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 [13] 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 first-class 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 considered equal up to β-conversion. The relations designed for proving termination of higher-order term rewriting systems, such as HORPO [42] and CPO [24], lack many of the desired properties (e.g., transitivity, stability under substitution).
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 $\{y \mapsto \lambda x.\, z\, x\,(\mathsf{f}\, x)\}$ also gives rise to a subterm $\mathsf{f}\,\mathsf{c}$ after β-reduction. The corresponding inference generates the clause $z\,\mathsf{c}\,\mathsf{a} \not\approx z\,\mathsf{b}\,(\mathsf{f}\,\mathsf{b})$.

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 [41], 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 substitution. Even on ground terms, we cannot obtain all desirable properties. We sacrifice compatibility with arguments (the property that $s' \succ s$ implies $s'\,t \succ s\,t$) and compensate for it with an *argument congruence* rule (ArgCong), as in Bentkamp et al. [13].

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 [39]), relying on heuristics to keep the combinatorial explosion under control. 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 $z\,(\mathsf{sk_x}\, z)\,(\mathsf{sk_y}\, z) \not\approx \mathsf{f}\,(\mathsf{sk_y}\, z)\,(\mathsf{sk_x}\, z)$. 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 challenge for higher-order reasoning [15], although similar difficulties arise with first-order sets and arrays [36]. 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. [68] to derive dynamic completeness of a given clause prover from ground static completeness.

We implemented the calculus in the Zipperposition prover [30] (Sect. 6). Our empirical evaluation includes benchmarks from the TPTP [63] and interactive verification problems exported from Isabelle/HOL [25] (Sect. 7). The results appear promising and suggest that an implementation inside a high-performance prover such as E [59] or Vampire [48] could compete against the rapidly evolving state of the art (Sect. 8).

An earlier version of this article was presented at the 2019 edition of CADE [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 [17] to rank-1 (top-level) polymorphism, as in TH1 [43]. In keeping with standard superposition, we consider only formulas in conjunctive normal form, without explicit quantifiers or Boolean type. We use Henkin semantics [16,33,37], as opposed to the standard semantics that serves as the foundation of the HOL systems [35]. By admitting nonstandard models, Henkin semantics is not subject to Gödel's first incompleteness theorem, allowing us to claim not only soundness but also 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 $\iota \in \Sigma_{\mathsf{ty}}$ and a binary function type constructor $\rightarrow \in \Sigma_{\mathsf{ty}}$ to be present. 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 $\tau \rightarrow \upsilon$ for $\rightarrow(\tau, \upsilon)$. 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 $\mathsf{diff} : \Pi\alpha, \beta.\, (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha$ in Σ. We use diff to express the polymorphic functional extensionality axiom.

A signature is a tuple $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma, \mathcal{V})$. The set of *raw λ-terms* is 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 simply 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 $\tau \rightarrow \upsilon$. If $s : \tau \rightarrow \upsilon$ and $t : \tau$, then the *application* $s\,t$ is a raw λ-term of type υ.

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.

The α-renaming rule is defined as $(\lambda x.\, t) \rightarrow_\alpha (\lambda y.\, t\{x \mapsto y\})$, 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*. A variable occurrence is *free* in a λ-term if it is not bound by a λ-expression. A λ-term is *ground* if it is built without using type variables and contains no free term variables. Using the spine notation [28], λ-terms can be decomposed in a unique way as a non-application *head* t applied to zero or more arguments: $t\, s_1 \ldots s_n$ or $t\, \bar{s}_n$ (abusing notation).

The β- and η-reduction rules are specified on λ-terms as $(\lambda x.\, t)\, u \rightarrow_\beta t\{x \mapsto u\}$ and $(\lambda x.\, t\, x) \rightarrow_\eta t$. For β, bound variables in t are implicitly renamed to avoid capture; for η, the variable x may not occur free in t. The λ-terms form equivalence classes modulo $\beta\eta$-reduction, called *$\beta\eta$-equivalence classes* or simply *terms*. 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 $\rightarrow_\beta$ and $\rightarrow_\eta$ 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 [39, 41, 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 $\bot$.

In general, a substitution $\{\bar{\alpha}_m, \bar{x}_n \mapsto \bar{\upsilon}_m, \bar{s}_n\}$, where each x_j has type τ_j and each s_j has type $\tau_j\{\bar{\alpha}_m \mapsto \bar{\upsilon}_m\}$, maps m type variables to m types and n term variables to n terms. The letters $\theta, \pi, \rho, \sigma$ are reserved for substitutions. Substitutions α-rename 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 $\sigma[\bar{x}_n \mapsto \bar{s}_n]$ denotes the substitution that replaces each x_i by s_i and that otherwise coincides with σ. 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.

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 $\mathcal{J}_{\mathsf{ty}}(\kappa) : \mathcal{U}^n \rightarrow \mathcal{U}$ with each n-ary type constructor κ, such that for all domains $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}$, the set $\mathcal{J}_{\mathsf{ty}}(\rightarrow)(\mathcal{D}_1, \mathcal{D}_2)$ is a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$. The semantics is *standard* if $\mathcal{J}_{\mathsf{ty}}(\rightarrow)(\mathcal{D}_1, \mathcal{D}_2)$ 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 D_i.

The comprehension principle states that every function designated by a λ-expression is contained in the corresponding domain. Loosely following Fitting [33, 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 $\lambda x.\, t$ and all valuations ξ. If a type interpretation $\mathcal{I}_{\mathsf{ty}}$ and a 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 [33, Proposition 2.18]. Given an interpretation $\mathcal{I}$ and a valuation ξ, an equation $s \approx t$ is true if

$[\![s]\!]_{\mathcal{I}}^{\xi}$ and $[\![t]\!]_{\mathcal{I}}^{\xi}$ 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.

Skolemization A problem expressed in higher-order logic must be transformed into clausal normal form before the calculi can be applied. This process works as in the first-order case, except for skolemization. The issue is that skolemization, when performed naively, is unsound for higher-order logic with a Henkin semantics [54, Sect. 6], because it introduces new functions that can be used to instantiate variables.

The core of this article is not affected by this because the problems are given in clausal form. For the implementation, we claim soundness only w.r.t. models that satisfy the axiom of choice, which is the semantics mandated by the TPTP THF format [64]. By contrast, refutational completeness holds w.r.t. arbitrary models as defined above.

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 symbols $\mathsf{true}, \mathsf{false} : \mathit{bool} \in \Sigma$, $\mathsf{implies} : \mathit{bool} \to \mathit{bool} \to \mathit{bool} \in \Sigma$, $\mathsf{forall} : \Pi\alpha.\ (\alpha \to \mathit{bool}) \to \mathit{bool}$, and $\mathsf{equal} : \Pi\alpha.\ \alpha \to \alpha \to \mathit{bool}$, characterized by the axioms

$$\begin{array}{cc} \mathsf{true} \not\approx \mathsf{false} & p \not\approx (\lambda x.\, \mathsf{true}) \lor \mathsf{forall}\langle\alpha\rangle\, p \approx \mathsf{true} \\ a \approx \mathsf{true} \lor a \approx \mathsf{false} & \mathsf{forall}\langle\alpha\rangle\, p \not\approx \mathsf{true} \lor p \approx (\lambda x.\, \mathsf{true}) \\ a \approx \mathsf{true} \lor \mathsf{implies}\, a\, b \approx \mathsf{true} & x \not\approx y \lor \mathsf{equal}\langle\alpha\rangle\, x\, y \approx \mathsf{true} \\ b \not\approx \mathsf{true} \lor \mathsf{implies}\, a\, b \approx \mathsf{true} & \mathsf{equal}\langle\alpha\rangle\, x\, y \not\approx \mathsf{true} \lor x \approx y \\ \mathsf{implies}\, a\, b \not\approx \mathsf{true} \lor a \not\approx \mathsf{true} \lor b \approx \mathsf{true} & \end{array}$$

Proxies for $\neg$, $\lor$, $\land$, and $\exists$ can be defined in terms of the above—for example, $\mathsf{not} \approx (\lambda a.\ \mathsf{implies}\ a\ \mathsf{false})$. Similarly, Hilbert choice can be axiomatized as the proxy $\mathsf{choice} : \Pi\alpha.\ (\alpha \to \mathit{bool}) \to \alpha$ characterized by the axiom $p\ x \not\approx \mathsf{true} \lor p\ (\mathsf{choice}\langle\alpha\rangle\ p) \approx \mathsf{true}$.

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) [64] and TH1 (polymorphic) [43]. 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.

3 The Calculus

Our *clausal λ-superposition calculus* is inspired by the extensional nonpurifying clausal λ-free higher-order superposition calculus described by Bentkamp et al. [13]. The text of this and the next section is partly based on that paper, and the associated journal article [12], (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 $\mathsf{f}\,\mathsf{a}\,\mathsf{a}$.

A term (i.e., a $\beta\eta$-equivalence class) t is defined to be a *green subterm* of a term s if either $s = t$ or $s = \mathsf{f}\langle\bar{\tau}\rangle\ \bar{s}$ for some function symbol f, types $\bar{\tau}$ and terms $\bar{s}$, where t is a green subterm of s_i for some i. 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 set of green positions of a term is defined analogously to the set of positions of a first-order term: ε is a green position of t, and if $t = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{s}$ and p is a green position of s_i, then $i.p$ is a green position of t. We denote the green subterm of s at the green position p by $s|_p$. We write $t = s\langle\!\langle u\rangle\!\rangle_p$ to express that u is a green subterm of t at the green position p and call $s\langle\!\langle\;\rangle\!\rangle_p$ a *green context*; we omit the subscript p if there are no ambiguities.

Another key notion is that of a "fluid" term. A (not necessarily green) subterm t of $s[t]$ is called *fluid* if (1) $t{\downarrow_{\beta\eta}}$ is of the form $y\,\bar{u}_n$, where y is not bound in $s[t]$ and $n \geq 1$, 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 that is not bound in $s[t]$. Intuitively, fluid subterms are terms whose η-short β-normal form can change radically as a result of instantiation. For example, applying $\{z \mapsto \lambda x.\,x\}$ to the fluid term $\lambda x.\,y\,\mathsf{a}\,(z\,x)$ makes the λ vanish: $(\lambda x.\,y\,\mathsf{a}\,x) = y\,\mathsf{a}$. Similarly, $(\lambda x.\,\mathsf{f}\,(y\,x)\,x)\{y \mapsto \lambda x.\,\mathsf{a}\} = (\lambda x.\,\mathsf{f}\,\mathsf{a}\,x) = \mathsf{f}\,\mathsf{a}$.

3.1 Term Order

The calculus is parameterized by a well-founded strict total order $\succ$ on ground terms satisfying the following criteria:

- *green subterm property*: $t\langle\!\langle s\rangle\!\rangle \succeq s$ (i.e., $t\langle\!\langle s\rangle\!\rangle \succ s$ or $t\langle\!\langle s\rangle\!\rangle = s$);
- *compatibility with green contexts*: $s' \succ s$ implies $t\langle\!\langle s'\rangle\!\rangle \succ t\langle\!\langle s\rangle\!\rangle$.

The literal and clause orders are defined as multiset extensions in the standard way [6]. 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.

For nonground terms, $\succ$ is extended to a strict partial order so that $t \succ s$ if and only if $t\theta \succ s\theta$ for all grounding substitutions θ. This makes $\succ$ stable under substitutions. In practice, one can underapproximate this extension of $\succ$ to nonground terms by any other extension of $\succ$ to nonground terms that is also stable under substitutions. We also introduce a quasiorder $\succsim$ such that $t \succsim s$ if and only if $t\theta \succeq s\theta$ for all grounding θ. The quasiorder $\succsim$ is more precise than the nonstrict order $\succeq$. For example, we have $x\,\mathsf{b} \not\succeq x\,\mathsf{a}$ because $x\,\mathsf{b} \neq x\,\mathsf{a}$ and $x\,\mathsf{b} \not\succ x\,\mathsf{a}$ by stability under substitutions with $\{x \mapsto \lambda y.\,c\}$. But we can have $x\,\mathsf{b} \succsim x\,\mathsf{a}$.

Our approach to derive a suitable order is to encode η-short β-normal forms into untyped λ-free higher-order terms and apply an order $\succ_{\mathsf{base}}$ such as the λ-free Knuth–Bendix order (KBO) [9], the λ-free lexicographic path order (LPO) [23], or the embedding path order (EPO) [10]. The encoding, denoted by $\lfloor\;\rfloor$, translates $\lambda x\!:\!\tau.\,t$ to $\mathsf{lam}\,\lfloor\tau\rfloor\,\lfloor t\rfloor$ and uses De Bruijn [27] symbols db_i to represent bound variables x. It replaces fluid terms t by fresh variables z_t and maps type arguments to term arguments, while erasing any other type information; thus, $\lfloor\lambda x\!:\!\iota.\,\lambda y\!:\!\iota.\,x\rfloor = \mathsf{lam}\,\iota\,(\mathsf{lam}\,\iota\,(\mathsf{db}_1\,\iota))$ and $\lfloor\mathsf{f}\langle\iota\rangle(y\,\mathsf{a})\rfloor = \mathsf{f}\,\iota\,z_{y\,\mathsf{a}}$. The use of De Bruijn indices and the monolithic encoding of fluid terms ensure stability under α-renaming and substitution.

