encoding. If $t = x$ or if t is fluid, $O(t)\rho = z_t\rho = O(t\theta)$. If $t = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u}$, then $O(t)\rho = \mathsf{f}_k(O(\bar{\tau})\rho, O(\bar{u})\rho) \overset{\text{IH}}{=} \mathsf{f}_k(O(\bar{\tau}\theta), O(\bar{u}\theta)) = O(\mathsf{f}\langle\bar{\tau}\theta\rangle\,(\bar{u}\theta)) = O(t\theta)$. If $t = (\lambda x\colon\tau.\ u)$ and t is not fluid, then $O(t)\rho = \mathsf{lam}(O(\tau)\rho, O(\mathcal{B}_x(u))\rho) \overset{\text{IH}}{=} \mathsf{lam}(O(\tau\theta), O(\mathcal{B}_x(u)\theta)) = \mathsf{lam}(O(\tau\theta), O(\mathcal{B}_x(u)\theta[x \mapsto x])) = O(\lambda x\colon\tau\theta.\ u\theta[x \mapsto x]) = O((\lambda x\colon\tau.\ u)\theta) = O(t\theta)$. □

4 Refutational Completeness

Besides soundness, the most important property of the Boolean-free λ-superposition calculus introduced in Sect. 3 is refutational completeness. We will prove static and dynamic refutational completeness of *HInf* w.r.t. $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$, which are defined as follows. For the precise definitions of inference systems and redundancy criteria, we refer to Waldmann et al. [71].

Definition 33 (Static refutational completeness) Let *Inf* be an inference system, and let $(Red_\mathrm{I}, Red_\mathrm{C})$ be a redundancy criterion. The inference system *Inf* is *statically refutationally complete* w.r.t. $(Red_\mathrm{I}, Red_\mathrm{C})$ if we have $N \models \bot$ if and only if $\bot \in N$ for every clause set N that is saturated w.r.t. *Inf* and Red_I.

Definition 34 (Dynamic refutational completeness) Let *Inf* be an inference system, and let $(Red_\mathrm{I}, Red_\mathrm{C})$ be a redundancy criterion. Let $(N_i)_i$ be a finite or infinite sequence over sets of clauses. Such a sequence is a *derivation* if $N_i \setminus N_{i+1} \subseteq Red_\mathrm{C}(N_{i+1})$ for all i. It is *fair* if all *Inf*-inferences from clauses in the limit inferior $\bigcup_i \bigcap_{j \geq i} N_j$ are contained in $\bigcup_i Red_\mathrm{I}(N_i)$. The inference system *Inf* is *dynamically refutationally complete* w.r.t. $(Red_\mathrm{I}, Red_\mathrm{C})$ if for every fair derivation $(N_i)_i$ such that $N_0 \models \bot$, we have $\bot \in N_i$ for some i.

4.1 Outline of the Proof

The proof proceeds in three steps, corresponding to the three levels GF, GH, and H introduced in Sect. 3.4:

1. We use Bachmair and Ganzinger's work on the refutational completeness of standard (first-order) superposition [6] to prove static refutational completeness of *GFInf*.
2. From the first-order model constructed in Bachmair and Ganzinger's proof, we derive a clausal higher-order model and thus prove static refutational completeness of *GHInf*.
3. We use the saturation framework by Waldmann et al. [71] to lift the static refutational completeness of *GHInf* to static and dynamic refutational completeness of *HInf*.

In the first step, since the inference system *GFInf* is standard ground superposition, we can make use of Bachmair and Ganzinger's results. Given a saturated clause set $N \subseteq \mathcal{C}_\mathrm{GF}$ with $\bot \notin N$, Bachmair and Ganzinger prove refutational completeness by constructing a term rewriting system R_N and showing that it can be viewed as an interpretation that is a model of N. This first step deals exclusively with ground first-order clauses.

In the second step, we derive refutational completeness of *GHInf*. Given a saturated clause set $N \subseteq \mathcal{C}_\mathrm{GH}$ with $\bot \notin N$, we use the first-order model $R_{\mathcal{F}(N)}$ of $\mathcal{F}(N)$ constructed in the first step to derive a clausal higher-order interpretation that is a model of N. Under the encoding $\mathcal{F}$, occurrences of the same symbol with different numbers of arguments are regarded as different symbols—e.g., $\mathcal{F}(\mathsf{f}) = \mathsf{f}_0$ and $\mathcal{F}(\mathsf{f}\,\mathsf{a}) = \mathsf{f}_1(\mathsf{a}_0)$. All λ-expressions $\lambda x.\, t$ are regarded as uninterpreted symbols $\mathsf{lam}_{\lambda x.\, t}$. The difficulty is to construct a higher-order interpretation that merges the first-order denotations of all f_i into a single higher-order denotation of f and to show that the symbols $\mathsf{lam}_{\lambda x.\, t}$ behave like $\lambda x.\, t$. This step relies on saturation w.r.t. the GArgCong rule—which connects a term of functional type with its value when applied to an argument x—and on the presence of the extensionality rule Ext.

In the third step, we employ the saturation framework by Waldmann et al. [71], which is based on Bachmair and Ganzinger's framework [7, Sect. 4], to prove refutational completeness of *HInf*. Both frameworks help calculus designers prove static and dynamic refutational completeness of nonground calculi. In addition, the framework by Waldmann et al. explicitly supports the redundancy criterion defined in Sect. 3.4, which can be used to justify the deletion of subsumed clauses. Moreover, their framework provides completeness theorems for prover architectures, such as the DISCOUNT loop.

The main proof obligation we must discharge to use the framework is that there should exist nonground inferences in *HInf* corresponding to all nonredundant inferences in *GHInf*. We face two specifically higher-order difficulties. First, in standard superposition, we can

avoid SUP inferences into variables x by exploiting the clause order's compatibility with contexts: If $t' \prec t$, we have $C\{x \mapsto t'\} \prec C\{x \mapsto t\}$, which allows us to show that SUP inferences into variables are redundant. This technique fails for higher-order variables x that occur applied in C, because the order lacks compatibility with arguments. This is why our SUP rule must perform some inferences into variables. The other difficulty also concerns applied variables. We must show that any nonredundant SUP inference in level GH into a position corresponding to a fluid term or a deeply occurring variable in level H can be lifted to a FLUIDSUP inference. This involves showing that the z variable in FLUIDSUP can represent arbitrary contexts around a term t.

For the entire proof of refutational completeness, $\beta\eta$-normalization is the proverbial barking dog that never bites. On level GH, the rules SUP, ERES, and EFACT preserve η-short β-normal form, and so does first-order term rewriting. Thus, we can completely ignore $\longrightarrow_\beta$ and $\longrightarrow_\eta$. On level H, instantiation can cause β- and η-reduction, but this poses no difficulties thanks to the clause order's stability under grounding substitutions.

4.2 The Ground First-Order Level

We use Bachmair and Ganzinger's results on standard superposition [6] to prove refutational completeness of GF. In the subsequent steps, we will also make use of specific properties of the model Bachmair and Ganzinger construct. The basis of Bachmair and Ganzinger's proof is that a term rewriting system R defines an interpretation $\mathcal{T}_{\mathrm{GF}}/R$ such that for every ground equation $s \approx t$, we have $\mathcal{T}_{\mathrm{GF}}/R \models s \approx t$ if and only if $s \leftrightarrow_R^* t$. Formally, $\mathcal{T}_{\mathrm{GF}}/R$ denotes the monomorphic first-order interpretation whose universes $\mathcal{U}_\tau$ consist of the R-equivalence classes over $\mathcal{T}_{\mathrm{GF}}$ containing terms of type τ. The interpretation $\mathcal{T}_{\mathrm{GF}}/R$ is term-generated—that is, for every element a of the universe of this interpretation and for any valuation ξ, there exists a ground term t such that $[\![t]\!]^\xi_{\mathcal{T}_{\mathrm{GF}}/R} = a$. To lighten notation, we will write R to refer to both the term rewriting system R and the interpretation $\mathcal{T}_{\mathrm{GF}}/R$.

The term rewriting system is constructed as follows:

Definition 35 Let $N \subseteq \mathcal{C}_{\mathrm{GF}}$. We first define sets of rewrite rules E_N^C and R_N^C for all $C \in N$ by induction on the clause order. Assume that E_N^D has already been defined for all $D \in N$ such that $D \prec C$. Then $R_N^C = \bigcup_{D \prec C} E_N^D$. Let $E_N^C = \{s \longrightarrow t\}$ if the following conditions are met:

(a) $C = C' \vee s \approx t$;
(b) $s \approx t$ is $\succsim$-maximal in C;
(c) $s \succ t$;
(d) C' is false in R_N^C;
(e) s is irreducible w.r.t. R_N^C.

Then C is said to *produce* $s \longrightarrow t$. Otherwise, $E_N^C = \emptyset$. Finally, $R_N = \bigcup_D E_N^D$.

Based on Bachmair and Ganzinger's work, Bentkamp et al. [10, Lemma 4.4 and Theorem 4.5] prove the following properties of R_N:

Lemma 36 *Let $\bot \notin N$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$ be saturated w.r.t. GFInf and $\mathrm{GFRed_I}$. If $C = C' \vee s \approx t \in N$ produces $s \longrightarrow t$, then $s \approx t$ is strictly $\succeq$-eligible in C and C' is false in R_N.*

Theorem 37 (Ground first-order static refutational completeness) *The inference system GFInf is statically refutationally complete w.r.t. $(\mathrm{GFRed_I}, \mathrm{GFRed_C})$. More precisely, if $N \subseteq \mathcal{C}_{\mathrm{GF}}$ is a clause set saturated w.r.t. GFInf and $\mathrm{GFRed_I}$ such that $\bot \notin N$, then R_N is a model of N.*

4.3 The Ground Higher-Order Level

In this subsection, let *GHSel* be a selection function on $\mathcal{C}_{\mathrm{GH}}$, let $N \subseteq \mathcal{C}_{\mathrm{GH}}$ be a clause set saturated w.r.t. $\mathit{GHInf}^{\mathit{GHSel}}$ and $\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}$ such that $\bot \notin N$. Clearly, $\mathcal{F}(N)$ is then saturated w.r.t. $\mathit{GFInf}^{\mathcal{F}(\mathit{GHSel})}$ and $\mathit{GFRed}_{\mathrm{I}}^{\mathcal{F}(\mathit{GHSel})}$.

We abbreviate $R_{\mathcal{F}(N)}$ as R. Given two terms $s,t \in \mathcal{T}_{\mathrm{GH}}$, we write $s \sim t$ to abbreviate $R \models \mathcal{F}(s) \approx \mathcal{F}(t)$, which is equivalent to $[\![\mathcal{F}(s)]\!]_R = [\![\mathcal{F}(t)]\!]_R$.

Lemma 38 *For all terms $t, s : \tau \to \upsilon$ in $\mathcal{T}_{\mathrm{GH}}$, the following statements are equivalent:*

1. $t \sim s$;
2. $t\,(\mathsf{diff}\,t\,s) \sim s\,(\mathsf{diff}\,t\,s)$;
3. $t\,u \sim s\,u$ *for all* $u \in \mathcal{T}_{\mathrm{GH}}$.

Proof (3) $\Rightarrow$ (2): Take $u := \mathsf{diff}\,t\,s$.

(2) $\Rightarrow$ (1): Since N is saturated, the GEXT inference that generates the clause $C = t\,(\mathsf{diff}\,t\,s) \not\approx s\,(\mathsf{diff}\,t\,s) \vee t \approx s$ is redundant—i.e., $C \in N \cup \mathit{GHRed}_{\mathrm{C}}(N)$—and hence $R \models \mathcal{F}(C)$ by Theorem 37 and the assumption that $\bot \notin N$. Therefore, it follows from $t\,(\mathsf{diff}\,t\,s) \sim s\,(\mathsf{diff}\,t\,s)$ that $t \sim s$.

(1) $\Rightarrow$ (3): We assume that $t \sim s$—i.e., $\mathcal{F}(t) \leftrightarrow_R^* \mathcal{F}(s)$. By induction on the number of rewrite steps between $\mathcal{F}(t)$ and $\mathcal{F}(s)$ and by transitivity of $\sim$, it suffices to show that $\mathcal{F}(t) \to_R \mathcal{F}(s)$ implies $t\,u \sim s\,u$. If the rewrite step $\mathcal{F}(t) \to_R \mathcal{F}(s)$ is not at the top level, then neither $s{\downarrow_{\beta\eta}}$ nor $t{\downarrow_{\beta\eta}}$ can be λ-expressions. Therefore, $(s{\downarrow_{\beta\eta}})\,(u{\downarrow_{\beta\eta}})$ and $(t{\downarrow_{\beta\eta}})\,(u{\downarrow_{\beta\eta}})$ are in η-short β-normal form, and there is an analogous rewrite step $\mathcal{F}(t\,u) \to_R \mathcal{F}(s\,u)$ using the same rewrite rule. It follows that $t\,u \sim s\,u$. If the rewrite step $\mathcal{F}(t) \to_R \mathcal{F}(s)$ is at the top level, $\mathcal{F}(t) \to \mathcal{F}(s)$ must be a rule of R. This rule must originate from a productive clause of the form $\mathcal{F}(C) = \mathcal{F}(C' \vee t \approx s)$. By Lemma 36, $\mathcal{F}(t \approx s)$ is strictly $\succeq$-eligible in $\mathcal{F}(C)$ w.r.t. $\mathcal{F}(\mathit{GHSel})$, and hence $t \approx s$ is strictly $\succeq$-eligible in C w.r.t. *GHSel*. Thus, the following GARGCONG inference ι applies:

$$\frac{C' \vee t \approx s}{C' \vee t\,u \approx s\,u}\,\textsc{GArgCong}$$

By saturation, ι is redundant w.r.t. N—i.e., $\mathit{concl}(\iota) \in N \cup \mathit{GHRed}_{\mathrm{C}}(N)$. By Theorem 37 and the assumption that $\bot \notin N$, $\mathcal{F}(\mathit{concl}(\iota))$ is then true in R. By Lemma 36, $\mathcal{F}(C')$ is false in R. Therefore, $\mathcal{F}(t\,u \approx s\,u)$ must be true in R. □

Lemma 39 *Let $s \in \mathcal{T}_{\mathrm{H}}$ and θ, θ' grounding substitutions such that $x\theta \sim x\theta'$ for all variables x and $\alpha\theta = \alpha\theta'$ for all type variables α. Then $s\theta \sim s\theta'$.*

Proof In this proof, we work directly on λ-terms. To prove the lemma, it suffices to prove it for any λ-term s. Here, for λ-terms t_1 and t_2, the notation $t_1 \sim t_2$ is to be read as $t_1{\downarrow_{\beta\eta}} \sim t_2{\downarrow_{\beta\eta}}$ because $\mathcal{F}$ is only defined on η-short β-normal terms.

DEFINITION We extend the syntax of λ-terms with a new polymorphic function symbol $\oplus : \Pi\alpha.\, \alpha \to \alpha \to \alpha$. We will omit its type argument. It is equipped with two reduction rules: $\oplus\,t\,s \to t$ and $\oplus\,t\,s \to s$. A *$\beta\oplus$-reduction step* is either a rewrite step following one of these rules or a β-reduction step.

The computability path order $\succ_{\mathsf{CPO}}$ [22] guarantees that

- $\oplus\, t\, s \succ_{\mathsf{CPO}} s$ by applying rule @▷;
- $\oplus\, t\, s \succ_{\mathsf{CPO}} t$ by applying rule @▷ twice;
- $(\lambda x.\, t)\, s \succ_{\mathsf{CPO}} t[x \mapsto s]$ by applying rule @β.

Since this order is moreover monotone, it decreases with $\beta\oplus$-reduction steps. The order is also well founded; thus, $\beta\oplus$-reductions terminate. And since the $\beta\oplus$-reduction steps describe a finitely branching term rewriting system, by Kőnig's lemma [44], there is a maximal number of $\beta\oplus$-reduction steps from each λ-term.

DEFINITION A λ-term is *term-ground* if it does not contain free term variables. It may contain polymorphic type arguments.

DEFINITION We introduce an auxiliary function $\mathcal{S}$ that essentially measures the size of a λ-term but assigns a size of 1 to term-ground λ-terms.

$$\mathcal{S}(s) = \begin{cases} 1 & \text{if } s \text{ is term-ground or is a bound or free variable or a symbol} \\ 1 + \mathcal{S}(t) & \text{if } s \text{ is not term-ground and has the form } \lambda x.\, t \\ \mathcal{S}(t) + \mathcal{S}(u) & \text{if } s \text{ is not term-ground and has the form } t\, u \end{cases}$$

We prove $s\theta \sim s\theta'$ by well-founded induction on s, θ, and θ' using the left-to-right lexicographic order on the triple $\big(n_1(s), n_2(s), n_3(s)\big) \in \mathbb{N}^3$, where

- $n_1(s)$ is the maximal number of $\beta\oplus$-reduction steps starting from $s\sigma$, where σ is the substitution mapping each term variable x to $\oplus\, x\theta\, x\theta'$;
- $n_2(s)$ is the number of free term variables occurring more than once in s;
- $n_3(s) = \mathcal{S}(s)$.

CASE 1: The λ-term s is term-ground. Then the lemma is trivial.

CASE 2: The λ-term s contains $k \geq 2$ free term variables. Then we can apply the induction hypothesis twice and use the transitivity of $\sim$ as follows. Let x be one of the free term variables in s. Let $\rho = \{x \mapsto x\theta\}$ the substitution that maps x to $x\theta$ and ignores all other variables. Let $\rho' = \theta'[x \mapsto x]$.

We want to invoke the induction hypothesis on $s\rho$ and $s\rho'$. This is justified because $s\sigma$ $\oplus$-reduces to $s\rho\sigma$ and to $s\rho'\sigma$. These $\oplus$-reductions have at least one step because x occurs in s and $k \geq 2$. Hence, $n_1(s) > n_1(s\rho)$ and $n_1(s) > n_1(s\rho')$.

