Superposition for Full Higher-order Logic

Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirović

Superposition for Full Higher-Order Logic

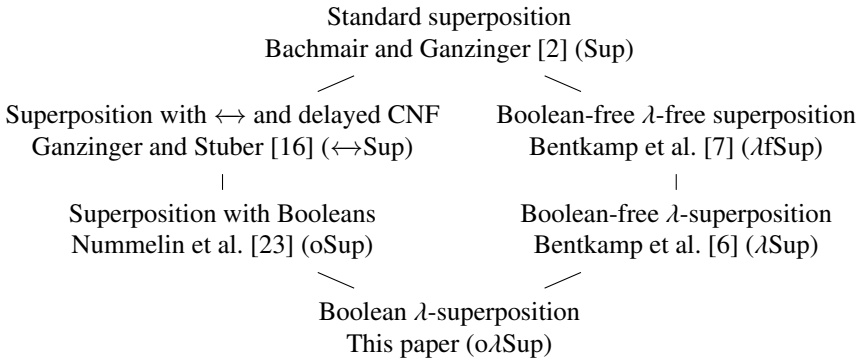
Alexander Bentkamp[1], Jasmin Blanchette[1,2,3],
Sophie Tourret[2,3], and Petar Vukmirović[1]

[1] Vrije Universiteit Amsterdam, Amsterdam, the Netherlands
{a.bentkamp,j.c.blanchette,p.vukmirovic}@vu.nl
[2] Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
sophie.tourret@inria.fr
[3] Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany

Abstract. We recently designed two calculi as stepping stones towards superposition for full higher-order logic: Boolean-free λ-superposition and superposition for first-order logic with interpreted Booleans. Stepping on these stones, we finally reach a sound and refutationally complete calculus for higher-order logic with polymorphism, extensionality, Hilbert choice, and Henkin semantics. In addition to the complexity of combining the calculus's two predecessors, new challenges arise from the interplay between λ-terms and Booleans. Our implementation in Zipperposition outperforms all other higher-order theorem provers and is on a par with an earlier, pragmatic prototype of Booleans in Zipperposition.

1 Introduction

Superposition is a leading calculus for first-order logic with equality. We have been wondering for some years whether it would be possible to gracefully generalize it to extensional higher-order logic and use it as the basis of a strong higher-order automatic theorem prover. Towards this goal, we have, together with colleagues, designed superposition-like calculi for three intermediate logics between first-order and higher-order logic. Now we are finally ready to assemble a superposition calculus for full higher-order logic. The filiation of our new calculus from Bachmair and Ganzinger's standard first-order superposition is as follows:



Our goal was to devise an efficient calculus for higher-order logic. To achieve it, we pursued two objectives. First, the calculus should be refutationally complete. Second, the calculus should coincide as much as possible with its predecessors oSup and λSup on the respective fragments of higher-order logic (which in turn essentially coincide with Sup on first-order logic). Achieving these objectives is the main contribution of this paper. We made an effort to keep the calculus simple, but often the refutational completeness proof forced our hand to add conditions or special cases.

Like oSup, our calculus oλSup operates on clauses that can contain Boolean subterms, and it interleaves clausification with other inferences. Like λSup, oλSup eagerly $\beta\eta$-normalizes terms, employs full higher-order unification, and relies on a fluid subterm superposition rule (FLUIDSUP) to simulate superposition inferences below applied variables—i.e., terms of the form $y\,t_1 \ldots t_n$ for $n \geq 1$.

Because oSup contains several superposition-like inference rules for Boolean subterms, our completeness proof requires dedicated *fluid Boolean subterm hoisting rules* (FLUIDBOOLHOIST, FLUIDLOOBHOIST), which simulate Boolean inferences below applied variables, in addition to FLUIDSUP, which simulates superposition inferences.

Due to restrictions related to the term order that parameterizes superposition, it is difficult to handle variables bound by unclausified quantifiers if these variables occur applied or in arguments of applied variables. We solve the issue by replacing such quantified terms $\forall y.\, t$ by equivalent terms $(\lambda y.\, t) \approx (\lambda y.\, \mathsf{T})$ in a preprocessing step.

We implemented our calculus in the Zipperposition prover and evaluated it on TPTP and Sledgehammer benchmarks. The new Zipperposition outperforms all other higher-order provers and is on a par with an ad hoc implementation of Booleans in the same prover by Vukmirović and Nummelin [30]. We refer to the technical report [8] for the completeness proof and a more detailed account of the calculus and its evaluation.

2 Logic

Our logic is higher-order logic (simple type theory) with rank-1 polymorphism, Hilbert choice, and functional and Boolean extensionality. Its syntax mostly follows Gordon and Melham [17]. We use the notation $\bar{a}_n$ or $\bar{a}$ to stand for the tuple $(a_1, \ldots, a_n)$ where $n \geq 0$. Deviating from Gordon and Melham, type arguments are explicit, written as $\mathsf{c}\langle\bar{\tau}_m\rangle$ for a symbol $\mathsf{c} : \Pi\bar{\alpha}_m.\, \upsilon$ and types $\bar{\tau}_m$. In the type signature Σ_{ty}, we require the presence of a nullary Boolean type constructor o and a binary function type constructor $\to$. In the term signature Σ, we require the presence of the logical symbols T, $\bot$, $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\forall$, $\exists$, $\approx$, and $\not\approx$. The logical symbols are shown in bold to distinguish them from the notation used for clauses below. Moreover, we require the presence of the Hilbert choice operator $\varepsilon \in \Sigma$. Although ε is interpreted in our semantics, we do not consider it a logical symbol. Our calculus will enforce the semantics of ε by an axiom, whereas the semantics of the logical symbols will be enforced by inference rules. We write $\mathcal{V}$ for the set of (term) variables. We use Henkin semantics, in the style of Fitting [15], with respect to which we can prove our calculus refutationally complete. In summary, our logic essentially coincides with the TPTP TH1 format [20].

We generally view terms modulo $\alpha\beta\eta$-equivalence. When defining operations that need to analyze the structure of terms, however, we use a custom normal form as the

default representative of a $\beta\eta$-equivalence class: The $\beta\eta\mathsf{Q}_\eta$*-normal* form $t{\downarrow_{\beta\eta\mathsf{Q}_\eta}}$ of a term t is obtained by bringing the term into η-short β-normal form and finally applying the rewrite rule $\mathsf{Q}\langle\tau\rangle\ s \longrightarrow_{\mathsf{Q}_\eta} \mathsf{Q}\langle\tau\rangle\ (\lambda x.\ s\ x)$ exhaustively whenever s is not a λ-expression. Here and elsewhere, Q stands for either $\forall$ or $\exists$.

On top of the standard higher-order terms, we install a clausal structure that allows us to formulate calculus rules in the style of first-order superposition. A literal $s \mathrel{\tilde{\approx}} t$ is an equation $s \approx t$ or disequation $s \not\approx t$ of terms s and t; both equations and disequations are unordered pairs. A clause $L_1 \vee \cdots \vee L_n$ is a finite multiset of literals L_j. The empty clause is written as $\bot$. This clausal structure does not restrict the logic, because an arbitrary term t of Boolean type can be written as the clause $t \approx \mathsf{T}$.

We considered excluding negative literals by encoding them as $(s \mathrel{\tilde{\not\approx}} t) \approx \bot$, following $\leftrightarrow$Sup [16]. However, this approach would make the conclusion of the equality factoring rule (EFACT) too large for our purposes. Regardless, the simplification machinery will allow us to reduce negative literals $t \not\approx \bot$ and $t \not\approx \mathsf{T}$ to $t \approx \mathsf{T}$ and $t \approx \bot$, respectively, thereby eliminating redundant representations of nonequational literals.

We let $\mathrm{CSU}(s,t)$ denote an arbitrary (preferably, minimal) complete set of unifiers for two terms s and t on the set of free variables of the clauses in which s and t occur. To compute such sets, Huet-style preunification [18] is not sufficient, and we must resort to a full unification procedure [19, 29]. To cope with the nontermination of such procedures, we use dovetailing as described by Vukmirović et al. [28, Sect. 5].

Some of the rules in our calculus introduce Skolem symbols, representing objects mandated by existential quantification. We assume that these symbols do not occur in the input problem. More formally, given a problem over a term signature Σ, our calculus operates on a Skolem-extended term signature Σ_{sk} that, in addition to all symbols from Σ, inductively contains symbols $\mathsf{sk}_{\Pi\bar{\alpha}.\,\forall\bar{x}.\,\exists z.\,t\,z} : \Pi\bar{\alpha}.\,\bar{\tau} \to \upsilon$ for all types υ, variables $z : \upsilon$, and terms $t : \upsilon \to \mathrm{o}$ over Σ_{sk}, where $\bar{\alpha}$ are the free type variables occurring in t and $\bar{x} : \bar{\tau}$ are the free term variables occurring in t, both in order of first occurrence.

