





Superposition with First-Class Booleans and Inprocessing Clausification (Technical Report)

Visa Nummelin¹, Alexander Bentkamp¹,
Sophie Tourret^{2,3}, and Petar Vukmirović¹

¹ Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
visa.nummelin@vu.nl a.bentkamp@vu.nl 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 present a complete superposition calculus for first-order logic with an interpreted Boolean type. Our motivation is to lay the foundation for refutationally complete calculi in more expressive logics with Booleans, such as higher-order logic, and to make superposition work efficiently on problems that would be obfuscated when using clausification as preprocessing. Working directly on formulas, our calculus avoids the costly axiomatic encoding of the theory of Booleans into first-order logic and offers various ways to interleave clausification with other derivation steps. We evaluate our calculus using the Zipperposition theorem prover, and observe that, with no tuning of parameters, our approach is on a par with the state-of-the-art approach.

1 Introduction

Superposition is a calculus for equational first-order logic that works on problems given in clausal normal form. Its immense success made preprocessing clausification a predominant mechanism in modern automatic theorem proving. However, this preprocessing is not without drawbacks. Clausification can transform simple problems, such as $s \rightarrow s$ where s is a large formula, in a way that hides its original simplicity from the superposition calculus. Ganzinger and Stuber’s superposition-like calculus [13] operates on clauses that contain formulas as well as terms and replaces preprocessing clausification by inprocessing—meaning processing during the operation of the calculus itself. Inprocessing clausification allows superposition’s powerful simplification engine to work on formulas. For example, unit equalities can rewrite formulas s and t in $s \leftrightarrow t$ before clausification duplicates the occurrences into $s \rightarrow t$ and $t \rightarrow s$. Whole formulas rather than simple literals can be removed by rules such as subsumption resolution [5].

Another issue with Boolean reasoning in the standard superposition calculus is that, in first-order logic, formulas cannot appear inside terms although this is

often desirable for problems coming from software verifiers or proof assistants. Instead, authors of such tools need to resort to translations. Kotelnikov et al. studied effects of these translations in detail. They showed that simple axioms such as the domain cardinality axiom for Booleans ($\forall(x : o). x \approx \top \vee x \approx \perp$) can severely slow down superposition provers. To support more efficient reasoning on problems with first-class Booleans, they describe the FOOL logic, which admits functions that take arguments of Boolean type and quantification over Booleans. They further describe two approaches to reason in FOOL: The first one [17] requires an additional rule in the superposition calculus, whereas the second one [16] is completely based on preprocessing.

Our calculus combines complementary advantages of Ganzinger and Stuber’s and of Kotelnikov et al.’s work. Following Kotelnikov et al., our logic (Sect. 2) is similar to FOOL and supports nesting formulas inside terms, as well as quantifying over Booleans. Following Ganzinger and Stuber, our calculus (Sect. 3) reasons with formulas and supports inprocessing clausification.

Our calculus also extends the two approaches. To reduce the number of possible inferences, we generalize Ganzinger and Stuber’s Boolean selection functions, which allow us to restrict the Boolean subterms in a clause on which inferences can be performed. The term order requirements of our calculus are less restrictive than Ganzinger and Stuber’s. In addition to the lexicographic path order (LPO), we also support the Knuth-Bendix order (KBO) [15], which is known to work better with superposition in practice.

Our proof of refutational completeness (Sect. 5) lays the foundation for complete calculi in more complex logics with Booleans. Indeed, Bentkamp et al. [8] devised a refutationally complete calculus for higher-order logic and prove it complete by transforming the model constructed in our proof into a higher-order model. Our completeness theorem incorporates a powerful redundancy criterion that allows for a variety of inprocessing clausification methods (Sect. 6).

We implemented our approach in the Zipperposition theorem prover (Sect. 7) and evaluated it on thousands of problems that target our logic ranging from TPTP to SMT-LIB to Sledgehammer-generated benchmarks (Sect. 8). Without fine-tuning, our new calculus performs as well as known techniques. Exploring the strategic choices that our calculus opens should lead to further performance improvements. In addition, we corroborate the claims of Ganzinger and Stuber concerning applicability of formula-based superposition reasoning: We find a set of 17 TPTP problems (out of 1000 randomly selected) that Zipperposition can solve only using the techniques described in this paper.

2 Logic

Our logic is a first-order logic with an interpreted Boolean type. It is essentially identical to the UF logic of SMT-LIB [6], including the Core theory, but without if-then-else and let expressions, which can be supported through simple translations. It also closely resembles Kotelnikov et al.’s FOOL [17], which additionally supports if-then-else and let expressions.

Throughout the report, we write tuples (a_1, \dots, a_n) as \bar{a}_n or \bar{a} .

Syntax We fix a set Σ_{ty} of types. We require that Σ_{ty} contains the type o of Booleans. A type declaration is a tuple of types $(\bar{\tau}_n, v) \in \Sigma_{\text{ty}}^{n+1}$, written as $\bar{\tau}_n \rightarrow v$. If $n = 0$, we simply write v for $() \rightarrow v$.

We fix a set Σ of (function) symbols f , each associated with a type declaration $\bar{\tau}_n \rightarrow v$, written as $f : \bar{\tau}_n \rightarrow v$ or f , and a countably infinite set \mathcal{V} of 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 that Σ contains the logical symbols $\mathbf{T}, \mathbf{\perp} : o$; $\neg : o \rightarrow o$; $\mathbf{\wedge}, \mathbf{\vee}, \rightarrow : (o \times o) \rightarrow o$; and the overloaded symbols $\approx, \not\approx : (\tau \times \tau) \rightarrow o$ for each $\tau \in \Sigma_{\text{ty}}$. The logical symbols are printed in bold to distinguish them from the notation used for clauses below. We use infix notation for the binary logical symbols. Moreover, we require that there is at least one nullary symbol for each type to avoid empty Herbrand universes. A *formula* is a term of Boolean type.

A signature is a pair $(\Sigma_{\text{ty}}, \Sigma)$. The set of *terms* is defined inductively as follows. Every $x : \tau \in \mathcal{V}$ is a term of type τ . If $f : \bar{\tau}_n \rightarrow v \in \Sigma$ and $\bar{t}_n : \bar{\tau}_n$ is a tuple of terms, then the application $f(\bar{t}_n)$ (or simply f if $n = 0$) is a term of type v . If $x : \tau$ and $t : o$, then the quantified terms $\forall x. t$ and $\exists x. t$ are terms of Boolean type. We view quantified terms modulo α -renaming. We also write these as $\forall p$ and $\exists p$, where $ps = \{x \mapsto s\}t$ is a term to term function of the given form.

The root of a term is x if the term is a variable x ; it is f if the term is an application $f(\bar{t}_n)$; and it is \forall or \exists if the term is a quantified term $\forall x. t$ or $\exists x. t$.

A variable occurrence is *free* in a term if it is not bound by \forall or \exists . A term is *ground* if it contains no free variables.