This application of the induction hypothesis gives us $s\rho\theta \sim s\rho\theta'$ and $s\rho'\theta \sim s\rho'\theta'$. Since $s\rho\theta = s\theta$ and $s\rho'\theta' = s\theta'$, this is equivalent to $s\theta \sim s\rho\theta'$ and $s\rho'\theta \sim s\theta'$. Since moreover $s\rho\theta' = s\rho'\theta$, we have $s\theta \sim s\theta'$ by transitivity of $\sim$. The following illustration visualizes the above argument:

$$\begin{array}{ccc} & s\rho & \\ \theta\swarrow & & \searrow\theta' \\ s\theta & \underset{\text{IH}}{\sim} & s\rho\theta' \end{array} \quad = \quad \begin{array}{ccc} & s\rho' & \\ \theta\swarrow & & \searrow\theta' \\ s\rho'\theta & \underset{\text{IH}}{\sim} & s\theta' \end{array}$$

CASE 3: The λ-term s contains a free term variable that occurs more than once. Then we rename variable occurrences apart by replacing each occurrence of each free term variable x by a fresh variable x_i, for which we define $x_i\theta = x\theta$ and $x_i\theta' = x\theta'$. Let s' be the resulting λ-term. Since $s\sigma = s'\sigma$, we have $n_1(s) = n_1(s')$. All free term variables occur only once in s'. Hence, $n_2(s) > 0 = n_2(s')$. Therefore, we can invoke the induction hypothesis on s' to obtain $s'\theta \sim s'\theta'$. Since $s\theta = s'\theta$ and $s\theta' = s'\theta'$, it follows that $s\theta \sim s\theta'$.

CASE 4: The λ-term s contains only one free term variable x, which occurs exactly once.

CASE 4.1: The λ-term s is of the form $\mathsf{f}\langle\bar{\tau}\rangle\,\bar{t}$ for some symbol f, some types $\bar{\tau}$, and some λ-terms $\bar{t}$. Then let u be the λ-term in $\bar{t}$ that contains x. We want to apply the induction hypothesis to u, which can be justified as follows. Consider the longest sequence of $\beta\oplus$-reductions from $u\sigma$. This sequence can be replicated inside $s\sigma = (\mathsf{f}\langle\bar{\tau}\rangle\,\bar{t})\sigma$. Therefore, the longest sequence of $\beta\oplus$-reductions from $s\sigma$ is at least as long—i.e., $n_1(s) \geq n_1(u)$. Since both s and u have only one free term variable occurrence, we have $n_2(s) = 0 = n_2(u)$. But $n_3(s) > n_3(u)$ because u is a term-nonground subterm of s.

Applying the induction hypothesis gives us $u\theta \sim u\theta'$. By definition of $\mathcal{F}$, we have $\mathcal{F}((\mathsf{f}\langle\bar{\tau}\rangle\,\bar{t})\theta) = \mathsf{f}_m^{\bar{\tau}\theta}\,\mathcal{F}(\bar{t}\theta)$ and analogously for θ', where m is the length of $\bar{t}$. By congruence of $\approx$ in first-order logic, it follows that $s\theta \sim s\theta'$.

CASE 4.2: The λ-term s is of the form $x\,\bar{t}$ for some λ-terms $\bar{t}$. Then we observe that, by assumption, $x\theta \sim x\theta'$. By applying Lemma 38 repeatedly, we have $x\theta\,\bar{t} \sim x\theta'\,\bar{t}$. Since x occurs only once, $\bar{t}$ is term-ground and hence $s\theta = x\theta\,\bar{t}$ and $s\theta' = x\theta'\,\bar{t}$. Therefore, $s\theta \sim s\theta'$.

CASE 4.3: The λ-term s is of the form $\lambda z.\,u$ for some λ-term u. Then we observe that to prove $s\theta \sim s\theta'$, it suffices to show that $s\theta\,(\mathsf{diff}\;s\theta\;s\theta') \sim s\theta'\,(\mathsf{diff}\;s\theta\;s\theta')$ by Lemma 38. Via $\beta\eta$-conversion, this is equivalent to $u\rho\theta \sim u\rho\theta'$ where $\rho = \{z \mapsto \mathsf{diff}\;(s\theta{\downarrow_{\beta\eta}})\;(s\theta'{\downarrow_{\beta\eta}})\}$. To prove $u\rho\theta \sim u\rho\theta'$, we apply the induction hypothesis on $u\rho$.

It remains to show that the induction hypothesis applies on $u\rho$. Consider the longest sequence of $\beta\oplus$-reductions from $u\rho\sigma$. Since $z\rho$ starts with the diff symbol, $z\rho$ will not cause more $\beta\oplus$-reductions than z. Hence, the same sequence of $\beta\oplus$-reductions can be applied inside $s\sigma = (\lambda z.\,u)\sigma$, proving that $n_1(s) \geq n_1(u\rho)$. Since both s and $u\rho$ have only one free term variable occurrence, $n_2(s) = 0 = n_2(u\rho)$. But $n_3(s) = \mathcal{S}(s) = 1 + \mathcal{S}(u)$ because s is term-nonground. Moreover, $\mathcal{S}(u) \geq \mathcal{S}(u\rho) = n_3(u\rho)$ because ρ replaces a variable by a ground λ-term. Hence, $n_3(s) > n_3(u\rho)$, which justifies the application of the induction hypothesis.

CASE 4.4: The λ-term s is of the form $(\lambda z.\,u)\,t_0\,\bar{t}$ for some λ-terms u, t_0, and $\bar{t}$. We apply the induction hypothesis on $s' = u\{z \mapsto t_0\}\,\bar{t}$. To justify it, consider the longest sequence of $\beta\oplus$-reductions from $s'\sigma$. Prepending the reduction $s\sigma \to_\beta s'\sigma$ to it gives us a longer sequence from $s\sigma$. Hence, $n_1(s) > n_1(s')$. The induction hypothesis gives us $s'\theta \sim s'\theta'$. Since $\sim$ is invariant under β-reductions, it follows that $s\theta \sim s\theta'$. □

We proceed by defining a higher-order interpretation $\mathcal{I}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}_{\mathsf{ty}}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$ derived from R. The interpretation R is an interpretation in monomorphic first-order logic. Let $\mathcal{U}_\tau$ be its universe for type τ and $\mathcal{J}$ its interpretation function.

To illustrate the construction, we will employ the following running example. Let the higher-order signature be $\Sigma_{\mathsf{ty}} = \{\iota, \to\}$ and $\Sigma = \{\mathsf{f} : \iota \to \iota,\, \mathsf{a} : \iota,\, \mathsf{b} : \iota\}$. The first-order signature accordingly consists of Σ_{ty} and $\Sigma_{\mathrm{GF}} = \{\mathsf{f}_0, \mathsf{f}_1, \mathsf{a}_0, \mathsf{b}_0\} \cup \{\mathsf{lam}_{\lambda x.\,t} \mid \lambda x.\,t \in \mathcal{T}_{\mathrm{GH}}\}$. We write $[t]$ for the equivalence class of $t \in \mathcal{T}_{\mathrm{GF}}$ modulo R. We assume that $[\mathsf{f}_0] = [\mathsf{lam}_{\lambda x.\,x}]$, $[\mathsf{a}_0] = [\mathsf{f}_1(\mathsf{a}_0)]$, and $[\mathsf{b}_0] = [\mathsf{f}_1(\mathsf{b}_0)]$, and that f_0, $\mathsf{lam}_{\lambda x.\,\mathsf{a}}$, $\mathsf{lam}_{\lambda x.\,\mathsf{b}_0}$, a_0, and b_0 are in disjoint equivalence classes. Hence, $\mathcal{U}_{\iota\to\iota} = \{[\mathsf{f}_0], [\mathsf{lam}_{\lambda x.\,\mathsf{a}}], [\mathsf{lam}_{\lambda x.\,\mathsf{b}}], \ldots\}$ and $\mathcal{U}_\iota = \{[\mathsf{a}_0], [\mathsf{b}_0]\}$.

When defining the universe $\mathcal{U}^{\mathrm{GH}}$ of the higher-order interpretation, we need to ensure that it contains subsets of function spaces, since $\mathcal{J}_{\mathsf{ty}}^{\mathrm{GH}}(\to)(\mathcal{D}_1, \mathcal{D}_2)$ must be a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$ for all $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}^{\mathrm{GH}}$. But the first-order universes $\mathcal{U}_\tau$ consist of equivalence classes of terms from $\mathcal{T}_{\mathrm{GF}}$ w.r.t. the rewriting system R, not of functions.

To repair this mismatch, we will define a family of functions $\mathcal{E}_\tau$ that give a meaning to the elements of the first-order universes $\mathcal{U}_\tau$. We will define a domain $\mathcal{D}_\tau$ for each ground type τ and then let $\mathcal{U}^{\mathrm{GH}}$ be the set of all these domains $\mathcal{D}_\tau$. Thus, there will be a one-to-one

correspondence between ground types and domains. Since the higher-order and first-order type signatures are identical (including $\to$, which is uninterpreted in first-order logic), we can identify higher-order and first-order types.

We define $\mathcal{E}_\tau$ and $\mathcal{D}_\tau$ in a mutual recursion. To ensure well definedness, we must simultaneously show that $\mathcal{E}_\tau$ is bijective. We start with nonfunctional types τ: Let $\mathcal{D}_\tau = \mathcal{U}_\tau$ and let $\mathcal{E}_\tau : \mathcal{U}_\tau \longrightarrow \mathcal{D}_\tau$ be the identity. Clearly, the identity is bijective. For functional types, we define

$$\mathcal{D}_{\tau\to\upsilon} = \big\{\varphi : \mathcal{D}_\tau \longrightarrow \mathcal{D}_\upsilon \mid \exists\, s : \tau \to \upsilon.\, \forall\, u : \tau.\, \varphi\big(\mathcal{E}_\tau\big(\llbracket\mathcal{F}(u)\rrbracket_R\big)\big) = \mathcal{E}_\upsilon\big(\llbracket\mathcal{F}(s\,u)\rrbracket_R\big)\big\}$$

$$\mathcal{E}_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \longrightarrow \mathcal{D}_{\tau\to\upsilon}$$

$$\mathcal{E}_{\tau\to\upsilon}(\llbracket\mathcal{F}(s)\rrbracket_R)\big(\mathcal{E}_\tau\big(\llbracket\mathcal{F}(u)\rrbracket_R\big)\big) = \mathcal{E}_\upsilon\big(\llbracket\mathcal{F}(s\,u)\rrbracket_R\big)$$

To verify that this equation is a valid definition of $\mathcal{E}_{\tau\to\upsilon}$, we must show that

- every element of $\mathcal{U}_{\tau\to\upsilon}$ is of the form $\llbracket\mathcal{F}(s)\rrbracket_R$ for some term s;
- every element of $\mathcal{D}_\tau$ is of the form $\mathcal{E}_\tau\big(\llbracket\mathcal{F}(u)\rrbracket_R\big)$ for some term u;
- the definition does not depend on the choice of such s and u;
- $\mathcal{E}_{\tau\to\upsilon}(\llbracket\mathcal{F}(s)\rrbracket_R) \in \mathcal{D}_{\tau\to\upsilon}$ for all s.

The first claim holds because R is term-generated and $\mathcal{F}$ is a bijection. The second claim holds because R is term-generated and $\mathcal{F}$ and $\mathcal{E}_\tau$ are bijections. To prove the third claim, we assume that there are other ground terms t and v such that $\llbracket\mathcal{F}(s)\rrbracket_R = \llbracket\mathcal{F}(t)\rrbracket_R$ and $\mathcal{E}_\tau\big(\llbracket\mathcal{F}(u)\rrbracket_R\big) = \mathcal{E}_\tau\big(\llbracket\mathcal{F}(v)\rrbracket_R\big)$. Since $\mathcal{E}_\tau$ is bijective, we have $\llbracket\mathcal{F}(u)\rrbracket_R = \llbracket\mathcal{F}(v)\rrbracket_R$. Using the $\sim$-notation, we can write this as $u \sim v$. Applying Lemma 39 to the term $x\,y$ and the substitutions $\{x \mapsto s, y \mapsto u\}$ and $\{x \mapsto t, y \mapsto v\}$, we obtain $s\,u \sim t\,v$—i.e., $\llbracket\mathcal{F}(s\,u)\rrbracket_R = \llbracket\mathcal{F}(t\,v)\rrbracket_R$. Thus, the definition of $\mathcal{E}_{\tau\to\upsilon}$ above does not depend on the choice of s and u. The fourth claim is obvious from the definition of $\mathcal{D}_{\tau\to\upsilon}$ and the third claim.

It remains to show that $\mathcal{E}_{\tau\to\upsilon}$ is bijective. For injectivity, we fix two terms $s, t \in \mathcal{T}_{\mathrm{GH}}$ such that for all $u \in \mathcal{T}_{\mathrm{GH}}$, we have $\llbracket\mathcal{F}(s\,u)\rrbracket_R = \llbracket\mathcal{F}(t\,u)\rrbracket_R$. By Lemma 38, $\llbracket\mathcal{F}(s)\rrbracket_R = \llbracket\mathcal{F}(t)\rrbracket_R$, which shows that $\mathcal{E}_{\tau\to\upsilon}$ is injective. For surjectivity, we fix an element $\varphi \in \mathcal{D}_{\tau\to\upsilon}$. By definition of $\mathcal{D}_{\tau\to\upsilon}$, there exists a term s such that $\varphi\big(\mathcal{E}_\tau\big(\llbracket\mathcal{F}(u)\rrbracket_R\big)\big) = \mathcal{E}_\upsilon\big(\llbracket\mathcal{F}(s\,u)\rrbracket_R\big)$ for all u. Hence, $\mathcal{E}_{\tau\to\upsilon}(\llbracket\mathcal{F}(s)\rrbracket_R) = \varphi$, proving surjectivity and therefore bijectivity of $\mathcal{E}_{\tau\to\upsilon}$. Below, we will usually write $\mathcal{E}$ instead of $\mathcal{E}_\tau$ since the type τ is determined by $\mathcal{E}_\tau$'s first argument.

In our running example, we thus have $\mathcal{D}_\iota = \mathcal{U}_\iota = \{[\mathsf{a}_0], [\mathsf{b}_0]\}$ and $\mathcal{E}_\iota$ is the identity $\mathcal{U}_\iota \longrightarrow \mathcal{D}_\iota$, $c \mapsto c$. The function $\mathcal{E}^0_{\iota\to\iota}$ maps $[\mathsf{f}_0]$ to the identity $\mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$, $c \mapsto c$; it maps $[\mathsf{lam}_{\lambda x.\,\mathsf{a}}]$ to the constant function $\mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$, $c \mapsto [\mathsf{a}_0]$; and it maps $[\mathsf{lam}_{\lambda x.\,\mathsf{b}}]$ to the constant function $\mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$, $c \mapsto [\mathsf{b}_0]$. The swapping function $[\mathsf{a}_0] \mapsto [\mathsf{b}_0], [\mathsf{b}_0] \mapsto [\mathsf{a}_0]$ is not in the image of $\mathcal{E}^0_{\iota\to\iota}$. Therefore, $\mathcal{D}_{\iota\to\iota}$ contains only the identity and the two constant functions, but not this swapping function.

We define the higher-order universe as $\mathcal{U}^{\mathrm{GH}} = \{\mathcal{D}_\tau \mid \tau \text{ ground}\}$. Moreover, we define $\mathcal{J}^{\mathrm{GH}}_{\mathsf{ty}}(\kappa)(\mathcal{D}_{\bar\tau}) = \mathcal{U}_{\kappa(\bar\tau)}$ for all $\kappa \in \Sigma_{\mathsf{ty}}$, completing the type interpretation $\mathcal{I}^{\mathrm{GH}}_{\mathsf{ty}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathsf{ty}})$. We define the interpretation function as $\mathcal{J}^{\mathrm{GH}}(\mathsf{f}, \mathcal{D}_{\bar\upsilon_m}) = \mathcal{E}(\mathcal{J}(\mathsf{f}_0^{\bar\upsilon_m}))$ for all $\mathsf{f} : \Pi\bar\alpha_m.\,\tau$.

In our example, we thus have $\mathcal{J}^{\mathrm{GH}}(\mathsf{f}) = \mathcal{E}([\mathsf{f}_0])$, which is the identity on $\mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$.

Finally, we need to define the designation function $\mathcal{L}^{\mathrm{GH}}$, which takes a valuation ξ and a λ-expression as arguments. Given a valuation ξ, we choose a grounding substitution θ such that $\mathcal{D}_{\alpha\theta} = \xi(\alpha)$ and $\mathcal{E}(\llbracket\mathcal{F}(x\theta)\rrbracket_R) = \xi(x)$ for all type variables α and all variables x. Such a substitution can be constructed as follows: We can fulfill the first equation in a unique way because there is a one-to-one correspondence between ground types and domains. Since $\mathcal{E}^{-1}(\xi(x))$ is an element of a first-order universe and R is term-generated, there exists a

ground term t such that $[\![t]\!]_R^\xi = \mathcal{E}^{-1}(\xi(x))$. Choosing one such t and defining $x\theta = \mathcal{F}^{-1}(t)$ gives us a grounding substitution θ with the desired property.

We define $\mathcal{L}^{\mathrm{GH}}(\xi,(\lambda x.\, t)) = \mathcal{E}([\![\mathcal{F}((\lambda x.\, t)\theta)]\!]_R)$. To prove that this is well defined, we assume that there exists another substitution θ' with the properties $\mathcal{D}_{\alpha\theta'} = \xi(\alpha)$ for all α and $\mathcal{E}([\![\mathcal{F}(x\theta')]\!]_R) = \xi(x)$ for all x. Then we have $\alpha\theta = \alpha\theta'$ for all α due to the one-to-one correspondence between domains and ground types. We have $[\![\mathcal{F}(x\theta)]\!]_R = [\![\mathcal{F}(x\theta')]\!]_R$ for all x because $\mathcal{E}$ is injective. By Lemma 39 it follows that $[\![\mathcal{F}((\lambda x.\, t)\theta)]\!]_R = [\![\mathcal{F}((\lambda x.\, t)\theta')]\!]_R$, which proves that $\mathcal{L}^{\mathrm{GH}}$ is well defined.