3 The Calculus

The oλSup calculus closely resembles λSup, augmented with rules for Boolean reasoning that are inspired by oSup. As in λSup, superposition-like inferences are restricted to certain first-order-like subterms, the *green subterms*, which we define inductively as follows: Every term t is a green subterm of t, and for all symbols $\mathsf{f} \in \Sigma \setminus \{\forall, \exists\}$, if t is a green subterm of u_i for some i, then t is a green subterm of $\mathsf{f}\langle\bar{\tau}\rangle\ \bar{u}$. For example, the green subterms of $\mathsf{f}\ (\mathsf{g}\ (\neg\,\mathsf{p}))\ (\forall\langle\tau\rangle\ (\lambda x.\,\mathsf{q}))\ (y\,\mathsf{a})\ (\lambda x.\,\mathsf{h}\,\mathsf{b})$ are the term itself, $\mathsf{g}\ (\neg\,\mathsf{p})$, $\neg\,\mathsf{p}$, p, $\forall\langle\tau\rangle\ (\lambda x.\,\mathsf{q})$, $y\,\mathsf{a}$, and $\lambda x.\,\mathsf{h}\,\mathsf{b}$. We write $s\langle t\rangle$ to denote a term s with a green subterm t and call the first-order-like context $s\langle\,\rangle$ a *green context*.

Following λSup, we call a term t *fluid* if (1) $t{\downarrow_{\beta\eta\mathsf{Q}_\eta}}$ is of the form $y\,\bar{u}_n$ where $n \geq 1$, or (2) $t{\downarrow_{\beta\eta\mathsf{Q}_\eta}}$ is a λ-expression and there exists a substitution σ such that $t\sigma{\downarrow_{\beta\eta\mathsf{Q}_\eta}}$ is not a λ-expression (due to η-reduction). Intuitively, fluid terms are terms whose normal form can change radically as a result of instantiation.

We define deeply occurring variables as in λSup, but exclude λ-expressions directly below quantifiers: A variable *occurs deeply* in a clause C if it occurs inside an argument of an applied variable or inside a λ-expression that is not directly below a quantifier.

Preprocessing. Our completeness theorem requires that quantified variables do not appear in certain higher-order contexts. We use preprocessing to eliminate problematic occurrences of quantifiers. The rewrite rules $\mathsf{A}_{\approx}$ and $\mathsf{E}_{\approx}$, which we collectively denote by $\mathsf{Q}_{\mathbin{\approx}}$, are defined as $\mathsf{A}\langle\tau\rangle \longrightarrow_{\mathsf{A}_{\approx}} \lambda y.\, y \mathbin{\approx} (\lambda x.\, \mathsf{T})$ and $\mathsf{E}\langle\tau\rangle \longrightarrow_{\mathsf{E}_{\approx}} \lambda y.\, y \not\approx (\lambda x.\, \bot)$ where the rewritten occurrence of $\mathsf{Q}\langle\tau\rangle$ is unapplied or has an argument of the form $\lambda x.\, v$ such that x occurs as a nongreen subterm of v. If either of these rewrite rules can be applied to a given term, the term is $\mathsf{Q}_{\mathbin{\approx}}$*-reducible*; otherwise, it is $\mathsf{Q}_{\mathbin{\approx}}$*-normal*.

For example, the term $\lambda y.\, \mathsf{E}\langle\iota \to \iota\rangle\, (\lambda x.\, \mathsf{g}\, x\, y\, (z\, y)\, (\mathsf{f}\, x))$ is $\mathsf{Q}_{\mathbin{\approx}}$-normal. A term may be $\mathsf{Q}_{\mathbin{\approx}}$-reducible because a quantifier appears unapplied (e.g., $\mathsf{g}\, \mathsf{E}\langle\iota\rangle$); a quantified variable occurs applied (e.g., $\mathsf{E}\langle\iota \to \iota\rangle\, (\lambda x.\, x\, \mathsf{a})$); a quantified variable occurs inside a nested λ-expression (e.g., $\mathsf{A}\langle\iota\rangle\, (\lambda x.\, \mathsf{f}\, (\lambda y.\, x))$); or a quantified variable occurs in the argument of a variable, either a free variable (e.g., $\mathsf{A}\langle\iota\rangle\, (\lambda x.\, z\, x)$) or a variable bound above the quantifier (e.g., $\lambda y.\, \mathsf{E}\langle\iota\rangle\, (\lambda x.\, y\, x)$).

A preprocessor $\mathsf{Q}_{\mathbin{\approx}}$-normalizes the input problem. Although inferences may produce $\mathsf{Q}_{\mathbin{\approx}}$-reducible clauses, we do not $\mathsf{Q}_{\mathbin{\approx}}$-normalize during the derivation process itself. Instead, $\mathsf{Q}_{\mathbin{\approx}}$-reducible ground instances of clauses will be considered redundant by the redundancy criterion. Thus, clauses whose ground instances are all $\mathsf{Q}_{\mathbin{\approx}}$-reducible can be deleted. However, there are $\mathsf{Q}_{\mathbin{\approx}}$-reducible clauses, such as $x\, \mathsf{A}\langle\iota\rangle \mathbin{\approx} \mathsf{a}$, that nevertheless have $\mathsf{Q}_{\mathbin{\approx}}$-normal ground instances. Such clauses must be kept because the completeness proof relies on their $\mathsf{Q}_{\mathbin{\approx}}$-normal ground instances.

In principle, we could omit the side condition of the $\mathsf{Q}_{\mathbin{\approx}}$-rewrite rules and eliminate all quantifiers. However, the calculus (especially, the redundancy criterion) performs better with quantifiers than with λ-expressions, which is why we restrict $\mathsf{Q}_{\mathbin{\approx}}$-normalization as much as the completeness proof allows. Extending the preprocessing to eliminate all Boolean terms as in Kotelnikov et al. [21] does not work for higher-order logic because Boolean terms can contain variables bound by enclosing λ-expressions.

Term Order. The calculus is parameterized by a well-founded strict total order $\succ$ on ground terms satisfying these four criteria: (O1) compatibility with green contexts—i.e., $s' \succ s$ implies $t\langle s'\rangle \succ t\langle s\rangle$; (O2) green subterm property—i.e. $t\langle s\rangle \succeq s$ where $\succeq$ is the reflexive closure of $\succ$; (O3) $u \succ \bot \succ \mathsf{T}$ for all terms $u \notin \{\mathsf{T}, \bot\}$; (O4) $\mathsf{Q}\langle\tau\rangle\, t \succ t\, u$ for all types τ, terms t, and terms u such that $\mathsf{Q}\langle\tau\rangle\, t$ and u are $\mathsf{Q}_{\mathbin{\approx}}$-normal and the only Boolean green subterms of u are T and $\bot$. The restriction of (O4) to $\mathsf{Q}_{\mathbin{\approx}}$-normal terms ensures that term orders fulfilling the requirements exist, but it forces us to preprocess the input problem. We extend $\succ$ to literals and clauses via the multiset extensions in the standard way [2, Sect. 2.4].

For nonground terms, $\succ$ is required to be a strict partial order such that $t \succ s$ implies $t\theta \succ s\theta$ for all grounding substitutions θ. As in λSup, we also introduce a nonstrict variant $\succsim$ for which we require that $t\theta \succeq s\theta$ for all grounding substitutions θ whenever $t \succsim s$, and similarly for literals and clauses.