A literal $s \approx t$ is an equation $s \approx t$ or a disequation $s \not\approx t$. Unlike terms constructed using the function symbols \approx and $\not\approx$, literals are unoriented—i.e., $s \approx t$ and $t \approx s$ denote the same literal. A clause $L_1 \vee \dots \vee L_n$ is a finite multiset of literals L_j . The empty clause is written as $\mathbf{\perp}$. Terms t of Boolean type are not literals. They must be encoded as $t \approx \mathbf{T}$ and $t \approx \mathbf{\perp}$, which we call *predicate literals*. Both are considered positive literals because they are equations, not disequations.

We have considered excluding negative literals $s \not\approx t$ by encoding them as $(s \approx t) \approx \mathbf{\perp}$, following Ganzinger and Stuber. However, this approach requires an additional term order condition to make the conclusion of equality factoring small enough, excluding KBO. To support both KBO and LPO, we allow negative literals. Regardless, our simplification mechanism will allow us to simplify negative literals of the form $t \not\approx \mathbf{\perp}$ and $t \not\approx \mathbf{T}$ into $t \approx \mathbf{T}$ and $t \approx \mathbf{\perp}$, respectively, thereby eliminating redundant representations of predicate literals.

Subterms and positions are inductively defined as follows. A position in a term is a tuple of natural numbers. For any term t , the empty position ε is a position of t , and t is the subterm of t at position ε . If t is the subterm of u_i at position p , then $i.p$ is a position of $f(\bar{u})$, and t is the subterm of $f(\bar{u})$ at position $i.p$. If t is the subterm of u at position p , then $1.p$ is a position of $\forall x. u$ and of $\exists x. u$, and t is the subterm of $\forall x. u$ and of $\exists x. u$ at position $1.p$.

For positions in clauses, natural numbers are not appropriate because clauses and literals are unordered. A position in a clause C is a tuple $L.s.p$ where $L = s \approx t$ is a literal in C and p is a position in s . The subterm of C at position $L.s.p$ is the subterm of s at position p . We write $s|_p$ to denote the subterm at position p in s . We write $s[u]_p$ to denote a term s with the subterm u at position p and call $s[\]_p$ a *context*; the position p may be omitted in this notation. A position p is *at or below* a position q if q is a prefix of p . A position p is *below* a position q if q is a proper prefix of p .

Substitutions are defined as usual in first-order logic and they rename quantified variables to avoid capture.

Semantics An interpretation $\mathcal{I} = (\mathcal{U}, \mathcal{J})$ is a pair, consisting of a universe \mathcal{U}_τ for each type $\tau \in \Sigma_{\text{ty}}$ and an *interpretation function* \mathcal{J} , which associates with each symbol $f : \bar{\tau} \rightarrow v$ and universe elements $\bar{a} \in \mathcal{U}_{\bar{\tau}}$ a universe element $\mathcal{J}(f)(\bar{a}) \in \mathcal{U}_v$. We require that $\mathcal{U}_o = \{0, 1\}$; $\mathcal{J}(\mathbf{T}) = 1$; $\mathcal{J}(\mathbf{F}) = 0$; $\mathcal{J}(\neg)(a) = 1 - a$; $\mathcal{J}(\wedge)(a, b) = \min\{a, b\}$; $\mathcal{J}(\vee)(a, b) = \max\{a, b\}$; $\mathcal{J}(\rightarrow)(a, b) = \max\{1 - a, b\}$; $\mathcal{J}(\approx)(c, d) = 1$ if $c = d$ and 0 otherwise; $\mathcal{J}(\neq)(c, d) = 0$ if $c = d$ and 1 otherwise; for all $a, b \in \mathcal{U}_o$ and $c, d \in \mathcal{U}_\tau$ where $\tau \in \Sigma_{\text{ty}}$.

A *valuation* is a function assigning an element $\xi(x) \in \mathcal{U}_\tau$ to each variable $x : \tau$. For an interpretation \mathcal{I} and a valuation ξ , the denotation of a term is inductively defined as $\llbracket x \rrbracket_{\mathcal{I}}^\xi = \xi(x)$ for a variable $x \in \mathcal{V}$; $\llbracket f(\bar{t}) \rrbracket_{\mathcal{I}}^\xi = \mathcal{J}(f)(\llbracket \bar{t} \rrbracket_{\mathcal{I}}^\xi)$ for a symbol $f \in \Sigma$ and appropriately typed terms \bar{t} ; and $\llbracket \forall x. t \rrbracket_{\mathcal{I}}^\xi = \min\{\llbracket t \rrbracket_{\mathcal{I}}^{\xi[x \mapsto a]} \mid a \in \mathcal{U}_\tau\}$, $\llbracket \exists x. t \rrbracket_{\mathcal{I}}^\xi = \max\{\llbracket t \rrbracket_{\mathcal{I}}^{\xi[x \mapsto a]} \mid a \in \mathcal{U}_\tau\}$ for a variable $x : \tau \in \mathcal{V}$ and a term $t : o$. For ground terms t , the denotation does not depend on the choice of the valuation ξ , which is why we sometimes write $\llbracket t \rrbracket_{\mathcal{I}}$ for $\llbracket t \rrbracket_{\mathcal{I}}^\xi$.

Given an interpretation \mathcal{I} and a valuation ξ , an equation $s \approx t$ is true if $\llbracket s \rrbracket_{\mathcal{I}}^\xi$ and $\llbracket t \rrbracket_{\mathcal{I}}^\xi$ are equal and it is false otherwise. A disequation $s \neq t$ 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. An 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 .

Some of our calculus rules introduce Skolem symbols, which are intended to be interpreted as witnesses for existentially quantified terms. Still, our semantics treats them as uninterpreted symbols. To achieve a satisfiability-preserving calculus, we assume that these symbols do not occur in the input problem. More precisely, we inductively extend the signature of the input problem by a symbol $\text{sk}_{\forall \bar{y}. \exists z. t} : \bar{\tau} \rightarrow v$ for each term of the form $\exists z. t$ over the extended signature, where v is the type of z and $\bar{y} : \bar{\tau}$ are the free variables occurring in $\exists z. t$, in order of first appearance.

3 The Calculus

Following standard superposition, our calculus employs a term order and a literal selection function to restrict the search space. To accommodate for quantified Boolean terms, we impose additional requirements on the term order. To support flexible reasoning with Boolean subterms, in addition to the literal selection function, we introduce a Boolean subterm selection function.

3.1 Term Order

The calculus is parameterized by a strict well-founded order \succ on ground terms that fulfills: (O1) $u \succ \perp \succ \top$ for any term u that is not \top or \perp ; (O2) $\forall x. t \succ \{x \mapsto u\}t$ and $\exists x. t \succ \{x \mapsto u\}t$ for any term u whose only Boolean subterms are \top and \perp ; (O3) subterm property; (O4) compatibility with contexts (not necessarily below \forall and \exists); (O5) totality. The order is extended to literals, clauses, and nonground terms as usual [3]. The nonground order then also enjoys (O6) stability under grounding substitutions.

Ganzinger and Stuber's term order restrictions are similar but incompatible with KBO. Using an encoding of our terms into untyped first-order logic we describe how both LPO and the transfinite variant of KBO [19] can satisfy conditions (O1)–(O6).