More precisely, the encoding $\lfloor\;\rfloor$ is composed of two steps, $\lfloor\;\rfloor_{\mathsf{db}}$ and $\lfloor\;\rfloor_{\mathsf{lam}}$. We also use the notation $\lceil\;\rceil_{\mathsf{db}}$ for the inverse of $\lfloor\;\rfloor_{\mathsf{db}}$. Given the signature $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma, \mathcal{V})$, $\lfloor\;\rfloor_{\mathsf{db}}$ encodes terms into $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma \uplus \{\mathsf{db}_i \mid i \in \mathbb{N}\}, \mathcal{V})$ by replacing each occurrence of a bound variable by db_i, where i is the number of λs in the term structure between the bound variable and its λ-binder. For types, we let $\lfloor\tau\rfloor_{\mathsf{db}} = \tau$. Then, $\lfloor\;\rfloor_{\mathsf{lam}}$ encodes these types and terms further as terms over the untyped λ-free signature $(\Sigma_{\mathsf{ty}} \uplus \Sigma \uplus \{\mathsf{lam}\} \uplus \{\mathsf{db}_i \mid i \in \mathbb{N}\}, \{z_t \mid t \text{ is a term}\})$. The type-to-term version of $\lfloor\ \rfloor_{\mathsf{lam}}$ is defined by $\lfloor\alpha\rfloor_{\mathsf{lam}} = \alpha$ and $\lfloor\kappa(\bar{\tau})\rfloor_{\mathsf{lam}} = \kappa\,\lfloor\bar{\tau}\rfloor_{\mathsf{lam}}$. The term-to-term version is defined by

$$\lfloor t\rfloor_{\mathsf{lam}} = \begin{cases} z_t & \text{if } t = x \text{ or if } \lceil t\rceil_{\mathsf{db}} \text{ is fluid} \\ \mathsf{f}\,\lfloor\bar{\tau}\rfloor_{\mathsf{lam}}\,\lfloor\bar{u}\rfloor_{\mathsf{lam}} & \text{if } t = \mathsf{f}\langle\bar{\tau}\rangle\,\bar{u} \\ \mathsf{lam}\,\lfloor\tau\rfloor_{\mathsf{lam}}\,\lfloor u\rfloor_{\mathsf{lam}} & \text{if } t = (\lambda x{:}\,\tau.\,u) \text{ and } \lceil t\rceil_{\mathsf{db}} \text{ is not fluid} \end{cases}$$

For any λ-terms t and s, let $\lfloor t\rfloor = \lfloor\lfloor t\rfloor_{\mathsf{db}}\rfloor_{\mathsf{lam}}$ and let $t \succ_{\mathsf{meta}} s$ be $\lfloor t\rfloor \succ_{\mathsf{base}} \lfloor s\rfloor$.

Lemma 1 *Let $\succ_{\mathsf{base}}$ be a strict partial order on λ-free terms. If the restriction of $\succ_{\mathsf{base}}$ to ground terms enjoys well-foundedness, totality, the green subterm property, and compatibility with green contexts, the restriction of $\succ_{\mathsf{meta}}$ to ground terms enjoys the same properties.*

Proof Transitivity and irreflexivity of $\succ_{\mathsf{meta}}$ each follow directly from the corresponding property of $\succ_{\mathsf{base}}$.

WELL-FOUNDEDNESS: If there existed an infinite descending chain $t_1 \succ_{\mathsf{meta}} t_2 \succ_{\mathsf{meta}} \cdots$ of ground terms, there would exist the chain $\lfloor t_1\rfloor \succ_{\mathsf{base}} \lfloor t_2\rfloor \succ_{\mathsf{base}} \cdots$, contradicting the well-foundedness of $\succ_{\mathsf{base}}$ on ground terms.

TOTALITY: By totality of $\succ_{\mathsf{base}}$, for any ground terms t and s, we have $\lfloor t\rfloor \succ_{\mathsf{base}} \lfloor s\rfloor$, $\lfloor t\rfloor \prec_{\mathsf{base}} \lfloor s\rfloor$, or $\lfloor t\rfloor = \lfloor s\rfloor$. In the first two cases, it follows that $t \succ_{\mathsf{meta}} s$ or $t \prec_{\mathsf{meta}} s$. In the last case, it follows that $t = s$ because the encoding $\lfloor\ \rfloor$ is clearly injective.

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

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

Lemma 2 *Let $\succ_{\mathsf{base}}$ be a strict partial order on λ-free terms. If $\succ_{\mathsf{base}}$ enjoys stability under substitution (w.r.t. λ-free terms), $\succ_{\mathsf{meta}}$ enjoys stability under substitution (w.r.t. $\beta\eta$-equivalence classes).*

Proof Let t be a λ-term, and let θ be a substitution. Define a λ-free substitution ρ such that $z_{\lfloor t\rfloor_{\mathsf{db}}}\rho = \lfloor t\theta\rfloor$ for each variable $z_{\lfloor t\rfloor_{\mathsf{db}}}$ in θ's domain and otherwise $x\rho = x$. Then $\lfloor t\rfloor\rho = \lfloor t\theta\rfloor$ for all t by induction on the definition of the encoding. If $\lfloor t\rfloor_{\mathsf{db}} = x$ or if t is fluid, $\lfloor t\rfloor\rho = z_{\lfloor t\rfloor_{\mathsf{db}}}\rho = \lfloor t\theta\rfloor$. If $\lfloor t\rfloor_{\mathsf{db}} = \mathsf{f}\langle\bar{\tau}\rangle\,\lfloor\bar{u}\rfloor_{\mathsf{db}}$, then $\lfloor t\rfloor\rho = \mathsf{f}\,(\lfloor\bar{\tau}\rfloor\rho)\,(\lfloor\bar{u}\rfloor\rho) \overset{\text{IH}}{=} \mathsf{f}\,\lfloor\bar{\tau}\theta\rfloor\,\lfloor\bar{u}\theta\rfloor = \lfloor\mathsf{f}\langle\bar{\tau}\theta\rangle\,(\bar{u}\theta)\rfloor = \lfloor t\theta\rfloor$. If $\lfloor t\rfloor_{\mathsf{db}} = (\lambda x:\tau.\,\lfloor u\rfloor_{\mathsf{db}})$ and t is not fluid, then $\lfloor t\rfloor\rho = \mathsf{lam}\,(\lfloor\tau\rfloor\rho)\,(\lfloor u\rfloor\rho) \overset{\text{IH}}{=} \mathsf{lam}\,\lfloor\tau\theta\rfloor\,\lfloor u\theta\rfloor = \lfloor\lambda x:\tau\theta.\,\lfloor u\theta\rfloor_{\mathsf{db}}\rfloor_{\mathsf{lam}} = \lfloor\lambda x:\tau\theta.\,u\theta\rfloor = \lfloor(\lambda x:\tau.\,u)\theta\rfloor = \lfloor t\theta\rfloor$.

Assume $t \succ_{\mathsf{meta}} s$ for some terms t and s. Then $\lfloor t\rfloor \succ_{\mathsf{base}} \lfloor s\rfloor$ and by stability under substitutions, $\lfloor t\rfloor\rho \succ_{\mathsf{base}} \lfloor s\rfloor\rho$. With the above observation, it follows that $\lfloor t\theta\rfloor \succ_{\mathsf{base}} \lfloor s\theta\rfloor$ and hence $t\theta \succ_{\mathsf{meta}} s\theta$. □

3.2 The Inference Rules

In addition to $\succ$, the calculus is parameterized by a selection function *sel* that maps each clause to a subclause consisting of negative literals. A literal $L\langle y\rangle$ may not be selected if $y\,\bar{u}_n$, with $n > 0$, is a $\succsim$-maximal term of the clause.

A literal L is (*strictly*) *eligible* in C if it is selected in C or if there are no selected literals in C and L is (strictly) maximal in C. A variable is *deep* in a clause C if it occurs inside a λ-expression or inside an argument of an applied variable; these cover all occurrences that may correspond to positions inside λ-expressions after applying a substitution. A variable that is not deep is said to be *shallow*.

We regard positive and negative superposition as two cases of a single rule

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \quad \overbrace{C' \vee [\neg]\, s\langle u\rangle \approx s'}^{C}}{(D' \vee C' \vee [\neg]\, s\langle t'\rangle \approx s')\sigma}\,\textsc{Sup}$$

with the following side conditions:

1. u is not a fluid subterm;
2. u is not a deep variable 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. $t\sigma \not\precsim t'\sigma$; 6. $s\langle u\rangle\sigma \not\precsim s'\sigma$; 7. $C\sigma \not\precsim D\sigma$;
8. $(t \approx t')\sigma$ is strictly eligible in $D\sigma$;
9. $([\neg]\, s\langle u\rangle \approx s')\sigma$ is eligible in $C\sigma$, and strictly 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 $\not\preceq$ is replaced by the more precise $\not\precsim$.

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

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

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

1. u is either a deep variable in C or a fluid subterm;
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:

$$\frac{C' \vee u \not\approx u'}{C'\sigma}\,\textsc{EqRes} \qquad\qquad \frac{C' \vee u' \approx v' \vee u \approx v}{(C' \vee v \not\approx v' \vee u \approx v')\sigma}\,\textsc{EqFact}$$

For EqRes: $\sigma \in \mathrm{CSU}(u,u')$ and $(u \not\approx u')\sigma$ is eligible in the premise. For EqFact: $\sigma \in \mathrm{CSU}(u,u')$, $u'\sigma \not\precsim v'\sigma$, $u\sigma \not\precsim v\sigma$, and $(u \approx v)\sigma$ is eligible in the premise.

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

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

where σ is the most general type substitution that ensures well-typedness of the conclusion. In particular, if the result type of s is not a type variable, σ is the identity substitution; and if the result type is a type variable, it is instantiated with $\alpha_1 \to \cdots \to \alpha_m \to \beta$, where $\bar{\alpha}_m$ and β are fresh. This yields infinitely many conclusions, one for each m. The literal $s\sigma \approx s'\sigma$ must be strictly eligible in $(C' \vee s \approx s')\sigma$, and $\bar{x}_n$ is a nonempty tuple of distinct fresh variables.

The rules are complemented by the polymorphic functional extensionality axiom:

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

In the sequel, we will omit the type arguments to diff since they can be inferred from the term arguments.

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

Lemma 3 (Substitution lemma) *Let $\mathcal{I} = (\mathcal{I}_{\mathsf{ty}}, \mathcal{J}, \mathcal{L})$ be a proper interpretation. Then*

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

for all terms $t \in \mathcal{T}_{\mathrm{H}}$, all types $\tau \in \mathcal{T}y_{\mathrm{H}}$, and all substitutions ρ, where $\xi'(\alpha) = [\![\alpha\rho]\!]^{\xi}_{\mathcal{I}_{\mathsf{ty}}}$ for all type variables α and $\xi'(x) = [\![x\rho]\!]^{\xi}_{\mathcal{I}}$ 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 $\bar{\upsilon}$,

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

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

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

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

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

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

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

If t is a λ-expression, 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

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

Lemma 4 *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 is true in $\mathcal{I}$ for all valuations ξ. Given a valuation ξ, define ξ' as in Lemma 3. Then, by Lemma 3, 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.

Theorem 5 (Soundness) *The inference rules* SUP, FLUIDSUP, EQRES, EQFACT, *and* ARGCONG *are sound.*

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

3.3 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.

Because it gives rise to flex–flex pairs, which are unification constraints where both sides are applied variables, FLUIDSUP can be very prolific. With applied variables 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, EQRES, and EQFACT). Due to order restrictions and fairness, we cannot postpone solving flex–flex pairs indefinitely. Thus, we cannot use Huet's pre-unification procedure [39] and must instead choose a full unification procedure such as Jensen and Pietrzykowski's [40], Snyder and Gallier's [61], or the one by Vukmirović et al. [66]. 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 keep the explosion under control. 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: $\{y \mapsto \lambda w\,x.\,y'\,w\,x\,\mathsf{c}\,\mathsf{d},\ z \mapsto y'\,\mathsf{a}\,\mathsf{b}\}$. Full higher-order unification is currently being investigated by two of this article's authors [66].