To construct a concrete order fulfilling these requirements, we define an encoding into untyped first-order terms, and compare these using a variant of the Knuth–Bendix order. In a first step, denoted O, the encoding translates fluid terms t as fresh variables z_t; nonfluid λ-expressions $\lambda x{:}\tau.\, u$ as $\mathsf{lam}(O(\tau), O(u))$; applied quantifiers $\mathsf{Q}\langle\tau\rangle(\lambda x{:}\tau.\, u)$ as $\mathsf{Q}_1(O(\tau), O(u))$; and other terms $\mathsf{f}\langle\bar{\tau}\rangle\, \bar{u}_k$ as $\mathsf{f}_k(O(\bar{\tau}), O(\bar{u}_k))$. Bound variables are encoded as constants db^i corresponding to De Bruijn indices. In a second step, denoted $\mathcal{P}$, the

encoding replaces Q_1 by Q'_1 and variables z by z' whenever they occur below lam. For example, $\forall\langle\iota\rangle(\lambda x.\,\mathsf{p}\,y\,y\,(\lambda u.\,\mathsf{f}\,y\,y\,(\forall\langle\iota\rangle(\lambda v.\,u))))$ is encoded as $\forall_1(\iota,\mathsf{p}_3(y,y,\mathsf{lam}(\mathsf{o},\mathsf{f}_3(y',y',\forall'_1(\iota,\mathsf{db}^1)))))$. The first-order terms can then be compared using a transfinite Knuth–Bendix order $\succ_{\mathsf{kb}}$ [22]. Let the weight of $\forall_1$ and $\exists_1$ be ω, the weight of T_0 and $\bot_0$ be 1, and the weights of all other symbols be less than ω. Let the precedence $>$ be total and $\bot_0, \mathsf{T}_0$ be the symbols of lowest precedence, with $\bot_0 > \mathsf{T}_0$. Then let $t \succ s$ if $O(\mathcal{P}(t)) \succ_{\mathsf{kb}} O(\mathcal{P}(s))$ and $t \succsim s$ if $O(\mathcal{P}(t)) \succeq_{\mathsf{kb}} O(\mathcal{P}(s))$.

Selection Functions. The calculus is also parameterized by a literal selection function and a Boolean subterm selection function. We define an element x of a multiset M to be $\trianglerighteq$-*maximal* for some relation $\trianglerighteq$ if for all $y \in M$ with $y \trianglerighteq x$, we have $y = x$. It is *strictly* $\trianglerighteq$-maximal if it is $\trianglerighteq$-maximal and occurs only once in M.

The literal selection function *HLitSel* maps each clause to a subset of *selected literals*. A literal may not be selected if it is positive and neither side is $\bot$. Moreover, a literal $L\langle y\rangle$ may not be selected if $y\,\bar{u}_n$, with $n \geq 1$, is a $\succeq$-maximal term of the clause.

The Boolean subterm selection function *HBoolSel* maps each clause C to a subset of *selected subterms* in C. Selected subterms must be green subterms of Boolean type. Moreover, a subterm s must not be selected if $s = \mathsf{T}$, if $s = \bot$, if s is a variable-headed term, if s is at the topmost position on either side of a positive literal, or if s contains a variable y as a green subterm, and $y\,\bar{u}_n$, with $n \geq 1$, is a $\succeq$-maximal term of the clause.

Eligibility. A literal L is (*strictly*) *eligible* w.r.t. a substitution σ in C if it is selected in C or there are no selected literals and no selected Boolean subterms in C and $L\sigma$ is (strictly) $\succsim$-maximal in $C\sigma$.

The eligible subterms of a clause C w.r.t. a substitution σ are inductively defined as follows: Any selected subterm is eligible. If a literal $L = s \mathrel{\dot{\approx}} t$ with $s\sigma \not\precsim t\sigma$ is either eligible and negative or strictly eligible and positive, then the subterm s is eligible. If a subterm t is eligible and the head of t is not $\approx$ or $\not\approx$, all direct green subterms of t are eligible. If a subterm t is eligible and t is of the form $u \approx v$ or $u \not\approx v$, then u is eligible if $u\sigma \not\precsim v\sigma$ and v is eligible if $u\sigma \not\succsim v\sigma$.

The Core Inference Rules. The calculus consists of the following core inference rules. The first five rules stem from λSup, with minor adaptions concerning Booleans:

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \quad C\langle u\rangle}{(D' \vee C\langle t'\rangle)\sigma}\,\textsc{Sup} \qquad \frac{\overbrace{C' \vee u \not\approx u'}^{C}}{C'\sigma}\,\textsc{ERes} \qquad \frac{\overbrace{C' \vee u' \approx v' \vee u \approx v}^{C}}{(C' \vee v \not\approx v' \vee u \approx v')\sigma}\,\textsc{EFact}$$

$$\frac{\overbrace{D' \vee t \approx t'}^{D} \quad C\langle u\rangle}{(D' \vee C\langle z\,t'\rangle)\sigma}\,\textsc{FluidSup} \qquad \frac{\overbrace{C' \vee s \approx s'}^{C}}{C'\sigma \vee s\sigma\,\bar{x}_n \approx s'\sigma\,\bar{x}_n}\,\textsc{ArgCong}$$

Sup 1. u is not fluid; 2. u is not a variable deeply occurring in C; 3. 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. u is eligible in C w.r.t. σ; 7. $C\sigma \not\precsim D\sigma$; 8. $t \approx t'$ is strictly eligible in D w.r.t. σ; 9. $t\sigma$ is not a fully applied logical symbol; 10. if $t'\sigma = \bot$, the subterm u is at the top level of a positive literal.

ERES 1. $\sigma \in \mathrm{CSU}(u, u')$; 2. $u \not\approx u'$ is eligible in C w.r.t. σ.

EFACT 1. $\sigma \in \mathrm{CSU}(u, u')$; 2. $u\sigma \not\precsim v\sigma$; 3. $(u \approx v)\sigma$ is $\succsim$-maximal in $C\sigma$; 4. $u\sigma \not\precsim v\sigma$; 5. nothing is selected in C.

FLUIDSUP 1. u is a variable deeply occurring in C or u is fluid; 2. z is a fresh variable; 3. $\sigma \in \mathrm{CSU}(z\,t, u)$; 4. $(z\,t')\sigma \neq (z\,t)\sigma$; 5.–10. as for SUP.

ARGCONG 1. $n > 0$; 2. σ is the most general type substitution that ensures well-typedness of the conclusion for a given n; 3. $\bar{x}_n$ is a tuple of distinct fresh variables; 4. the literal $s \approx s'$ is strictly eligible in C w.r.t. σ.

The following rules are concerned with Boolean reasoning and originate from oSup. They have been adapted to support polymorphism and applied variables.

$$\frac{C\langle u\rangle}{(C\langle \bot\rangle \vee u \approx \top)\sigma}\ \textsc{BoolHoist} \qquad \frac{\overbrace{C' \vee s \approx s'}^{C}}{C'\sigma}\ \textsc{FalseElim}$$

$$\frac{C\langle u\rangle}{(C\langle \bot\rangle \vee x \approx y)\sigma}\ \textsc{EqHoist} \qquad \frac{C\langle u\rangle}{C\langle t'\rangle\sigma}\ \textsc{BoolRw}$$

$$\frac{C\langle u\rangle}{(C\langle \top\rangle \vee x \approx y)\sigma}\ \textsc{NeqHoist} \qquad \frac{C\langle u\rangle}{C\langle y\,(\mathsf{sk}_{\Pi\bar{\alpha}.\,\forall\bar{x}.\,\exists z.\,\neg y\sigma\,z}\langle\bar{\alpha}\rangle\,\bar{x})\rangle\sigma}\ \textsc{ForallRw}$$

$$\frac{C\langle u\rangle}{(C\langle \bot\rangle \vee y\,x \approx \top)\sigma}\ \textsc{ForallHoist} \qquad \frac{C\langle u\rangle}{C\langle y\,(\mathsf{sk}_{\Pi\bar{\alpha}.\,\forall\bar{x}.\,\exists z.\,y\sigma\,z}\langle\bar{\alpha}\rangle\,\bar{x})\rangle\sigma}\ \textsc{ExistsRw}$$

$$\frac{C\langle u\rangle}{(C\langle \top\rangle \vee y\,x \approx \bot)\sigma}\ \textsc{ExistsHoist}$$

BOOLHOIST 1. σ is a type unifier of the type of u with the Boolean type o (i.e., the identity if u is Boolean or $\{\alpha \mapsto \mathrm{o}\}$ if u is of type α for some type variable α); 2. the head of u is neither a variable nor a logical symbol; 3. u is eligible in C; 4. the occurrence of u is not at the top level of a positive literal.

EQHOIST, NEQHOIST, FORALLHOIST, EXISTSHOIST 1. $\sigma \in \mathrm{CSU}(u,\ x \mathrel{\approx\!\!\!\approx} y)$, $\sigma \in \mathrm{CSU}(u,\ x \mathrel{\not\approx\!\!\!\approx} y)$, $\sigma \in \mathrm{CSU}(u, \forall\langle\alpha\rangle\, y)$, or $\sigma \in \mathrm{CSU}(u, \exists\langle\alpha\rangle\, y)$, respectively; 2. x, y, and α are fresh variables; 3. u is eligible in C w.r.t. σ; 4. if the head of u is a variable, it must be applied and the affected literal must be of the form $u \approx \top$, $u \approx \bot$, or $u \approx v$ where v is a variable-headed term.