Our encoding represents bound variables by De Bruijn indices, which become new constant symbols db_n for $n \in \mathbb{N}$. Quantifiers are represented by two new unary function symbols, also denoted by \forall and \exists . All other symbols are simply identified with their untyped counterpart. Regardless of symbol precedence or symbol weights, KBO and LPO enjoy properties (O3)–(O6) when applied to the encoded terms because they are ground-total simplification orders. They are even compatible with contexts below quantifiers.

To satisfy (O1) and (O2), let the precedence for LPO be $\top < \perp < f < \forall < \exists < \text{db}_0 < \text{db}_1 < \dots$ where f is any other symbol. For KBO, we can use the same symbol precedence and a symbol weight function \mathcal{W} that assigns each symbol ordinal weights (of the form $\omega a + b$ with $a, b \in \mathbb{N}$), where $\mathcal{W}(\top) = \mathcal{W}(\perp) = 1$, $\mathcal{W}(\forall) = \mathcal{W}(\exists) = \omega$, and $\mathcal{W}(f) \in \mathbb{N} \setminus \{0\}$ for any other symbol f .

It is easy to check that for both orders (O1) is satisfied. As replacing variable x with u from (O2) in $\forall x. t$ (or $\exists x. t$) only decreases de Bruijn indices or replaces them by $u \prec \text{db}_0$, LPO satisfies (O2). A term u without quantified subterms has a weight $< \omega$. Hence, the ω -component of $\{x \mapsto u\}t$ is smaller than that of $\forall x. t$ for 1. Consequently, (O2) holds in case of KBO.

3.2 Selection and Eligibility

Following an idea of Ganzinger and Stuber, we parameterize our calculus with two selection functions: one selecting literals and one selecting Boolean subterms.

Definition 1 (Selection functions). The calculus is parameterized by a literal selection function $FLSel$ and a Boolean subterm selection function $FBSel$. The function $FLSel$ maps each clause to a subset of its literals. The selection function $FBSel$ maps each clause C to a subset of the positions of Boolean subterms in C . The literals $FLSel(C)$ and the positions $FBSel(C)$ are *selected* in C . The following selection restrictions apply: (S1) A literal can only be selected if it is negative or of the form $s \approx \perp$. (S2) A Boolean subterm can only be selected if it is not \top , \perp , or a variable. (S3) A Boolean subterm can only be selected if its occurrence is not below a quantifier. (S4) The topmost terms on either side of a positive literal cannot be selected.

The interplay of maximality w.r.t. term order, literal and Boolean selection functions gives rise to a new notion of eligibility:

Definition 2 (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 σL is (*strictly*) maximal in σC . The eligible positions of a clause C w.r.t. a substitution σ are inductively defined as follows: (E1) Any selected position is eligible. (E2) If a literal $s \approx t$ with $\sigma s \not\approx \sigma t$ is either eligible and negative or strictly eligible and positive, then $L.s.\varepsilon$ is eligible. (E3) If the position p is eligible and the root of $\sigma(C|_p)$ is not $\approx, \not\approx, \forall$, or \exists , the positions of all direct subterms are eligible. (E4) If the position p is eligible and $\sigma(C|_p)$ is of the form $s \approx t$ or $s \not\approx t$, the position of s is eligible if $\sigma s \not\approx \sigma t$ and the position of t is eligible if $\sigma s \approx \sigma t$. The substitution σ is left implicit if it is the identity substitution.

3.3 The Core Inference Rules

The following inference rules form our calculus:

$$\begin{array}{c}
\frac{\overbrace{D'}^{D} \vee t \approx t' \quad C[u]}{\sigma(D' \vee C[t'])} \text{SUP} \qquad \frac{\overbrace{C'}^C \vee u' \approx v' \vee u \approx v}{\sigma(C' \vee v \not\approx v' \vee u \approx v')} \text{FACTOR} \\
\\
\frac{\overbrace{C'}^C \vee u \not\approx u'}{\sigma C'} \text{IRREFL} \qquad \frac{\overbrace{C'}^C \vee s \approx t}{\sigma C'} \perp \text{ELIM} \qquad \frac{C[u]}{\sigma C[t']} \text{BOOLRW} \\
\\
\frac{C[\forall z. v]}{C[\{z \mapsto \text{sk}_{\forall \bar{y}. \exists z. \neg v(\bar{y})\}v]} \forall \text{RW} \qquad \frac{C[\exists z. v]}{C[\{z \mapsto \text{sk}_{\forall \bar{y}. \exists z. v(\bar{y})\}v]} \exists \text{RW} \\
\\
\frac{C[u]}{C[\perp] \vee u \approx \top} \text{BOOLHOIST} \qquad \frac{C[s \approx t]}{C[\perp] \vee s \approx t} \approx \text{HOIST} \qquad \frac{C[s \not\approx t]}{C[\top] \vee s \approx t} \not\approx \text{HOIST} \\
\\
\frac{C[\forall x. t]}{C[\perp] \vee \{x \mapsto y\}t \approx \top} \forall \text{HOIST} \qquad \frac{C[\exists x. t]}{C[\top] \vee \{x \mapsto y\}t \approx \perp} \exists \text{HOIST}
\end{array}$$

The rules are subject to the following side conditions:

- SUP (1) $\sigma = \text{mgu}(t, u)$; (2) u is not a variable; (3) $\sigma t \not\approx \sigma t'$; (4) $D < C[u]$; (5) the position of u is eligible in C w.r.t. σ ; (6) $t \approx t'$ is strictly eligible in D w.r.t. σ ; (7) the root of t is not a logical symbol; (8) if $\sigma t' = \perp$, the subterm u is at the top level of a positive literal.
- FACTOR (1) $\sigma = \text{mgu}(u, u')$; (2) $\sigma u \not\approx t \notin \sigma C$ for any term t ; (3) no Boolean subterm and no literal is selected in C ; (4) σu is a maximal term in σC ; (5) σv is maximal in $\{t \mid \sigma u \approx t \in \sigma C\}$.

- IRREFL (1) $\sigma = \text{mgu}(u, u')$; (2) $u \not\approx u'$ is eligible in C w.r.t. σ .
- \perp ELIM (1) $\sigma = \text{mgu}(s \approx t, \perp \approx \top)$; (2) $s \approx t$ is strictly eligible in C w.r.t. σ .
- BOOLRW (1) (t, t') is one of the following pairs, where x is a fresh variable:
 $(\neg\perp, \top)$, $(\neg\top, \perp)$, $(\perp \wedge \perp, \perp)$, $(\top \wedge \perp, \perp)$, $(\perp \wedge \top, \perp)$, $(\top \wedge \top, \top)$, $(\perp \vee \perp, \perp)$,
 $(\top \vee \perp, \top)$, $(\perp \vee \top, \top)$, $(\top \vee \top, \top)$, $(\perp \rightarrow \perp, \top)$, $(\top \rightarrow \perp, \perp)$, $(\perp \rightarrow \top, \top)$,
 $(\top \rightarrow \top, \top)$, $(x \approx x, \top)$, $(x \not\approx x, \perp)$; (2) $\sigma = \text{mgu}(t, u)$; (3) u is not a variable; (4) the position of u is eligible in C w.r.t. σ .
- \star RW (where $\star \in \{\forall, \exists\}$) (1) v is a term that may refer to z ; (2) \bar{y} are the free variables occurring in $\forall z.v$ and $\exists z.v$, respectively, in order of first appearance; (3) the position of the indicated subterm is eligible in C ; (4) for \forall RW, $C[\top]$ is not a tautology; (5) for \exists RW, $C[\perp]$ is not a tautology. (In an implementation, the tautology check can be approximated by checking if the affected literal is of the form $\forall z.v \approx \top$ or $\exists z.v \approx \perp$.)
- BOOLHOIST (1) u is a Boolean term whose root is an uninterpreted predicate; (2) the position of u is eligible in C ; (3) u is not a variable; (4) u is not at the top level of a positive literal.
- \star HOIST (where $\star \in \{\approx, \not\approx, \forall, \exists\}$) (1) the position of the indicated subterm is eligible in C ; (2) y is a fresh variable.