The calculus is a graceful generalization of standard superposition, except for the extensionality axiom. From $\mathsf{g}\,x \approx \mathsf{f}\,x\,x$, the axiom can be used to derive clauses such as $(\lambda x.\,y\,x\,(\mathsf{g}\,x)) \approx (\lambda x.\,y\,x\,(\mathsf{f}\,x\,x))$, which are useless if the problem is first-order. This could be avoided if we could find a way to make the positive literal $y \approx z$ 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.

Example 6 Prefix subterms such as g in the term $\mathsf{g}\,\mathsf{a}$ are not green subterms and thus cannot be superposed into. ARGCONG gives us access to those positions. Consider the clauses $\mathsf{g}\,\mathsf{a} \not\approx \mathsf{f}\,\mathsf{a}$ and $\mathsf{g} \approx \mathsf{f}$. An ARGCONG inference from $\mathsf{g} \approx \mathsf{f}$ generates $\mathsf{g}\,x \approx \mathsf{f}\,x$. This clause can be used for a SUP inference into the first clause, yielding $\mathsf{f}\,\mathsf{a} \not\approx \mathsf{f}\,\mathsf{a}$ and thus $\bot$ by EQRES.

Example 7 Applied variables give rise to subtle situations with no counterparts in first-order logic. Consider the clauses $\mathsf{f}\,\mathsf{a} \approx \mathsf{c}$ and $\mathsf{h}\,(y\,\mathsf{b})\,(y\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f}\,\mathsf{b}))\,(\mathsf{g}\,\mathsf{c})$, where $\mathsf{f}\,\mathsf{a} \succ \mathsf{c}$. 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 $\mathsf{f}\,\mathsf{a}$ of $\mathsf{g}\,(\mathsf{f}\,\mathsf{a}) = (\lambda x.\,\mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (y\,\mathsf{a})\theta$.

FLUIDSUP's z variable 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 $\{y \mapsto \lambda x.\, z\,(\mathsf{f}\,x)\}$. The resulting clause is $\mathsf{h}\,(z\,(\mathsf{f\,b}))\,(z\,\mathsf{c}) \not\approx \mathsf{h}\,(\mathsf{g}\,(\mathsf{f\,b}))\,(\mathsf{g\,c})$, from which $\bot$ follows by EQRES.

Example 8 The clause set consisting of $\mathsf{f\,a} \approx \mathsf{c}$, $\mathsf{f\,b} \approx \mathsf{d}$, and $\mathsf{g\,c} \not\approx y\,\mathsf{a} \lor \mathsf{g\,d} \not\approx y\,\mathsf{b}$ has a similar flavor. EQRES is applicable on either literal of the third clause, but the computed unifier, $\{y \mapsto \lambda x.\, \mathsf{g\,c}\}$ or $\{y \mapsto \lambda x.\, \mathsf{g\,d}\}$, is not the right one. Again, we need FLUIDSUP.

Example 9 Third-order clauses containing subterms of the form $y\,(\lambda x.\, t)$ can be even more stupefying. The clause set containing $\mathsf{f\,a} \approx \mathsf{c}$ and $\mathsf{h}\,(y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\,y \not\approx \mathsf{h}\,(\mathsf{g\,c})\,(\lambda w\,x.\, w\,x)$ is unsatisfiable. To see why, apply $\theta = \{y \mapsto \lambda w\,x.\, w\,x\}$ to the second clause, yielding $\mathsf{h}\,(\mathsf{g}\,(\mathsf{f\,a}))\,(\lambda w\,x.\, w\,x) \not\approx \mathsf{h}\,(\mathsf{g\,c})\,(\lambda w\,x.\, w\,x)$. Let $\mathsf{f\,a} \succ \mathsf{c}$. A SUP inference is possible between the first clause and this ground instance of the second one. But at the nonground level, the subterm $\mathsf{f\,a}$ is not clearly localized: $\mathsf{g}\,(\mathsf{f\,a}) = (\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (\lambda w\,x.\, w\,x)\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a} = (y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a})\theta$. The FLUIDSUP rule can cope with this. One of the unifiers of $z\,(\mathsf{f\,a})$ and $y\,(\lambda x.\, \mathsf{g}\,(\mathsf{f}\,x))\,\mathsf{a}$ will be $\{y \mapsto \lambda w\,x.\, w\,x,\ z \mapsto \mathsf{g}\}$, yielding the clearly unsatisfiable clause $\mathsf{h}\,(\mathsf{g\,c})\,(\lambda w\,x.\, w\,x) \not\approx \mathsf{h}\,(\mathsf{g\,c})\,(\lambda w\,x.\, w\,x)$.

Example 10 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 clauses $\mathsf{g\,a} \approx \mathsf{b}$, $\mathsf{h}\,(\lambda y.\, x\,y\,\mathsf{g}\,z) \approx \mathsf{c}$, and $\mathsf{h}\,(\mathsf{f\,b}) \not\approx \mathsf{c}$. Applying $\{x \mapsto \lambda y'\,w\,z'.\, \mathsf{f}\,(w\,\mathsf{a})\,y'\}$ to the second clause yields $\mathsf{h}\,(\lambda y.\, (\lambda y'\,w\,z'.\, \mathsf{f}\,(w\,\mathsf{a})\,y')\,y\,\mathsf{g}\,z) \approx \mathsf{c}$, which β-reduces to $\mathsf{h}\,(\lambda y.\, \mathsf{f}\,(\mathsf{g\,a})\,y) \approx \mathsf{c}$ and $\beta\eta$-reduces to $\mathsf{h}\,(\mathsf{f}\,(\mathsf{g\,a})) \approx \mathsf{c}$. A SUP inference is possible between the first clause set and this new ground clause, generating the clause $\mathsf{h}\,(\mathsf{f\,b}) \approx \mathsf{c}$. By also considering λ-expressions, the FLUIDSUP rule is applicable at the nonground level to derive this clause.

3.4 Redundancy Criterion

A redundant (or composite) 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 its type arguments and argument count. For example, $\mathcal{F}(\mathsf{f\,a}) = \mathsf{f}_1(\mathsf{a}_0)$ and $\mathcal{F}(\mathsf{g}\langle\iota\rangle) = \mathsf{g}_0^\iota$. 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 $\{\mathsf{g} \approx \mathsf{f},\ \mathsf{g\,a} \not\approx \mathsf{f\,a}\}$ 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 $\{\mathsf{b}_0 \approx \mathsf{a}_0,\ \mathsf{c}_0 \not\approx \mathsf{d}_0\}$ are satisfiable in first-order logic.

Given a higher-order signature $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma, \mathcal{V})$, 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 $\rightarrow$ is uninterpreted in first-order logic. For each ground instance $\mathsf{f}\langle\bar{\upsilon}\rangle : \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau$ 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 result type $\tau_{j+1} \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau$, 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 layers of logics: the higher-order layer H over a given signature $(\Sigma_{\mathsf{ty}}, \mathcal{V}_{\mathsf{ty}}, \Sigma, \mathcal{V})$, the ground higher-order layer GH over the signature $(\Sigma_{\mathsf{ty}}, \{\}, \Sigma, \{\})$, and the ground monomorphic first-order layer 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, $\mathcal{T}y_{\mathrm{H}}$, $\mathcal{T}y_{\mathrm{GH}}$, and $\mathcal{T}y_{\mathrm{GF}}$ 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 layers has an entailment relation $\models$. A clause set N_1 entails a clause set N_2, denoted $N_1 \models N_2$, if any 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 layers are connected via two functions $\mathcal{G}$ and $\mathcal{F}$. The grounding function $\mathcal{G}$ maps terms from $\mathcal{T}_{\mathrm{H}}$ to the set of their ground instances in $\mathcal{T}_{\mathrm{GH}}$ and clauses from $\mathcal{C}_{\mathrm{H}}$ to the set of their ground instances in $\mathcal{C}_{\mathrm{GH}}$. The encoding $\mathcal{F} : \mathcal{T}_{\mathrm{GH}} \to \mathcal{T}_{\mathrm{GF}}$ is recursively defined as

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

using η-short β-normal representatives of terms. The encoding $\mathcal{F}$ is extended to map GH clauses to GF clauses in the obvious way. 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 $\mathcal{F}^{-1}(C) \succ \mathcal{F}^{-1}(D)$. The property that $\succ$ on clauses is the multiset extension of $\succ$ on literals, which in turn is the multiset extension of $\succ$ on terms, is maintained because $\mathcal{F}^{-1}$ maps the multiset representations elementwise.

A key property of $\mathcal{F}$ is that green subterms in GH correspond to subterms in GF. This allows us to show that well-foundedness, totality on ground terms, compatibility with contexts, and the subterm property hold for $\succ$ on GF terms. Moreover, the subterms considered by SUP and FLUIDSUP include all the subterms exposed to the redundancy criterion.

Lemma 11 *Let $s, t \in \mathcal{T}_{\mathrm{GH}}$. We have $\mathcal{F}(t\langle\!\langle s \rangle\!\rangle_p) = \mathcal{F}(t)[\mathcal{F}(s)]_p$. In other words, s is a green subterm of t at position p if and only if $\mathcal{F}(s)$ is a subterm of $\mathcal{F}(t)$ at position p.*

Proof Analogous to Lemma 3.8 of Bentkamp et al. [12].

Lemma 12 *Well-foundedness, totality, compatibility with contexts, and the subterm property hold for $\succ$ in $\mathcal{T}_{\mathrm{GF}}$.*

Proof Analogous to Lemma 3.10 of Bentkamp et al. [12].

The saturation procedures of superposition provers aggressively 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 $\mathsf{b} \not\approx \mathsf{a} \lor x \approx \mathsf{c}$. The proof of refutational completeness of resolution and superposition provers relies on the well-foundedness of the strict subsumption relation [58, Sect. 7]. 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 $\sqsupseteq$ as $\sqsupseteq_{\mathrm{subs}} \cap \sqsupseteq_{\mathrm{synt}}$, where $\sqsupseteq_{\mathrm{subs}}$ stands for "subsumed by" and $\sqsupseteq_{\mathrm{synt}}$ for "'syntactically larger than or of same syntactic size as." Then let $D \sqsupset C$ if $D \sqsupseteq C$ and $C \not\sqsupseteq D$. 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. [68].

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 $D \in \mathcal{G}(C)$.

Along with the three layers of logics, we consider three inference systems: *HInf*, *GHInf*, and *GFInf*. *HInf* is the inference system described in Sect. 3.2. For uniformity, we regard the extensionality axiom as a premise-free inference rule EXT whose conclusion is the (EXT) axiom. *GHInf* consists of all SUP, EQRES, and EQFACT inferences from *HInf* whose premises and conclusion are ground, 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 eligible in $C' \vee s \approx s'$ and $\bar{u}_n$ is a nonempty tuple of ground terms. *GFInf* contains all SUP, EQRES, and EQFACT inferences from *GHInf* translated by $\mathcal{F}$. It coincides exactly with standard first-order superposition. Given a SUP, EQRES, or EQFACT inference $\iota \in \mathit{GHInf}$, let $\mathcal{F}(\iota)$ denote the corresponding inference in *GFInf*.

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.

Each of the three inference systems is parameterized by a selection function *sel*. Occasionally, we will make this dependency explicit, writing $\mathit{HInf}^{\mathit{sel}}$ for *HInf* and similarly for *GHInf* and *GFInf*. For each selection function *sel* on $\mathcal{C}_{\mathrm{GH}}$, via the bijection $\mathcal{F}$, we can obtain a corresponding selection function on $\mathcal{C}_{\mathrm{GF}}$, which we denote $\mathcal{F}(\mathit{sel})$. There is, however, no general way to derive the right selection function on $\mathcal{C}_{\mathrm{GH}}$ from a selection function on $\mathcal{C}_{\mathrm{H}}$. In the refutational completeness proof, given a saturated clause set $N \subseteq \mathcal{C}_{\mathrm{H}}$ and a selection function on $\mathcal{C}_{\mathrm{H}}$, we need a selection function on $\mathcal{C}_{\mathrm{GH}}$ such that for each clause $C \in \mathcal{G}(N)$ there exists a clause $D \in N$ with $C \in \mathcal{G}(D)$ and corresponding selected literals. Since the saturated clause set N is not known during a derivation, our redundancy criterion must not depend on it. Therefore, we consider 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. Given a selection function *sel* on $\mathcal{C}_{\mathrm{H}}$, let $\mathcal{G}(\mathit{sel})$ denote the set of such selection functions on $\mathcal{C}_{\mathrm{GH}}$.