In our example, for all ξ we have $\mathcal{L}^{\mathrm{GH}}(\xi,\lambda x.\, x) = \mathcal{E}([\mathsf{lam}_{\lambda x.\, x}]) = \mathcal{E}([\mathsf{f}_0])$, which is the identity. If $\xi(y) = [\mathsf{a}_0]$, then $\mathcal{L}^{\mathrm{GH}}(\xi,\lambda x.\, y) = \mathcal{E}([\mathsf{lam}_{\lambda x.\, \mathsf{a}}])$, which is the constant function $c \mapsto [\mathsf{a}_0]$. Similarly, if $\xi(y) = [\mathsf{b}_0]$, then $\mathcal{L}^{\mathrm{GH}}(\xi,\lambda x.\, y)$ is the constant function $c \mapsto [\mathsf{b}_0]$.

This concludes the definition of the interpretation $\mathcal{I}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathsf{ty}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$. It remains to show that $\mathcal{I}^{\mathrm{GH}}$ is proper. In a proper interpretation, the denotation $[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}}$ of a term t does not depend on the representative of t modulo $\beta\eta$, but since we have not yet shown $\mathcal{I}^{\mathrm{GH}}$ to be proper, we cannot rely on this property. For this reason, we use λ-terms in the following three lemmas and mark all $\beta\eta$-reductions explicitly.

The higher-order interpretation $\mathcal{I}^{\mathrm{GH}}$ relates to the first-order interpretation R as follows:

Lemma 40 *Given a ground λ-term t, we have $[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(t{\downarrow_{\beta\eta}})]\!]_R)$.*

Proof By induction on t. Assume that $[\![s]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(s{\downarrow_{\beta\eta}})]\!]_R)$ for all proper subterms s of t. If t is of the form $\mathsf{f}\langle\bar\tau\rangle$, then

$$\begin{aligned}
[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}} &= \mathcal{J}^{\mathrm{GH}}(\mathsf{f}, \mathcal{D}_{\bar\tau}) \\
&= \mathcal{E}(\mathcal{J}(\mathsf{f}_0, \mathcal{U}_{\mathcal{F}(\bar\tau)})) \\
&= \mathcal{E}([\![\mathsf{f}_0\langle\mathcal{F}(\bar\tau)\rangle]\!]_R) \\
&= \mathcal{E}([\![\mathcal{F}(\mathsf{f}\langle\bar\tau\rangle)]\!]_R) \\
&= \mathcal{E}([\![\mathcal{F}(\mathsf{f}\langle\bar\tau\rangle{\downarrow_{\beta\eta}})]\!]_R) = \mathcal{E}([\![\mathcal{F}(t{\downarrow_{\beta\eta}})]\!]_R)
\end{aligned}$$

If t is an application $t = t_1\, t_2$, where t_1 is of type $\tau \to \upsilon$, then

$$\begin{aligned}
[\![t_1\, t_2]\!]_{\mathcal{I}^{\mathrm{GH}}} &= [\![t_1]\!]_{\mathcal{I}^{\mathrm{GH}}}([\![t_2]\!]_{\mathcal{I}^{\mathrm{GH}}}) \\
&\overset{\text{IH}}{=} \mathcal{E}_{\tau\to\upsilon}([\![\mathcal{F}(t_1{\downarrow_{\beta\eta}})]\!]_R)(\mathcal{E}_\tau([\![\mathcal{F}(t_2{\downarrow_{\beta\eta}})]\!]_R)) \\
&\overset{\text{Def } \mathcal{E}}{=} \mathcal{E}_\upsilon([\![\mathcal{F}((t_1\, t_2){\downarrow_{\beta\eta}})]\!]_R)
\end{aligned}$$

If t is a λ-expression, then

$$\begin{aligned}
[\![\lambda x.\, u]\!]^\xi_{\mathcal{I}^{\mathrm{GH}}} &= \mathcal{L}^{\mathrm{GH}}(\xi,(\lambda x.\, u)) \\
&= \mathcal{E}([\![\mathcal{F}((\lambda x.\, u)\theta{\downarrow_{\beta\eta}})]\!]_R) \\
&= \mathcal{E}([\![\mathcal{F}((\lambda x.\, u){\downarrow_{\beta\eta}})]\!]_R)
\end{aligned}$$

where θ is a substitution such that $\mathcal{D}_{\alpha\theta} = \xi(\alpha)$ and $\mathcal{E}([\![\mathcal{F}(x\theta)]\!]_R) = \xi(x)$. □

We need to show that the interpretation $\mathcal{I}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}_{\mathsf{ty}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$ is proper. In the proof, we will need to employ the following lemma, which is very similar to the substitution lemma (Lemma 18), but we must prove it here for our particular interpretation $\mathcal{I}^{\mathrm{GH}}$ because we have not shown that $\mathcal{I}^{\mathrm{GH}}$ is proper yet.

Lemma 41 (Substitution lemma) *We have $[\![\tau\rho]\!]^\xi_{\mathcal{I}^{\mathrm{GH}}_{\mathsf{ty}}} = [\![\tau]\!]^{\xi'}_{\mathcal{I}^{\mathrm{GH}}_{\mathsf{ty}}}$ and $[\![t\rho]\!]^\xi_{\mathcal{I}^{\mathrm{GH}}} = [\![t]\!]^{\xi'}_{\mathcal{I}^{\mathrm{GH}}}$ for all λ-terms t, all $\tau \in \mathcal{T}y_{\mathrm{H}}$ and all grounding substitutions ρ, where $\xi'(\alpha) = [\![\alpha\rho]\!]^\xi_{\mathcal{I}^{\mathrm{GH}}_{\mathsf{ty}}}$ for all type variables α and $\xi'(x) = [\![x\rho]\!]^\xi_{\mathcal{I}^{\mathrm{GH}}}$ for all term variables x.*

Proof We proceed by induction on the structure of τ and t. The proof is identical to the one of Lemma 18, except for the last step, which uses properness of the interpretation, a property we cannot assume here. However, here, we have the assumption that ρ is a grounding substitution. Therefore, if t is a λ-expression, we argue as follows:

$$\begin{aligned}
[\![(\lambda z.\, u)\rho]\!]^{\xi}_{\mathcal{I}^{\mathrm{GH}}} &= [\![(\lambda z.\, u\rho')]\!]^{\xi}_{\mathcal{I}^{\mathrm{GH}}} && \text{where } \rho'(z) = z \text{ and } \rho'(x) = \rho(x) \text{ for } x \neq z \\
&= \mathcal{L}^{\mathrm{GH}}(\xi, (\lambda z.\, u\rho')) && \text{by the definition of the term denotation} \\
&= \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\rho\theta{\downarrow_{\beta\eta}})]\!]^{\xi}_{R}) && \text{for some } \theta \text{ by the definition of } \mathcal{L}^{\mathrm{GH}} \\
&= \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\rho{\downarrow_{\beta\eta}})]\!]^{\xi}_{R}) && \text{because } (\lambda z.\, u)\rho \text{ is ground} \\
&\overset{*}{=} \mathcal{L}^{\mathrm{GH}}(\xi', \lambda z.\, u) && \text{by the definition of } \mathcal{L}^{\mathrm{GH}} \text{ and Lemma 40} \\
&= [\![\lambda z.\, u]\!]^{\xi'}_{\mathcal{I}^{\mathrm{GH}}} && \text{by the definition of the term denotation}
\end{aligned}$$

The step $*$ is justified as follows: We have $\mathcal{L}^{\mathrm{GH}}(\xi', \lambda z.\, u) = \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\theta'{\downarrow_{\beta\eta}})]\!]^{\xi}_{R})$ by the definition of $\mathcal{L}^{\mathrm{GH}}$, if θ' is a substitution such that $\mathcal{D}_{\alpha\theta'} = \xi'(\alpha)$ for all α and $\mathcal{E}([\![\mathcal{F}(x\theta'{\downarrow_{\beta\eta}})]\!]^{\xi}_{R}) = \xi'(x)$ for all x. By the definition of ξ' and by Lemma 40, ρ is such a substitution. Hence, $\mathcal{L}^{\mathrm{GH}}(\xi', \lambda z.\, u) = \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\rho{\downarrow_{\beta\eta}})]\!]^{\xi}_{R})$. □

Lemma 42 *The interpretation $\mathcal{I}^{\mathrm{GH}}$ is proper.*

Proof We must show that $[\![(\lambda x.\, t)]\!]^{\xi}_{\mathcal{I}^{\mathrm{GH}}}(a) = [\![t]\!]^{\xi[x\mapsto a]}_{\mathcal{I}^{\mathrm{GH}}}$ for all λ-expressions $\lambda x.\, t$, all valuations ξ, and all values a.

$$\begin{aligned}
[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{I}^{\mathrm{GH}}}(a) &= \mathcal{L}^{\mathrm{GH}}(\xi, \lambda x.\, t)(a) && \text{by the definition of } [\![\ ]\!]_{\mathcal{I}^{\mathrm{GH}}} \\
&= \mathcal{E}([\![\mathcal{F}((\lambda x.\, t)\theta{\downarrow_{\beta\eta}})]\!]_{R})(a) && \text{by the definition of } \mathcal{L}^{\mathrm{GH}} \text{ for some } \theta \\
& && \text{such that } \mathcal{E}([\![\mathcal{F}(z\theta)]\!]_{R}) = \xi(z) \text{ for all } z \\
& && \text{and } \mathcal{D}_{\alpha\theta} = \xi(\alpha) \text{ for all } \alpha \\
&= \mathcal{E}([\![\mathcal{F}(((\lambda x.\, t)\theta\ s){\downarrow_{\beta\eta}})]\!]_{R}) && \text{by the definition of } \mathcal{E} \\
& && \text{where } \mathcal{E}([\![\mathcal{F}(s)]\!]_{R}) = a \\
&= \mathcal{E}([\![\mathcal{F}(t(\theta[x \mapsto s]){\downarrow_{\beta\eta}})]\!]_{R}) && \text{by } \beta\text{-reduction} \\
&= [\![t(\theta[x \mapsto s])]\!]_{\mathcal{I}^{\mathrm{GH}}} && \text{by Lemma 40} \\
&= [\![t]\!]^{\xi[x\mapsto a]}_{\mathcal{I}^{\mathrm{GH}}} && \text{by Lemma 41}
\end{aligned}$$

□

Lemma 43 *$\mathcal{I}^{\mathrm{GH}}$ is a model of N.*

Proof By Lemma 40, we have $[\![t]\!]_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(t)]\!]_{R})$ for all $t \in \mathcal{T}_{\mathrm{GH}}$. Since $\mathcal{E}$ is a bijection, it follows that any (dis)equation $s \mathrel{\tilde{\approx}} t \in \mathcal{C}_{\mathrm{GH}}$ is true in $\mathcal{I}^{\mathrm{GH}}$ if and only if $\mathcal{F}(s \mathrel{\tilde{\approx}} t)$ is true in R. Hence, a clause $C \in \mathcal{C}_{\mathrm{GH}}$ is true in $\mathcal{I}^{\mathrm{GH}}$ if and only if $\mathcal{F}(C)$ is true in R. By Theorem 37 and the assumption that $\bot \notin N$, the interpretation R is a model of $\mathcal{F}(N)$—that is, for all clauses $C \in N$, $\mathcal{F}(C)$ is true in R. Hence, all clauses $C \in N$ are true in $\mathcal{I}^{\mathrm{GH}}$ and therefore $\mathcal{I}^{\mathrm{GH}}$ is a model of N. □

We summarize the results of this subsection in the following theorem:

Theorem 44 (Ground static refutational completeness) *Let GHSel be a selection function on $\mathcal{C}_{\mathrm{GH}}$. Then the inference system $\mathit{GHInf}^{\mathit{GHSel}}$ is statically refutationally complete w.r.t. $(\mathit{GHRed}_{\mathrm{I}}, \mathit{GHRed}_{\mathrm{C}})$. In other words, if $N \subseteq \mathcal{C}_{\mathrm{GH}}$ is a clause set saturated w.r.t. $\mathit{GHInf}^{\mathit{GHSel}}$ and $\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}$, then $N \models \bot$ if and only if $\bot \in N$.*

The construction of $\mathcal{I}^{\mathrm{GH}}$ relies on specific properties of R. It would not work with an arbitrary first-order interpretation. Transforming a higher-order interpretation into a first-order interpretation is easier:

Lemma 45 *Given a clausal higher-order interpretation $\mathcal{I}$ on* GH*, there exists a first-order interpretation $\mathcal{I}^{\mathrm{GF}}$ on* GF *such that for any clause $C \in \mathcal{C}_{\mathrm{GH}}$ the truth values of C in $\mathcal{I}$ and of $\mathcal{F}(C)$ in $\mathcal{I}^{\mathrm{GF}}$ coincide.*

Proof Let $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$ be a clausal higher-order interpretation. Let $\mathcal{U}_\tau^{\mathrm{GF}} = [\![\tau]\!]_{\mathcal{I}_{\mathsf{ty}}}$ be the first-order type universe for the ground type τ. For a symbol $\mathsf{f}_j^{\bar{\upsilon}} \in \Sigma_{\mathrm{GF}}$, let $\mathcal{J}^{\mathrm{GF}}(\mathsf{f}_j^{\bar{\upsilon}}) = [\![\mathsf{f}\langle\bar{\upsilon}\rangle]\!]_{\mathcal{J}}$ (up to currying). For a symbol $\mathsf{lam}_{\lambda x.t} \in \Sigma_{\mathrm{GF}}$, let $\mathcal{J}^{\mathrm{GF}}(\mathsf{lam}_{\lambda x.t}) = [\![\lambda x.\, t]\!]_{\mathcal{J}}$. This defines a first-order interpretation $\mathcal{I}^{\mathrm{GF}} = (\mathcal{U}^{\mathrm{GF}}, \mathcal{J}^{\mathrm{GF}})$.

We need to show that for any $C \in \mathcal{C}_{\mathrm{GH}}$, $\mathcal{I} \models C$ if and only if $\mathcal{I}^{\mathrm{GF}} \models \mathcal{F}(C)$. It suffices to show that $[\![t]\!]_{\mathcal{J}} = [\![\mathcal{F}(t)]\!]_{\mathcal{J}^{\mathrm{GF}}}$ for all terms $t \in \mathcal{T}_{\mathrm{GH}}$. We prove this by induction on the structure of the η-short β-normal form of t. If t is a λ-expression, this is obvious. If t is of the form $\mathsf{f}\langle\bar{\upsilon}\rangle\, \bar{s}_j$, then $\mathcal{F}(t) = \mathsf{f}_j^{\bar{\upsilon}}(\mathcal{F}(\bar{s}_j))$ and hence $[\![\mathcal{F}(t)]\!]_{\mathcal{J}^{\mathrm{GF}}} = \mathcal{J}^{\mathrm{GF}}(\mathsf{f}_j^{\bar{\upsilon}})([\![\mathcal{F}(\bar{s}_j)]\!]_{\mathcal{J}^{\mathrm{GF}}}) = [\![\mathsf{f}\langle\bar{\upsilon}\rangle]\!]_{\mathcal{J}}([\![\mathcal{F}(\bar{s}_j)]\!]_{\mathcal{J}^{\mathrm{GF}}}) \overset{\mathrm{IH}}{=} [\![\mathsf{f}\langle\bar{\upsilon}\rangle]\!]_{\mathcal{J}}([\![\bar{s}_j]\!]_{\mathcal{J}}) = [\![t]\!]_{\mathcal{J}}$. □

4.4 The Nonground Higher-Order Level

To lift the result to the nonground level, we employ the saturation framework of Waldmann et al. [71]. It is easy to see that the entailment relation $\models$ on GH is a consequence relation in the sense of the framework. We need to show that our redundancy criterion on GH is a redundancy criterion in the sense of the framework and that $\mathcal{G}$ is a grounding function in the sense of the framework:

Lemma 46 *The redundancy criterion for* GH *is a redundancy criterion in the sense of Sect. 2 of the saturation framework.*

Proof We must prove the conditions (R1) to (R4) of the saturation framework. Adapted to our context, they state the following for all clause sets $N, N' \subseteq \mathcal{C}_{\mathrm{GH}}$:

(R1) if $N \models \bot$, then $N \setminus \mathit{GHRed}_{\mathrm{C}}(N) \models \bot$;
(R2) if $N \subseteq N'$, then $\mathit{GHRed}_{\mathrm{C}}(N) \subseteq \mathit{GHRed}_{\mathrm{C}}(N')$ and $\mathit{GHRed}_{\mathrm{I}}(N) \subseteq \mathit{GHRed}_{\mathrm{I}}(N')$;
(R3) if $N' \subseteq \mathit{GHRed}_{\mathrm{C}}(N)$, then $\mathit{GHRed}_{\mathrm{C}}(N) \subseteq \mathit{GHRed}_{\mathrm{C}}(N \setminus N')$ and $\mathit{GHRed}_{\mathrm{I}}(N) \subseteq \mathit{GHRed}_{\mathrm{I}}(N \setminus N')$;
(R4) if $\iota \in \mathit{GHInf}$ and $\mathit{concl}(\iota) \in N$, then $\iota \in \mathit{GHRed}_{\mathrm{I}}(N)$.

The proof is analogous to the proof of Lemma 4.12 of Bentkamp et al. [10], using Lemma 45. □

Lemma 47 *The grounding functions $\mathcal{G}^{\mathit{GHSel}}$ for $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$ are grounding functions in the sense of Sect. 3 of the saturation framework.*

Proof We must prove the conditions (G1), (G2), and (G3) of the saturation framework. Adapted to our context, they state the following:

(G1) $\mathcal{G}(\bot) = \{\bot\}$;
(G2) for every $C \in \mathcal{C}_{\mathrm{H}}$, if $\bot \in \mathcal{G}(C)$, then $C = \bot$;
(G3) for every $\iota \in \mathit{HInf}$, $\mathcal{G}^{\mathit{GHSel}}(\iota) \subseteq \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(\mathit{concl}(\iota)))$.