3.4 Rationale for the Rules

In addition to the standard superposition rules SUP, FACTOR, and IRREFL, our calculus contains various rules to deal with Booleans. For each logical symbol and quantifier, we must consider the case where it is true and the case where it is false. Whenever possible, we prefer rules that rewrite the Boolean subterm in place (with names ending in RW). When this cannot be done in a satisfiability-preserving way, we resort to rules hoisting the Boolean subterm into a dedicated literal (with names ending in HOIST). For terms rooted by an uninterpreted predicate, the rule BOOLHOIST only deals with the case that the term is false. If it is true, we rely on SUP to rewrite it to \top eventually.

Example 3. The clause $a \wedge \neg a \approx \top$ can be refuted by the core inferences as follows. First we derive $a \approx \top$ (displayed on the left) and then we use it to derive \perp (displayed on the right). In this and the following example, we assume eager selection of literals whenever the selection restrictions allow it.

$$\begin{array}{c}
 \frac{a \wedge \neg a \approx \top}{\perp \wedge \neg a \approx \top \vee a \approx \top} \text{BOOLHOIST} \\
 \frac{\perp \wedge \neg \perp \approx \top \vee a \approx \top \vee a \approx \top}{\perp \wedge \top \approx \top \vee a \approx \top \vee a \approx \top} \text{BOOLRW} \\
 \frac{\perp \approx \top \vee a \approx \top \vee a \approx \top}{\perp \approx \top \vee a \approx \top} \text{BOOLRW} \\
 \frac{\perp \approx \top \vee a \approx \top}{\top \not\approx \top \vee a \approx \top} \text{IRREFL} \\
 \frac{\top \not\approx \top \vee a \approx \top}{a \approx \top} \text{FACTOR}
 \end{array}
 \qquad
 \begin{array}{c}
 \frac{a \wedge \neg a \approx \top \quad a \approx \top}{\top \wedge \neg a \approx \top} \text{SUP} \\
 \frac{\top \wedge \neg \top \approx \top}{\top \wedge \perp \approx \top} \text{BOOLRW} \\
 \frac{\top \wedge \perp \approx \top}{\perp \approx \top} \text{BOOLRW} \\
 \frac{\perp \approx \top}{\perp} \text{ELIM}
 \end{array}$$

The derivation illustrates how **BOOLHOIST** and **SUP** replace uninterpreted predicates by \mathbf{T} and \perp to allow **BOOLRW** to eliminate the surrounding logical symbols.

Example 4. The clause $(\exists x. \forall y. y \not\approx x) \approx \mathbf{T}$ can be refuted as follows:

$$\frac{\frac{\frac{(\exists x. \forall y. y \not\approx x) \approx \mathbf{T}}{(\forall y. y \not\approx \text{sk}_{\exists x. \forall y. y \not\approx x}) \approx \mathbf{T}}{\perp \approx \mathbf{T} \vee (y' \not\approx \text{sk}_{\exists x. \forall y. y \not\approx x}) \approx \mathbf{T}} \exists\text{RW}}{\perp \approx \mathbf{T}} \text{VHOIST}}{\frac{(y' \not\approx \text{sk}_{\exists x. \forall y. y \not\approx x}) \approx \mathbf{T}}{\perp \approx \mathbf{T}} \not\approx\text{RW}}{\perp} \perp\text{ELIM}} \perp\text{ELIM}$$

This example shows how quantifiers can be Skolemized or hoisted to the clausal level.

Our calculus is a graceful generalization of superposition: if the input problem does not contain any Boolean terms, it coincides with standard superposition. Our **FACTOR** rule is even a bit more restrictive by excluding equality factoring inferences such as

$$\frac{c \not\approx a \vee b \approx f(x) \vee b \approx g(y)}{c \not\approx a \vee f(x) \not\approx g(y) \vee b \approx f(x)}$$

where $c \succ b$ and $b, f(x), g(y)$ are mutually incomparable.

However, most standard superposition provers support nonequational predicates by encoding them as $t \approx \mathbf{T}$ and $t \not\approx \mathbf{T}$, where \mathbf{T} is considered an auxiliary symbol that does not receive any special treatment by the prover. In this respect, our calculus diverges slightly from standard superposition since our normal representation is $t \approx \mathbf{T}$ and $t \approx \perp$.

Example 5. Consider the unsatisfiable clause set consisting of $f(\mathbf{a}, \mathbf{a}) \approx \mathbf{T}$ and $f(x, \mathbf{a}) \not\approx \mathbf{T} \vee f(\mathbf{a}, x) \not\approx \mathbf{T}$. In standard superposition, it can be refuted by two superposition inferences from the first clause, followed by two equality resolution inferences. Our calculus also allows for these inferences, but performs in addition a **BOOLHOIST** inference on $f(x, \mathbf{a})$ or $f(\mathbf{a}, x)$ in the second clause.

To avoid this **BOOLHOIST** inference, we can bring the two clauses in our normal representation of predicate literals, yielding $f(\mathbf{a}, \mathbf{a}) \approx \mathbf{T}$ and $f(x, \mathbf{a}) \approx \perp \vee f(\mathbf{a}, x) \approx \perp$. But then the **SUP** inference is no longer applicable because in this representation with positive literals the two occurrences of the terms $f(x, \mathbf{a})$ and $f(\mathbf{a}, x)$ in the second clause are not eligible w.r.t. $\{x \mapsto \mathbf{a}\}$. Instead, our calculus now requires an **FACTOR** inference on the second clause, whose conclusion contains the necessary eligible occurrence of $f(\mathbf{a}, \mathbf{a})$.

As Ganzinger and Stuber observed, selection can be used to faithfully imitate the behavior of ordered resolution. The same holds for imitating the support for nonequational predicates in the standard superposition calculus. In our example, we can select the subterm $f(x, \mathbf{a})$ to make it eligible and to avoid the applicability of **FACTOR**, thus achieving the same refutation as standard superposition without additional inferences.