FALSEELIM 1. $\sigma \in \mathrm{CSU}(s \approx s', \bot \approx \top)$; 2. $s \approx s'$ is strictly eligible in C w.r.t. σ.

BOOLRW 1. $\sigma \in \mathrm{CSU}(t, u)$ and (t, t') is one of the following pairs, where y is a fresh variable: $(\neg\bot, \top)$, $(\neg\top, \bot)$, $(\bot \wedge \bot, \bot)$, $(\top \wedge \bot, \bot)$, $(\bot \wedge \top, \bot)$, $(\top \wedge \top, \top)$, $(\bot \vee \bot, \bot)$, $(\top \vee \bot, \top)$, $(\bot \vee \top, \top)$, $(\top \vee \top, \top)$, $(\bot \to \bot, \top)$, $(\top \to \bot, \bot)$, $(\bot \to \top, \top)$, $(\top \to \top, \top)$, $(y \mathrel{\approx\!\!\!\approx} y, \top)$, $(y \mathrel{\not\approx\!\!\!\approx} y, \bot)$; 2. u is not a variable; 3. u is eligible in C w.r.t. σ; 4. if the head of u is a variable, it must be applied and the affected literal must be of the form $u \approx \top$, $u \approx \bot$, or $u \approx v$ where v is a variable-headed term.

FORALLRW, EXISTSRW 1. $\sigma \in \mathrm{CSU}(\forall\langle\beta\rangle\, y, u)$ and $\sigma \in \mathrm{CSU}(\exists\langle\beta\rangle\, y, u)$, respectively, where β is a fresh type variable, y is a fresh term variable, $\bar{\alpha}$ are the free type variables and $\bar{x}$ are the free term variables occurring in $y\sigma$ in order of first occurrence;

2. u is not a variable; 3. u is eligible in C w.r.t. σ; 4. if the head of u is a variable, it must be applied and the affected literal must be of the form $u \approx \mathsf{T}$, $u \approx \bot$, or $u \approx v$ where v is a variable-headed term; 5. for FORALLRW, the indicated occurrence of u is not in a literal $u \approx \mathsf{T}$, and for EXISTSRW, the indicated occurrence of u is not in a literal $u \approx \bot$.

Like SUP, also the Boolean rules must be simulated in fluid terms. The following rules are Boolean counterparts of FLUIDSUP:

$$\frac{C\langle u\rangle}{(C\langle z\,\bot\rangle \vee x \approx \mathsf{T})\sigma}\,\text{FLUID-BOOLHOIST} \qquad \frac{C\langle u\rangle}{(C\langle z\,\mathsf{T}\rangle \vee x \approx \bot)\sigma}\,\text{FLUID-LOOBHOIST}$$

FLUIDBOOLHOIST 1. u is fluid; 2. z and x are fresh variables; 3. $\sigma \in \mathrm{CSU}(z\,x,\,u)$; 4. $(z\,\bot)\sigma \neq (z\,x)\sigma$; 5. $x\sigma \neq \mathsf{T}$ and $x\sigma \neq \bot$; 6. u is eligible in C w.r.t. σ.
FLUIDLOOBHOIST Like the above but with $\bot$ replaced by T in condition 4.

In addition to the inference rules, our calculus relies on two axioms, below. Axiom (EXT), from λSup, embodies functional extensionality; the expression $\mathsf{diff}\langle\alpha,\beta\rangle$ abbreviates $\mathsf{sk}_{\Pi\alpha\beta.\,\forall z\,y.\,\exists x.\,z\,x \not\approx y\,x}\langle\alpha,\beta\rangle$. Axiom (CHOICE) characterizes the Hilbert choice operator ε.

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

$$y\,x \approx \bot \vee y\,(\varepsilon\langle\alpha\rangle\,y) \approx \mathsf{T} \qquad \text{(CHOICE)}$$

Rationale for the Rules. Most of the calculus's rules are adapted from its precursors. SUP, ERES, and EFACT are already present in Sup, with slightly different side conditions. Notably, as in λfSup and λSup, SUP inferences are required only into green contexts. Other subterms are accessed indirectly via ARGCONG and (EXT).

The rules BOOLHOIST, EQHOIST, NEQHOIST, FORALLHOIST, EXISTSHOIST, FALSEELIM, BOOLRW, FORALLRW, and EXISTSRW, concerned with Boolean reasoning, stem from oSup, which was inspired by $\leftrightarrow$Sup. Except for BOOLHOIST and FALSEELIM, these rules have a condition stating that "if the head of u is a variable, it must be applied and the affected literal must be of the form $u \approx \mathsf{T}$, $u \approx \bot$, or $u \approx v$ where v is a variable-headed term." The inferences at variable-headed terms permitted by this condition are our form of primitive substitution [1,18], a mechanism that blindly substitutes logical connectives and quantifiers for variables z with a Boolean result type.

Example 1. Our calculus can prove that Leibniz equality implies equality (i.e., if two values behave the same for all predicates, they are equal) as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{z\,\mathsf{a} \approx \bot \vee z\,\mathsf{b} \approx \mathsf{T}}{(x\,\mathsf{a} \mathrel{\approx\!\!\!\approx} y\,\mathsf{a}) \approx \bot \vee \bot \approx \mathsf{T} \vee x\,\mathsf{b} \approx y\,\mathsf{b}}\,\text{EQHOIST}}{\mathsf{T} \approx \bot \vee \bot \approx \mathsf{T} \vee w\,\mathsf{a}\,\mathsf{b}\,\mathsf{b} \approx w\,\mathsf{b}\,\mathsf{a}\,\mathsf{b}}\,\text{BOOLRW}}{\bot \approx \mathsf{T} \vee w\,\mathsf{a}\,\mathsf{b}\,\mathsf{b} \approx w\,\mathsf{b}\,\mathsf{a}\,\mathsf{b}}\,\text{FALSEELIM}}{w\,\mathsf{a}\,\mathsf{b}\,\mathsf{b} \approx w\,\mathsf{b}\,\mathsf{a}\,\mathsf{b}}\,\text{FALSEELIM} \qquad \mathsf{a} \not\approx \mathsf{b}}{\dfrac{\mathsf{a} \not\approx \mathsf{a}}{\bot}\,\text{ERES}}\,\text{SUP}$$

The EqHoist inference, applied on $z\,\mathsf{b}$, illustrates how our calculus introduces logical symbols without a dedicated primitive substitution rule. Although $\approx$ does not appear in the premise, we still need to apply EqHoist on $z\,\mathsf{b}$ with $\mathrm{CSU}(z\,\mathsf{b},\, x_0 \approx y_0) = \{\{z \mapsto \lambda v.\, x\,v \approx y\,v,\ x_0 \mapsto x\,\mathsf{b},\ y_0 \mapsto y\,\mathsf{b}\}\}$. Other calculi [1, 9, 18, 26] would apply an explicit primitive substitution rule instead, yielding essentially $(x\,\mathsf{a} \approx y\,\mathsf{a}) \approx \bot \vee (x\,\mathsf{b} \approx y\,\mathsf{b}) \approx \top$. However, in our approach this clause is subsumed and could be discarded immediately. By hoisting the equality to the clausal level, we bypass the redundancy criterion.

Next, BoolRw can be applied to $x\,\mathsf{a} \approx y\,\mathsf{a}$ with $\mathrm{CSU}(x\,\mathsf{a} \approx y\,\mathsf{a},\, y_0 \approx y_0) = \{\{x \mapsto \lambda v.\, w\,\mathsf{a}\,v\,v,\ y \mapsto \lambda v.\, w\,v\,\mathsf{a}\,v,\ y_0 \mapsto w\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\}\}$. The two FalseElim steps remove the $\bot \approx \top$ literals. Then Sup is applicable with the unifier $\{w \mapsto \lambda x_1\, x_2\, x_3.\, x_2\} \in \mathrm{CSU}(\mathsf{b},\, w\,\mathsf{a}\,\mathsf{b}\,\mathsf{b})$, and ERes derives the contradiction.

Like in λSup, the FluidSup rule is responsible for simulating superposition inferences below applied variables, other fluid terms, and deeply occurring variables. Complementarily, FluidBoolHoist and FluidLoobHoist simulate the various Boolean inference rules below fluid terms. Initially, we considered adding a fluid version of each rule that operates on Boolean subterms, but we discovered that FluidBoolHoist and FluidLoobHoist suffice to achieve refutational completeness.