Given a selection function *sel* on $\mathcal{C}_{\mathrm{H}}$, a selection function $\mathit{gsel} \in \mathcal{G}(\mathit{sel})$, and an inference $\iota \in \mathit{HInf}^{\mathit{sel}}$, we define the set $\mathcal{G}^{\mathit{gsel}}(\iota)$ of ground instances of ι to be all inferences $\iota' \in \mathit{GHInf}^{\mathit{gsel}}$ 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, EQFACT to EQFACT, EQRES to EQRES, EXT to GEXT, and ARGCONG to GARGCONG inferences, but it is also possible that $\mathcal{G}^{\mathit{gsel}}(\iota)$ 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 $\iota \in \mathit{GFInf}^{\mathit{gsel}}$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$, let $\iota \in \mathit{GFRed}_{\mathrm{I}}^{\mathit{gsel}}(N)$ if $\mathit{prems}(\iota) \cap \mathit{GFRed}_{\mathrm{C}}(N) \neq \varnothing$ or $\{D \in N \mid D \prec \mathit{mprem}(\iota)\} \models \mathit{concl}(\iota)$.
- Given $\iota \in \mathit{GHInf}^{\mathit{gsel}}$ and $N \subseteq \mathcal{C}_{\mathrm{GH}}$, let $\iota \in \mathit{GHRed}_{\mathrm{I}}^{\mathit{gsel}}(N)$ if
  - ι is not a GARGCONG or GEXT inference and $\mathcal{F}(\iota) \in \mathit{GFRed}_{\mathrm{I}}^{\mathcal{F}(\mathit{gsel})}(\mathcal{F}(N))$; or
  - ι is a GARGCONG or GEXT inference and $\mathit{concl}(\iota) \in N \cup \mathit{GHRed}_{\mathrm{C}}(N)$.
- Given $\iota \in \mathit{HInf}^{\mathit{sel}}$ and $N \subseteq \mathcal{C}_{\mathrm{H}}$, let $\iota \in \mathit{HRed}_{\mathrm{I}}(N)$ if $\mathcal{G}^{\mathit{gsel}}(\iota) \subseteq \mathit{GHRed}_{\mathrm{I}}(\mathcal{G}(N))$ for all $\mathit{gsel} \in \mathcal{G}(\mathit{sel})$.

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 a redundancy criterion $(\mathit{Red}_{\mathrm{I}}, \mathit{Red}_{\mathrm{C}})$ if every inference from clauses in N is in $\mathit{Red}_{\mathrm{I}}(N)$.

4 Refutational Completeness

Besides soundness, the most important property of the clausal λ-superposition calculus introduced in Sect. 3 is refutational completeness. Parts of the proof are inspired by Bentkamp et al. [13].

4.1 Outline of the Proof

The proof proceeds in three steps, corresponding to the three layers 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. [68] 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 GEXT.

In the third step, we employ the saturation framework by Waldmann et al. [68] to prove refutational completeness of *HInf*. The main proof obligation the framework leaves to us is that there 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 on layer GH into a position corresponding to a fluid term or a deep variable on layer 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 dog that did not bark. On layer GH, the rules SUP, EQRES, and EQFACT preserve η-short β-normal form, and so does first-order term rewriting. Thus, we can completely ignore $\rightarrow_\beta$

and $\rightarrow_\eta$. On layer H, instantiation can cause β- and η-reduction, but this poses no difficulties thanks to the clause order's stability under substitution.

4.2 The Ground First-Order Layer

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. 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 $E_N^C = \{s \rightarrow t\}$ if the following conditions are met:

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

Then C is said to *produce* $s \rightarrow t$. Otherwise, $E_N^C = \emptyset$. Finally, $R_N = \bigcup_D E_N^D$. Based on Bachmair and Ganzinger's work, Bentkamp et al. [12, Lemma 4.2, Theorem 4.3] prove the following properties of R_N:

Lemma 13 *Let $\bot \notin N$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$ be saturated w.r.t. GFInf and GFRed$_{\mathrm{I}}$. If $C = C' \vee s \approx t \in N$ produces $s \rightarrow t$, then $s \approx t$ is strictly eligible in C and C' is false in R_N.*

Theorem 14 (Ground first-order static refutational completeness) *Let $\bot \notin N$ and $N \subseteq \mathcal{C}_{\mathrm{GF}}$ be saturated w.r.t. GFInf and GFRed$_{\mathrm{I}}$. Then R_N is a model of N.*

4.3 The Ground Higher-Order Layer

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

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 15 *For all terms $t, s : \tau \rightarrow \upsilon$ 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) ⇒ (2): Take $u = \mathsf{diff}\,t\,s$.

(2) ⇒ (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 14. Therefore, it follows from $t\,(\mathsf{diff}\,t\,s) \sim s\,(\mathsf{diff}\,t\,s)$ that $t \sim s$.

(1) ⇒ (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 $\mathcal{F}(t) \rightarrow_R \mathcal{F}(s)$ implies $t\,u \sim s\,u$. If the rewrite step $\mathcal{F}(t) \rightarrow_R \mathcal{F}(s)$ 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 $\mathcal{F}(t\,u) \rightarrow_R \mathcal{F}(s\,u)$ using the same rewrite rule. It follows that $t\,u \sim s\,u$. If the rewrite step $\mathcal{F}(t) \rightarrow_R \mathcal{F}(s)$ is at the top level, $\mathcal{F}(t) \rightarrow \mathcal{F}(s)$ must be a rule of R. This rule must come from a productive clause of the form $\mathcal{F}(C) = \mathcal{F}(C' \vee t \approx s)$. By Lemma 13, $\mathcal{F}(t \approx s)$ is strictly eligible in $\mathcal{F}(C)$ w.r.t. $\mathcal{F}(\mathit{gsel})$, and hence $t \approx s$ is strictly eligible in C w.r.t. gsel. Moreover, t and s have functional type. Thus, there is this GARGCONG inference ι:

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

By saturation, ι is redundant w.r.t. N—i.e., we have $\mathit{concl}(\iota) \in N \cup \mathit{GHRed}_{\mathrm{C}}(N)$. By Theorem 14, $\mathcal{F}(\mathit{concl}(\iota))$ is then true in R. By Lemma 13, $\mathcal{F}(C')$ is false in R. Therefore, $\mathcal{F}(t\,u \approx s\,u)$ must be true in R. ☐

Lemma 16 *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 statement, 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 $\oplus : \Pi\alpha.\, \alpha \rightarrow \alpha \rightarrow \alpha$. We will omit its type argument. It is equipped with two reduction rules: $\oplus\,t\,s \rightarrow t$ and $\oplus\,t\,s \rightarrow s$.

The computability path order $\succ_{\mathsf{CPO}}$ [24] guarantees that

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

Thus this order is compatible with the $\oplus$-reduction rules and β-reduction. Since this order is moreover monotone and well-founded, these reduction rules terminate. And since the reduction rules describe a finitely branching term rewrite system, by Kőnig's lemma [45], 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 term variables occurring more than once in s; and
- $n_3(s) = \mathcal{S}(s)$.

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 $s\rho'\sigma$. 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 $s\theta = s\rho\theta$, $s\rho\theta' = s\rho'\theta$, and $s\rho'\theta' = s\theta'$, this implies $s\theta \sim s\theta'$ by transitivity of $\sim$, as illustrated below:

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

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 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 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 term variable occurrence, we have $n_2(s) = 0 = n_2(u)$. But $n_3(s) > n_3(u)$ because u is a non-term-ground 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 15 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 15. 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 is applicable 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 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 non-term-ground. 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 $s\sigma \rightarrow_\beta s'\sigma$ 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 $\mathcal{I}^{\text{GH}} = (\mathcal{U}^{\text{GH}}, \mathcal{J}_{\text{ty}}^{\text{GH}}, \mathcal{J}^{\text{GH}}, \mathcal{L}^{\text{GH}})$ 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. When defining the universe $\mathcal{U}^{\text{GH}}$ of the higher-order interpretation, we need to ensure that it contains subsets of function spaces, since $\mathcal{J}_{\text{ty}}^{\text{GH}}(\rightarrow)(\mathcal{D}_1, \mathcal{D}_2)$ must be a subset of the function space from $\mathcal{D}_1$ to $\mathcal{D}_2$ for all $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{U}^{\text{GH}}$. But the first-order universes $\mathcal{U}_\tau$ consist of equivalence classes of terms from $\mathcal{T}_{\text{GF}}$ w.r.t. the rewrite system R, not of functions.

Towards this end, 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 $\mathcal{U}^{\text{GH}}$ be the set of all these domains $\mathcal{D}_\tau$. Thus, there will be a one-to-one correspondence between ground types and domains.

We define $\mathcal{E}_\tau$ and $\mathcal{D}_\tau$ in a mutual induction and prove that $\mathcal{E}_\tau$ is a bijection simultaneously. 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. We proceed by defining $\mathcal{E}_{\tau\to\upsilon}$ and $\mathcal{D}_{\tau\to\upsilon}$. We assume that $\mathcal{E}_\tau$, $\mathcal{E}_\upsilon$, $\mathcal{D}_\tau$, and $\mathcal{D}_\upsilon$ have already been defined and that $\mathcal{E}_\tau$, $\mathcal{E}_\upsilon$ are bijections. To ensure that $\mathcal{E}_{\tau\to\upsilon}$ will be bijective, we first define an injective function $\mathcal{E}^0_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \longrightarrow \mathcal{D}_\tau \longrightarrow \mathcal{D}_\upsilon$, define $\mathcal{D}_{\tau\to\upsilon}$ as its image $\mathcal{E}^0_{\tau\to\upsilon}(\mathcal{U}_{\tau\to\upsilon})$, and finally define $\mathcal{E}_{\tau\to\upsilon}$ as $\mathcal{E}^0_{\tau\to\upsilon}$ with its codomain restricted to $\mathcal{D}_{\tau\to\upsilon}$:

$$\mathcal{E}^0_{\tau\to\upsilon} : \mathcal{U}_{\tau\to\upsilon} \longrightarrow \mathcal{D}_\tau \longrightarrow \mathcal{D}_\upsilon$$
$$\mathcal{E}^0_{\tau\to\upsilon}(\llbracket \mathcal{F}(s) \rrbracket_R)\big(\mathcal{E}_\tau\big(\llbracket \mathcal{F}(u) \rrbracket_R\big)\big) = \mathcal{E}_\upsilon\big(\llbracket \mathcal{F}(s\, u) \rrbracket_R\big)$$

This is a valid definition because each element of $\mathcal{U}_{\tau\to\upsilon}$ is of the form $\llbracket \mathcal{F}(s) \rrbracket_R$ for some s and each element of $\mathcal{D}_\tau$ is of the form $\mathcal{E}_\tau\big(\llbracket \mathcal{F}(u) \rrbracket_R\big)$ for some u. This function is well defined if it does not depend on the choice of s and u. To show this, 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 $\llbracket \mathcal{F}(u) \rrbracket_R = \llbracket \mathcal{F}(v) \rrbracket_R$. Applying Lemma 16 to the term $x\, y$ and the substitutions $\{x \mapsto s, y \mapsto u\}$ and $\{x \mapsto t, y \mapsto v\}$, we obtain $\llbracket \mathcal{F}(s\, u) \rrbracket_R = \llbracket \mathcal{F}(t\, v) \rrbracket_R$, indicating that $\mathcal{E}^0_{\tau\to\upsilon}$ is well defined. It remains to show that $\mathcal{E}^0_{\tau\to\upsilon}$ is injective as a function from $\mathcal{U}_{\tau\to\upsilon}$ to $\mathcal{D}_\tau \longrightarrow \mathcal{D}_\upsilon$. Assume two terms $s, t \in \mathcal{T}_{\text{GH}}$ such that for all $u \in \mathcal{T}_{\text{GH}}$, we have $\llbracket \mathcal{F}(s\, u) \rrbracket_R = \llbracket \mathcal{F}(t\, u) \rrbracket_R$. By Lemma 15, it follows that $\llbracket \mathcal{F}(s) \rrbracket_R = \llbracket \mathcal{F}(t) \rrbracket_R$, which concludes the proof that $\mathcal{E}^0_{\tau\to\upsilon}$ is injective.

We define $\mathcal{D}_{\tau\to\upsilon} = \mathcal{E}^0_{\tau\to\upsilon}(\mathcal{U}_{\tau\to\upsilon})$ and $\mathcal{E}_{\tau\to\upsilon}(a) = \mathcal{E}^0_{\tau\to\upsilon}(a)$. This ensures that $\mathcal{E}_{\tau\to\upsilon}$ is bijective and concludes the inductive definition of $\mathcal{D}$ and $\mathcal{E}$. In the following, we will usually write $\mathcal{E}$ instead of $\mathcal{E}_\tau$, since the type τ is determined by first argument of $\mathcal{E}_\tau$.

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

We define the interpretation function as $\mathcal{J}^{\text{GH}}(\mathsf{f}, \mathcal{D}_{\bar{\upsilon}_m}) = \mathcal{E}(\mathcal{J}(\mathsf{f}_0^{\bar{\upsilon}_m}))$ for all $\mathsf{f} : \Pi\bar{\alpha}_m.\, \tau$.