4 Satisfiability Preservation

Most of our calculus rules are sound. However, since \forall RW and \exists RW introduce fresh Skolem symbols, they are only satisfiability preserving. To prove these properties, we first show the substitution lemma for our logic:

Lemma 6 (Substitution lemma). *Let $\mathcal{I} = (\mathcal{U}, \mathcal{J})$ be an interpretation. Then*

$$\llbracket \sigma t \rrbracket_{\mathcal{I}}^{\xi} = \llbracket t \rrbracket_{\mathcal{I}}^{\xi'}$$

for all terms t , all substitutions σ , and all valuations ξ, ξ' such that $\xi'(x) = \llbracket \sigma x \rrbracket_{\mathcal{I}}^{\xi}$ for all variables x .

Proof. We prove this by structural induction on t . If $t = x$ for some $x \in \mathcal{V}$, then

$$\llbracket \sigma t \rrbracket_{\mathcal{I}}^{\xi} = \xi'(x) = \llbracket t \rrbracket_{\mathcal{I}}^{\xi'}$$

If $t = f(\bar{t})$ for some $f \in \Sigma$ and terms \bar{t} , then

$$\llbracket \sigma t \rrbracket_{\mathcal{I}}^{\xi} = \llbracket f(\sigma \bar{t}) \rrbracket_{\mathcal{I}}^{\xi} = \mathcal{J}(f)(\llbracket \sigma \bar{t} \rrbracket_{\mathcal{I}}^{\xi}) \stackrel{\text{IH}}{=} \mathcal{J}(f)(\llbracket \bar{t} \rrbracket_{\mathcal{I}}^{\xi'}) = \llbracket t \rrbracket_{\mathcal{I}}^{\xi'}$$

If $t = \forall x. s$ for some term s , then we can assume without loss of generality that $\sigma x = x$ and that x does not appear in σy for any $y \neq x$. We have

$$\begin{aligned} \llbracket \sigma t \rrbracket_{\mathcal{I}}^{\xi} &= \llbracket \forall x. \sigma s \rrbracket_{\mathcal{I}}^{\xi} = \min \{ \llbracket \sigma s \rrbracket_{\mathcal{I}}^{\xi[x \mapsto a]} \mid a \in \mathcal{U}_{\tau} \} \\ &\stackrel{\text{IH}}{=} \min \{ \llbracket s \rrbracket_{\mathcal{I}}^{\xi'[x \mapsto a]} \mid a \in \mathcal{U}_{\tau} \} = \llbracket \forall x. s \rrbracket_{\mathcal{I}}^{\xi'} = \llbracket t \rrbracket_{\mathcal{I}}^{\xi'} \end{aligned}$$

The induction hypothesis is applicable because $\xi'[x \mapsto a](x) = a = \llbracket x \rrbracket_{\mathcal{I}}^{\xi[x \mapsto a]} = \llbracket \sigma x \rrbracket_{\mathcal{I}}^{\xi[x \mapsto a]}$ and for all $y \neq x$ we have $\xi'[x \mapsto a](y) = \xi'(y) = \llbracket \sigma y \rrbracket_{\mathcal{I}}^{\xi} = \llbracket \sigma y \rrbracket_{\mathcal{I}}^{\xi[x \mapsto a]}$. An analogous argument applies if $t = \exists x. s$. \square

Lemma 7. *Let C be a clause and \mathcal{I} an interpretation. If $\mathcal{I} \models C$, then $\mathcal{I} \models \sigma C$ for all substitutions σ .*

Proof. Let $s \approx t$ be a literal of C and ξ a valuation. Since $\mathcal{I} \models C$, this literal is true w.r.t. \mathcal{I} and ξ . Let ξ' be defined as in Lemma 6. Then $\llbracket \sigma s \rrbracket_{\mathcal{I}}^{\xi} = \llbracket s \rrbracket_{\mathcal{I}}^{\xi'}$ and $\llbracket \sigma t \rrbracket_{\mathcal{I}}^{\xi} = \llbracket t \rrbracket_{\mathcal{I}}^{\xi'}$. Thus, $\sigma s \approx \sigma t$ is true w.r.t. \mathcal{I} and ξ' . Since this holds for arbitrary literals $s \approx t$ in C and arbitrary valuations, we have $\mathcal{I} \models \sigma C$. \square

Lemma 8. *All rules of $FInf$ except for \forall RW and \exists RW are sound.*

Proof. For SUP, FACTOR, IRREFL, and \perp ELIM, we can show soundness by applying Lemma 7 to the premises and, given an interpretation and a valuation, making a case distinction on the truth of literals, using the fact that σ is an mgu.

For BOOLHOIST, \approx HOIST, $\not\approx$ HOIST, \forall HOIST, and \exists HOIST, given an interpretation and a valuation, we can simply make a case distinction on whether the affected subterm of the premise is true or false.

For BOOLRW, we apply Lemma 7 to the premise. Since σ is an mgu and since both elements of each listed pair have identical denotations under any interpretation and valuation, soundness follows. \square

Theorem 9 (Satisfiability preservation). *If the initial problem does not contain any Skolem symbols, all rules of $FInf$ preserve satisfiability.*

Proof. Except for $\forall RW$ and $\exists RW$, Lemma 8 implies satisfiability preservation. We will argue for $\forall RW$ here. For $\exists RW$, we can argue analogously. Assume that the current clause set N and in particular the premise $C[\forall z. v]$ of the $\forall RW$ inference has a model $\mathcal{I} = (\mathcal{U}, \mathcal{J})$. We define a new interpretation $\mathcal{I}' = (\mathcal{U}, \mathcal{J}')$, redefining the interpretation of $sk_{\forall \bar{y}. \exists z. \neg v} : \bar{\tau} \rightarrow \tau$. For all other symbols, let the interpretation function \mathcal{J}' be identical with the old interpretation function \mathcal{J} . What remains is to define $\mathcal{J}'(sk_{\forall \bar{y}. \exists z. \neg v})(\bar{a})$ for each $\bar{a} \in \mathcal{U}_{\bar{\tau}}$.

Let $\bar{a} \in \mathcal{U}_{\bar{\tau}}$ and ξ be a valuation such that $\xi(\bar{y}) = \bar{a}$. If $\llbracket \forall z. v \rrbracket_{\mathcal{I}}^{\xi} = 1$, then by the definition of term denotation, for all $a \in \mathcal{U}_{\tau}$, $\llbracket v \rrbracket_{\mathcal{I}}^{\xi[z \mapsto a]} = 1$. Otherwise $\llbracket \forall z. v \rrbracket_{\mathcal{I}}^{\xi} = 0$ and hence there exists an $a \in \mathcal{U}_{\tau}$ such that $\llbracket v \rrbracket_{\mathcal{I}}^{\xi[z \mapsto a]} = 0$. In both cases there exists an $a \in \mathcal{U}_{\tau}$ such that $\llbracket v \rrbracket_{\mathcal{I}}^{\xi[z \mapsto a]} = \llbracket \forall z. v \rrbracket_{\mathcal{I}}^{\xi}$. We define $\mathcal{J}'(sk_{\forall \bar{y}. \exists z. \neg v})(\bar{a}) = a$.