Clearly, $C = \bot$ if and only if $\bot \in \mathcal{G}(C)$ if and only if $\mathcal{G}(C) = \{\bot\}$, proving (G1) and (G2). For every $\iota \in \mathit{HInf}$, by the definition of $\mathcal{G}^{\mathit{GHSel}}$, we have $\mathit{concl}(\mathcal{G}^{\mathit{GHSel}}(\iota)) \subseteq \mathcal{G}(\mathit{concl}(\iota))$, and thus (G3) by (R4). □

To lift the completeness result of the previous subsection to the nonground calculus *HInf*, we employ Theorem 14 of the saturation framework, which, adapted to our context, is stated as follows. The theorem uses the notation $\mathit{Inf}(N)$ to denote the set of *Inf*-inferences whose premises are in N, for an inference system *Inf* and a clause set N. Moreover, it uses Herbrand entailment $\models_{\mathcal{G}}$ on $\mathcal{C}_{\mathrm{H}}$, which is defined so that $N_1 \models_{\mathcal{G}} N_2$ if and only if $\mathcal{G}(N_1) \models \mathcal{G}(N_2)$.

Theorem 48 (Lifting theorem) *If $\mathit{GHInf}^{\mathit{GHSel}}$ is statically refutationally complete w.r.t. $(\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}, \mathit{GHRed}_{\mathrm{C}})$ for every $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$, and if for every $N \subseteq \mathcal{C}_{\mathrm{H}}$ that is saturated w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$ there exists a $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$ such that $\mathit{GHInf}^{\mathit{GHSel}}(\mathcal{G}(N)) \subseteq \mathcal{G}^{\mathit{GHSel}}(\mathit{HInf}(N)) \cup \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(N))$, then also HInf is statically refutationally complete w.r.t. $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$ and $\models_{\mathcal{G}}$.*

Proof This is almost an instance of Theorem 14 of the saturation framework. We take $\mathcal{C}_{\mathrm{H}}$ for $\mathbf{F}$, $\mathcal{C}_{\mathrm{GH}}$ for $\mathbf{G}$, and $\mathcal{G}(\mathit{HSel})$ for Q. It is easy to see that the entailment relation $\models$ on GH is a consequence relation in the sense of the framework. By Lemma 46 and 47, $(\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}, \mathit{GHRed}_{\mathrm{C}})$ is a redundancy criterion in the sense of the framework, and $\mathcal{G}^{\mathit{GHSel}}$ are grounding functions in the sense of the framework, for all $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$. The redundancy criterion $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$ matches exactly the intersected lifted redundancy criterion $\mathit{Red}^{\cap \mathcal{G}, \sqsupset}$ of the saturation framework. Theorem 14 of the saturation framework states the theorem only for $\sqsupset = \varnothing$. By Lemma 16 of the saturation framework, it also holds if $\sqsupset \neq \varnothing$. □

Let $N \subseteq \mathcal{C}_{\mathrm{H}}$ be a clause set saturated w.r.t. *HInf* and $\mathit{HRed}_{\mathrm{I}}$. We assume that *HSel* fulfills the selection restriction that a literal $L\langle y\rangle$ must not be selected if $y\,\bar{u}_n$, with $n > 0$, is a $\succeq$-maximal term of the clause, as required in Definition 9. For the above theorem to apply, we need to show that there exists a selection function $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$ such that all inferences $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ with $\mathit{prems}(\iota) \in \mathcal{G}(N)$ are liftable or redundant. Here, for ι to be *liftable* means that ι is a $\mathcal{G}^{\mathit{GHSel}}$-ground instance of a *HInf*-inference from N; for ι to be *redundant* means that $\iota \in \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(N))$.

To choose the right selection function $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$, we observe that each ground clause $C \in \mathcal{G}(N)$ must have at least one corresponding clause $D \in N$ such that C is a ground instance of D. We choose one of them for each $C \in \mathcal{G}(N)$, which we denote by $\mathcal{G}^{-1}(C)$. Then let *GHSel* select those literals in C that correspond to literals selected by *HSel* in $\mathcal{G}^{-1}(C)$. With respect to this selection function *GHSel*, we can show that all inferences from $\mathcal{G}(N)$ are liftable or redundant:

Lemma 49 *Let $\mathcal{G}^{-1}(C) = D \in N$ and $D\theta = C$. Let σ and ρ be substitutions such that $x\sigma\rho = x\theta$ for all variables x in D. Let L be a (strictly) $\succeq$-eligible literal in C w.r.t. GHSel. Then there exists a (strictly) $\succsim$-eligible literal L' in D w.r.t. σ and HSel such that $L'\theta = L$.*

Proof If $L \in \mathit{GHSel}(C)$, then there exists L' such that $L'\theta = L$ and $L' \in \mathit{HSel}(D)$ by the definition of $\mathcal{G}^{-1}$. Otherwise, L is $\succeq$-maximal in C. Since $C = D\sigma\rho$, there are literals L' in $D\sigma$ such that $L'\rho = L$. Choose L' to be a $\succsim$-maximal among them. Then L' is $\succsim$-maximal in $D\sigma$ because for any literal $L'' \in D$ with $L'' \succsim L'$, we have $L''\rho \succeq L'\rho = L$ and hence $L''\rho = L$ by $\succeq$-maximality of L.

If L is strictly $\succeq$-maximal in C, L' is also strictly $\succsim$-maximal in $D\sigma$ because a duplicate of L' in $D\sigma$ would imply a duplicate of L in C. □

Lemma 50 (Lifting of ERES, EFACT, GARGCONG, and GEXT) *All* ERES, EFACT, GARGCONG, *and* GEXT *inferences from* $\mathcal{G}(N)$ *are liftable.*

Proof ERES: Let $\iota \in GHInf^{GHSel}$ be an ERES inference with $prems(\iota) \in \mathcal{G}(N)$. Then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s\theta \not\approx s'\theta}{C'\theta}\,\text{ERES}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \not\approx s'$ and the literal $s\theta \not\approx s'\theta$ is $\succeq$-eligible w.r.t. *GHSel*. Since $s\theta$ and $s'\theta$ are unifiable and ground, we have $s\theta = s'\theta$. Thus, there exists an idempotent $\sigma \in \mathrm{CSU}(s, s')$ such that for some substitution ρ and for all variables x in C, we have $x\sigma\rho = x\theta$. By Lemma 49, we may assume without loss of generality that $s \not\approx s'$ is $\succsim$-eligible in C w.r.t. σ and *HSel*. Hence, the following inference $\iota' \in HInf$ applies:

$$\frac{C' \vee s \not\approx s'}{C'\sigma}\,\text{ERES}$$

Then ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.

EFACT: Analogously, if $\iota \in GHInf^{GHSel}$ is an EFACT inference with $prems(\iota) \in \mathcal{G}(N)$, then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s'\theta \approx t'\theta \vee s\theta \approx t\theta}{C'\theta \vee t\theta \not\approx t'\theta \vee s\theta \approx t'\theta}\,\text{EFACT}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s' \approx t' \vee s \approx t$, the literal $s\theta \approx t\theta$ is $\succeq$-eligible in C w.r.t. *GHSel*, and $s\theta \not\prec t\theta$. Then $s \not\prec t$. Moreover, $s\theta$ and $s'\theta$ are unifiable and ground. Hence, $s\theta = s'\theta$ and there exists an idempotent $\sigma \in \mathrm{CSU}(s, s')$ such that for some substitution ρ and for all variables x in C, we have $x\sigma\rho = x\theta$. By Lemma 49, we may assume without loss of generality that $s \approx t$ is $\succsim$-eligible in C w.r.t. σ and *HSel*. It follows that the following inference $\iota' \in HInf$ is applicable:

$$\frac{C' \vee s' \approx t' \vee s \approx t}{(C' \vee t \not\approx t' \vee s \approx t')\sigma}\,\text{EFACT}$$

Then ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.

GARGCONG: Let $\iota \in GHInf^{GHSel}$ be a GARGCONG inference with $prems(\iota) \in \mathcal{G}(N)$. Then ι is of the form

$$\frac{C\theta \;=\; C'\theta \vee s\theta \approx s'\theta}{C'\theta \vee s\theta\,\bar{u}_n \approx s'\theta\,\bar{u}_n}\,\text{GARGCONG}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \approx s'$, the literal $s\theta \approx s'\theta$ is strictly $\succeq$-eligible w.r.t. *GHSel*, and $s\theta$ and $s'\theta$ are of functional type. It follows that s and s' have either a functional or a polymorphic type. Let σ be the most general substitution such that $s\sigma$ and $s'\sigma$ take n arguments. By Lemma 49, we may assume without loss of generality that $s \not\approx s'$ is strictly $\succsim$-eligible in C w.r.t. σ and *HSel*. Hence the following inference $\iota' \in HInf$ is applicable:

$$\frac{C' \vee s \approx s'}{C'\sigma \vee s\sigma\,\bar{x}_n \approx s'\sigma\,\bar{x}_n}\,\text{ARGCONG}$$

Since σ is the most general substitution that ensures well-typedness of the conclusion, ι is a ground instance of ι' and is therefore liftable.

GEXT: The conclusion of a GEXT inference in *GHInf* is by definition a ground instance of the conclusion of an EXT inference in *HInf*. Hence, the GEXT inference is a ground instance of the EXT inference. Therefore it is liftable. □

Some of the SUP inferences in *GHInf* are liftable as well:

Lemma 51 (Instances of green subterms) *Let s be a λ-term in η-short β-normal form, let σ be a substitution, and let p be a green position of both s and $s\sigma{\downarrow_{\beta\eta}}$. Then $(s|_p)\sigma{\downarrow_{\beta\eta}} = (s\sigma{\downarrow_{\beta\eta}})|_p$.*

Proof By induction on p. If $p = \varepsilon$, then $(s|_p)\sigma{\downarrow_{\beta\eta}} = s\sigma{\downarrow_{\beta\eta}} = (s\sigma{\downarrow_{\beta\eta}})|_p$. If $p = i.p'$, then $s = \mathsf{f}\langle\bar{\tau}\rangle\, s_1 \ldots s_n$ and $s\sigma = \mathsf{f}\langle\bar{\tau}\sigma\rangle\,(s_1\sigma)\ldots(s_n\sigma)$, where $1 \leq i \leq n$ and p' is a green position of s_i. Clearly, $\beta\eta$-normalization steps of $s\sigma$ can take place only in proper subterms. So $s\sigma{\downarrow_{\beta\eta}} = \mathsf{f}\langle\bar{\tau}\sigma\rangle\,(s_1\sigma{\downarrow_{\beta\eta}})\ldots(s_n\sigma{\downarrow_{\beta\eta}})$. Since $p = i.p'$ is a green position of $s\sigma{\downarrow_{\beta\eta}}$, p' must be a green position of $(s_i\sigma){\downarrow_{\beta\eta}}$. By the induction hypothesis, $(s_i|_{p'})\sigma{\downarrow_{\beta\eta}} = (s_i\sigma{\downarrow_{\beta\eta}})|_{p'}$. Therefore $(s|_p)\sigma{\downarrow_{\beta\eta}} = (s|_{i.p'})\sigma{\downarrow_{\beta\eta}} = (s_i|_{p'})\sigma{\downarrow_{\beta\eta}} = (s_i\sigma{\downarrow_{\beta\eta}})|_{p'} = (s\sigma{\downarrow_{\beta\eta}})|_p$. □

Lemma 52 (Lifting of SUP) *Let $\iota \in GHInf^{GHSel}$ be a* SUP *inference*

$$\frac{\overbrace{D'\theta \vee t\theta \approx t'\theta}^{D\theta} \quad \overbrace{C'\theta \vee s\theta\langle t\theta\rangle_p \mathrel{\dot{\approx}} s'\theta}^{C\theta}}{D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle_p \mathrel{\dot{\approx}} s'\theta}\ \text{SUP}$$

where $\mathcal{G}^{-1}(D\theta) = D = D' \vee t \approx t' \in N$, $s\theta = s\theta\langle t\theta\rangle_p$, and $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \mathrel{\dot{\approx}} s' \in N$. We assume that s, t, $s\theta$, and $t\theta$ are represented by λ-terms in η-short β-normal form. Let p' be the longest prefix of p that is a green position of s. Since ε is a green position of s, the longest prefix always exists. Let $u = s|_{p'}$. Suppose one of the following conditions applies: (i) *u is a deeply occurring variable in C;* (ii) *$p = p'$ and the variable condition holds for D and C; or* (iii) *$p \neq p'$ and u is not a variable. Then ι is liftable.*

Proof The SUP inference conditions for ι are that $t\theta \approx t'\theta$ is strictly $\succeq$-eligible, $s\theta \mathrel{\dot{\approx}} s'\theta$ is strictly $\succeq$-eligible if positive and $\succeq$-eligible if negative, $D\theta \not\asymp C\theta$, $t\theta \not\asymp t'\theta$, and $s\theta \not\asymp s'\theta$. We assume that s, t, $s\theta$, and $t\theta$ are represented by λ-terms in η-short β-normal form. By Lemma 51, $u\theta$ agrees with $s\theta|_{p'}$ (considering both as terms rather than as λ-terms).

CASE 1: We have (a) $p = p'$, (b) u is not fluid, and (c) u is not a variable deeply occurring in C. Then $u\theta = s\theta|_{p'} = s\theta|_p = t\theta$. Since θ is a unifier of u and t, there exists an idempotent $\sigma \in \mathrm{CSU}(t,u)$ such that for some substitution ρ and for all variables x occurring in D and C, we have $x\sigma\rho = x\theta$. The inference conditions can be lifted: (Strict) eligibility of $t\theta \approx t'\theta$ and $s\theta \mathrel{\dot{\approx}} s'\theta$ w.r.t. *GHSel* implies (strict) eligibility of $t \approx t'$ and $s \mathrel{\dot{\approx}} s'$ w.r.t. σ and *HSel*; $D\theta \not\asymp C\theta$ implies $D \not\asymp C$; $t\theta \not\asymp t'\theta$ implies $t \not\asymp t'$; and $s\theta \not\asymp s'\theta$ implies $s \not\asymp s'$. Moreover, by (a) and (c), condition (ii) must hold and thus the variable condition holds for D and C. Hence there is the following SUP inference $\iota' \in HInf$:

$$\frac{D' \vee t \approx t' \quad C' \vee s\langle u\rangle_p \mathrel{\dot{\approx}} s'}{(D' \vee C' \vee s\langle t'\rangle_p \mathrel{\dot{\approx}} s')\sigma}\ \text{SUP}$$

Then ι is the $\sigma\rho$-ground instance of ι' and therefore liftable.

CASE 2: We have (a) $p \neq p'$, or (b) u is fluid, or (c) u is a variable deeply occurring in C. We will first show that (a) implies (b) or (c). Suppose (a) holds but neither (b) nor (c) holds.

Then condition (iii) must hold—i.e., u is not a variable. Moreover, since (b) does not hold, u cannot have the form $y\,\bar{u}_n$ for a variable y and $n \geq 1$. If u were of the form $\mathsf{f}\langle\bar{\tau}\rangle\, s_1 \ldots s_n$ with $n \geq 0$, $u\theta$ would have the form $\mathsf{f}\langle\bar{\tau}\theta\rangle\,(s_1\theta)\ldots(s_n\theta)$, but then there is some $1 \leq i \leq n$ such that $p'.i$ is a prefix of p and $s|_{p'.i}$ is a green subterm of s, contradicting the maximality of p'. So u must be a λ-expression, but since $t\theta$ is a proper green subterm of $u\theta$, $u\theta$ cannot be a λ-expression, yielding a contradiction. We may thus assume that (b) or (c) holds.

Let $p = p'.p''$. Let z be a fresh variable. Define a substitution θ' that maps this variable z to $\lambda y.\,(s\theta|_{p'})\langle y\rangle_{p''}$ and any other variable w to $w\theta$. Clearly, $(z\,t)\theta' = (s\theta|_{p'})\langle t\theta\rangle_{p''} = s\theta|_{p'} = u\theta = u\theta'$. Since θ' is a unifier of u and $z\,t$, there exists an idempotent $\sigma \in \mathrm{CSU}(z\,t, u)$ such that for some substitution ρ, for $x = z$, and for all variables x in C and D, we have $x\sigma\rho = x\theta'$. As in case 1, (strict) eligibility of the ground literals implies (strict) eligibility of the nonground literals. Moreover, by construction of θ', $t\theta' = t\theta \neq t'\theta = t'\theta'$ implies $(z\,t)\theta' \neq (z\,t')\theta'$, and thus $(z\,t)\sigma \neq (z\,t')\sigma$. Since we also have (b) or (c), there is the following inference ι':

$$\frac{D' \vee t \approx t' \quad C' \vee s\langle u\rangle_{p'} \mathrel{\dot{\approx}} s'}{(D' \vee C' \vee s\langle z\,t'\rangle_{p'} \mathrel{\dot{\approx}} s')\sigma}\ \textsc{FluidSup}$$

Then ι is the $\sigma\rho$-ground instance of ι' and therefore liftable. □

The other SUP inferences might not be liftable, but they are redundant:

Lemma 53 *Let $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ be a* SUP *inference from $\mathcal{G}(N)$ not covered by Lemma 52. Then $\iota \in \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(N))$.*

Proof Let $C\theta = C'\theta \vee s\theta \approx s'\theta$ and $D\theta = D'\theta \vee t\theta \approx t'\theta$ be the premises of ι, where $s\theta \approx s'\theta$ and $t\theta \approx t'\theta$ are the literals involved in the inference, $s\theta \succ s'\theta$, $t\theta \succ t'\theta$, and C', D', s, s', t, t' are the respective subclauses and terms in $C = \mathcal{G}^{-1}(C\theta)$ and $D = \mathcal{G}^{-1}(D\theta)$. Then the inference ι has the form

$$\frac{D'\theta \vee t\theta \approx t'\theta \quad C'\theta \vee s\theta\langle t\theta\rangle \approx s'\theta}{D'\theta \vee C'\theta \vee s\theta\langle t'\theta\rangle \approx s'\theta}\ \textsc{Sup}$$

To show that $\iota \in \mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(N))$, it suffices to show $\{D \in \mathcal{F}(\mathcal{G}(N)) \mid D \prec \mathcal{F}(C\theta)\} \models \mathcal{F}(\mathit{concl}(\iota))$. To this end, let $\mathcal{I}$ be an interpretation in GF such that $\mathcal{I} \models \{D \in \mathcal{F}(\mathcal{G}(N)) \mid D \prec \mathcal{F}(C\theta)\}$. We need to show that $\mathcal{I} \models \mathcal{F}(\mathit{concl}(\iota))$. If $\mathcal{F}(D'\theta)$ is true in $\mathcal{I}$, then obviously $\mathcal{I} \models \mathcal{F}(\mathit{concl}(\iota))$. So we assume that $\mathcal{F}(D'\theta)$ is false in $\mathcal{I}$. Since $C\theta \succ D\theta$ by the SUP order conditions, it follows that $\mathcal{I} \models \mathcal{F}(t\theta \approx t'\theta)$. Therefore, it suffices to show $\mathcal{I} \models \mathcal{F}(C\theta)$.