Example 2. The clause set consisting of $\mathsf{h}\,(y\,\mathsf{b}) \not\approx \mathsf{h}\,(\mathsf{g}\,\bot) \vee \mathsf{h}\,(y\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,\top)$ and $\mathsf{a} \not\approx \mathsf{b}$ highlights the need for FluidBoolHoist and its companion. The set is unsatisfiable because the instantiation $\{y \mapsto \lambda x.\, \mathsf{g}\,(x \approx \mathsf{a})\}$ produces the clause $\mathsf{h}\,(\mathsf{g}\,(\mathsf{b} \approx \mathsf{a})) \not\approx \mathsf{h}\,(\mathsf{g}\,\bot) \vee \mathsf{h}\,(\mathsf{g}\,(\mathsf{a} \approx \mathsf{a})) \not\approx \mathsf{h}\,(\mathsf{g}\,\top)$, which is unsatisfiable in conjunction with $\mathsf{a} \not\approx \mathsf{b}$.

The literal selection function can select either literal in the first clause. ERes is applicable in either case, but the unifiers $\{y \mapsto \lambda x.\, \mathsf{g}\,\bot\}$ and $\{y \mapsto \lambda x.\, \mathsf{g}\,\top\}$ do not lead to a contradiction. Instead, we need to apply FluidBoolHoist if the first literal is selected or FluidLoobHoist if the second literal is selected. In the first case, the derivation is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{h}\,(y\,\mathsf{b}) \not\approx \mathsf{h}\,(\mathsf{g}\,\bot) \vee \mathsf{h}\,(y\,\mathsf{a}) \not\approx \mathsf{h}\,(\mathsf{g}\,\top)}{\mathsf{h}\,(z'\,\mathsf{b}\,\bot) \not\approx \mathsf{h}\,(\mathsf{g}\,\bot) \vee \mathsf{h}\,(z'\,\mathsf{a}\,(x'\mathsf{a})) \not\approx \mathsf{h}\,(\mathsf{g}\,\top) \vee x'\,\mathsf{b} \approx \top}\,\textsc{FluidBoolHoist}}{\mathsf{h}\,(\mathsf{g}\,(x'\,\mathsf{a})) \not\approx \mathsf{h}\,(\mathsf{g}\,\top) \vee x'\,\mathsf{b} \approx \top}\,\textsc{ERes} \qquad \mathsf{h}\,(\mathsf{g}\,(x''\,\mathsf{a} \approx x'''\,\mathsf{a})) \not\approx \mathsf{h}\,(\mathsf{g}\,\top) \vee \bot \approx \top \vee x''\,\mathsf{b} \approx x'''\,\mathsf{b}}{\mathsf{a} \not\approx \mathsf{b} \qquad \mathsf{h}\,(\mathsf{g}\,(\mathsf{a} \approx x'''\,\mathsf{a})) \not\approx \mathsf{h}\,(\mathsf{g}\,\top) \vee \bot \approx \top \vee \mathsf{a} \not\approx x'''\,\mathsf{b}}\,\textsc{Sup}}{\mathsf{h}\,(\mathsf{g}\,\top) \not\approx \mathsf{h}\,(\mathsf{g}\,\top) \vee \bot \approx \top \vee \mathsf{a} \not\approx \mathsf{a}}\,\textsc{BoolRw}}{\bot \approx \top \vee \mathsf{a} \not\approx \mathsf{a}}\,\textsc{ERes}}{\bot \approx \top}\,\textsc{ERes}}{\bot}\,\textsc{FalseElim}$$

The FluidBoolHoist inference uses the unifier $\{y \mapsto \lambda u.\, z'\,u\,(x'\,u),\ z \mapsto \lambda u.\, z'\,\mathsf{b}\,u,\ x \mapsto x'\,\mathsf{b}\} \in \mathrm{CSU}(z\,x, y\,\mathsf{b})$. We apply ERes to the first literal of the resulting clause, with unifier $\{z' \mapsto \lambda u v.\, \mathsf{g}\,v\} \in \mathrm{CSU}(\mathsf{h}\,(z'\,\mathsf{b}\,\bot),\ \mathsf{h}\,(\mathsf{g}\,\bot))$. Next, we apply EqHoist with the unifier $\{x' \mapsto \lambda u.\, x''\,u \approx x'''\,u,\ w \mapsto x''\,\mathsf{b},\ w' \mapsto x'''\,\mathsf{b}\} \in \mathrm{CSU}(x'\,\mathsf{b}, w \approx w')$ to the literal

created by FLUIDBOOLHOIST, effectively performing a primitive substitution. The resulting clause can superpose into $\mathsf{a} \not\approx \mathsf{b}$ with the unifier $\{x'' \mapsto \lambda u.\, u\} \in \mathrm{CSU}(x''\,\mathsf{b}, \mathsf{b})$. The two sides of the interpreted equality in the first literal can then be unified, allowing us to apply BOOLRW with the unifier $\{y \mapsto \mathsf{a},\ x''' \mapsto \lambda u.\, \mathsf{a}\} \in \mathrm{CSU}(y \mathbin{\mathbf{\approx}} y,\ \mathsf{a} \mathbin{\mathbf{\approx}} x'''\,\mathsf{b})$. Finally, applying ERES twice and FALSEELIM once yields the empty clause.

Remarkably, none of the provers that participated in the CASC-J10 competition can solve this two-clause problem within a minute. Satallax finds a proof after 72 s and LEO-II after over 7 minutes. Our new Zipperposition implementation solves it in 3 s.

The Redundancy Criterion. In first-order superposition, a clause is considered redundant if all its ground instances are entailed by $\prec$-smaller ground instances of other clauses. In essence, this will also be our definition, but we will use a different notion of ground instances and a different notion of entailment.

Given a clause C, let its *ground instances* $\mathcal{G}(C)$ be the set of all clauses of the form $C\theta$ for some substitution θ such that $C\theta$ is ground and $\mathsf{Q}_{\mathbf{\approx}}$-normal, and for all variables x occurring in C, the only Boolean green subterms of $x\theta$ are T and $\bot$. The rationale of this definition is to ensure that ground instances of the conclusion of FORALLHOIST, EXISTSHOIST, FORALLRW, and EXISTSRW inferences are smaller than the corresponding instances of their premise by property (O4).

The redundancy criterion's notion of entailment is defined via an encoding into a weaker logic, following λfSup and λSup. In this paper, the weaker logic is ground first-order logic with interpreted Booleans—the ground fragment of the logic of oSup. Its signature $(\Sigma_{\mathrm{ty}}, \Sigma_{\mathrm{GF}})$ is derived from our higher-order signature $(\Sigma_{\mathrm{ty}}, \Sigma)$ as follows. The type constructors Σ_{ty} are the same in both signatures, but $\to$ is an uninterpreted type constructor in first-order logic. For each ground instance $\mathsf{f}\langle\bar{\upsilon}\rangle : \tau_1 \to \cdots \to \tau_n \to \tau$ of a symbol $\mathsf{f} \in \Sigma$, we introduce a first-order symbol $\mathsf{f}_j^{\bar{\upsilon}} \in \Sigma_{\mathrm{GF}}$ with argument types $\bar{\tau}_j$ and result type $\tau_{j+1} \to \cdots \to \tau_n \to \tau$, for each j. Moreover, for each ground term $\lambda x.\, t$, we introduce a symbol $\mathsf{lam}_{\lambda x.\, t} \in \Sigma_{\mathrm{GF}}$ of the same type. The symbols $\bot_0$, T_0, $\neg_1$, $\wedge_2$, $\vee_2$, $\to_2$, $\mathbin{\mathbf{\approx}}_2^\tau$, and $\mathbin{\not\mathbf{\approx}}_2^\tau$ are identified with the corresponding first-order logical symbols.