This definition ensures that, by Lemma 6,

$$\llbracket \{z \mapsto sk_{\forall \bar{y}. \exists z. \neg v}(\bar{y})\}v \rrbracket_{\mathcal{I}'}^{\xi} = \llbracket v \rrbracket_{\mathcal{I}'}^{\xi[z \mapsto a]} = \llbracket v \rrbracket_{\mathcal{I}}^{\xi[z \mapsto a]} = \llbracket \forall z. v \rrbracket_{\mathcal{I}}^{\xi} \quad (*)$$

If N already contains the symbol $sk_{\forall \bar{y}. \exists z. \neg v}$, it must have been introduced by $\forall RW$ or $\exists RW$. Without loss of generality, we can therefore assume that \mathcal{I} interprets $sk_{\forall \bar{y}. \exists z. \neg v}$ already as defined above and hence $\mathcal{I} = \mathcal{I}'$. If N does not contain $sk_{\forall \bar{y}. \exists z. \neg v}$, \mathcal{I}' is also a model of N . In both cases, \mathcal{I}' being a model of N and in particular $C[\forall z. v]$ implies that \mathcal{I}' is a model of $N \cup \{C[\{z \mapsto sk_{\forall \bar{y}. \exists z. \neg v}(\bar{y})\}v]\}$. Thus, satisfiability is preserved. \square

4.1 The Redundancy Criterion

As in standard first-order superposition, we define the redundancy criterion first on ground clauses and ground inferences, and lift it to nonground clauses and nonground inferences in a second step. The distinction of a ground calculus G and our calculus F without a ground restriction will also be used in the refutational completeness proof. Let \mathcal{T}_F and \mathcal{C}_F denote the set of all terms and of all clauses, respectively. Let \mathcal{T}_G and \mathcal{C}_G denote the set of ground terms and of ground clauses, respectively. Moreover, in the following, we define a ground inference system $GInf$ on G that—unlike the ground inference system of standard first-order superposition—is not contained in the inference system $FInf$ on F .

The inference system $GInf$ is parameterized by a literal selection function $GLSel$, a Boolean subterm selection function $GBSel$, and a witness function \mathbf{w} . The functions $GLSel$ and $GBSel$ are subject to the same restrictions as specified in Definition 1. The witness function \mathbf{w} maps a clause $C \in \mathcal{C}_G$ and a subterm $\exists x. v$ of C to a term $\mathbf{w}(C, \exists x. v) \in \mathcal{T}_G$. This function will be used to provide an applied skolem function for $\exists x. v$. We require that $\exists x. v \succ \{x \mapsto \mathbf{w}(C, \exists x. v)\}v$ and $\forall x. v \succ \{x \mapsto \mathbf{w}(C, \exists x. \neg v)\}v$. Let Q be the set of all triples $(GLSel, GBSel, \mathbf{w})$ that satisfy the above requirements. Let $q \in Q$ be such a parameter triple.

The term order is a fourth parameter of $GInf^q$, but unlike the other parameters, we let it coincide with the term order on F and it can therefore be fixed globally.

The inference rules of $GInf^q$ include the rules SUP, FACTOR, IRREFL, \perp ELIM, BOOLRW, \approx HOIST, and $\not\approx$ HOIST on ground clauses. In addition, $GInf^q$ contains the following rules:

$$\frac{C[\forall z. v]}{C[\{z \mapsto \mathbf{w}(C, \exists x. \neg v)\}v]} \text{G}\forall\text{RW} \quad \frac{C[\exists z. v]}{C[\{z \mapsto \mathbf{w}(C, \exists x. v)\}v]} \text{G}\exists\text{RW}$$

where

- v is a term that may contain the loose bound variable z
- the position of the indicated subterm is eligible in C
- for $\text{G}\forall\text{RW}$, $C[\mathbf{T}]$ is not a tautology, and for $\text{G}\exists\text{RW}$, $C[\perp]$ is not a tautology.

$$\frac{C[\forall x. t]}{C[\perp] \vee \{x \mapsto u\}t \approx \mathbf{T}} \text{G}\forall\text{HOIST} \quad \frac{C[\exists x. t]}{C[\mathbf{T}] \vee \{x \mapsto u\}t \approx \perp} \text{G}\exists\text{HOIST}$$

where

- u is a ground term whose only Boolean subterms are \mathbf{T} and \perp
- the position of the indicated subterm is eligible in C .

Definition 10 (Redundancy on G). Following Bachmair and Ganzinger [4, Sect. 4.2.2], we define a ground clause C to be *redundant* w.r.t. a set N of ground clauses if there exist clauses $C_1, \dots, C_k \in N$ such that $C_1, \dots, C_k \models C$ and $C \succ C_i$ for all $1 \leq i \leq k$. We write $GRed_C(N)$ for the set of redundant ground clauses w.r.t. N . A ground inference with main premise C , side premises C_1, \dots, C_n , and conclusion D is called *redundant* w.r.t. N if there exist clauses $D_1, \dots, D_k \prec C$ in N such that $D_1, \dots, D_k, C_1, \dots, C_n \models D$. We write $GRed_I(N)$ for the set of redundant ground inferences w.r.t. N .

Lemma 11. *Our redundancy criterion on G is a redundancy criterion in the sense of the saturation framework.*

Proof. By B&G Handbook Theorem 4.7, this is a redundancy criterion in the sense of B&G. If the ground inference system is reductive, it is also a redundancy criterion in the sense of Waldmann et al.'s framework by B&G Handbook Theorem 4.8. \square

Definition 12 (Grounding function). Given a clause $C \in \mathcal{C}_F$, let $\Gamma C \subseteq \mathcal{C}_G$ be the set of all ground clauses of the form γC where γ is a substitution such that for all variables x , the only Boolean subterms of γx are \perp and \mathbf{T} .

Given a parameter triple $q \in Q$ and an inference $\iota \in FInf$, we define the set $I^q(\iota)$ of ground instances of ι to be all inferences $\iota' \in GInf^q$ such that $\text{prems}(\iota') = \gamma \text{prems}(\iota)$ and $\text{concl}(\iota') = \gamma \text{concl}(\iota)$ for some substitution γ and such that ι and ι' stem from the same inference rule or from BOOLRW and GBOOLRW, \forall HOIST and \forall HOIST, or \exists HOIST and \exists HOIST, respectively.

Definition 13 (Redundancy on F). A nonground clause C is redundant w.r.t. clauses N if C is strictly subsumed by a clause in N (strictly meaning that C is subsumed by but does not subsume the other clause) or every ground instance of C is redundant w.r.t. ground instances of N according to the prior definition. We write $FRed_C(N)$ for the set of redundant clauses w.r.t. N . Similarly a nonground inference is redundant if it is redundant at the level of ground instances regardless of the parameterization of the grounding. We write $FRed_I(N)$ for the set of redundant inferences w.r.t. N .

A clause set N is *saturated* w.r.t. an inference system and the inference component Red_I of a redundancy criterion if every inference from clauses in N is in $Red_I(N)$.

4.2 Simplification Rules

The redundancy criterion is a graceful generalization of the criterion of standard superposition. Thus, the standard simplification and deletion rules, such as deletion of trivial literals and clauses, subsumption, and demodulation, can be justified. Demodulation below quantifiers is justified if the term order is compatible with contexts below quantifiers.