Let p be the green position in $s\theta$ where ι takes place and p' be the longest prefix of p that is a green subterm of s. Let $u = s|_{p'}$. Since Lemma 52 does not apply to ι, u is not a deeply occurring variable; if $p = p'$, the variable condition does not hold for D and C; and if $p \neq p'$, u is a variable. This means either the green position p does not exist in s, because it is below an unapplied variable that does not occur deeply in C, or $s|_p$ is an unapplied variable that does not occur deeply in C and for which the variable condition does not hold.

CASE 1: The green position p does not exist in s because it is below a variable x that does not occur deeply in C. Then $t\theta$ is a green subterm of $x\theta$ and hence a green subterm of $x\theta\,\bar{w}$ for any arguments $\bar{w}$. Let v be the term that we obtain by replacing $t\theta$ by $t'\theta$ in $x\theta$ at the relevant position. Since $\mathcal{I} \models \mathcal{F}(t\theta \approx t'\theta)$, by congruence, $\mathcal{I} \models \mathcal{F}(x\theta\,\bar{w} \approx v\,\bar{w})$ for any arguments $\bar{w}$. Hence, $\mathcal{I} \models \mathcal{F}(C\theta)$ if and only if $\mathcal{I} \models \mathcal{F}(C\{x \mapsto v\}\theta)$ by congruence. Here, it is crucial that the variable does not occur deeply in C because congruence does not hold in $\mathcal{F}$-encoded terms below λ-binders. By the inference conditions, we have $t\theta \succ t'\theta$, which

implies $\mathcal{F}(C\theta) \succ \mathcal{F}(C\{x \mapsto v\}\theta)$ by compatibility with green contexts. Therefore, by the assumption about $\mathcal{I}$, we have $\mathcal{I} \models \mathcal{F}(C\{x \mapsto v\}\theta)$ and hence $\mathcal{I} \models \mathcal{F}(C\theta)$.

CASE 2: The term $s|_p$ is a variable x that does not occur deeply in C and for which the variable condition does not hold. From this, we know that $C\theta \succeq C''\theta$, where $C'' = C\{x \mapsto t'\}$. We cannot have $C\theta = C''\theta$ because $x\theta = t\theta \neq t'\theta$ and x occurs in C. Hence, we have $C\theta \succ C''\theta$. By the definition of $\mathcal{I}$, $C\theta \succ C''\theta$ implies $\mathcal{I} \models \mathcal{F}(C''\theta)$. We will use equalities that are true in $\mathcal{I}$ to rewrite $\mathcal{F}(C\theta)$ into $\mathcal{F}(C''\theta)$, which implies $\mathcal{I} \models \mathcal{F}(C\theta)$ by congruence.

By saturation, every ARGCONG inference ι' from D is in $\mathit{HRed}_\mathrm{I}(N)$—i.e., $\mathcal{G}(\mathit{concl}(\iota')) \subseteq \mathcal{G}(N) \cup \mathit{GHRed}_\mathrm{C}(\mathcal{G}(N))$. Hence, $D'\theta \vee t\theta\,\bar{u} \approx t'\theta\,\bar{u}$ is in $\mathcal{G}(N) \cup \mathit{GHRed}_\mathrm{C}(\mathcal{G}(N))$ for any ground arguments $\bar{u}$.

We observe that whenever $t\theta\,\bar{u}$ and $t'\theta\,\bar{u}$ are smaller than the $\succeq$-maximal term of $C\theta$ for some arguments $\bar{u}$, we have

$$\mathcal{I} \models \mathcal{F}(t\theta\,\bar{u}) \approx \mathcal{F}(t'\theta\,\bar{u}) \qquad (*)$$

To show this, we assume that $t\theta\,\bar{u}$ and $t'\theta\,\bar{u}$ are smaller than the $\succeq$-maximal term of $C\theta$ and we distinguish two cases: If $t\theta$ is smaller than the $\succeq$-maximal term of $C\theta$, all terms in $D'\theta$ are smaller than the $\succeq$-maximal term of $C\theta$ and hence $D'\theta \vee t\theta\,\bar{u} \approx t'\theta\,\bar{u} \prec C\theta$. If, on the other hand, $t\theta$ is equal to the $\succeq$-maximal term of $C\theta$, then $t\theta\,\bar{u}$ and $t'\theta\,\bar{u}$ are smaller than $t\theta$. Hence $t\theta\,\bar{u} \approx t'\theta\,\bar{u} \prec t\theta \approx t'\theta$ and $D'\theta \vee t\theta\,\bar{u} \approx t'\theta\,\bar{u} \prec D\theta \prec C\theta$. In both cases, since $D'\theta$ is false in $\mathcal{I}$, by the definition of $\mathcal{I}$, we have $(*)$.

Next, we show the equivalence of $C\theta$ and $C''\theta$ via rewriting with equations of the form $(*)$ where $t\theta\,\bar{u}$ and $t'\theta\,\bar{u}$ are smaller than the $\succeq$-maximal term of $C\theta$. Since x does not occur deeply in C, every occurrence of x in C is not inside a λ-expression and not inside an argument of an applied variable. Therefore, all occurrences of x in C are in a green subterm of the form $x\,\bar{v}$ for some terms $\bar{v}$ that do not contain x. Hence, every occurrence of x in C corresponds to a subterm $\mathcal{F}((x\,\bar{v})\theta) = \mathcal{F}(t\theta\,\bar{v}\theta)$ in $\mathcal{F}(C\theta)$ and to a subterm $\mathcal{F}((x\,\bar{v})\{x \mapsto t'\}\theta) = \mathcal{F}(t'\theta\,\bar{v}\{x \mapsto t'\}\theta) = \mathcal{F}(t'\theta\,\bar{v}\theta)$ in $\mathcal{F}(C''\theta)$. These are the only positions where $C\theta$ and $C''\theta$ differ.

To justify the necessary rewrite steps from $\mathcal{F}(t\theta\,\bar{v}\theta)$ into $\mathcal{F}(t'\theta\,\bar{v}\theta)$ using $(*)$, we must show that $\mathcal{F}(t\theta\,\bar{v}\theta)$ and $\mathcal{F}(t'\theta\,\bar{v}\theta)$ are smaller than the $\succeq$-maximal term in $\mathcal{F}(C\theta)$ for the relevant $\bar{v}$. If $\bar{v}$ is the empty tuple, we do not need to show this because $\mathcal{I} \models \mathcal{F}(t\theta \approx t'\theta)$ follows from $\mathcal{F}(D\theta)$'s being true and $\mathcal{F}(D'\theta)$'s being false. If $\bar{v}$ is nonempty, it suffices to show that $x\,\bar{v}$ is not a $\succeq$-maximal term in C. Then $\mathcal{F}(t\theta\,\bar{v}\theta)$ and $\mathcal{F}(t'\theta\,\bar{v}\theta)$, which correspond to the term $x\,\bar{v}$ in C, cannot be $\succeq$-maximal in $\mathcal{F}(C\theta)$ and $\mathcal{F}(C''\theta)$. Hence they must be smaller than the $\succeq$-maximal term in $\mathcal{F}(C\theta)$ because they are subterms of $\mathcal{F}(C\theta)$ and $\mathcal{F}(C''\theta) \prec \mathcal{F}(C\theta)$, respectively.

To show that $x\,\bar{v}$ is not a $\succeq$-maximal term in C, we make a case distinction on whether $s\theta \mathrel{\tilde{\approx}} s'\theta$ is selected in $C\theta$ or $s\theta$ is the $\succeq$-maximal term in $C\theta$. One of these must hold because $s\theta \mathrel{\tilde{\approx}} s'\theta$ is $\succeq$-eligible in $C\theta$. If it is selected, by the selection restrictions, x cannot be the head of a $\succeq$-maximal term of C. If $s\theta$ is the $\succeq$-maximal term in $C\theta$, we can argue that x is a green subterm of s and, since x does not occur deeply, s cannot be of the form $x\,\bar{v}$ for a nonempty $\bar{v}$. This justifies the necessary rewrites between $\mathcal{F}(C\theta)$ and $\mathcal{F}(C''\theta)$ and it follows that $\mathcal{I} \models \mathcal{F}(C\theta)$. □

With these properties of our inference systems in place, Theorem 48 guarantees static and dynamic refutational completeness of *HInf* w.r.t. HRed_I. However, this theorem gives us refutational completeness w.r.t. the Herbrand entailment $\models_\mathcal{G}$, defined so that $N_1 \models_\mathcal{G} N_2$ if $\mathcal{G}(N_1) \models \mathcal{G}(N_2)$, whereas our semantics is Tarski entailment $\models$, defined so that $N_1 \models N_2$ if any model of N_1 is a model of N_2. To repair this mismatch, we use the following lemma,

which can be proved along the lines of Lemma 4.19 of Bentkamp et al. [10], using Lemma 18 and Lemma 19.

Lemma 54 *For $N \subseteq \mathcal{C}_{\mathrm{H}}$, we have $N \models_{\mathcal{G}} \bot$ if and only if $N \models \bot$.*

Theorem 55 (Static refutational completeness) *The inference system HInf is statically refutationally complete w.r.t. $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$. In other words, if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. HInf and $\mathit{HRed}_{\mathrm{I}}$, then we have $N \models \bot$ if and only if $\bot \in N$.*

Proof We apply Theorem 48. By Theorem 44, $\mathit{GHInf}^{\mathit{GHSel}}$ is statically refutationally complete for all $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$. By Lemmas 50, 52, and 53, for every saturated $N \subseteq \mathcal{C}_{\mathrm{H}}$, there exists a selection function $\mathit{GHSel} \in \mathcal{G}(\mathit{HSel})$ such that all inferences $\iota \in \mathit{GHInf}^{\mathit{GHSel}}$ with $\mathit{prems}(\iota) \in \mathcal{G}(N)$ either are $\mathcal{G}^{\mathit{GHSel}}$-ground instances of *HInf*-inferences from N or belong to $\mathit{GHRed}_{\mathrm{I}}^{\mathit{GHSel}}(\mathcal{G}(N))$.

Theorem 48 implies that if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. *HInf* and $\mathit{HRed}_{\mathrm{I}}$, then $N \models_{\mathcal{G}} \bot$ if and only if $\bot \in N$. By Lemma 54, this also holds for the Tarski entailment $\models$. That is, if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. *HInf* and $\mathit{HRed}_{\mathrm{I}}$, then $N \models \bot$ if and only if $\bot \in N$. □

From static completeness, we can easily derive dynamic completeness:

Theorem 56 (Dynamic refutational completeness) *The inference system HInf is dynamically refutationally complete w.r.t. $(\mathit{HRed}_{\mathrm{I}}, \mathit{HRed}_{\mathrm{C}})$ as specified in Definition 34.*

Proof By Theorem 17 of the saturation framework, this follows from Theorem 55 and Lemma 54. □

5 Extensions

In addition to the simplification rules presented in Section 3.5, the core calculus can be extended with various optional rules for higher-order reasoning. Like the previous rules, they are not necessary for refutational completeness but can allow the prover to find more direct proofs. Most of these rules are concerned with the areas covered by the FLUIDSUP rule and the extensionality axiom.

Two of the optional rules below rely on the notion of "orange subterms."

Definition 57 A λ-term t is an *orange subterm* of a λ-term s if $s = t$; or if $s = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{s}$ and t is an orange subterm of s_i for some i; or if $s = x\,\bar{s}$ and t is an orange subterm of s_i for some i; or if $s = (\lambda x.\, u)$ and t is an orange subterm of u.

For example, in the term $\mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,(y\,\mathsf{b})\,(\lambda x.\,\mathsf{h}\,\mathsf{c}\,(\mathsf{g}\,x))$, the orange subterms are all the green subterms—a, $\mathsf{g}\,\mathsf{a}$, $y\,\mathsf{b}$, $\lambda x.\,\mathsf{h}\,\mathsf{c}\,(\mathsf{g}\,x)$ and the whole term—and in addition b, c, x, $\mathsf{g}\,x$, and $\mathsf{h}\,\mathsf{c}\,(\mathsf{g}\,x)$. Following Convention 1, this notion is lifted to $\beta\eta$-equivalence classes via representatives in η-short β-normal form. We write $t = s\langle\!\langle \bar{x}_n.\, u \rangle\!\rangle$ to indicate that u is an orange subterm of t, where $\bar{x}_n$ are the variables bound in the *orange context* around u, from outermost to innermost. If $n = 0$, we simply write $t = s\langle\!\langle u \rangle\!\rangle$.

Once a term $s\langle\!\langle \bar{x}_n.\, u \rangle\!\rangle$ has been introduced, we write $s\langle\!\langle \bar{x}_n.\, u' \rangle\!\rangle_\eta$ to denote the same context with a different subterm u' at that position. The η subscript is a reminder that u' is not necessarily an orange subterm of $s\langle\!\langle \bar{x}_n.\, u' \rangle\!\rangle_\eta$ due to potential applications of η-reduction. For example, if $s\langle\!\langle x.\,\mathsf{g}\,x\,x \rangle\!\rangle = \mathsf{h}\,\mathsf{a}\,(\lambda x.\,\mathsf{g}\,x\,x)$, then $s\langle\!\langle x.\,\mathsf{f}\,x \rangle\!\rangle_\eta = \mathsf{h}\,\mathsf{a}\,(\lambda x.\,\mathsf{f}\,x) = \mathsf{h}\,\mathsf{a}\,\mathsf{f}$.

Demodulation in Orange Contexts Demodulation, which destructively rewrites using an equality $t \approx t'$, is available at green positions, as described in Section 3.5. In addition, a variant of demodulation rewrites in orange contexts:

$$\frac{t \approx t' \quad C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle}{t \approx t' \quad C\langle s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\rangle \quad s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle \approx s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta} \lambda\textsc{DemodExt}$$

where the term $t\sigma$ may refer to the bound variables $\bar{x}$. The following side conditions apply:

1. $s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\downarrow_{\beta\eta}$ is a λ-expression or a term of the form $y\,\bar{u}_n$ with $n > 0$;
2. $s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle \succ s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta$;
3. $C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle \succ s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle \approx s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta$

Condition 3 ensures that the second premise is redundant w.r.t. the conclusions and may be removed. The double bar indicates that the conclusions collectively make the premises redundant and can replace them.

The third conclusion, which is entailed by $t \approx t'$ and (EXT), could be safely omitted if the corresponding (EXT) instance is smaller than the second premise. But in general, the third conclusion is necessary for the proof, and the variant of λDEMODEXT that omits it—let us call it λDEMOD—might not preserve refutational completeness.

An instance of λDEMODEXT, where $\mathsf{g}\,z$ is rewritten to $\mathsf{f}\,z\,z$ under a λ-binder, follows:

$$\frac{\mathsf{g}\,x \approx \mathsf{f}\,x\,x \quad \mathsf{k}\,(\lambda z.\,\mathsf{h}\,(\mathsf{g}\,z)) \approx \mathsf{c}}{\mathsf{g}\,x \approx \mathsf{f}\,x\,x \quad \mathsf{k}\,(\lambda z.\,\mathsf{h}\,(\mathsf{f}\,z\,z)) \approx \mathsf{c} \quad (\lambda z.\,\mathsf{h}\,(\mathsf{g}\,z)) \approx (\lambda z.\,\mathsf{h}\,(\mathsf{f}\,z\,z))} \lambda\textsc{DemodExt}$$

Lemma 58 *λDEMODEXT is sound and preserves refutational completeness of the calculus.*

Proof Soundness of the first conclusion is obvious. Soundness of the second and third conclusions follows from congruence and extensionality using the premises. Preservation of completeness is justified by redundancy. Specifically, we justify the deletion of the second premise by showing that it is redundant w.r.t. the conclusions—i.e., if for every ground instance $C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle\theta \in \mathcal{G}(C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle)$, its encoding $\mathcal{F}(C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle\theta)$ is entailed by $\mathcal{F}(\mathcal{G}(N))$, where N are the conclusions of λDEMODEXT. The first conclusion cannot help us prove redundancy because $s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\theta\downarrow_{\beta\eta}$ might be a λ-expression and then $\mathcal{F}(s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\theta)$ is a symbol that is unrelated to $\mathcal{F}(t\sigma\theta)$. Instead, we use the θ-instances of the last two conclusions. By Lemma 23, $\mathcal{F}(C\langle s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\rangle\theta)$ has $\mathcal{F}(s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\theta)$ as a subterm. If this subterm is replaced by $\mathcal{F}(s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\theta)$, we obtain $\mathcal{F}(C\langle s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\rangle\theta)$. Hence, the $\mathcal{F}$-encodings of the θ-instances of the last two conclusions entail the $\mathcal{F}$-encoding of the θ-instance of the second premise by congruence. Due to the side condition that the second premise is larger than the second and third conclusion, by stability under grounding substitutions, the θ-instances of the last two conclusions must be smaller than the θ-instance of the second premise. Thus, the second premise is redundant. □

Pruning Arguments of Variables The next simplification rule can be used to prune arguments of applied variables if the arguments can be expressed as functions of the remaining arguments. For example, the clause $C[y\,\mathsf{a}\,\mathsf{b}\,(\mathsf{f}\,\mathsf{b}\,\mathsf{a}),\, y\,\mathsf{b}\,\mathsf{d}\,(\mathsf{f}\,\mathsf{d}\,\mathsf{b})]$, in which y occurs twice, can be simplified to $C[y'\,\mathsf{a}\,\mathsf{b},\, y'\,\mathsf{b}\,\mathsf{d}]$. Here, for each occurrence of y, the third argument can be computed by applying f to the second and first arguments. The rule can also be used to remove the repeated arguments in $y\,\mathsf{b}\,\mathsf{b} \not\approx y\,\mathsf{a}\,\mathsf{a}$, the static argument a in $y\,\mathsf{a}\,\mathsf{c} \not\approx y\,\mathsf{a}\,\mathsf{b}$, and all four arguments in $y\,\mathsf{a}\,\mathsf{b} \not\approx z\,\mathsf{b}\,\mathsf{d}$. It is stated as

$$\frac{C}{C\sigma}\textsc{PruneArg}$$

where the following conditions apply:

1. $\sigma = \{y \mapsto \lambda\bar{x}_j.\, y'\, \bar{x}_{j-1}\}$; 2. y' is a fresh variable; 3. $C \sqsupset C\sigma$;
4. the minimum number k of arguments passed to any occurrence of y in the clause C is at least j;
5. there exists a term t containing no variables bound in the clause such that for all terms of the form $y\,\bar{s}_k$ occurring in the clause we have $s_j = t\,\bar{s}_{j-1}\,s_{j+1}\ldots s_k$.