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 $\llbracket t \rrbracket_R^{\xi} = \mathcal{E}^{-1}(\xi(x))$. Choosing one such t and defining $x\theta = \mathcal{F}^{-1}(t)$ gives us a grounding substitution θ with the desired property.

We define $\mathcal{L}^{\mathrm{GH}}(\xi, (\lambda x.\, t)) = \mathcal{E}(\llbracket \mathcal{F}((\lambda x.\, t)\theta) \rrbracket_R)$. 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 $\mathcal{E}(\llbracket \mathcal{F}(x\theta') \rrbracket_R) = \xi(x)$ 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 $\llbracket \mathcal{F}(x\theta) \rrbracket_R = \llbracket \mathcal{F}(x\theta') \rrbracket_R$ for all x because $\mathcal{E}$ is injective. By Lemma 16 it follows that $\llbracket \mathcal{F}((\lambda x.\, t)\theta) \rrbracket_R = \llbracket \mathcal{F}((\lambda x.\, t)\theta') \rrbracket_R$, which proves that $\mathcal{L}^{\mathrm{GH}}$ is well defined.

This concludes the definition of the interpretation $\mathcal{I}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}_{\mathsf{ty}}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$. It remains to show that $\mathcal{I}^{\mathrm{GH}}$ is proper. In a proper interpretation, the denotation $\llbracket t \rrbracket_{\mathcal{I}^{\mathrm{GH}}}$ 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 17 *For all ground λ-terms t, we have*

$$\llbracket t \rrbracket_{\mathcal{I}^{\mathrm{GH}}} = \mathcal{E}(\llbracket \mathcal{F}(t{\downarrow_{\beta\eta}}) \rrbracket_R)$$

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

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

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

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

If t is a λ-expression, then

$$\begin{aligned}
\llbracket \lambda x.\, u \rrbracket_{\mathcal{I}^{\mathrm{GH}}}^{\xi} &= \mathcal{L}^{\mathrm{GH}}(\xi, (\lambda x.\, u)) \\
&= \mathcal{E}(\llbracket \mathcal{F}((\lambda x.\, u)\theta{\downarrow_{\beta\eta}}) \rrbracket_R) \\
&= \mathcal{E}(\llbracket \mathcal{F}(\lambda x.\, u{\downarrow_{\beta\eta}}) \rrbracket_R)
\end{aligned}$$

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

We need to show that the interpretation $\mathcal{I}^{\mathrm{GH}} = (\mathcal{U}^{\mathrm{GH}}, \mathcal{J}_{\mathsf{ty}}^{\mathrm{GH}}, \mathcal{J}^{\mathrm{GH}}, \mathcal{L}^{\mathrm{GH}})$ is proper. In the proof, we will need to employ the following lemma, which is very similar to the substitution lemma (Lemma 4), 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 18 (Substitution lemma)

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

for all λ-terms t, all $\tau \in \mathcal{T}y_{\mathrm{H}}$ and all grounding substitutions ρ, where $\xi'(\alpha) = [\![\alpha\rho]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}_{\mathrm{ty}}}$ for all type variables α and $\xi'(x) = [\![x\rho]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}}$ 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 3, 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:

$$\begin{aligned}
[\![(\lambda z.\, u)\rho]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}} &= [\![(\lambda z.\, u\rho')]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}} && \text{where } \rho'(z) = z \text{ and } \rho'(x) = \rho(x) \text{ for } x \neq z \\
&= \mathcal{L}^{\mathrm{GH}}(\xi, (\lambda z.\, u\rho')) && \text{by the definition of the term denotation} \\
&= \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\rho\theta_{\xi}{\downarrow_{\beta\eta}})]\!]^{\xi}_{R}) && \text{by the definition of } \mathcal{L}^{\mathrm{GH}} \\
&= \mathcal{E}([\![\mathcal{F}((\lambda z.\, u)\rho{\downarrow_{\beta\eta}})]\!]^{\xi}_{R}) && \text{because } (\lambda z.\, u)\rho \text{ is ground} \\
&\overset{*}{=} \mathcal{L}^{\mathrm{GH}}(\xi', \lambda z.\, u) && \text{by the definition of } \mathcal{L}^{\mathrm{GH}} \text{ and Lemma 17} \\
&= [\![\lambda z.\, u]\!]^{\xi'}_{\mathcal{J}^{\mathrm{GH}}} && \text{by the definition of the term denotation}
\end{aligned}$$

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 17, 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 19 *The interpretation $\mathcal{J}^{\mathrm{GH}}$ is proper.*

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

$$\begin{aligned}
[\![\lambda x.\, t]\!]^{\xi}_{\mathcal{J}^{\mathrm{GH}}}(a) &= \mathcal{L}^{\mathrm{GH}}(\xi, \lambda x.\, t)(a) && \text{by the definition of } [\![\ ]\!]_{\mathcal{J}^{\mathrm{GH}}} \\
&= \mathcal{E}([\![\mathcal{F}((\lambda x.\, t)\theta{\downarrow_{\beta\eta}})]\!]_{R})(a) && \text{by the definition of } \mathcal{L}^{\mathrm{GH}} \text{ for some } \theta \\
& && \text{such that } \mathcal{E}([\![\mathcal{F}(z\theta)]\!]_{R}) = \xi(z) \text{ for all } z \\
& && \text{and } \mathcal{D}_{\alpha\theta} = \xi(\alpha) \text{ for all } \alpha \\
&= \mathcal{E}([\![\mathcal{F}((\lambda x.\, t)\theta\, s{\downarrow_{\beta\eta}})]\!]_{R}) && \text{by the definition of } \mathcal{E} \\
& && \text{where } \mathcal{E}([\![\mathcal{F}(s)]\!]_{R}) = a \\
&= \mathcal{E}([\![\mathcal{F}(t(\theta[x \mapsto s]){\downarrow_{\beta\eta}})]\!]_{R}) && \text{by } \beta\text{-reduction} \\
&= [\![t(\theta[x \mapsto s])]\!]_{\mathcal{J}^{\mathrm{GH}}} && \text{by Lemma 17} \\
&= [\![t]\!]^{\xi[x\mapsto a]}_{\mathcal{J}^{\mathrm{GH}}} && \text{by Lemma 18}
\end{aligned}$$

□

Lemma 20 *$\mathcal{J}^{\mathrm{GH}}$ is a model of N.*

Proof By Lemma 17, we have $[\![t]\!]_{\mathcal{J}^{\mathrm{GH}}} = \mathcal{E}([\![\mathcal{F}(t)]\!]_{R})$ for all $t \in \mathcal{T}_{\mathrm{GH}}$. Since $\mathcal{E}$ is a bijection, it follows that any (dis)equation $[\neg]\, s \approx t \in \mathcal{C}_{\mathrm{GH}}$ is true in $\mathcal{J}^{\mathrm{GH}}$ if and only if $\mathcal{F}([\neg]\, s \approx t)$ is true in R. Hence, a clause $C \in \mathcal{C}_{\mathrm{GH}}$ is true in $\mathcal{J}^{\mathrm{GH}}$ if and only if $\mathcal{F}(C)$ is true in R. By Theorem 14, 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 $\mathcal{J}^{\mathrm{GH}}$ and therefore $\mathcal{J}^{\mathrm{GH}}$ is a model of N. □

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

Theorem 21 (Ground static refutational completeness) *Let $N \subseteq \mathcal{C}_{\mathrm{GH}}$ be a clause set saturated w.r.t. $GHInf^{gsel}$ and $GHRed^{gsel}_{\mathrm{I}}$ for some selection function $gsel$. Then $N \models \bot$ if and only if $\bot \in N$.*

4.4 The Nonground Higher-Order Layer

To lift the result to the nonground layer, we employ the saturation framework of Waldmann et al. [68]. 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 22 *Given a clausal higher-order interpretation* $\mathfrak{I}$ *on* GH, *there exists a first-order interpretation* $\mathfrak{I}^{\mathrm{GF}}$ *on* GF *such that for any clause* $C \in \mathcal{C}_{\mathrm{GH}}$ *the truth values of* C *in* $\mathfrak{I}$ *and of* $\mathcal{F}(C)$ *in* $\mathfrak{I}^{\mathrm{GF}}$ *coincide.*

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

We need to show that for any $C \in \mathcal{C}_{\mathrm{GH}}$, $\mathfrak{I} \models C$ if and only if $\mathfrak{I}^{\mathrm{GF}} \models \mathcal{F}(C)$. It suffices to show that $[\![t]\!]_{\mathfrak{I}} = [\![\mathcal{F}(t)]\!]_{\mathfrak{I}^{\mathrm{GF}}}$ 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 $\mathcal{F}(t) = \mathsf{f}^{\bar{\upsilon}}_j(\mathcal{F}(\bar{s}_j))$ and hence $[\![\mathcal{F}(t)]\!]_{\mathfrak{I}^{\mathrm{GF}}} = \mathcal{J}^{\mathrm{GF}}(\mathsf{f}^{\bar{\upsilon}}_j)([\![\mathcal{F}(\bar{s}_j)]\!]_{\mathfrak{I}^{\mathrm{GF}}}) = [\![\mathsf{f}\langle\bar{\upsilon}\rangle]\!]_{\mathfrak{I}}([\![\mathcal{F}(\bar{s}_j)]\!]_{\mathfrak{I}^{\mathrm{GF}}}) \overset{\text{IH}}{=} [\![\mathsf{f}\langle\bar{\upsilon}\rangle]\!]_{\mathfrak{I}}([\![\bar{s}_j]\!]_{\mathfrak{I}}) = [\![t]\!]_{\mathfrak{I}}$. □

Lemma 23 *The redundancy criterion for* GH *is a redundancy criterion in the sense of the framework.*

Proof Analogous to the proof of Lemma 4.12 of Bentkamp et al. [12]. □

Lemma 24 *Given a selection function* sel *on* $\mathcal{C}_{\mathrm{H}}$, *the grounding functions* $\mathcal{G}^{gsel}$ *for* $gsel \in \mathcal{G}(sel)$ *are grounding functions in the sense of the framework.*

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

Let sel be a selection function on $\mathcal{C}_{\mathrm{H}}$ fulfilling the selection restriction that a literal $L\langle y\rangle$ may not be selected if $y\,\bar{u}_n$, with $n > 0$, is a $\succsim$-maximal term of the clause. Let $N \subseteq \mathcal{C}_{\mathrm{H}}$ be a clause set saturated w.r.t. HInf^{sel} and $\mathit{HRed}^{sel}_{\mathrm{I}}$. For the lifting mechanism of the saturation framework to apply, we need to show that there exists a selection function $gsel \in \mathcal{G}(sel)$ such that all inferences $\iota \in \mathit{GHInf}^{gsel}$ with $\mathit{prems}(\iota) \in \mathcal{G}(N)$ are liftable or redundant. Here, ι being *liftable* means that ι is a $\mathcal{G}^{gsel}$-ground instance of a HInf^{sel}-inference from N; ι being *redundant* means that $\iota \in \mathit{GHRed}^{gsel}_{\mathrm{I}}(\mathcal{G}(N))$.

To choose the right selection function $gsel \in \mathcal{G}(sel)$, 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 $\mathcal{G}^{-1}(C)$. Then let $gsel$ select those literals in C that correspond to the literals selected by sel in $\mathcal{G}^{-1}(C)$. With respect to this selection function $gsel$, we can show that all inferences from $\mathcal{G}(N)$ are liftable or redundant:

Lemma 25 (Lifting of EQRES, EQFACT, GARGCONG, and GEXT) *All* EQRES, EQFACT, GARGCONG, *and* GEXT *inferences are liftable.*

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

$$\frac{C\theta \;=\; C'\theta \vee s\theta \not\approx s'\theta}{C'\theta}\,\textsc{EqRes}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \not\approx s'$ and the literal $s\theta \not\approx s'\theta$ is eligible w.r.t. *gsel*. It follows that $s \not\approx s'$ is eligible in C w.r.t. *sel*. Moreover, $s\theta$ and $s'\theta$ are unifiable and ground, and therefore $s\theta = s'\theta$. Thus, there exists some corresponding $\sigma \in \mathrm{CSU}(s, s')$ and there is this inference $\iota' \in HInf^{sel}$:

$$\frac{C' \vee s \not\approx s'}{C'\sigma}\,\textsc{EqRes}$$

Since $\sigma \in \mathrm{CSU}(s, s')$, we have $x\sigma\sigma = x\sigma$ and $x\sigma\rho = x\theta$ for all variables x in C for some substitution ρ. Therefore, ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.