Some calculus rules can act as simplifications. \perp ELIM can always be a simplification. Given a clause on which both \star RW and \star HOIST apply, where $\star \in \{\forall, \exists\}$, the clause can be replaced by the conclusions of these rules. If \star RW does not apply because of condition 4 or 5, \star HOIST alone can be a simplification. Also justified by redundancy, the rules BOOLHOIST and \star HOIST can simultaneously replace all occurrences of the eligible subterm they act on. For example, applying \approx HOIST to $p(x \approx y) \approx \top \vee q(x \approx y) \approx \perp$ yields $p(\perp) \approx \top \vee q(\perp) \approx \perp \vee x \approx y$.

While experimenting with our implementation, we have observed that the following simplification rule from Vampire [18] can substantially shorten proofs:

$$\frac{s \not\approx t \vee C[s]}{s \not\approx t \vee C[t]} \text{LOCALRW}$$

In this rule, we require $s \succ t$.

Interpreting literals of the form $s \approx \top$ as $s \not\approx \perp$ and $s \approx \perp$ as $s \not\approx \top$ we can apply the rule even to these positive literals. This especially convenient with rules such as BOOLHOIST. Consider the clause $C = p^i(\perp) \approx \perp \vee q \approx \perp$, assume no literal is selected and the Boolean selection function always selects a subterm $p(\perp)$. Applying BOOLHOIST to C we get $p(\perp) \approx \top \vee p^{i-1}(\perp) \approx \perp \vee q \approx \perp$. This can then be simplified to a tautological clause $p(\perp) \approx \top \vee p(\perp) \approx \perp \vee q \approx \perp$ using $i - 2$ LOCALRW steps. If we did not use LOCALRW, BOOLHOIST would produce $i - 2$ intermediary clauses starting from C , none of which would be recognized as a tautology.

Many rules of our calculus replace subterms with \top or \perp . After this replacement, resulting terms can be simplified using Boolean equivalences that specify

the behavior of logical operations on \top and \perp . To this end, we use the rule BOOLSIMP [31], similar to *simp* of Leo-III [25, Sect. 4.2.1]:

$$\frac{C[s]}{C[t]} \text{BOOLSIMP}$$

This rule replaces s with t whenever $s \approx t$ is contained in a predefined set of tautological equations. In addition to all equations that Leo-III uses for *simp*, we also include the following ones:

$$\begin{aligned} (\top \rightarrow u) &\approx u & (\perp \rightarrow u) &\approx \top & (u \rightarrow \perp) &\approx \neg u & (u \rightarrow \top) &\approx \top \\ (u \rightarrow \neg u) &\approx \neg u & (\neg u \rightarrow u) &\approx u & (u \rightarrow u) &\approx \top & & \\ (u_1 \rightarrow \dots \rightarrow u_n \rightarrow v) &\approx \top & \text{where } u_i &= \neg u_j & \text{for some } i \neq j & & & \\ (u_1 \rightarrow \dots \rightarrow u_n \rightarrow v_1 \vee \dots \vee v_m) &\approx \top & \text{where } u_i &= v_j & \text{for some } i \text{ and } j & & & \\ (u_1 \wedge \dots \wedge u_n \rightarrow v_1 \vee \dots \vee v_m) &\approx \top & \text{where } u_i &= v_j & \text{for some } i \text{ and } j & & & \end{aligned}$$

It is easy to check that applying any equivalence reduces the size of s (assuming \neg is not greater than \rightarrow in weight and precedence). Using BOOLSIMP and \perp ELIM, the twelve steps of Example 3 can be replaced by just two simplification steps.

BOOLSIMP simplifies terms with logical symbol roots if one argument is either \top or \perp or if two arguments are identical. Thus, after simplification, BOOLRW applies only in two remaining cases: if all arguments of a logical symbol are distinct variables and if the sides of a (dis)equation are different and unifiable. This observation can be used to streamline the implementation of BOOLRW.

5 Refutational Completeness

Our calculus (*FInf*, *FRed*)—like every other saturation-based calculus—proves a statement S given a set of axioms T by deriving \perp from $N = T \cup \{\neg S\}$. In this section we are concerned with proving the refutational completeness of our calculus—i.e., that it always derives \perp when the input formula N is such that $N \models \perp$. Refutational completeness comes in two variants, static and dynamic completeness. Static completeness applies only on *saturated* sets of clauses—i.e., sets N such that every inference with premises in N is redundant to N .

Definition 14 (Static Completeness). *A calculus is statically complete if for any saturated set of clauses N such that $N \models \perp$ then $\perp \in N$.*

Dynamic completeness is a property that applies on fair derivations. Here a *derivation* is a finite or infinite sequence of clause sets $(N_j)_j$ such that, at each step j , $N_j \models \perp$ if and only if $N_{j+1} \models \perp$ and the deleted clauses in $N_j \setminus N_{j+1}$ are redundant w.r.t. N_{j+1} . Moreover a derivation is *fair* if every inference from $N_\infty := \bigcup_i \bigcap_{j \geq i} N_j$ is redundant w.r.t. some N_j .

Definition 15 (Dynamic Completeness). *A calculus is dynamically complete if for any fair derivation $(N_j)_j$ such that $N_0 \models \perp$, $\perp \in N_\infty$.*

We follow the usual schema for proving the dynamic completeness of a saturation-based calculus: We start by proving the static completeness of the ground variant of the calculus following Bachmair and Ganzinger’s approach [4] and then we lift this result to the non-ground level and dynamic completeness by relying on the saturation framework of Waldmann et al. [33], that generalizes the lifting technique from Bachmair and Ganzinger.

5.1 Ground Layer

In the spirit of Bachmair and Ganzinger’s completeness proof for first-order logic, our proof idea is, given a saturated set N such that $\perp \notin N$, to construct a term rewrite system that can be viewed as a model of N .

We assume the reader familiar with term-rewriting notions and syntax [1]. Our term rewrite systems are essentially standard first-order term rewrite systems. We generalize them to our terms with interpreted Booleans by treating all quantified terms as if they were constants, meaning that a term rewrite system does not rewrite below quantifiers. For example, the rewrite rule $\forall x. q \longrightarrow \forall x. p$ can rewrite $f(\forall x. q)$ into $f(\forall x. p)$, but the rewrite rule $q \longrightarrow p$ cannot.

It is well-known that in first-order logic term rewrite systems can be used to describe interpretations. In the following, we will show under which requirements a term rewrite system on our logic can also be viewed as an interpretation.