Clauses with a static argument correspond to the case $t := (\lambda\bar{x}_{j-1}\,x_{j+1}\ldots x_k.\,u)$, where u is the static argument (containing no variables bound in t) and j is its index in y's argument list. The repeated argument case corresponds to $t := (\lambda\bar{x}_{j-1}\,x_{j+1}\ldots x_k.\,x_i)$, where i is the index of the repeated argument's mate.

Lemma 59 PRUNEARG *is sound and preserves refutational completeness of the calculus.*

Proof The rule is sound because it simply applies a substitution to C. It preserves completeness because the premise C is redundant w.r.t. the conclusion $C\sigma$. This is because the sets of ground instances of C and $C\sigma$ are the same and $C \sqsupset C\sigma$. Clearly $C\sigma$ is an instance of C. We will show the inverse: that C is an instance of $C\sigma$. Let $\rho = \{y' \mapsto \lambda\bar{x}_{j-1}\,x_{j+1}\ldots x_k.\,y\,\bar{x}_{j-1}\,(t\,\bar{x}_{j-1}\,x_{j+1}\ldots x_k)\,x_{j+1}\ldots x_k\}$. We show $C\sigma\rho = C$. Consider an occurrence of y in C. By the side conditions, it will have the form $y\,\bar{s}_k\,\bar{u}$, where $s_j = t\,\bar{s}_{j-1}\,s_{j+1}\ldots s_k$. Hence, $(y\,\bar{s}_k)\sigma\rho = (y'\,\bar{s}_{j-1}\,s_{j+1}\ldots s_k)\rho = y\,\bar{s}_{j-1}\,(t\,\bar{s}_{j-1}\,s_{j+1}\ldots s_k)\,s_{j+1}\ldots s_k = y\,\bar{s}_k$. Thus, $C\sigma\rho = C$. □

We designed an algorithm that efficiently computes the subterm u of the term $t = (\lambda x_1\ldots x_{j-1}\,x_{j+1}\ldots x_k.\,u)$ occurring in the side conditions of PRUNEARG. The algorithm is incomplete, but our tests suggest that it discovers most cases of prunable arguments that occur in practice. The algorithm works by maintaining a mapping of pairs (y,i) of functional variables y and indices i of their arguments to a set of candidate terms for u. For an occurrence $y\,\bar{s}_n$ of y and for an argument s_j, the algorithm approximates this set by computing all possible ways in which subterms of s_j that are equal to any other s_i can be replaced with the variable x_i corresponding to the ith argument of y. The candidate sets for all occurrences of y are then intersected. An arbitrary element of the final intersection is returned as the term u.

Example 60 Suppose that $y\,\mathsf{a}\,(\mathsf{f}\,\mathsf{a})\,\mathsf{b}$ and $y\,z\,(\mathsf{f}\,z)\,\mathsf{b}$ are the only occurrences of y in a clause. The initial mapping is $\{1 \mapsto \mathcal{T}_{\mathrm{H}}, 2 \mapsto \mathcal{T}_{\mathrm{H}}, 3 \mapsto \mathcal{T}_{\mathrm{H}}\}$. After computing the ways in which each argument can be expressed using the remaining ones for the first occurrence and intersecting the sets, we get $\{1 \mapsto \{\mathsf{a}\}, 2 \mapsto \{\mathsf{f}\,\mathsf{a}, \mathsf{f}\,x_1\}, 3 \mapsto \{\mathsf{b}\}\}$, where x_1 represents y's first argument. Finally, after computing the corresponding sets for the second occurrence of y and intersecting them with the previous candidate sets, we get $\{1 \mapsto \emptyset, 2 \mapsto \{\mathsf{f}\,x_1\}, 3 \mapsto \{\mathsf{b}\}\}$. The final mapping shows that we can remove the second argument, since it can be expressed as a function of the first argument: $t = (\lambda x_1\,x_3.\,\mathsf{f}\,x_1\,x_3)$. We can also remove the third argument, since its value is fixed: $t = (\lambda x_1\,x_3.\,\mathsf{b})$.

Example 61 Suppose that $y\,(\lambda x.\,\mathsf{a})\,(\mathsf{f}\,\mathsf{a})\,\mathsf{c}$ and $y\,(\lambda x.\,\mathsf{b})\,(\mathsf{f}\,\mathsf{b})\,\mathsf{d}$ are the only occurrences of y in a clause. Here, PRUNEARG can be used to eliminate the second argument by taking $t := (\lambda x_1\,x_3.\,\mathsf{f}\,(x_1\,x_3))$, but our algorithm fails to detect this.

Alternatives to Axiom (EXT) Following the literature [34, 62], we provide a rule for negative extensionality:

$$\frac{C' \vee s \not\approx s'}{C' \vee s\,(\mathsf{sk}\langle\bar{\alpha}\rangle\bar{y}) \not\approx s'\,(\mathsf{sk}\langle\bar{\alpha}\rangle\bar{y})}\ \textsc{NegExt}$$

The following conditions apply:

1. sk is a fresh Skolem symbol;
2. $s \not\approx s'$ is $\succsim$-eligible in the premise;
3. $\bar{\alpha}$ and $\bar{y}$ are the type and term variables occurring free in the literal $s \not\approx s'$.

Negative extensionality can be applied as an inference rule at any time or as a simplification rule during preprocessing of the initial problem. The rule uses Skolem terms $\mathsf{sk}\,\bar{y}$ rather than $\mathsf{diff}\,s\,s'$ because they tend to be more compact.

Lemma 62 (NEGEXT's satisfiability preservation) *Let $N \subseteq \mathcal{C}_{\mathrm{H}}$, and let E be the conclusion of a* NEGEXT *inference from N. If $N \cup \{(\mathrm{EXT})\}$ is satisfiable, then $N \cup \{(\mathrm{EXT}), E\}$ is satisfiable.*

Proof Let $\mathcal{I}$ be a model of $N \cup \{(\mathrm{EXT})\}$. We need to construct a model of $N \cup \{(\mathrm{EXT}), E\}$. Since (EXT) holds in $\mathcal{I}$, so does its instance $s\,(\mathsf{diff}\,s\,s') \not\approx s'\,(\mathsf{diff}\,s\,s') \vee s \approx s'$. We extend the model $\mathcal{I}$ to a model $\mathcal{I}'$, interpreting sk such that $\mathcal{I}' \models \mathsf{sk}\langle\bar{\alpha}\rangle\bar{y} \approx \mathsf{diff}\,s\,s'$. The Skolem symbol sk takes the free type and term variables of $s \not\approx s'$ as arguments, which include all the free variables of $\mathsf{diff}\,s\,s'$, allowing us to extend $\mathcal{I}$ in this way.

By assumption, the premise $C' \vee s \not\approx s'$ is true in $\mathcal{I}$ and hence in $\mathcal{I}'$. Since the above instance of (EXT) holds in $\mathcal{I}$, it also holds in $\mathcal{I}'$. Hence, the conclusion $C' \vee s\,(\mathsf{sk}\langle\bar{\alpha}_m\rangle\bar{y}_n) \not\approx s'\,(\mathsf{sk}\langle\bar{\alpha}_m\rangle\bar{y}_n)$ also holds, which can be seen by resolving the premise against the (EXT) instance and unfolding the defining equation of sk. □

One reason why the extensionality axiom is so prolific is that both sides of its maximal literal, $y\,(\mathsf{diff}\,y\,z) \not\approx z\,(\mathsf{diff}\,y\,z)$, are fluid. As a pragmatic alternative to the axiom, we introduce the "abstracting" rules ABSSUP, ABSERES, and ABSEFACT with the same premises as the core SUP, ERES, and EFACT, respectively. We call these rules collectively ABS. Each new rule shares all the side conditions of the corresponding core rule except that of the form $\sigma \in \mathrm{CSU}(s,t)$. Instead, it lets σ be the most general unifier of s and t's types and adds this condition: Let $v\langle s_1,\ldots,s_n\rangle = s\sigma$ and $v\langle t_1,\ldots,t_n\rangle = t\sigma$, where $v\langle\,\rangle$ is the largest common green context of $s\sigma$ and $t\sigma$. If any s_i is of functional type and the core rule has conclusion $E\sigma$, the new rule has conclusion $E\sigma \vee s_1 \not\approx t_1 \vee \cdots \vee s_n \not\approx t_n$. The NEGEXT rule can then be applied to those literals $s_i \not\approx t_i$ whose sides have functional type. Essentially the same idea was proposed by Bhayat and Reger as *unification with abstraction* in the context of combinatory superposition [19, Sect. 3.1]. The approach regrettably does not fully eliminate the need for axiom (EXT), as Visa Nummelin demonstrated via the following example.

Example 63 Consider the unsatisfiable clause set consisting of $\mathsf{h}\,x \approx \mathsf{f}\,x$, $\mathsf{k}\,\mathsf{h} \approx \mathsf{k}\,\mathsf{g}$, and $\mathsf{k}\,\mathsf{g} \not\approx \mathsf{k}\,\mathsf{f}$, where k takes at most one argument and $\mathsf{h} \succ \mathsf{g} \succ \mathsf{f}$. The only nonredundant ABS inference applicable is ABSERES on the third clause, resulting in $\mathsf{g} \not\approx \mathsf{f}$. Applying EXTNEG further produces $\mathsf{g}\,\mathsf{sk} \not\approx \mathsf{f}\,\mathsf{sk}$. The set consisting of all five clauses is saturated.

A different approach is to instantiate the extensionality axiom with arbitrary terms s, s' of the same functional type:

$$\frac{}{s\,(\mathsf{diff}\,s\,s') \not\approx s'\,(\mathsf{diff}\,s\,s') \vee s \approx s'}\ \mathrm{EXTINST}$$

We would typically choose s, s' among the green subterms occurring in the current clause set. Intuitively, if we think in terms of eligibility, EXTINST demands $s\,(\mathsf{diff}\,s\,s') \approx s'\,(\mathsf{diff}\,s\,s')$ to be proved before $s \approx s'$ can be used. This can be advantageous because simplifying inferences (based on matching) will often be able to rewrite the applied terms $s\,(\mathsf{diff}\,s\,s')$ and $s'\,(\mathsf{diff}\,s\,s')$. In contrast, ABS assume $s \approx s'$ and delay the proof obligation that $s\,(\mathsf{diff}\,s\,s') \approx$

$s'\,(\mathsf{diff}\,s\,s')$. This can create many long clauses, which will be subject to expensive generating inferences (based on full unification).

Superposition can be generalized to orange subterms as follows:

$$\frac{D' \vee t \approx t' \quad C' \vee s\langle\!\langle \bar{x}.\, u \rangle\!\rangle \mathrel{\dot{\approx}} s'}{(D' \vee C' \vee s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta \mathrel{\dot{\approx}} s')\sigma\rho}\,\lambda\textsc{Sup}$$

where the substitution ρ is defined as follows: Let $P_y = \{y\}$ for all type and term variables $y \notin \bar{x}$. For each i, let P_{x_i} be defined recursively as the union of all P_y such that y occurs free in the λ-expression that binds x_i in $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma$ or that occurs free in the corresponding subterm of $s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta\sigma$. Then ρ is defined as $\{x_i \mapsto \mathsf{sk}_i\langle\bar{\alpha}_i\rangle\,\bar{y}_i \text{ for each } i\}$, where $\bar{y}_i$ are the term variables in P_{x_i} and $\bar{\alpha}_i$ are the type variables in P_{x_i} and the type variables occurring in the type of the λ-expression binding x_i. In addition, SUP's side conditions and the following conditions apply:

10. $\bar{x}$ has length $n > 0$;
11. $\bar{x}\sigma = \bar{x}$;
12. the variables $\bar{x}$ do not occur in $y\sigma$ for all variables y in u.

The substitution ρ introduces Skolem terms to represent bound variables that would otherwise escape their binders.

Example 64 We can shorten the derivation of Example 17 by applying λSUP as follows:

$$\frac{\overbrace{n \approx \mathsf{zero} \vee \mathsf{div}\; n\; n \approx \mathsf{one}}^{C_{\mathrm{div}}} \quad \overbrace{\mathsf{prod}\; K\; (\lambda k.\, \mathsf{div}\; (\mathsf{succ}\; k)\; (\mathsf{succ}\; k)) \not\approx \mathsf{one}}^{C_{\mathrm{nc}}}}{\mathsf{succ}\;\mathsf{sk} \approx \mathsf{zero} \vee \mathsf{prod}\; K\; (\lambda k.\, \mathsf{one}) \not\approx \mathsf{one}}\,\lambda\textsc{Sup}$$

From this conclusion, $\bot$ can be derived using only SUP and ERES inferences. We thus avoid both FLUIDSUP and (EXT).

The rule can be justified in terms of paramodulation and extensionality, with the Skolem terms standing for diff terms:

Lemma 65 (λSUP's satisfiability preservation) *Let $N \subseteq \mathcal{C}_{\mathrm{H}}$, and let E be the conclusion of a λSUP inference from N. If $N \cup \{(\mathrm{EXT})\}$ is satisfiable, then $N \cup \{(\mathrm{EXT}), E\}$ is satisfiable.*

Proof Let $\mathcal{I}$ be a model of $N \cup \{(\mathrm{EXT})\}$. We need to construct a model of $N \cup \{(\mathrm{EXT}), E\}$. For each i, let v_i be the λ-expression binding x_i in the term $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma$ in the rule. Let v_i' be the variant of v_i in which the relevant occurrence of $u\sigma$ is replaced by $t'\sigma$. We define a substitution π recursively by $x_i\pi = \mathsf{diff}\;(v_i\pi)\;(v_i'\pi)$ for all i. This definition is well founded because the variables x_j with $j \geq i$ do not occur freely in v_i and v_i'. We extend the model $\mathcal{I}$ to a model $\mathcal{I}'$, interpreting sk_i such that $\mathcal{I}' \models \mathsf{sk}_i\langle\bar{\alpha}_i\rangle\,\bar{y}_i \approx \mathsf{diff}\;(v_i\pi)\;(v_i'\pi)$ for each i. Since the free type and term variables of any $x_i\pi$ are necessarily contained in P_{x_i}, the arguments of sk_i include the free variables of $\mathsf{diff}\;(v_i\pi)\;(v_i'\pi)$, allowing us to extend $\mathcal{I}$ in this way.

By assumption, the premises of the λSUP inference are true in $\mathcal{I}$ and hence in $\mathcal{I}'$. We need to show that the conclusion $(D' \vee C' \vee s\langle\!\langle \bar{x}.\, t' \rangle\!\rangle_\eta \mathrel{\dot{\approx}} s')\sigma\rho$ is also true in $\mathcal{I}'$. Let ξ be a valuation. If $\mathcal{I}', \xi \models (D' \vee C')\sigma\rho$, we are done. So we assume that $D'\sigma\rho$ and $C'\sigma\rho$ are false in $\mathcal{I}'$ under ξ. In the following, we omit '$\mathcal{I}', \xi \models$', but all equations ($\approx$) are meant to be true in $\mathcal{I}'$ under ξ. Assuming $D'\sigma\rho$ and $C'\sigma\rho$ are false, we will show inductively that $v_i\pi \approx v_i'\pi$ for all $i = k, \ldots, 1$. By this assumption, the premises imply that $t\sigma\rho \approx t'\sigma\rho$ and $s\langle\!\langle \bar{x}.\, u \rangle\!\rangle\sigma\rho \mathrel{\dot{\approx}} s'\sigma\rho$. Due to the way we constructed $\mathcal{I}'$, we have $w\pi \approx w\rho$ for any term w.

Hence, we have $t\sigma\pi \approx t'\sigma\pi$. The terms $v_k\pi\,(\mathsf{diff}\,(v_k\pi)\,(v'_k\pi))$ and $v'_k\pi\,(\mathsf{diff}\,(v_k\pi)\,(v'_k\pi))$ are the respective result of applying π to the body of the λ-expressions v_k and v'_k. Therefore, by congruence, $t\sigma\pi \approx t'\sigma\pi$ and $t\sigma = u\sigma$ imply that $v_k\pi\,(\mathsf{diff}\,(v_k\pi)\,(v'_k\pi)) \approx v'_k\pi\,(\mathsf{diff}\,(v_k\pi)\,(v'_k\pi))$. The extensionality axiom then implies $v_k\pi \approx v'_k\pi$.