We define an encoding $\mathcal{F}$ of $\mathsf{Q}_{\mathbf{\approx}}$-normal ground higher-order terms into this ground first-order logic recursively as follows: $\mathcal{F}(\mathbf{\forall}\langle\tau\rangle\,(\lambda x.\, t)) = \forall x.\, \mathcal{F}(t)$ and $\mathcal{F}(\mathbf{\exists}\langle\tau\rangle\,(\lambda x.\, t)) = \exists x.\, \mathcal{F}(t)$ for applied quantifiers; $\mathcal{F}(\lambda x.\, t) = \mathsf{lam}_{\lambda x.\, t}$ for λ-expressions; and $\mathcal{F}(\mathsf{f}\langle\bar{\upsilon}\rangle\ \bar{s}_j) = \mathsf{f}_j^{\bar{\upsilon}}(\mathcal{F}(\bar{s}_j))$ for other terms. For quantified variables, we define $\mathcal{F}(x) = x$. Here, $\mathsf{Q}_{\mathbf{\approx}}$-normality is crucial to ensure that bound variables do not occur applied or within λ-expressions. The definition of green subterms is devised such that green subterms correspond to first-order subterms via the encoding $\mathcal{F}$, with the exception of first-order subterms below quantifiers. The encoding $\mathcal{F}$ is extended to clauses by mapping each literal and each side of a literal individually. From the entailment relation $\models$ for the ground first-order logic, we derive an entailment relation $\models_{\mathcal{F}}$ on $\mathsf{Q}_{\mathbf{\approx}}$-normal ground higher-order clauses by defining $M \models_{\mathcal{F}} N$ if $\mathcal{F}(M) \models \mathcal{F}(N)$. This relation is weaker than standard higher-order entailment; for example, $\{\mathsf{f} \approx \mathsf{g}\} \not\models_{\mathcal{F}} \{\mathsf{f}\,\mathsf{a} \approx \mathsf{g}\,\mathsf{a}\}$ (because of the subscripts added by $\mathcal{F}$) and $\{\mathsf{p}\,(\lambda x.\, \mathsf{T})\} \not\models_{\mathcal{F}} \{\mathsf{p}\,(\lambda x.\, \neg\bot)\}$ (because of the lam symbols used by $\mathcal{F}$).

Using $\models_{\mathcal{F}}$, we define a clause C to be *redundant* w.r.t. a clause set N if for every $D \in \mathcal{G}(C)$, we have $\{E \in \mathcal{G}(N) \mid E \prec D\} \models_{\mathcal{F}} D$ or there exists a clause $C' \in N$ such that $C \sqsupset C'$ and $D \in \mathcal{G}(C')$. The tiebreaker $\sqsupset$ can be an arbitrary well-founded partial order

on clauses; in practice, we use a well-founded restriction of the ill-founded strict subsumption relation [6, Sect. 3.4]. We denote the set of redundant clauses w.r.t. a clause set N by $Red_{\mathrm{C}}(N)$. Note that $\models_{\mathcal{F}}$ is weak enough to ensure that the ArgCong inference rule and axiom (Ext) are not immediately redundant and can fulfill their purpose.

For first-order superposition, an inference is considered redundant if for each of its ground instances, a premise is redundant or the conclusion is entailed by clauses smaller than the main premise. For most inference rules, our definition follows this idea, using $\models_{\mathcal{F}}$ for entailment; other rules need nonstandard notions of ground instances and redundancy. The definition of inference redundancy presented below is simpler than the more sophisticated notion in our technical report. Nonetheless, the redundant inferences below are a strict subset of the redundant inferences of our report and thus completeness also holds using the notion below. For the few prover optimizations based on inference redundancy that we know about (e.g., simultaneous superposition [4]), the following criterion suffices.

For Sup, ERes, EFact, BoolHoist, FalseElim, EqHoist, NeqHoist, and BoolRw, we define ground instances as usual: *Ground instances* are all inferences obtained by applying a grounding substitution to premises and conclusion such that the result adheres to the conditions of the given rule w.r.t. selection functions that select literals and subterms as in the original premise. For FluidSup and FluidBoolHoist, we define ground instances in the same way except that we require that ground instances adhere to the conditions of Sup or BoolHoist, respectively. For ForallRw, ExistsRw, ForallHoist, ExistsHoist, which do not have ground instances in the sense above, we define a *ground instance* as any inference that is obtained by applying the unifier σ to the premise and then applying a grounding substitution to premise and conclusion, regardless of whether the resulting inference is an inference of our calculus.

For all rules except FluidLoobHoist and ArgCong, we define an inference to be *redundant* w.r.t. a clause set N if for each ground instance ι, a premise of ι is redundant w.r.t. $\mathcal{G}(N)$ or the conclusion of ι is entailed w.r.t. $\models_{\mathcal{F}}$ by clauses from $\mathcal{G}(N)$ that are smaller than the main (i.e., rightmost) premise of ι. For the rules FluidLoobHoist and ArgCong, as well as axioms (Ext) and (Choice)—viewed as premiseless inferences—we define an inference to be *redundant* w.r.t. a clause set N if all ground instances of its conclusion are contained in $\mathcal{G}(N)$ or redundant w.r.t. $\mathcal{G}(N)$. We denote the set of redundant inferences w.r.t. N by $Red_{\mathrm{I}}(N)$.

Simplification Rules. Our redundancy criterion is strong enough to support counterparts of most simplification rules implemented in Schulz's first-order E [25, Sect. 2.3.1 and 2.3.2]. Deletion of duplicated literals, deletion of resolved literals, syntactic tautology deletion, negative simplify-reflect, and clause subsumption adhere to our redundancy criterion. Positive simplify-reflect, equality subsumption, and rewriting (demodulation) of positive and negative literals are supported if they are applied on green subterms or on other subterms that are encoded into first-order subterms by $\mathcal{G}$ and $\mathcal{F}$. Semantic tautology deletion can be applied as well, using $\models_{\mathcal{F}}$; moreover, for positive literals, the rewriting clause must be smaller than the rewritten clause.

Under some circumstances, inference rules can be applied as simplifications. The FalseElim and BoolRw rules can be applied as a simplification if σ is the identity. If the head of u is $\forall$, ForallHoist and ForallRw can both be applied and, together,

serve as one simplification rule. The same holds for ExistsHoist and ExistsRw if the head of u is $\exists$. For all of these rules, the eligibility conditions can be ignored.

Clausification. Like oSup, our calculus does not require the input problem to be clausified during the preprocessing, and it supports higher-order analogues of the three inprocessing clausification methods introduced by Nummelin et al. *Inner delayed clausification* relies on our core calculus rules to destruct logical symbols. *Outer delayed clausification* adds the following clausification rules to the calculus:

$$\frac{s \approx \mathsf{T} \vee C}{oc(s, C)}\,\textsc{PosOuterClaus} \qquad \frac{s \approx \bot \vee C}{oc(\neg s, C)}\,\textsc{NegOuterClaus}$$

$$\frac{s \approx t \vee C}{s \approx \bot \vee t \approx \mathsf{T} \vee C \quad s \approx \mathsf{T} \vee t \approx \bot \vee C}\,\textsc{EqOuterClaus}$$

$$\frac{s \not\approx t \vee C}{s \approx \bot \vee t \approx \bot \vee C \quad s \approx \mathsf{T} \vee t \approx \mathsf{T} \vee C}\,\textsc{NeqOuterClaus}$$

The double bars identify simplification rules (i.e., the conclusions make the premise redundant and can replace it). The first two rules require that s has a logical symbol as its head, whereas the last two require that s and t are Boolean terms other than T and $\bot$. The function oc distributes the logical symbols over the clause C—e.g., $oc(s \rightarrow t, C) = \{s \approx \bot \vee t \approx \mathsf{T} \vee C\}$, and $oc(\neg(s \vee t), C) = \{s \approx \bot \vee C, t \approx \bot \vee C\}$. It is easy to check that our redundancy criterion allows us to replace the premise of the OuterClaus rules with their conclusion. Nonetheless, we apply EqOuterClaus and NeqOuterClaus as inferences because the premises might be useful in their original form.

Besides the two delayed clausification methods, a third inprocessing clausification method is *immediate* clausification. This clausifies the input problem's outer Boolean structure in one swoop, resulting in a set of higher-order clauses. If unclausified Boolean terms rise to the top during saturation, the same algorithm is run to clausify them.

Unlike delayed clausification, immediate clausification is a black box and is unaware of the proof state other than the Boolean term it is applied to. Delayed clausification, on the other hand, clausifies the term step by step, allowing us to interleave clausification with the strong simplification machinery of superposition provers. It is especially powerful in higher-order contexts: Examples such as $y\,\mathsf{p}\,\mathsf{q} \not\approx (\mathsf{p} \vee \mathsf{q})$ can be refuted directly by equality resolution, rather than via more explosive rules on the clausified form.

4 Refutational Completeness

Our calculus is dynamically refutationally complete for problems in $\mathsf{Q}_{\approx}$-normal form. The full proof can be found in our technical report [8].