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

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

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s' \approx t' \vee s \approx t$, the literal $s\theta \approx t\theta$ is eligible in C w.r.t. *gsel*, and $s\theta \not\prec t\theta$. Hence, $s \approx t$ is eligible in C w.r.t. *sel* and $s \not\prec t$. Moreover, $s\theta$ and $s'\theta$ are unifiable and ground. Hence, $s\theta = s'\theta$. Thus, there exists some corresponding $\sigma \in \mathrm{CSU}(s, s')$ and there is this inference $\iota' \in HInf^{sel}$:

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

Since $\sigma \in \mathrm{CSU}(s, s')$, we have $x\sigma\sigma = x\sigma$ and $x\sigma\rho = x\theta$ for all variables x in C for some substitution ρ. Hence, ι is the $\sigma\rho$-ground instance of ι' and is therefore liftable.

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

$$\frac{C\theta \;=\; C'\theta \vee s\theta \approx s'\theta}{C'\theta \vee s\theta\,\bar{u} \approx s'\theta\,\bar{u}}\,\textsc{GArgCong}$$

where $\mathcal{G}^{-1}(C\theta) = C = C' \vee s \approx s'$, the literal $s\theta \approx s'\theta$ is strictly eligible w.r.t. *gsel*, and $s\theta$ and $s'\theta$ are of functional type. It follows that $s \not\approx s'$ is eligible in C w.r.t. *sel*, and that s and s' have either a functional or a polymorphic type. Hence there is this inference $\iota' \in HInf^{sel}$:

$$\frac{C' \vee s \approx s'}{(C' \vee s\,\bar{x} \approx s'\,\bar{x})\sigma}\,\textsc{ArgCong}$$

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 the 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 26 (Instances of green subterms) *Let s be a λ-term in η-short β-normal form, let σ be a substitution, 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 $1 \le i \le n$ 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 induction, $(s_i|_{p'})\sigma{\downarrow_{\beta\eta}} = (s_i\sigma{\downarrow_{\beta\eta}})|_{p'}$. 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 27 (Lifting of SUP) *Let $\iota \in \mathit{GHInf}^{\mathit{gsel}}$ be a SUP inference*

$$\frac{\overbrace{D'\theta \vee t\theta \approx t'\theta}^{D\theta} \quad \overbrace{C'\theta \vee [\neg]\, s\theta\langle t\theta\rangle_p \approx s'\theta}^{C\theta}}{D'\theta \vee C'\theta \vee [\neg]\, s\theta\langle t'\theta\rangle_p \approx s'\theta}\,\textsc{Sup}$$

where $\mathcal{G}^{-1}(D\theta) = D = D' \vee t \approx t'$ and $\mathcal{G}^{-1}(C\theta) = C = C' \vee [\neg]\, s \approx s'$. 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 deep variable in C;* (ii) *$p = p'$ and the variable condition holds for D and C or* (iii) *$p \ne p'$ 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 eligible, $[\neg]\, s\theta \approx s'\theta$ is strictly eligible if positive and eligible if negative, $D\theta \not\succsim C\theta$, $t\theta \not\precsim t'\theta$, and $s\theta \not\precsim s'\theta$. We assume that s, t, $s\theta$, and $t\theta$ are represented by λ-terms in η-short β-normal form. By Lemma 26, $u\theta$ agrees with $s\theta|_{p'}$ (considering both as terms rather than as λ-terms).

CASE 1: We have (a) $p = p'$, and (b) u is not fluid, and (c) if u is a variable, it is not deep in C. Then $u\theta = s\theta|_{p'} = s\theta|_p = t\theta$. Since θ is a unifier of u and t, there exists some corresponding $\sigma \in \mathrm{CSU}(t,u)$. The inference conditions can be lifted: (Strict) eligibility of $t\theta \approx t'\theta$ and $[\neg]\, s\theta \approx s'\theta$ w.r.t. *gsel* implies (strict) eligibility of $t \approx t'$ and $[\neg]\, s \approx s'$ w.r.t. *sel*; $D\theta \not\succsim C\theta$ implies $D \not\succsim C$; $t\theta \not\precsim t'\theta$ implies $t \not\precsim t'$; and $s\theta \not\precsim s'\theta$ implies $s \not\precsim s'$. 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 $\iota' \in \mathit{HInf}$:

$$\frac{D' \vee t \approx t' \quad C' \vee [\neg]\, s\langle u\rangle_p \approx s'}{(D' \vee C' \vee [\neg]\, s\langle t'\rangle_p \approx s')\sigma}\,\textsc{Sup}$$

Since $\sigma \in \mathrm{CSU}(t,u)$, we have $x\sigma\sigma = x\sigma$ and $x\sigma\rho = x\theta$ for all variables x occurring in D and C for some substitution ρ. Hence, ι is the $\sigma\rho$-ground instance of ι' and therefore liftable.

CASE 2: We have (a) $p \ne p'$; or (b) u is fluid; or (c) u is a deep variable 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 $n \ge 1$. If u were of the form $\mathsf{f}\langle\bar{\tau}\rangle\, s_1 \ldots s_n$ with $n \ge 0$, $u\theta$ would have the form $\mathsf{f}\langle\bar{\tau}\theta\rangle\,(s_1\theta)\ldots(s_n\theta)$, but then there is some $1 \le i \le n$ 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 some corresponding $\sigma \in \mathrm{CSU}(z\,t, u)$. 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':

$$\frac{D' \vee t \approx t' \qquad C' \vee [\neg]\, s\langle u\rangle_{p'} \approx s'}{(D' \vee C' \vee [\neg]\, s\langle z\,t'\rangle_{p'} \approx s')\sigma}\,\textsc{FluidSup}$$

Since $\sigma \in \mathrm{CSU}(z\,t, u)$, we have $x\sigma\sigma = x\sigma$ and $x\sigma\rho = x\theta'$ for $x = z$ and for all variables x in C and D for some substitution ρ. Hence, 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 28 *Let $\iota \in \mathit{GHInf}^{gsel}$ be a* SUP *inference that Lemma 27 does not apply to. Then $\iota \in \mathit{GHRed}_{\mathrm{I}}^{gsel}(\mathcal{G}(N))$.*

Proof Let $C\theta = C'\theta \vee [\neg]\, s\theta \approx s'\theta$ and $D\theta = D'\theta \vee t\theta \approx t'\theta$ be the premises of ι, where $[\neg]\, s\theta \approx s'\theta$ 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 looks like this:

$$\frac{D'\theta \vee t\theta \approx t'\theta \qquad C'\theta \vee [\neg]\, s\theta\langle t\theta\rangle \approx s'\theta}{D'\theta \vee C'\theta \vee [\neg]\, s\theta\langle t'\theta\rangle \approx s'\theta}\,\textsc{Sup}$$

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

Let p be the position in $s\theta$ where the ι takes place and p' be the longest prefix of p that is a green subterm of s. Let $u = s|_{p'}$. Since Lemma 27 does not apply to ι, u is not a deep variable; if $p = p'$, the variable condition does not hold for D and C; and if $p \neq p'$, u is a variable. That means either the position p does not exist in s because it is below an unapplied variable that is shallow in C, or $s|_p$ is an unapplied variable that is shallow in C and for which the variable condition does not hold.

CASE 1: The position p does not exist in s because it is below a variable x that is shallow in C. Then $t\theta$ is an argument subterm of $x\theta$ and hence an argument 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 $\mathfrak{I} \models \mathcal{F}(t\theta \approx t'\theta)$, by congruence, $\mathfrak{I} \models \mathcal{F}(x\theta\,\bar{w} \approx v\,\bar{w})$ for any arguments $\bar{w}$. Hence, $\mathfrak{I} \models \mathcal{F}(C\theta)$ if and only if $\mathfrak{I} \models \mathcal{F}(C\{x \mapsto v\}\theta)$ by congruence. Here, it is crucial that the variable is shallow 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 $\mathfrak{I}$, we have $\mathfrak{I} \models \mathcal{F}(C\{x \mapsto v\}\theta)$ and hence $\mathfrak{I} \models \mathcal{F}(C\theta)$.

CASE 2: The term $s|_p$ is a variable x that is shallow 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 $HRed_{\mathrm{I}}^{sel}(N)$—i.e., $\mathcal{G}(concl(\iota')) \subseteq \mathcal{G}(N) \cup GHRed_{\mathrm{C}}(\mathcal{G}(N))$. Hence, $D'\theta \vee t\theta\,\bar{u} \approx t'\theta\,\bar{u}$ is in $\mathcal{G}(N) \cup GHRed_{\mathrm{C}}(\mathcal{G}(N))$ for any ground arguments $\bar{u}$.

We observe that whenever $t\theta\,\bar{u}$ and $t'\theta\,\bar{u}$ are smaller than the 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 maximal term of $C\theta$ and we distinguish two cases: If $t\theta$ is smaller than the maximal term of $C\theta$, all terms in $D'\theta$ are smaller than the 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 maximal term of $C\theta$, $t\theta\,\bar{u}$ 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 $\mathcal{I} \models \mathcal{F}(t\theta\,\bar{u}) \approx \mathcal{F}(t'\theta\,\bar{u})$.

We proceed to 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 maximal term of $C\theta$. Since x is shallow 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 places 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 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 $\mathcal{F}(D\theta)$ being true and $\mathcal{F}(D'\theta)$ being false. If $\bar{v}$ is nonempty, it suffices to show that $x\,\bar{v}$ is not a 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 maximal in $\mathcal{F}(C\theta)$ and $\mathcal{F}(C''\theta)$ either. Hence they must be smaller than the 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 maximal term in C, we make a case distinction on whether $[\neg]s\theta \approx s'\theta$ is selected in $C\theta$ or $s\theta$ is the maximal term in $C\theta$. One of these must hold because $[\neg]s\theta \approx s'\theta$ is eligible in $C\theta$. If it is selected, by the selection restrictions, x cannot be the head of a maximal term of C. If $s\theta$ is the maximal term in $C\theta$, we can argue that x is a green subterm of s and, since x is shallow, 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, the saturation framework guarantees static and dynamic refutational completeness of $HInf^{sel}$ w.r.t. $HRed_{\mathrm{I}}^{sel}$. However, the framework gives us refutational completeness w.r.t. the Herbrand entailment $\models_{\mathcal{G}}$, defined as $N_1 \models_{\mathcal{G}} N_2$ if $\mathcal{G}(N_1) \models \mathcal{G}(N_2)$, whereas our semantics is Tarski entailment $\models$—i.e., $N_1 \models N_2$ if a model of N_1 implies that N_2 has a model. To repair this mismatch, we prove the following lemma, which can be shown as Lemma 4.18 of Bentkamp et al. [12].

Lemma 29 *For $N \subseteq \mathcal{C}_{\mathrm{H}}$, we have $N \models_{\mathcal{G}} \bot$ if and only if $N \models \bot$.*

Theorem 30 (Static refutational completeness) *Let $N \subseteq \mathcal{C}_{\mathrm{H}}$ be a clause set saturated w.r.t. $HInf^{sel}$ and $HRed_{\mathrm{I}}^{sel}$. Then $N \models \bot$ if and only if $\bot \in N$.*

Proof We apply Theorem 45 of Waldmann et al. We take H for **F**, GH for **G**, and $\mathcal{F}(sel)$ 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 23 and 24, $(\mathit{GHRed}_{\mathrm{I}}^{gsel}, \mathit{GHRed}_{\mathrm{C}})$ is a redundancy criterion in the sense of the framework, and $\mathcal{G}^{gsel}$ are grounding functions in the sense of the framework, for all $gsel \in \mathcal{F}(sel)$. The redundancy criterion $(\mathit{HRed}_{\mathrm{I}}^{sel}, \mathit{HRed}_{\mathrm{C}})$ matches exactly the intersected lifted redundancy criterion $\mathit{Red}^{\cap,\sqsupset}$ of Waldmann et al. By Theorem 21, GHInf^{gsel} is statically refutationally complete for all $gsel \in \mathcal{F}(sel)$. By Lemmas 25, 27, and 28, for every saturated $N \subseteq \mathcal{C}_{\mathrm{H}}$, there exists a selection function $gsel \in \mathcal{G}(sel)$ such that all inferences $\iota \in \mathit{GHInf}^{gsel}$ with $\mathit{prems}(\iota) \in \mathcal{G}(N)$ either are $\mathcal{G}^{gsel}$-ground instances of HInf^{sel}-inferences from N or belong to $\mathit{GHRed}_{\mathrm{I}}^{gsel}(\mathcal{G}(N))$.