It follows directly from the definition of π that for all i, $v_i\pi\,(\mathsf{diff}\,(v_i\pi)\,(v'_i\pi)) = s_i\langle\!\langle v_{i+1}\pi\rangle\!\rangle$ and $v'_i\pi\,(\mathsf{diff}\,(v_i\pi)\,(v'_i\pi)) = s_i\langle\!\langle v'_{i+1}\pi\rangle\!\rangle$ for some context $s_i\langle\!\langle\,\rangle\!\rangle$. The subterms $v_{i+1}\pi$ of $s_i\langle\!\langle v_{i+1}\pi\rangle\!\rangle$ and $v'_{i+1}\pi$ of $s_i\langle\!\langle v'_{i+1}\pi\rangle\!\rangle$ may be below applied variables but not below λs. Since substitutions avoid capture, in v_i and v'_i, π only substitutes x_j with $j < i$, but in v_{i+1} and v'_{i+1}, it substitutes all x_j with $j \leq i$. By an induction using these equations, congruence, and the extensionality axiom, we can derive from $v_k\pi \approx v'_k\pi$ that $v_1\pi \approx v'_1\pi$. Since $\mathcal{I}' \models w\pi \approx w\rho$ for any term w, we have $v_1\rho \approx v'_1\rho$. By congruence, it follows that $s\langle\!\langle\bar{x}.\,u\rangle\!\rangle\sigma\rho \approx s\langle\!\langle\bar{x}.\,t'\rangle\!\rangle_\eta\sigma\rho$. With $s\langle\!\langle\bar{x}.\,u\rangle\!\rangle\sigma\rho \mathrel{\dot\approx} s'\sigma\rho$, it follows that $(s\langle\!\langle\bar{x}.\,t'\rangle\!\rangle_\eta \mathrel{\dot\approx} s')\sigma\rho$. Hence, the conclusion of the λSUP inference is true in $\mathcal{I}'$. □

Alternatives to FLUIDSUP The next rule, *duplicating flex subterm superposition*, is a lightweight substitute for FLUIDSUP:

$$\frac{D' \vee t \approx t' \quad C' \vee s\langle y\,\bar{u}_n\rangle \mathrel{\dot\approx} s'}{(D' \vee C' \vee s\langle z\,\bar{u}_n\,t'\rangle \mathrel{\dot\approx} s')\rho\sigma}\,\textsc{DupSup}$$

where $n > 0$, $\rho = \{y \mapsto \lambda\bar{x}_n.\,z\,\bar{x}_n\,(w\,\bar{x}_n)\}$, and $\sigma \in \mathrm{CSU}(t, w\,(\bar{u}_n\rho))$ for fresh variables w, z. The order and eligibility restrictions are as for SUP. The rule can be understood as the composition of an inference that applies the substitution ρ and of a paramodulation inference into the subterm $w\,(\bar{u}_n\rho)$ of $s\langle z\,(\bar{u}_n\rho)\,(w\,(\bar{u}_n\rho))\rangle$. DUPSUP is general enough to replace FLUIDSUP in Examples 13 and 14 but not in Example 15. On the other hand, FLUIDSUP's unification problem is usually a flex–flex pair, whereas DUPSUP yields a less explosive flex–rigid pair unless t is variable-headed.

The last rule, *flex subterm superposition*, is an even more lightweight substitute for FLUIDSUP:

$$\frac{D' \vee t \approx t' \quad C' \vee s\langle y\,\bar{u}_n\rangle \mathrel{\dot\approx} s'}{(D' \vee C' \vee s\langle t'\rangle \mathrel{\dot\approx} s')\sigma}\,\textsc{FlexSup}$$

where $n > 0$ and $\sigma \in \mathrm{CSU}(t, y\,\bar{u}_n)$. The order and eligibility restrictions are as for SUP.

6 Implementation

Zipperposition [27, 28] is an open source superposition prover written in OCaml.[1] Its raw performance might not be comparable to highly optimized provers such as E and Vampire, but its code is easier to maintain and modify. Our rough estimate is that it is about three times slower than E. Originally designed for polymorphic first-order logic (TF1 [21]), Zipperposition was later extended by Cruanes with an incomplete higher-order mode based on pattern unification [53]. Bentkamp et al. [12] extended it further with a complete λ-free clausal higher-order mode. We have now implemented a clausal higher-order mode based on our calculus. We use the order $\succ_\lambda$ (Sect. 3.6) derived from the Knuth–Bendix order [45] and the lexicographic path order [43]. We currently use the corresponding nonstrict order $\succeq_\lambda$ as $\succsim$.

[1] `https://github.com/sneeuwballen/zipperposition`

Except for FLUIDSUP, the core calculus rules already existed in Zipperposition in a similar form. To improve efficiency, we extended the prover to use a higher-order generalization [68] of fingerprint indices [58] to find inference partners for all new binary inference rules. To speed up the computation of the SUP conditions, we omit the condition $C\sigma \not\precsim D\sigma$ in the implementation, at the cost of performing additional inferences. Among the optional rules, we implemented λDEMOD, PRUNEARG, NEGEXT, ABS, EXTINST, λSUP, DUPSUP, and FLEXSUP. For λDEMOD and λSUP, demodulation, subsumption, and other standard simplification rules (as implemented in E [59]), we use pattern unification. For generating inference rules that require enumerations of complete sets of unifiers, we use the complete procedure of Vukmirović et al. [68]. It has better termination behavior, produces fewer redundant unifiers, and can be implemented more efficiently than procedures such as Jensen and Pietrzykowski's [38] and Snyder and Gallier's [61]. The set of fluid terms is overapproximated in the implementation by the set of terms that are either nonground λ-expressions or terms of the form $y\,\bar{u}_n$ with $n > 0$. To efficiently retrieve candidates for ABS inferences without slowing down superposition term indexing structures, we implemented dedicated indexing for clauses that are eligible for ABS inferences [70, Sect. 3.3].

Zipperposition implements a DISCOUNT-style given clause procedure [5]. The proof state is represented by a set A of active clauses and a set P of passive clauses. To interleave nonterminating unification with other computation, we added a set T containing possibly infinite sequences of scheduled inferences. These sequences are stored as finite instructions of how to compute the inferences. Initially, all clauses are in P. At each iteration of the main loop, the prover heuristically selects a *given clause* C from P. If P is empty, sequences from T are evaluated to generate more clauses into P; if no clause can be produced in this way, A is saturated and the prover stops. Assuming a given clause C could be selected, it is first simplified using A. Clauses in A are then simplified w.r.t. C, and any simplified clause is moved to P. Then C is added to A and all sequences representing nonredundant inferences between C and A are added to T. This maintains the invariant that all nonredundant inferences between clauses in A have been scheduled or performed. Then some of the scheduled inferences in T are performed and the conclusions are put into P.

We can view the above loop as an instance of the abstract Zipperposition loop prover ZL of Waldmann et al. [71, Example 34]. Their Theorem 32 allows us to obtain dynamic completeness for this prover architecture from our static completeness result (Theorem 54). This requires that the sequences in T are visited fairly, that clauses in P are chosen fairly, and that simplification terminates, all of which are guaranteed by our implementation.

The unification procedure we use returns a sequence of either singleton sets containing the unifier or an empty set signaling that a unifier is still not found. Empty sets are returned to give back control to the caller of unification procedure and avoid getting stuck on nonterminating problems. These sequences of unifier subsingletons are converted into sequences containing subsingletons of clauses representing inference conclusions.

7 Evaluation

The evaluation consists of two parts: an assessment of the extensions described in Sect. 5 and a comparison of our prototype implementation with Zipperposition's modes for less expressive logics and with other higher-order provers. The experiments were run on StarExec

	−NE,−PA	−NE	−PA	Base
TH0	446 (0)	446 (0)	447 (0)	447 (0)
SH-λ	431 (0)	433 (0)	433 (0)	436 (1)

Fig. 1 Number of problems proved without rules included in the base configuration

	Base	+λD	+λS0	+λS1	+λS2	+λS4	+λS8	+λS1024
TH0	447 (0)	448 (0)	449 (0)	449 (0)	449 (0)	449 (0)	449 (0)	449 (0)
SH-λ	436 (1)	435 (4)	430 (1)	429 (0)	429 (0)	429 (0)	429 (0)	429 (0)

Fig. 2 Number of problems proved using rules that perform rewriting under λ-binders

	Base	+ABS	+EXTINST	+(EXT)
TH0	447 (0)	450 (1)	450 (1)	376 (0)
SH-λ	436 (11)	430 (11)	402 (1)	365 (2)

Fig. 3 Number of problems proved using rules that perform extensionality reasoning

	−FLEXSUP	Base	−FLEXSUP,+DUPSUP	−FLEXSUP,+FLUIDSUP
TH0	446 (0)	447 (0)	448 (1)	447 (0)
SH-λ	469 (10)	436 (4)	451 (3)	461 (7)

Fig. 4 Number of problems proved with rules that perform superposition into fluid terms

nodes equipped with Intel Xeon E5-2609 0 CPUs clocked at 2.40 GHz. Following CASC 2019 [65], we set 180 s as the CPU time limit. Our results are publicly available.[2]

Evaluation of Extensions In the first part, we assess the usefulness of the extensions described in Sect. 5. We used both standard TPTP benchmarks [64] and Sledgehammer-generated benchmarks [52]. From the TPTP, version 7.2.0, we used all 499 monomorphic higher-order theorems in TH0 syntax without interpreted Booleans and arithmetic (*TH0*). The Sledgehammer benchmarks, corresponding to Isabelle's Judgment Day suite [23], were regenerated to target clausal higher-order logic (*SH-*λ). They comprise 1253 problems, each generated from 256 Isabelle facts (definitions and lemmas).

We fixed a reasonable *base* configuration of Zipperposition parameters. For each extension, we then changed the corresponding parameters and observed the effect on the success rate. The base configuration uses the complete variant of the unification procedure of Vukmirović et al. [68]. It also includes the optional rules NEGEXT and PRUNEARG, substitutes FLEXSUP for the highly explosive FLUIDSUP, and excludes axiom (EXT). This configuration is not refutationally complete.

The rules NEGEXT (*NE*) and PRUNEARG (*PA*) were added to the base configuration because our informal experiments showed that they usually help. Fig. 1 confirms this, although the effect is small. In all tables, $+R$ denotes the inclusion of a rule R not present in the base, and $-R$ denotes the exclusion of a rule R present in the base. Numbers given in parentheses denote the number of problems that are solved only by the given configuration and by no other configuration in the same table.

The rules λDEMOD (λD) and λSUP extend the calculus to perform some rewriting under λ-binders. While experimenting with the calculus, we noticed that for some configurations,

[2] `https://doi.org/10.5281/zenodo.4032969`

λSUP performs better when the number of fresh Skolem symbols it introduces overall is bounded by some parameter n. As Fig. 2 shows, the inclusion of these rules has a different effect on the two benchmark sets. On the other hand, different choices of n for λSUP (denoted by λSn) do not seem to influence the success rate much.

The evaluation of the ABS and EXTINST rules and axiom (EXT), presented in Fig. 3, confirms our intuition that including the extensionality axiom is severely detrimental to performance. The +(EXT) configuration solves two unique problems on SH-λ benchmarks, but this small success is coincidental, since (EXT) is not even referenced in the generated proofs.

The FLEXSUP rule included in the base configuration underperformed. Even the FLUIDSUP and DUPSUP rules outperform FLEXSUP, as shown in Fig. 4. This effect is especially visible on SH-λ benchmarks. On TPTP, the differences are negligible.

Most of the extensions have a stronger effect on SH-λ than on TH0. A possible explanation is that the Boolean-free TH0 benchmark subset consists mostly of problems that are simple to solve using most prover parameters. On the other hand, SH-λ benchmarks are of varying difficulty and can thus benefit more from changing prover parameters.

Main Evaluation In the second part, we seek to answer the following research questions:

1. What is the overhead of our implementation on first-order problems, compared with first-order superposition?
2. How does our implementation compare with λ-free clausal superposition on λ-free problems?
3. How does the complete implementation of our calculus compare with incomplete variants?
4. How does our implementation compare with other higher-order provers on Boolean-free higher-order benchmarks?

We needed more benchmarks to answer questions 1 and 2. From the TPTP, we used 1000 randomly selected first-order (*FO*) problems in CNF, FOF, or TFF syntax without arithmetic. We partitioned the TH0 problems used above into those containing no λ-expressions (*TH0*λ*f*, 452 problems) and those containing λ-expressions (*TH0*λ, 47 problems). To make the SH-λ problems accessible to λ-free clausal higher-order provers, we regenerated them using λ-lifted supercombinators (*SH-ll*), as described by Meng and Paulson [52].

To answer questions 1 and 2, we ran Zipperposition in first-order (*FOZip*) and λ-free (*λfreeZip*) modes, as well as in a mode that encodes curried applications using a distinguished binary symbol @ before using first-order Zipperposition (@+*FOZip*). To answer question 3, we evaluated the implementation of our calculus in Zipperposition in three configurations: *λZip-base*, *λZip-pragmatic*, and *λZip-full*. The configuration λZip-base is the base described above. The configuration λZip-pragmatic builds on λZip-base by disabling FLEXSUP and replacing complete unification with the pragmatic variant pv^2_{1121} of the unification procedure [68]. The configuration λZip-full is a refutationally complete extension of λZip-base that substitutes FLUIDSUP for FLEXSUP and includes axiom (EXT).

To answer question 4, we selected all contenders in the THF division of CASC 2019 as representatives of the state of the art: CVC4 1.8 prerelease [9], Leo-III 1.4 [62], Satallax 3.4 [24], and Vampire 4.4 [18]. We also included Ehoh [69], the λ-free clausal higher-order mode of E 2.4. Leo-III and Satallax are cooperative higher-order provers that can be set up to regularly invoke first-order provers as terminal proof procedures. To assess the performance of their core calculi, we also evaluated them with first-order backends disabled. We denote these "uncooperative" configurations by *Leo-III-uncoop* and *Satallax-uncoop*, as opposed to the standard versions *Leo-III-coop* and *Satallax-coop*. To demonstrate the best

	FO	TH0λf	TH0λ	SH-ll	SH-λ
CVC4	539	424	31	696	650
Ehoh	681	418	–	691	–
Leo-III-uncoop	198	389	42	226	234
Leo-III-coop	582	**438**	43	683	674
Satallax-uncoop	–	398	43	489	507
Satallax-coop	–	432	43	602	616
Vampire	**729**	432	42	**718**	707
FOZip	399	–	–	–	–
@+FOZip	363	400	–	478	–
λfreeZip	395	398	–	538	–
λZip-base	388	408	39	420	436
λZip-pragmatic	396	411	33	496	503
λZip-full	177	339	34	353	361
Zip-uncoop	514	426	**46**	661	677
Zip-coop	625	434	**46**	710	**717**

Fig. 5 Number of problems proved by the different provers

performance of Zipperposition, we evaluated it in a portfolio mode that runs the prover in various configurations (*Zip-uncoop*). We also evaluated a cooperative version of the portfolio which, in some configurations, invokes Ehoh as backend on higher-order problems after a predefined time (*Zip-coop*). In this version, Zipperposition encodes selected clauses from the proof state to λ-free higher-order logic supported by Ehoh [69]. On first-order problems, we ran Ehoh, Vampire, and Zip-uncoop using the provers' respective first-order modes.

A summary of these experiments is presented in Fig. 5. Regarding question 1, we observe that λZip-pragmatic incurs less than 1% overhead and λZip-base incurs less than 3% overhead compared with FOZip, which is very reasonable. Regarding question 2, the numbers show that λZip-pragmatic outperforms λfreeZip on TH0λf problems and but falls behind λfreeZip on SH-ll problems. Regarding question 3, we see that λZip-full has substantially more overhead and performs worse than λZip-pragmatic and λZip-base on almost all benchmark sets, due to the explosive extensionality axiom and FLUIDSUP rule.

Regarding question 4, we learn that, except on TH0λ problems, both λZip-base and λZip-pragmatic outperform Leo-III-uncoop (which also runs a fixed configuration) by substantial margins. In addition, Zip-uncoop outperforms Satallax-uncoop (which also uses a portfolio). Our most competitive configuration, Zip-coop, emerges as the winner on both problem sets containing λ-expressions. The raw evaluation data show that, on higher-order TPTP benchmarks, Zip-coop does not solve any problems that no other cooperative prover solves. This probably says more about the benchmark set than about the prover. By contrast, on SH-ll benchmarks Zip-coop uniquely solves 21 problems, and on SH-λ benchmarks, it uniquely solves 27 problems, w.r.t. other cooperative provers.

8 Discussion and Related Work

Bentkamp et al. [12] introduced four calculi for λ-free clausal higher-order logic organized along two axes: *intensional* versus *extensional*, and *nonpurifying* versus *purifying*. The purifying calculi flatten the clauses containing applied variables, thereby eliminating the need for superposition into variables. As we extended their work to support λ-expressions, we found the purification approach problematic and gave it up because it needs x to be smaller

than $x\,t$, which is impossible to achieve with a term order on $\beta\eta$-equivalence classes. We also quickly gave up our attempt at supporting intensional higher-order logic. Extensionality is the norm for higher-order unification [30] and is mandated by the TPTP THF format [66] and in proof assistants such as HOL4, HOL Light, Isabelle/HOL, Lean, Nuprl, and PVS.

Bentkamp et al. viewed their approach as "a stepping stone towards full higher-order logic." It already included a notion analogous to green subterms and an ARGCONG rule, which help cope with the complications occasioned by β-reduction.