Theorem 3 (Dynamic refutational completeness). *Let $(N_i)_i$ be a derivation—i.e., $N_i \setminus N_{i+1} \subseteq \mathit{Red}_{\mathrm{C}}(N_{i+1})$ for all i. Let N_0 be $\mathsf{Q}_{\approx}$-normal and such that $N_0 \models \bot$. Moreover, assume that $(N_i)_i$ is fair—i.e., all inferences from clauses in the limit inferior $\bigcup_i \bigcap_{j \geq i} N_j$ are contained in $\bigcup_i \mathit{Red}_{\mathrm{I}}(N_i)$. Then we have $\bot \in N_i$ for some i.*

Following the completeness proof of λSup, our proof is structured in three levels of logics. For each, we define a calculus and show that it is refutationally complete: ground monomorphic first-order logic with an interpreted Boolean type (GF); the $\mathsf{Q}_{\approx}$-normal ground fragment of higher-order logic (GH); and higher-order logic (H).

The logic of the GF level is the ground fragment of oSup's logic. The GF calculus is a ground version of oSup, which Nummelin et al. showed refutationally complete. It consists of ground first-order equivalents of our rules, excluding ARGCONG, FLUIDBOOLHOIST, and FLUIDLOOBHOIST, which are specific to higher-order logic. The counterparts to FORALLHOIST and EXISTSHOIST enumerate ground terms instead of producing free variables, to stay within the ground fragment. For compatibility with the nonground level, the conclusions of FORALLRW and EXISTSRW cannot contain concrete Skolem functions. Instead, the GF calculus is parameterized by a witness function that can assign an arbitrary term to each occurrence of a quantifier in a clause. This witness function is used to retrieve the Skolem terms in the GF equivalents of FORALLRW and EXISTSRW.

On the next level, the GH calculus includes inference rules isomorphic to the GF rules, transferred to higher-order logic via $\mathcal{F}^{-1}$. Moreover, it contains an ARGCONG variant that enumerates ground terms instead of introducing fresh variables, as well as rules enumerating ground instances of axioms (EXT) and (CHOICE). We prove refutational completeness of the GH calculus by constructing a higher-order interpretation based on the model constructed for the completeness proof of the GF level. This proof step is analogous to the corresponding step in λSup's proof, but we must also consider $\mathsf{Q}_{\approx}$-normality and the logical symbols.

To lift completeness to the H level, we use the saturation framework of Waldmann et al. [31]. The main proof obligation it leaves us to show is that nonredundant GH inferences can be lifted to corresponding nonground H inferences. For this lifting, we must choose a suitable GH witness function and appropriate GH selection functions for literals and Boolean subterms, given a saturated clause set at the H level and the H selection functions. Then the saturation framework guarantees static refutational completeness w.r.t. Herbrand entailment, which is the entailment relation induced by the grounding function $\mathcal{G}$. We then show that this implies dynamic refutational completeness w.r.t. $\models$ for $\mathsf{Q}_{\approx}$-normal initial clause sets.

5 Implementation

We implemented our calculus in the Zipperposition prover [14], whose OCaml source code makes it convenient to prototype calculus extensions. Except for the presence of axioms (EXT) and (CHOICE), the new code gracefully extends Zipperposition's implementation of oSup in the sense that oλSup coincides with oSup on first-order problems. The same cannot be said w.r.t. λSup on Boolean-free problems because of the FLUIDBOOLHOIST and FLUIDLOOBHOIST rules, which are triggered by any applied variable. From the implementation of λSup, we inherit the given clause procedure, which supports infinitely branching inferences, as well as calculus extensions and heuristics [28]. From the implementation of oSup, we inherit the simplification rule BOOLSIMP, a mainstay of our Boolean simplification machinery.

As in the implementation of λSup, we approximate fluid terms as terms that are either nonground λ-expressions or terms of the form $x\,\bar{s}_n$ with $n > 0$. Two slight, accidental discrepancies are that we also count variable occurrences below quantifiers as deep and perform EFACT inferences even if the maximal literal is selected. Since we expect FLUIDBOOLHOIST and FLUIDLOOBHOIST to be highly explosive, we penalize them and all of their offspring. In addition to various λSup extensions [6, Sect. 5], we also use all the rules for Boolean reasoning described by Vukmirović and Nummelin [30] except for the BOOLEF rules.

6 Evaluation

We evaluate the calculus implementation in Zipperposition and compare it with other higher-order provers. Our experiments were performed on StarExec Miami servers equipped with Intel Xeon E5-2620 v4 CPUs clocked at 2.10 GHz. We used all 2606 TH0 theorems from the TPTP 7.3.0 library [27] and 1253 "Judgment Day" problems [12] generated using Sledgehammer (SH) [24] as our benchmark set. An archive containing the benchmarks and the raw evaluation results is publicly available [5].

Calculus Evaluation. In this first part, we evaluate selected parameters of Zipperposition by varying only the studied parameter in a fixed well-performing configuration. This base configuration disables axioms (CHOICE) and (EXT) and the FLUID- rules. It uses the unification procedure of Vukmirović et al. [29] in its complete variant—i.e., the variant that produces a complete set of unifiers. It uses none of the early Boolean rules described by Vukmirović and Nummelin [30]. The preprocessor $\mathsf{Q}_{\approx}$ is disabled as well. All of the completeness-preserving simplification rules listed in Sect. 3 are enabled. The configuration uses immediate clausification. We set the CPU time limit to 30 s in all three experiments.

In the first experiment, we assess the overhead incurred by the FLUID- rules. These rules unify with a term whose head is a fresh variable. Thus, we expected that they needed to be tightly controlled to achieve good performance. To test our hypothesis, we simultaneously modified the parameters of these three rules. In Figure 1, the *off* mode simply disables the rules, the *pragmatic* mode uses a terminating incomplete unification algorithm (the pragmatic variant of Vukmirović et al. [29]), and the *complete* mode uses a complete unification algorithm. The results show that disabling FLUID- rules altogether achieves the best performance. However, on TPTP problems, *complete* finds 35 proofs not found by *off*, and *pragmatic* finds 22 proofs not found by *off*. On Sledgehammer benchmarks, this effect is much weaker, likely because the Sledgehammer benchmarks require less higher-order reasoning: *complete* finds only one new proof over *off*, and *pragmatic* finds only four.

In the second experiment, we explore the clausification methods introduced at the end of Sect. 3: *inner* delayed clausification, *outer* delayed clausification, and *immediate* clausification. The modes *inner* and *outer* employ oSup's RENAME rule, which renames Boolean terms headed by logical symbols using a Tseitin-like transformation if they occur at least four times in the proof state. Vukmirović and Nummelin [30] observed that *outer* clausification can greatly help prove higher-order problems, and we expected

	off	*pragmatic*	*complete*
TPTP	**1642**	1591	1619
SH	**467**	431	437

Fig. 1. Evaluation of FLUID- rules

	inner	*outer*	*immediate*
TPTP	1323	**1670**	1642
SH	406	**470**	467

Fig. 2. Evaluation of clausification method

	off	$p = 64$	$p = 16$	$p = 4$	$p = 1$
TPTP	**1642**	1617	1613	1615	1594
SH	**467**	458	458	459	445

Fig. 3. Evaluation of axiom (CHOICE)

	TPTP	ofSH	SH
CVC4 1.8	1796	680	619
Leo-III 1.5.2	2104	681	621
Vampire 4.5	2131	692	681
Satallax 3.5	2162	573	587
Zip (CASC-J10)	2301	**734**	**736**
New Zip	**2320**	724	720

Fig. 4. Evaluation of all competitive higher-order provers

it to perform well for our calculus, too. The results are shown in Figure 2. The results confirm our hypothesis: The *outer* mode outperforms *immediate* on both TPTP and Sledgehammer benchmarks. The *inner* mode performs worst, but on Sledgehammer benchmarks, it proves 17 problems beyond the reach of the other two. Interestingly, several of these problems contain axioms of the form $\phi \to \psi$, and applying superposition and demodulation to these axioms is preferable to clausifying them.

In the third experiment, we investigate the effect of axiom (CHOICE), which is necessary to achieve refutational completeness. To evaluate (CHOICE), we either disabled it in a configuration labeled *off* or set the axiom's penalty p to different values. In Zipperposition, penalties are propagated through inference and simplification rules and are used to increase the heuristic weight of clauses, postponing the selection of penalized clauses. The results are shown in Figure 3. As expected, disabling (CHOICE), or at least penalizing it heavily, improves performance. Yet enabling (CHOICE) can be crucial: For 19 TPTP problems, the proofs are found when (CHOICE) is enabled and $p = 4$, but not when the rule is disabled. On Sledgehammer problems, this effect is weaker, with only two new problems proved for $p = 4$.