If $\sqsupset = \varnothing$, Theorem 45 of Waldmann et al. implies that if $N \subseteq \mathcal{C}_{\mathrm{H}}$ is a clause set saturated w.r.t. HInf^{sel} and $\mathit{HRed}_{\mathrm{I}}^{sel}$, then $N \models_{\mathcal{G}} \bot$ if and only if $\bot \in N$. By Lemma 47 of Waldmann et al., this also holds if $\sqsupset \neq \varnothing$. By Lemma 29, 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^{sel} and $\mathit{HRed}_{\mathrm{I}}^{sel}$, then $N \models \bot$ if and only if $\bot \in N$. □

From static refutational completeness, we can easily derive dynamic refutational completeness. Let $(N_i)_i$ be a (finite or infinite) sequence over sets of clauses from $\mathcal{C}_{\mathrm{H}}$. Such a sequence is called a *derivation* if $N_i \setminus N_{i+1} \subseteq \mathit{HRed}_{\mathrm{C}}(N_{i+1})$ for all i. It is called *fair* if all HInf-inferences from clauses in $\bigcup_i \bigcap_{j \geq i} N_j$ are contained in $\bigcup_i \mathit{HRed}_{\mathrm{I}}(N_i)$.

Theorem 31 (Dynamic refutational completeness) *For every fair derivation $(N_i)_i$ such that $N_0 \models \{\bot\}$, we have $\bot \in N_i$ for some i.*

Proof By Theorem 48 of Waldmann et al., this follows from Theorem 30 and Lemma 29. □

5 Extensions

The calculus can be extended to make it more practical. The familiar simplification machinery can be adapted to higher-order terms by considering green contexts instead of arbitrary contexts. Optional inference rules provide lightweight alternatives to the FLUIDSUP rule and the extensionality axiom.

Two of the optional rules below rely on the notion of "orange subterms." 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. 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 the convention introduced in Sect. 2, 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, which destructively rewrites using an equality $t \approx t'$, is available at green positions. A variant rewrites inside λ-expressions:

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

where $s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle\!\downarrow_{\beta\eta}$ is a λ-expression or an applied variable. The term $t\sigma$ may refer to the bound variables $\bar{x}$. There is one side condition: the second premise is larger ($\succ$) than the second and third conclusion. This ensures that this premise is redundant w.r.t. these 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, with $y := (\lambda x.\, s\langle\!\langle \bar{x}.\, t\sigma \rangle\!\rangle)$ and $z := (\lambda x.\, s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta)$, is smaller than the second premise. But in general, it would appear that the third conclusion is necessary and that the variant of λDEMODEXT that omits it—let us call it λDEMOD—does not preserve refutational completeness.

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

$$\frac{\mathsf{g}\, x \approx \mathsf{f}\, x\, x \qquad \mathsf{k}\, (\lambda z.\, \mathsf{h}\, (\mathsf{g}\, z)) \approx \mathsf{c}}{\mathsf{g}\, x \approx \mathsf{f}\, x\, x \qquad \mathsf{k}\, (\lambda z.\, \mathsf{h}\, (\mathsf{f}\, z\, z)) \approx \mathsf{c} \qquad (\lambda z.\, \mathsf{h}\, (\mathsf{g}\, z)) \approx (\lambda z.\, \mathsf{h}\, (\mathsf{f}\, z\, z))}\ \lambda\text{DEMODEXT}$$

Lemma 32 *λDEMODEXT is sound and preserves refutational completeness of the calculus.*

Proof Soundness of the first conclusion is obvious. Soundness of the second and third conclusion 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. By definition, it is redundant 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 11, $\mathcal{F}(C\langle s\langle\!\langle \bar{x}.\, t'\sigma \rangle\!\rangle_\eta\rangle\theta)$ 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 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. $\square$

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

$$\frac{C}{C\sigma}\ \text{PRUNEARG}$$

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

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 33 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.

For example, 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 the clause C. 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: $t = (\lambda x_1\, x_3.\, \mathsf{b})$. An example where our procedure fails is the pair of occurrences $y\, (\lambda x.\, \mathsf{a})\, (\mathsf{f}\, \mathsf{a})\, \mathsf{c}$ and $y\, (\lambda x.\, \mathsf{b})\, (\mathsf{f}\, \mathsf{b})\, \mathsf{d}$. 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 will not detect this.

Following the literature [36, 62], we provide a rule for negative extensionality:

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

where sk is a fresh Skolem symbol, $\bar{\alpha}$ and $\bar{y}$ are the type and term variables occurring free in the the literal $s \not\approx s'$, and $s \not\approx s'$ is eligible in the premise. Negative extensionality can also be applied as a simplification rule to all literals in 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 34 (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 $s\, (\mathsf{diff}\, s\, s') \not\approx s'\, (\mathsf{diff}\, s\, s') \vee s \approx s'$, where $\tau \to \upsilon$ is the type of s and s'. We extend the model $\mathcal{I}$ to a model $\mathcal{I}'$, interpreting sk such that $\mathcal{I}' \models \mathsf{sk}\langle\bar{\alpha}\rangle\bar{y} \approx \mathsf{diff}\, s\, s'$ for each i. 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. $\square$

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 terms. As a pragmatic alternative to the axiom, we introduce the rules EXTSUP, EXTEQRES, and EXTEQFACT with the same premises as the core SUP, EQRES and EQFACT, respectively. We collectively call these rules EXTINF. Each new rule shares all the side conditions of the core rules except that of the form $\sigma \in \mathrm{CSU}(s,t)$. Instead, it sets σ to be the most general unifier of the types of s and t and adds the following condition: Let $s = v\langle\!\langle s_1,\ldots,s_n\rangle\!\rangle$ and $t = v\langle\!\langle t_1,\ldots,t_n\rangle\!\rangle$, where $v\sigma$ 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 $E \vee s_1 \not\approx t_1 \vee s_n \not\approx t_n$. The rule NEGEXT can then be applied to literals $s_i \not\approx t_i$ whose sides have functional type.

A different approach is to instantiate the extensionality axiom with arbitrary terms s, s' of the same functional type, which presumably appear as green subterms in the current clause set:

$$\frac{}{s\,(\mathsf{diff}\,s\,s') \not\approx s'\,(\mathsf{diff}\,s\,s') \vee s \approx s'}\;\textsc{ExtInst}$$

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, EXTINF 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:

$$\frac{D' \vee t \approx t' \qquad C' \vee [\neg]\, s\langle\!\langle \bar{x}.\, u\rangle\!\rangle \approx s'}{(D' \vee C' \vee [\neg]\, s\langle\!\langle \bar{x}.\, t'\rangle\!\rangle_\eta \approx s')\sigma\rho}\;\lambda\textsc{Sup}$$

SUP's side conditions apply. We also require that $\bar{x}$ has length $n > 0$, $\bar{x}\sigma = \bar{x}$, and the variables $\bar{x}$ do not occur in $y\sigma$ for all variables y in u. Moreover, let $P_y = \{y\}$ for all type and term variables $y \notin \bar{x}$. For each i, let P_{x_i} be recursively defined 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 $s\langle\!\langle \bar{x}.\, t'\rangle\!\rangle_\eta\sigma$. The substitution 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. This substitution introduces Skolem terms to represent bound variables that would otherwise escape their binders. The rule can be justified in terms of paramodulation and extensionality, with the Skolem terms standing for diff terms. An instance of the rule follows:

$$\frac{n \approx \mathsf{zero} \vee \mathsf{div}\;n\;n \approx \mathsf{one} \qquad \mathsf{prod}\;K\;(\lambda k.\,\mathsf{div}\;(\mathsf{succ}\;k)\;(\mathsf{succ}\;k)) \not\approx \mathsf{one}}{\mathsf{succ}\;\mathsf{sk} \approx \mathsf{zero} \vee \mathsf{prod}\;K\;(\lambda k.\,\mathsf{one}) \not\approx \mathsf{one}}\;\lambda\textsc{Sup}$$

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.

Lemma 35 **(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 $\mathcal{J}$ be a model of $N \cup \{(\text{EXT})\}$. We need to construct a model of $N \cup \{(\text{EXT}), E\}$. 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 $\mathcal{J}$ to a model $\mathcal{J}'$, interpreting sk_i such that $\mathcal{J}' \models \mathsf{sk}_i\langle\bar{\alpha}_i\rangle\,\bar{y}_i \approx \mathsf{diff}\,(v_i\pi)\,(v_i'\pi)$ 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 $\mathcal{J}$ in this way.

By assumption, the premises of the λSUP inference are true in $\mathcal{J}$ and hence in $\mathcal{J}'$. We need to show that the conclusion $(D' \vee C' \vee [\neg]\, s\langle\!\langle \bar{x}.\, t'\rangle\!\rangle_\eta \approx s')\sigma\rho$ is also true in $\mathcal{J}'$. Let ξ be a valuation. If $\mathcal{J}', \xi \models (D' \vee C')\sigma\rho$, we are done. So we assume that $D'\sigma\rho$ and $C'\sigma\rho$ are false in $\mathcal{J}'$ under ξ. In the following, we omit '$\mathcal{J}', \xi \models$', but all equations ($\approx$) are meant to be true in $\mathcal{J}'$ 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 $[\neg]\, s\langle\!\langle \bar{x}.\, u\rangle\!\rangle\sigma\rho \approx s'\sigma\rho$. Due to the way we constructed $\mathcal{J}'$, 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 $\mathcal{J}' \models w\pi \approx w\rho$ 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 $[\neg]\, s\langle\!\langle \bar{x}.\, u\rangle\!\rangle\sigma\rho \approx s'\sigma\rho$, it follows that $([\neg]\, s\langle\!\langle \bar{x}.\, t'\rangle\!\rangle_\eta \approx s')\sigma\rho$. Hence, the conclusion of the λSUP inference is true in $\mathcal{J}'$. □

The next rule, *duplicating flex subterm superposition*, is a lightweight alternative to FLUIDSUP:

$$\frac{D' \vee t \approx t' \quad C' \vee [\neg]\, s\langle y\,\bar{u}_n\rangle \approx s'}{(D' \vee C' \vee [\neg]\, s\langle z\,\bar{u}_n\, t'\rangle \approx s')\rho\sigma}\;\text{DUPSUP}$$

where $n \geq 1$, $\rho = \{y \mapsto \lambda\bar{x}_n.\, z\,\bar{x}_n\,(w\,\bar{x}_n)\}$, and $\sigma \in \text{CSU}(t, w\,(\bar{u}_n\rho))$ 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 7 and 8 but not in Example 9. 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 an applied variable. We conjecture that DUPSUP, in conjunction with an extended SUP rule that considers the green subterms of $\bar{u}_n$, constitutes a complete alternative to FLUIDSUP for fluid subterms of the form $y\,\bar{u}_n$ if the types of t, all u_j, and $y\,\bar{u}_n$ are not functional or type variables.

The last rule, *flex subterm superposition*, is an even more lightweight alternative to FLUIDSUP:

$$\frac{D' \vee t \approx t' \quad C' \vee [\neg]\, s\langle y\,\bar{u}_n\rangle \approx s'}{(D' \vee C' \vee [\neg]\, s\langle t'\rangle \approx s')\sigma}\;\text{FLEXSUP}$$

where $n \geq 1$ and $\sigma \in \text{CSU}(t, y\,\bar{u}_n)$. The order and eligibility restrictions are as for SUP.

6 Implementation

Zipperposition [29,30] is an open source superposition prover written in OCaml.[1] Originally designed for polymorphic first-order logic (TF1 [22]), it was later extended with an incomplete higher-order mode based on pattern unification [53]. Bentkamp et al. [13] 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 $\succ_{\mathsf{meta}}$ described in Section 3.1 induced by a λ-free KBO [9] with the precedence $\mathsf{lam} \succ \cdots \succ \mathsf{db}_k \succ \cdots \succ \mathsf{db}_1 \succ \mathsf{db}_0$. We currently use $\succeq$ as the nonstrict term order but could improve precision by employing a more precise computable approximation of $\succsim$.

Except for FLUIDSUP, the core calculus rules already existed in Zipperposition in a similar form. To improve efficiency, we added appropriate indexes for all new inference rules that have two premises. Among the optional rules, we implemented λDEMOD, PRUNEARG, NEGEXT, EXTINF, 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. [66].

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 the passive set. 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. Once it is selected, C is simplified using A. Clauses in A are simplified w.r.t. C and moved to P if simplified. Then C is added to A and all nonredundant inferences between C and A are scheduled into 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. Finally the loop restarts.

We can view the above loop as an instance of the abstract Zipperposition loop prover ZL of Waldmann et al. [68, Example 69]. Theorem 67 of Waldmann et al. allows us to obtain dynamic completeness for this prover architecture from our static completeness result (Theorem 30), given a strategy that visits the sequences in T fairly and that chooses clauses in P fairly.

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

We evaluated our prototype implementation of the calculus in Zipperposition with other higher-order provers and with Zipperposition's modes for less expressive logics. All of the experiments were performed on StarExec nodes equipped with Intel Xeon E5-2609 0 CPUs clocked at 2.40 GHz. Following CASC-27,[2] we use 180 s as the CPU time limit.