Our Boolean-free λ-superposition calculus joins the family of proof systems for higher-order logic. It is related to Andrews's higher-order resolution [1], Huet's constrained resolution [36], Jensen and Pietrzykowski's ω-resolution [38], Snyder's higher-order E-resolution [60], Benzmüller and Kohlhase's extensional higher-order resolution [14], Benzmüller's higher-order unordered paramodulation and RUE resolution [13], and Bhayat and Reger's combinatory superposition [19]. A noteworthy variant of higher-order unordered paramodulation is Steen and Benzmüller's higher-order ordered paramodulation [62], whose order restrictions undermine refutational completeness but yield better empirical results. Other approaches are based on analytic tableaux [8, 46, 47, 55], connections [2], sequents [50], and satisfiability modulo theories (SMT) [9]. Andrews [3] and Benzmüller and Miller [15] provide excellent surveys of higher-order automation.

Combinatory superposition was developed shortly after λ-superposition and is closely related. It is modeled on the intensional nonpurifying calculus by Bentkamp et al. and targets extensional polymorphic clausal higher-order logic. Both combinatory and λ-superposition gracefully generalize the highly successful first-order superposition rules without sacrificing refutational completeness, and both are equipped with a redundancy criterion, which earlier refutationally complete higher-order calculi lack. In particular, PRUNEARG is a versatile simplification rule that could be useful in other provers. Combinatory superposition's distinguishing feature is that it uses SKBCI combinators to represent λ-expressions. Combinators can be implemented more easily starting from a first-order prover; β-reduction amounts to demodulation. However, according to its developers [19], "Narrowing terms with combinator axioms is still explosive and results in redundant clauses. It is also never likely to be competitive with higher-order unification in finding complex unifiers." Among the drawbacks of λ-superposition are the need to solve flex–flex pairs eagerly and the explosion caused by the extensionality axiom. We believe that this is a reasonable trade-off, especially for large problems with a substantial first-order component.

Our prototype Zipperposition joins the league of automatic theorem provers for higher-order logic. We list some of its rivals. TPS [4] is based on the connection method and expansion proofs. LEO [14] and LEO-II [17] implement variants of RUE resolution. Leo-III [62] is based on higher-order paramodulation. Satallax [24] implements a higher-order tableau calculus guided by a SAT solver. LEO-II, Leo-III, and Satallax integrate external first-order provers as terminal proof procedures. AgsyHOL [50] is based on a focused sequent calculus guided by narrowing. The SMT solvers CVC4 and veriT have recently been extended to higher-order logic [9]. Vampire now implements both combinatory superposition and a version of standard superposition in which first-order unification is replaced by restricted combinatory unification [18].

Half a century ago, Robinson [56] proposed to reduce higher-order logic to first-order logic via a translation. "Hammer" tools such as Sledgehammer [54], Miz$\mathbb{AR}$ [67], HOLyHammer [42], and CoqHammer [29] have since popularized this approach in proof assistants. The translation must eliminate the λ-expressions, typically using SKBCI combinators or λ-lifting [52], and encode typing information [20].

9 Conclusion

We presented the Boolean-free λ-superposition calculus, which targets a clausal fragment of extensional polymorphic higher-order logic. With the exception of a functional extensionality axiom, it gracefully generalizes standard superposition. Our prototype prover Zipperposition shows promising results on TPTP and Isabelle benchmarks. In future work, we plan to pursue five main avenues of investigation.

We first plan to *extend the calculus to support Booleans and Hilbert choice.* Booleans are notoriously explosive. We want to experiment with both axiomatizations and native support in the calculus. Native support would likely take the form of a primitive substitution rule that enumerates predicate instantiations [2], delayed clausification rules [32], and rules for reasoning about Hilbert choice.

We want to investigate techniques to *curb the explosion caused by functional extensionality.* The extensionality axiom reintroduces the search space explosion that the calculus's order restrictions aim at avoiding. Maybe we can replace it by more restricted inference rules without compromising refutational completeness.

We will also look into approaches to *curb the explosion caused by higher-order unification.* Our calculus suffers from the need to solve flex–flex pairs. Existing procedures [38,61, 68] enumerate redundant unifiers. This can probably be avoided to some extent. It could also be useful to investigate unification procedures that would delay imitation/projection choices via special schematic variables, inspired by Libal's representation of regular unifiers [49].

We clearly need to *fine-tune and develop heuristics.* We expect heuristics to be a fruitful area for future research in higher-order reasoning. Proof assistants are an inexhaustible source of easy-looking benchmarks that are beyond the power of today's provers. Whereas "hard higher-order" may remain forever out of reach, we believe that there is a substantial "easy higher-order" fragment that awaits automation.

Finally, we plan to *implement the calculus in a state-of-the-art prover.* A suitable basis for an optimized implementation of the calculus would be Ehoh, the λ-free clausal higher-order version of E developed by Vukmirović, Blanchette, Cruanes, and Schulz [69].

Acknowledgment Simon Cruanes patiently explained Zipperposition's internals and allowed us to continue the development of his prover. Christoph Benzmüller and Alexander Steen shared insights and examples with us, guiding us through the literature and clarifying how the Leos work. Maria Paola Bonacina and Nicolas Peltier gave us some ideas on how to treat the extensionality axiom as a theory axiom, ideas we have yet to explore. Mathias Fleury helped us set up regression tests for Zipperposition. Ahmed Bhayat, Tomer Libal, and Enrico Tassi shared their insights on higher-order unification. Andrei Popescu and Dmitriy Traytel explained the terminology surrounding the λ-calculus. Haniel Barbosa, Daniel El Ouraoui, Pascal Fontaine, Visa Nummelin, and Hans-Jörg Schurr were involved in many stimulating discussions. Christoph Weidenbach made this collaboration possible. Ahmed Bhayat, Wan Fokkink, Nicolas Peltier, Mark Summerfield, and the anonymous reviewers suggested several textual improvements. The maintainers of StarExec let us use their service for the evaluation. We thank them all.

Bentkamp, Blanchette, and Vukmirović's research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka). Bentkamp and Blanchette also benefited from the Netherlands Organization for Scientific Research (NWO) Incidental Financial Support scheme. Blanchette has received funding from the NWO under the Vidi program (project No. 016.Vidi.189.037, Lean Forward).

References

1. Andrews, P.B.: Resolution in type theory. J. Symb. Log. **36**(3), 414–432 (1971)
2. Andrews, P.B.: On connections and higher-order logic. J. Autom. Reason. **5**(3), 257–291 (1989)

3. Andrews, P.B.: Classical type theory. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, vol. II, pp. 965–1007. Elsevier and MIT Press (2001)
4. Andrews, P.B., Bishop, M., Issar, S., Nesmith, D., Pfenning, F., Xi, H.: TPS: A theorem-proving system for classical type theory. J. Autom. Reason. **16**(3), 321–353 (1996)
5. Avenhaus, J., Denzinger, J., Fuchs, M.: DISCOUNT: A system for distributed equational deduction. In: J. Hsiang (ed.) RTA-95, *LNCS*, vol. 914, pp. 397–402. Springer (1995)
6. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. **4**(3), 217–247 (1994)
7. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, vol. I, pp. 19–99. Elsevier and MIT Press (2001)
8. Backes, J., Brown, C.E.: Analytic tableaux for higher-order logic with choice. J. Autom. Reason. **47**(4), 451–479 (2011)
9. Barbosa, H., Reynolds, A., Ouraoui, D.E., Tinelli, C., Barrett, C.W.: Extending SMT solvers to higher-order logic. In: P. Fontaine (ed.) CADE-27, *LNCS*, vol. 11716, pp. 35–54. Springer (2019)
10. Bentkamp, A., Blanchette, J., Cruanes, S., Waldmann, U.: Superposition for lambda-free higher-order logic. Log. Meth. Comput. Sci. **17**(2), 1:1–1:38 (2021)
11. Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P., Waldmann, U.: Superposition with lambdas. In: P. Fontaine (ed.) CADE-27, *LNCS*, vol. 11716, pp. 55–73. Springer (2019)
12. Bentkamp, A., Blanchette, J.C., Cruanes, S., Waldmann, U.: Superposition for lambda-free higher-order logic. In: D. Galmiche, S. Schulz, R. Sebastiani (eds.) IJCAR 2018, *LNCS*, vol. 10900, pp. 28–46. Springer (2018)
13. Benzmüller, C.: Extensional higher-order paramodulation and RUE-resolution. In: H. Ganzinger (ed.) CADE-16, *LNCS*, vol. 1632, pp. 399–413. Springer (1999)
14. Benzmüller, C., Kohlhase, M.: Extensional higher-order resolution. In: C. Kirchner, H. Kirchner (eds.) CADE-15, *LNCS*, vol. 1421, pp. 56–71. Springer (1998)
15. Benzmüller, C., Miller, D.: Automation of higher-order logic. In: J.H. Siekmann (ed.) Computational Logic, *Handbook of the History of Logic*, vol. 9, pp. 215–254. Elsevier (2014)
16. Benzmüller, C., Paulson, L.C.: Multimodal and intuitionistic logics in simple type theory. Log. J. IGPL **18**(6), 881–892 (2010)
17. Benzmüller, C., Sultana, N., Paulson, L.C., Theiss, F.: The higher-order prover LEO-II. J. Autom. Reason. **55**(4), 389–404 (2015)
18. Bhayat, A., Reger, G.: Restricted combinatory unification. In: P. Fontaine (ed.) CADE-27, *LNCS*, vol. 11716, pp. 74–93. Springer (2019)
19. Bhayat, A., Reger, G.: A combinator-based superposition calculus for higher-order logic. In: N. Peltier, V. Sofronie-Stokkermans (eds.) IJCAR 2020, Part I, *LNCS*, vol. 12166, pp. 278–296. Springer (2020)
20. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. Log. Meth. Comput. Sci. **12**(4) (2016)
21. Blanchette, J.C., Paskevich, A.: TFF1: The TPTP typed first-order form with rank-1 polymorphism. In: M.P. Bonacina (ed.) CADE-24, *LNCS*, vol. 7898, pp. 414–420. Springer (2013)
22. Blanqui, F., Jouannaud, J.P., Rubio, A.: The computability path ordering. Log. Meth. Comput. Sci. **11**(4) (2015)
23. Böhme, S., Nipkow, T.: Sledgehammer: Judgement Day. In: J. Giesl, R. Hähnle (eds.) IJCAR 2010, *LNCS*, vol. 6173, pp. 107–121. Springer (2010)
24. Brown, C.E.: Satallax: An automatic higher-order prover. In: B. Gramlich, D. Miller, U. Sattler (eds.) IJCAR 2012, *LNCS*, vol. 7364, pp. 111–117. Springer (2012)
25. de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. Indag. Math **75**(5), 381–392 (1972)
26. Cervesato, I., Pfenning, F.: A linear spine calculus. J. Log. Comput. **13**(5), 639–688 (2003)
27. Cruanes, S.: Extending superposition with integer arithmetic, structural induction, and beyond. Ph.D. thesis, École polytechnique (2015)
28. Cruanes, S.: Superposition with structural induction. In: C. Dixon, M. Finger (eds.) FroCoS 2017, *LNCS*, vol. 10483, pp. 172–188. Springer (2017)
29. Czajka, Ł., Kaliszyk, C.: Hammer for Coq: Automation for dependent type theory. J. Autom. Reason. **61**(1-4), 423–453 (2018)
30. Dowek, G.: Higher-order unification and matching. In: J.A. Robinson, A. Voronkov (eds.) Handbook of Automated Reasoning, vol. II, pp. 1009–1062. Elsevier and MIT Press (2001)
31. Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer (2002)
32. Ganzinger, H., Stuber, J.: Superposition with equivalence reasoning and delayed clause normal form transformation. Information and Computation **199**(1–2), 3–23 (2005)
33. Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)

34. Gupta, A., Kovács, L., Kragl, B., Voronkov, A.: Extensional crisis and proving identity. In: F. Cassez, J. Raskin (eds.) ATVA 2014, *LNCS*, vol. 8837, pp. 185–200. Springer (2014)
35. Henkin, L.: Completeness in the theory of types. J. Symb. Log. **15**(2), 81–91 (1950)
36. Huet, G.P.: A mechanization of type theory. In: N.J. Nilsson (ed.) IJCAI-73, pp. 139–146. William Kaufmann (1973)
37. Huet, G.P.: A unification algorithm for typed lambda-calculus. Theor. Comput. Sci. **1**(1), 27–57 (1975)
38. Jensen, D.C., Pietrzykowski, T.: Mechanizing ω-order type theory through unification. Theor. Comput. Sci. **3**(2), 123–171 (1976)
39. Jouannaud, J.P., Rubio, A.: Rewrite orderings for higher-order terms in eta-long beta-normal form and recursive path ordering. Theor. Comput. Sci. **208**(1–2), 33–58 (1998)
40. Jouannaud, J.P., Rubio, A.: Polymorphic higher-order recursive path orderings. J. ACM **54**(1), 2:1–2:48 (2007)
41. Kaliszyk, C., Sutcliffe, G., Rabe, F.: TH1: The TPTP typed higher-order form with rank-1 polymorphism. In: P. Fontaine, S. Schulz, J. Urban (eds.) PAAR-2016, *CEUR Workshop Proceedings*, vol. 1635, pp. 41–55. CEUR-WS.org (2016)
42. Kaliszyk, C., Urban, J.: HOL(y)Hammer: Online ATP service for HOL Light. Math. Comput. Sci. **9**(1), 5–22 (2015)
43. Kamin, S., Lévy, J.J.: Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois (1980)
44. Kőnig, D.: Über eine Schlussweise aus dem Endlichen ins Unendliche. Acta Sci. Math. (Szeged) **3499/2009**(3:2–3), 121–130 (1927)
45. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: J. Leech (ed.) Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press (1970)
46. Kohlhase, M.: Higher-order tableaux. In: P. Baumgartner, R. Hähnle, J. Posegga (eds.) TABLEAUX '95, *LNCS*, vol. 918, pp. 294–309. Springer (1995)
47. Konrad, K.: HOT: A concurrent automated theorem prover based on higher-order tableaux. In: J. Grundy, M.C. Newey (eds.) TPHOLs '98, *LNCS*, vol. 1479, pp. 245–261. Springer (1998)
48. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: N. Sharygina, H. Veith (eds.) CAV 2013, *LNCS*, vol. 8044, pp. 1–35. Springer (2013)
49. Libal, T.: Regular patterns in second-order unification. In: A.P. Felty, A. Middeldorp (eds.) CADE-25, *LNCS*, vol. 9195, pp. 557–571. Springer (2015)
50. Lindblad, F.: A focused sequent calculus for higher-order logic. In: S. Demri, D. Kapur, C. Weidenbach (eds.) IJCAR 2014, *LNCS*, vol. 8562, pp. 61–75. Springer (2014)
51. Mayr, R., Nipkow, T.: Higher-order rewrite systems and their confluence. Theor. Comput. Sci. **192**(1), 3–29 (1998)
52. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reason. **40**(1), 35–60 (2008)
53. Miller, D.: A logic programming language with lambda-abstraction, function variables, and simple unification. J. Log. Comput. **1**(4), 497–536 (1991)
54. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: G. Sutcliffe, S. Schulz, E. Ternovska (eds.) IWIL-2010, *EPiC*, vol. 2, pp. 1–11. EasyChair (2012)
55. Robinson, J.: Mechanizing higher order logic. In: B. Meltzer, D. Michie (eds.) Machine Intelligence, vol. 4, pp. 151–170. Edinburgh University Press (1969)
56. Robinson, J.: A note on mechanizing higher order logic. In: B. Meltzer, D. Michie (eds.) Machine Intelligence, vol. 5, pp. 121–135. Edinburgh University Press (1970)
57. Schulz, S.: E - a brainiac theorem prover. AI Commun. **15**(2-3), 111–126 (2002)
58. Schulz, S.: Fingerprint indexing for paramodulation and rewriting. In: B. Gramlich, D. Miller, U. Sattler (eds.) IJCAR 2012, *LNCS*, vol. 7364, pp. 477–483. Springer (2012)
59. Schulz, S., Cruanes, S., Vukmirovic, P.: Faster, higher, stronger: E 2.3. In: P. Fontaine (ed.) CADE-27, *LNCS*, vol. 11716, pp. 495–507. Springer (2019)
60. Snyder, W.: Higher order *E*-unification. In: M.E. Stickel (ed.) CADE-10, *LNCS*, vol. 449, pp. 573–587. Springer (1990)
61. Snyder, W., Gallier, J.H.: Higher-order unification revisited: Complete sets of transformations. J. Symb. Comput. **8**(1/2), 101–140 (1989)
62. Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: D. Galmiche, S. Schulz, R. Sebastiani (eds.) IJCAR 2018, *LNCS*, vol. 10900, pp. 108–116. Springer (2018)
63. Sutcliffe, G.: The 10th IJCAR automated theorem proving system competition—CASC-J10 Accepted in *AI Commun.*
64. Sutcliffe, G.: The TPTP problem library and associated infrastructure—from CNF to TH0, TPTP v6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017)

65. Sutcliffe, G.: The CADE-27 automated theorem proving system competition—CASC-27. AI Commun. **32**(5-6), 373–389 (2019)
66. Sutcliffe, G., Benzmüller, C., Brown, C.E., Theiss, F.: Progress in the development of automated theorem proving for higher-order logic. In: R.A. Schmidt (ed.) CADE-22, *LNCS*, vol. 5663, pp. 116–130. Springer (2009)
67. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reason. **50**(2), 229–241 (2013)
68. Vukmirović, P., Bentkamp, A., Nummelin, V.: Efficient full higher-order unification. In: Z.M. Ariola (ed.) FSCD 2020, *LIPIcs*, vol. 167, pp. 5:1–5:17. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2020)
69. Vukmirović, P., Blanchette, J.C., Cruanes, S., Schulz, S.: Extending a brainiac prover to lambda-free higher-order logic. In: T. Vojnar, L. Zhang (eds.) TACAS 2019, *LNCS*, vol. 11427, pp. 192–210. Springer (2019)
70. Vukmirović, P., Nummelin, V.: Boolean reasoning in a higher-order superposition prover. In: Practical Aspects of Automated Reasoning (PAAR 2020) (2020)
71. Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: N. Peltier, V. Sofronie-Stokkermans (eds.) IJCAR 2020, Part I, *LNCS*, vol. 12166, pp. 316–334. Springer (2020)