Prover Comparison. In this second part, we compare Zipperposition's performance with other higher-order provers. Like at CASC-J10, the wall-clock time limit was 120 s, the CPU time limit was 960 s, and the provers were run on StarExec Miami. We used the following versions of all systems that took part in the THF division: CVC4 1.8 [3], Leo-III 1.5.2 [26], Satallax 3.5 [13], and Vampire 4.5 [11]. The developers of Vampire have informed us that its higher-order schedule is optimized for running on a single core. As a result, the prover suffers some degradation of performance when running on multiple cores. We evaluate both the version of Zipperposition that took part in CASC-J10 (*Zip*) and the updated version of Zipperposition that supports our new calculus (*New Zip*). Zip's portfolio of prover configurations is based on λSup and techniques described by Vukmirović and Nummelin [30]. New Zip's portfolio is specially designed for our

new calculus and optimized for TPTP problems. To assess the performance of Boolean reasoning, we used Sledgehammer benchmarks generated both with native Booleans (SH) and with an encoding into Boolean-free higher-order logic (ofSH). For technical reasons, the encoding also performs λ-lifting, but this minor transformation should have little impact on results [6, Sect. 7].

The results are shown in Figure 4. The two versions of Zipperposition are ahead of all other provers on both benchmark sets. This shows that, with thorough parameter tuning, higher-order superposition outperforms tableaux, which had been the state of the art in higher-order reasoning for a decade. The updated version of New Zip beats Zip on TPTP problems but lags behind Zip on Sledgehammer benchmarks as we have yet to further explore more general heuristics that work well with our new calculus. The Sledgehammer benchmarks fail to demonstrate the superiority of native Booleans reasoning compared with an encoding, and in fact CVC4 and Leo-III perform dramatically better on the encoded Boolean problems, suggesting that there is room for tuning.

7 Conclusion

We have created a superposition calculus for higher-order logic that is refutationally complete. Most of the key ideas have been developed in previous work by us and colleagues, but combining them in the right way has been challenging. A key idea was to $\mathsf{Q}_{\approx}$-normalize away inconvenient terms.

Unlike earlier refutationally complete calculi for full higher-order logic based on resolution or paramodulation, our calculus employs a term order, which restricts the proof search, and a redundancy criterion, which can be used to add various simplification rules while keeping refutational completeness. These two mechanisms are undoubtedly major factors in the success of first-order superposition, and it is very fortunate that we could incorporate both in a higher-order calculus. An alternative calculus with the same two mechanisms could be achieved by combining oSup with Bhayat and Reger's combinatory superposition [10]. The article on λSup [6, Sect. 8] discusses related work in more detail.

The evaluation results show that our calculus is an excellent basis for higher-order theorem proving. In future work, we want to experiment further with the different parameters of the calculus (for example, with Boolean subterm selection heuristics) and implement it in a state-of-the-art prover such as E.

Acknowledgment. Uwe Waldmann provided advice and carefully checked the completeness proof. Visa Nummelin led the design of the oSup calculus. Simon Cruanes helped us with the implementation. Martin Desharnais generated the Sledgehammer benchmarks. Christoph Benzmüller, Ahmed Bhayat, Mathias Fleury, Herman Geuvers, Giles Reger, Alexander Steen, Mark Summerfield, Geoff Sutcliffe, and the anonymous reviewers helped us in various ways. 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). Blanchette's research has received funding from the Netherlands Organization for Scientific Research (NWO) under the Vidi program (project No. 016.Vidi.189.037, Lean Forward).

References

[1] Andrews, P.B.: On connections and higher-order logic. J. Autom. Reason. **5**(3), 257–291 (1989)

[2] Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. **4**(3), 217–247 (1994)

[3] Barrett, C.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: CAV. LNCS, vol. 6806, pp. 171–177. Springer (2011)

[4] Benanav, D.: Simultaneous paramodulation. In: Stickel, M.E. (ed.) CADE-10. LNCS, vol. 449, pp. 442–455. Springer (1990)

[5] Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P.: Superposition for full higher-order logic (supplementary material), https://doi.org/10.5281/zenodo.4534759

[6] Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P., Waldmann, U.: Superposition with lambdas, accepted in J. Autom. Reason. Preprint at https://arxiv.org/abs/2102.00453v1 (2021)

[7] Bentkamp, A., Blanchette, J.C., Cruanes, S., Waldmann, U.: Superposition for lambda-free higher-order logic. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS, vol. 10900, pp. 28–46. Springer (2018)

[8] Bentkamp, A., Blanchette, J.C., Tourret, S., Vukmirović, P.: Superposition for full higher-order logic (technical report). Technical report (2021), https://matryoshka-project.github.io/pubs/hosup_report.pdf

[9] Benzmüller, C., Paulson, L.C., Theiss, F., Fietzke, A.: LEO-II—A cooperative automatic theorem prover for higher-order logic. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 162–170. Springer (2008)

[10] Bhayat, A., Reger, G.: Set of support for higher-order reasoning. In: Konev, B., Urban, J., Rümmer, P. (eds.) PAAR-2018. CEUR Workshop Proceedings, vol. 2162, pp. 2–16. CEUR-WS.org (2018)

[11] Bhayat, A., Reger, G.: A combinator-based superposition calculus for higher-order logic. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020, Part I. LNCS, vol. 12166, pp. 278–296. Springer (2020)

[12] Böhme, S., Nipkow, T.: Sledgehammer: Judgement Day. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 107–121. Springer (2010)

[13] Brown, C.E.: Satallax: An automatic higher-order prover. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 111–117. Springer (2012)

[14] Cruanes, S.: Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond. Ph.D. thesis, École polytechnique (2015)

[15] Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer (2002)

[16] Ganzinger, H., Stuber, J.: Superposition with equivalence reasoning and delayed clause normal form transformation. Information and Computation **199**(1–2), 3–23 (2005)

[17] Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)

[18] Huet, G.P.: A mechanization of type theory. In: Nilsson, N.J. (ed.) IJCAI-73. pp. 139–146. William Kaufmann (1973)

[19] Jensen, D.C., Pietrzykowski, T.: Mechanizing ω-order type theory through unification. Theor. Comput. Sci. **3**(2), 123–171 (1976)

[20] Kaliszyk, C., Sutcliffe, G., Rabe, F.: TH1: The TPTP typed higher-order form with rank-1 polymorphism. In: Fontaine, P., Schulz, S., Urban, J. (eds.) PAAR-2016. CEUR Workshop Proceedings, vol. 1635, pp. 41–55. CEUR-WS.org (2016)

[21] Kotelnikov, E., Kovács, L., Suda, M., Voronkov, A.: A clausal normal form translation for FOOL. In: Benzmüller, C., Sutcliffe, G., Rojas, R. (eds.) GCAI 2016. EPiC, vol. 41, pp. 53–71. EasyChair (2016)

[22] Ludwig, M., Waldmann, U.: An extension of the Knuth-Bendix ordering with LPO-like properties. In: Dershowitz, N., Voronkov, A. (eds.) LPAR-14. LNCS, vol. 4790, pp. 348–362. Springer (2007)
[23] Nummelin, V., Bentkamp, A., Tourret, S., Vukmirović, P.: Superposition with first-class Booleans and inprocessing clausification. In: Platzer, A., Sutcliffe, G. (eds.) CADE-28. LNCS, Springer (2021)
[24] Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) IWIL-2010. EPiC, vol. 2, pp. 1–11. EasyChair (2012)
[25] Schulz, S.: E - a brainiac theorem prover. AI Commun. **15**(2-3), 111–126 (2002)
[26] Steen, A., Benzmüller, C.: The higher-order prover Leo-III. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS, vol. 10900, pp. 108–116. Springer (2018)
[27] 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)
[28] Vukmirović, P., Bentkamp, A., Blanchette, J., Cruanes, S., Nummelin, V., Tourret, S.: Making higher-order superposition work. In: Platzer, A., Sutcliffe, G. (eds.) CADE-28. LNCS, Springer (2021)
[29] Vukmirović, P., Bentkamp, A., Nummelin, V.: Efficient full higher-order unification. In: Ariola, Z.M. (ed.) FSCD 2020. LIPIcs, vol. 167, pp. 5:1–5:17. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2020)
[30] Vukmirović, P., Nummelin, V.: Boolean reasoning in a higher-order superposition prover. In: PAAR-2020. CEUR Workshop Proceedings, vol. 2752, pp. 148–166. CEUR-WS.org (2020)
[31] Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020, Part I. LNCS, vol. 12166, pp. 316–334. Springer (2020)