[1] `https://github.com/sneeuwballen/zipperposition`
[2] `http://tptp.cs.miami.edu/CASC/`

	−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	+EXTINF	+EXTINST	+(EXT)
TH0	447 (0)	450 (1)	450 (1)	376 (0)
SH-λ	436 (11)	430 (11)	402 (1)	364 (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

We used both standard TPTP benchmarks [63] and Sledgehammer-generated benchmarks [52]. From the TPTP, we randomly selected 1000 first-order (FO) problems in CNF, FOF, or TFF syntax without arithmetic and all 499 monomorphic higher-order theorems in TH0 syntax without first-class Booleans and arithmetic. We partitioned the TH0 problems into those containing no λ-expressions (TH0λf, 452 problems) and those containing λ-expressions (TH0λ, 47 problems). The Sledgehammer benchmarks, corresponding to Isabelle's Judgment Day suite [25], were regenerated to target clausal higher-order logic. They comprise 2506 problems, divided in two groups: SH-λ preserves λ-expressions, whereas SH-ll encodes them as λ-lifted supercombinators [52] to make the problems accessible to λ-free clausal higher-order provers. Each group of problems is generated from 256 Isabelle facts (definitions and lemmas). Our results are publicly available.[3]

Evaluation of Extensions To assess the usefulness of the extensions described in Sect. 5, we fixed a *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. [66]. It also includes the optional rules NEGEXT and PRUNEARG, substitutes FLEXSUP for the highly explosive FLUIDSUP, and excludes the (EXT) axiom. The base 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. Figure 1 confirms this, although the effect is small. In the figures, $+R$ 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 given configuration and no other configuration in the same figure.

[3] `http://matryoshka.gforge.inria.fr/pubs/lamsup_article_results.zip`

The rules λDEMOD (λD) and λSUP extend the calculus to perform some rewriting under λ-binders. While experimenting with the calculus, we noticed that, in some configurations, λSUP performs better when the number of fresh Skolem symbols it introduces overall is bounded by some parameter n. As Fig. 2 shows, inclusion of these rules has different effect on the two benchmark sets. Different choices of n for λSUP (denoted by λSn) do not seem to influence the success rate much.

The evaluation of the EXTINF and EXTINST rules and the (EXT) axiom, presented in Fig. 3, confirms our intuition that including the extensionality axiom is severely detrimental to performance. The raw data reveal that there are no cases in which the axiom helped solve a problem that could not be solved by using one of the two rules instead, even though the +(EXT) configuration solved 2 unique problems on SH-λ benchmarks.

The FLEXSUP rule included in the base configuration did not perform as well as we expected. Even the FLUIDSUP and DUPSUP rules outperformed FLEXSUP, as shown in Fig. 4. This effect is especially visible on SH-λ benchmarks. On TPTP, the differences are negligible.

Main Evaluation We selected all contenders in the THF division of CASC-27 as representatives of the state of the art: CVC4 1.8 prerelease [8], Leo-III 1.4 [62], Satallax 3.4 [26], and Vampire 4.4 [19]. We also included Ehoh [67], 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 evaluated them with first-order backends disabled. We denote these "uncooperative" configurations by Leo-III-uncoop and Satallax-uncoop respectively, as opposed to the standard versions Leo-III-coop and Satallax-coop.

To evaluate the overhead our calculus incurs on first-order or λ-free higher-order problems, we ran Zipperposition in first-order (FOZip) and λ-free (λfreeZip) modes, as well as in a mode that performs the applicative encoding before using first-order Zipperposition (@+FOZip). We evaluated the implementation of our calculus in Zipperposition (λZip) in three configurations: base, pragmatic, and full. Pragmatic builds on base by disabling FLEXSUP and replacing complete unification with the pragmatic variant procedure pv^2_{1121} of

	FO	TH0λf	TH0λ	SH-ll	SH-λ
CVC4	540	424	31	696	650
Ehoh	681	418	–	691	–
Leo-III-uncoop	198	389	42	226	234
Leo-III-coop	580	**438**	43	672	668
Satallax-uncoop	–	398	43	489	507
Satallax-coop	–	432	43	602	616
Vampire	**731**	432	42	**719**	708
FOZip	399	–	–	–	–
@+FOZip	363	400	–	478	–
λfreeZip	395	398	–	538	–
λZip-base	388	408	39	420	436
λZip-pragmatic	396	411	33	496	504
λZip-full	171	339	34	353	361
Zip-uncoop	514	427	45	608	620
Zip-coop	631	434	**46**	715	**709**

Fig. 5 Number of problems proved by the different provers

Vukmirović et al. Full is a refutationally complete extension of base that substitutes FLUIDSUP for FLEXSUP and includes the (EXT) axiom. Finally, we evaluated Zipperposition in a portfolio mode that runs the prover in various configurations (Zip-uncoop). We also evaluated a cooperative version of the portfolio that uses Ehoh as backend on higher-order problems (Zip-coop). 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. In the pragmatic configuration, our calculus outperformed λfreeZip on TH0λf problems and incurred less than 1% overhead compared with FOZip, but fell behind λfreeZip on SH-ll problems. The full configuration suffers greatly from the explosive extensionality axiom and FLUIDSUP rule.

Both base and pragmatic configurations outperformed Leo-III-uncoop, which runs a fixed configuration, by a substantial margin. Similarly, Zip-uncoop outperformed Satallax-uncoop, which runs a portfolio of configurations. Our most competitive configuration, Zip-coop, emerges as the winner on both problem sets containing λ-expressions.

8 Discussion and Related Work

Bentkamp et al. [13] 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 [32] and is mandated by the TPTP THF format [64] 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 clausal 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 [38], Jensen and Pietrzykowski's ω-resolution [40], Snyder's higher-order E-resolution [60], Benzmüller and Kohlhase's extensional higher-order resolution [15], Benzmüller's higher-order unordered paramodulation and RUE resolution [14], and Bhayat and Reger's combinatory superposition [20]. 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 [7, 46, 47, 56], connections [2], sequents [50], and satisfiability modulo theories (SMT) [8]. Andrews [3] and Benzmüller and Miller [16] 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 [20], "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 [15] and LEO-II [18] implement variants of RUE resolution. Leo-III [62] is based on higher-order paramodulation. Satallax [26] implements a higher-order tableau calculus guided by a SAT solver. LEO-II, Leo-III, and Satallax integrate first-order provers as terminal 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 [8]. Vampire now implements both combinatory superposition and an incomplete variant called restricted combinatory unification [19].

Half a century ago, Robinson [57] proposed to reduce higher-order logic to first-order logic via a translation. "Hammer" tools such as Sledgehammer [55], Miz$\mathbb{AR}$ [65], HOLy-Hammer [44], and CoqHammer [31] 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 [21].

9 Conclusion

We presented the clausal 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 [34], 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 [40, 61, 67] 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 et al. [67].

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, and Hans-Jörg Schurr were involved in many stimulating discussions. Christoph Weidenbach made this collaboration possible. Ahmed Bhayat, 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. Backes, J., Brown, C.E.: Analytic tableaux for higher-order logic with choice. J. Autom. Reason. **47**(4), 451–479 (2011)
8. 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)
9. Becker, H., Blanchette, J.C., Waldmann, U., Wand, D.: A transfinite Knuth–Bendix order for lambda-free higher-order terms. In: L. de Moura (ed.) CADE-26, *LNCS*, vol. 10395, pp. 432–453. Springer (2017)
10. Bentkamp, A.: Formalization of the embedding path order for lambda-free higher-order terms. Archive of Formal Proofs (2018). `http://isa-afp.org/entries/Lambda_Free_EPO.html`
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. To be submitted to a journal. `http://matryoshka.gforge.inria.fr/pubs/lfhosup_article.pdf`
13. 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)
14. Benzmüller, C.: Extensional higher-order paramodulation and RUE-resolution. In: H. Ganzinger (ed.) CADE-16, *LNCS*, vol. 1632, pp. 399–413. Springer (1999)
15. Benzmüller, C., Kohlhase, M.: Extensional higher-order resolution. In: C. Kirchner, H. Kirchner (eds.) CADE-15, *LNCS*, vol. 1421, pp. 56–71. Springer (1998)
16. 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)
17. Benzmüller, C., Paulson, L.C.: Multimodal and intuitionistic logics in simple type theory. Log. J. IGPL **18**(6), 881–892 (2010)
18. Benzmüller, C., Sultana, N., Paulson, L.C., Theiss, F.: The higher-order prover LEO-II. J. Autom. Reason. **55**(4), 389–404 (2015)

19. Bhayat, A., Reger, G.: Restricted combinatory unification. In: P. Fontaine (ed.) CADE-27, *LNCS*, vol. 11716, pp. 74–93. Springer (2019)
20. Bhayat, A., Reger, G.: A combinator-based superposition calculus for higher-order logic (technical report). Technical report (2020). `https://github.com/vprover/vampire_publications/blob/master/paper_drafts/comb_sup_report.pdf`
21. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. Log. Meth. Comput. Sci. **12**(4) (2016)
22. 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)
23. Blanchette, J.C., Waldmann, U., Wand, D.: A lambda-free higher-order recursive path order. In: J. Esparza, A.S. Murawski (eds.) FoSSaCS 2017, *LNCS*, vol. 10203, pp. 461–479. Springer (2017)
24. Blanqui, F., Jouannaud, J.P., Rubio, A.: The computability path ordering. Log. Meth. Comput. Sci. **11**(4) (2015)
25. Böhme, S., Nipkow, T.: Sledgehammer: Judgement Day. In: J. Giesl, R. Hähnle (eds.) IJCAR 2010, *LNCS*, vol. 6173, pp. 107–121. Springer (2010)
26. 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)
27. 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)
28. Cervesato, I., Pfenning, F.: A linear spine calculus. J. Log. Comput. **13**(5), 639–688 (2003)
29. Cruanes, S.: Extending superposition with integer arithmetic, structural induction, and beyond. Ph.D. thesis, École polytechnique (2015)
30. Cruanes, S.: Superposition with structural induction. In: C. Dixon, M. Finger (eds.) FroCoS 2017, *LNCS*, vol. 10483, pp. 172–188. Springer (2017)
31. Czajka, Ł., Kaliszyk, C.: Hammer for Coq: Automation for dependent type theory (2018)
32. 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)
33. Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer (2002)
34. Ganzinger, H., Stuber, J.: Superposition with equivalence reasoning and delayed clause normal form transformation. Information and Computation **199**(1–2), 3–23 (2005)
35. Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)
36. 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)
37. Henkin, L.: Completeness in the theory of types. J. Symb. Log. **15**(2), 81–91 (1950)
38. Huet, G.P.: A mechanization of type theory. In: N.J. Nilsson (ed.) IJCAI-73, pp. 139–146. William Kaufmann (1973)
39. Huet, G.P.: A unification algorithm for typed lambda-calculus. Theor. Comput. Sci. **1**(1), 27–57 (1975)
40. Jensen, D.C., Pietrzykowski, T.: Mechanizing ω-order type theory through unification. Theor. Comput. Sci. **3**(2), 123–171 (1976)
41. 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)
42. Jouannaud, J.P., Rubio, A.: Polymorphic higher-order recursive path orderings. J. ACM **54**(1), 2:1–2:48 (2007)
43. 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)
44. Kaliszyk, C., Urban, J.: HOL(y)Hammer: Online ATP service for HOL Light. Math. Comput. Sci. **9**(1), 5–22 (2015)
45. Kőnig, D.: Über eine Schlussweise aus dem Endlichen ins Unendliche. Acta Sci. Math. (Szeged) **3499/2009**(3:2-3), 121–130 (1927)
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. Miller, D.A.: A compact representation of proofs. Studia Logica **46**(4), 347–370 (1987)
55. 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)
56. Robinson, J.: Mechanizing higher order logic. In: B. Meltzer, D. Michie (eds.) Machine Intelligence, vol. 4, pp. 151–170. Edinburgh University Press (1969)
57. 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)
58. Schlichtkrull, A., Blanchette, J.C., Traytel, D., Waldmann, U.: Formalizing Bachmair and Ganzinger's ordered resolution prover. In: D. Galmiche, S. Schulz, R. Sebastiani (eds.) IJCAR 2018, *LNCS*, vol. 10900, pp. 89–107. Springer (2018)
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 TPTP problem library and associated infrastructure—from CNF to TH0, TPTP v6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017)
64. 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)
65. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. J. Autom. Reason. **50**(2), 229–241 (2013)
66. Vukmirović, P., Bentkamp, A., Nummelin, V.: Efficient full higher-order unification (technical report). Technical report (2020). `http://matryoshka.gforge.inria.fr/pubs/hounif_report.pdf`
67. 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, pp. 192–210. Springer (2019)
68. Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving (technical report). Technical report (2020). `http://matryoshka.gforge.inria.fr/pubs/saturate_report_v2.pdf`