Definition 16 (Interpretable Rewrite System). Let R be a rewrite system over \mathcal{T}_G , R is *interpretable* when:

- (I1) for all Boolean terms $t \in \mathcal{T}_G$, either $t \leftrightarrow_R^* \mathbf{T}$ or $t \leftrightarrow_R^* \mathbf{\perp}$;
- (I2) $\neg \mathbf{\perp} \leftrightarrow_R^* \mathbf{T}$; $\neg \mathbf{T} \leftrightarrow_R^* \mathbf{\perp}$; and corresponding requirements for $\mathbf{\wedge}$, $\mathbf{\vee}$, and $\mathbf{\rightarrow}$;
- (I3) $s \approx s' \leftrightarrow_R^* \mathbf{T}$ if and only if $s \leftrightarrow_R^* s'$ for all $s, s' \in \mathcal{T}_G$; and corresponding requirements for $\not\approx$;
- (I4) $\forall x. s \leftrightarrow_R^* \mathbf{T}$ if and only if $\{x \mapsto u\}s \leftrightarrow_R^* \mathbf{T}$ for all $u \in \mathcal{T}_G$; and corresponding requirements for \exists .

We define an interpretation $(\mathcal{U}, \mathcal{J})$ based on an interpretable rewrite system R . We use R to denote both the rewrite system and the interpretation.

For each type τ , let \mathcal{U}_τ be the set of equivalence classes $[t]$ of terms $t \in \mathcal{T}_G$ modulo \leftrightarrow_R^* . Let $\mathcal{J}(f)(\bar{a}) = [f(\bar{t})]$ where \bar{t} are terms from the equivalence classes \bar{a} , respectively. This does not depend on the choice of \bar{t} because if $\bar{t} \leftrightarrow_R^* \bar{t}'$, then $f(\bar{t}) \leftrightarrow_R^* f(\bar{t}')$.

We identify $[\mathbf{T}]$ with 1 and $[\mathbf{\perp}]$ with 0. By (I1), this ensures that $\mathcal{U}_o = \{0, 1\}$, $\mathcal{J}(\mathbf{T}) = 1$, and $\mathcal{J}(\mathbf{\perp}) = 0$. (I2) ensures that $\mathcal{J}(\neg)$, $\mathcal{J}(\mathbf{\wedge})$, $\mathcal{J}(\mathbf{\vee})$, and $\mathcal{J}(\mathbf{\rightarrow})$ adhere to the requirements of an interpretation. (I3) ensures that $\mathcal{J}(\approx)$ and $\mathcal{J}(\not\approx)$ adhere to the requirements of an interpretation and (I4) does the same for quantifiers.

Lemma 17. *Let R be an interpretable rewrite system. Then $[[t]]_R = [t]$ for all $t \in \mathcal{T}_G$.*

Proof. By induction on the structure of t .

If $t = f(\bar{s})$, then $\llbracket t \rrbracket_R = \mathcal{J}(f)(\llbracket \bar{s} \rrbracket_R) \stackrel{\text{IH}}{=} \mathcal{J}(f)([\bar{s}]) = [f(\bar{s})] = [t]$. If $t = \exists x. s$, then, using the substitution lemma (Lemma 6),

$$\begin{aligned} \llbracket t \rrbracket_R &= \min \{ \llbracket s \rrbracket_R^{\{x \mapsto [u]\}} \mid u \in \mathcal{T}_G \} \\ &= \min \{ \llbracket \{x \mapsto u\} s \rrbracket_R \mid u \in \mathcal{T}_G \} \\ &\stackrel{\text{IH}}{=} \min \{ [\{x \mapsto u\} s] \mid u \in \mathcal{T}_G \} \\ &\stackrel{\text{(I4)}}{=} [\forall x. s] = [t]. \end{aligned}$$

If $t = \forall x. s$, we argue analogously. \square

This lemma shows that $R \models t \approx t'$ if and only if $t \leftrightarrow_R^* t'$, as in first-order logic. In the following step, we define a closure operation on term rewrite systems that allows us to turn a term rewrite system into an interpretable one under some conditions.

Definition 18 (Boolean closure of a term rewrite system). Let R be a ground term rewrite system. We define Δ_R^s and R^s by induction over all terms s .

- (A1) Let $\Delta_R^s = \emptyset$ if s is not Boolean, if s is reducible by R^s , or if $s = \mathbf{T}$ or $s = \mathbf{\perp}$
- (A2) Otherwise, let $\Delta_R^s = \{s \longrightarrow \mathbf{T}\}$ if one of the following conditions holds:
 - (i) $s = \neg \mathbf{\perp}$;
 - (ii) $s = \mathbf{T} \wedge \mathbf{T}$;
 - (iii) $s = \mathbf{T} \vee t$ or $s = t \vee \mathbf{T}$;
 - (iv) $s = \mathbf{\perp} \rightarrow t$ or $s = t \rightarrow \mathbf{T}$;
 - (v) $s = t \approx t$;
 - (vi) $s = t \not\approx t'$ and $t \neq t'$;
 - (vii) $s = \forall x. t$ and $\{x \mapsto u\}t \longrightarrow_{R^s}^* \mathbf{T}$ for all ground terms u in which all Boolean subterms are either \mathbf{T} or $\mathbf{\perp}$;
 - (viii) $s = \exists x. t$ and $\{x \mapsto u\}t \longrightarrow_{R^s}^* \mathbf{T}$ for some ground term u in which all Boolean subterms are either \mathbf{T} or $\mathbf{\perp}$.
- (A3) Otherwise, let $\Delta_R^s = \{s \longrightarrow \mathbf{\perp}\}$.

Let $R^s = R \cup \bigcup_{u \prec s} \Delta_R^u$ and $R^* = R \cup \bigcup_u \Delta_R^u$.

Lemma 19. *Let R be without critical pairs and oriented by \succ . Assume for each rule $s' \longrightarrow t' \in R$ that all proper Boolean subterms of s' are \mathbf{T} and $\mathbf{\perp}$ and that the root of s' is not a logical symbol. Let s be a Boolean term. Then*

- (1) R^s and R^* are oriented by \succ and hence terminating.
- (2) R^s and R^* do not have critical pairs and are thus confluent.
- (3) The normal form of any Boolean term smaller than s w.r.t. R^s is $\mathbf{\perp}$ or \mathbf{T} .
- (4) The normal form of any Boolean term w.r.t. R^* is $\mathbf{\perp}$ or \mathbf{T} .
- (5) R^* is an interpretable rewrite system.

Proof. (1) is obvious from Definition 18 and (O1).

For (2), suppose there is a critical pair. Since R does not have critical pairs, one of the rules of the critical pair must come from some Δ_R^u for some term u . Due to condition (A1), Δ_R^u cannot form a critical pair with some Δ_R^v . Thus, the other rule of the critical pair must stem from R , say $s' \rightarrow t' \in R$. Also due to condition (A1), s' cannot be smaller or equal to u because that would make u reducible by R^u . But $s' \succ u$ is not possible either because then s' would contain a proper Boolean subterm that is neither \top nor \perp by condition (A1) and by (O3). Contradiction by (O5).

The points (3) and (4) are obvious from Definition 18.

For (5), Condition (I1) follows from part (4). For (I2), since R does not contain any rules reducing terms rooted by logical symbols, R cannot reduce $\neg\perp$, $\neg\top$, or similar terms. Therefore, (I2) follows directly from the definition of the Boolean closure.

For (I3), first assume that $s \leftrightarrow_{R^*}^* s'$. Let u be their common normal form. Then $s \approx s' \rightarrow_{R^*}^* u \approx u$. Since u cannot be reduced further, and R does not contain any rules reducing terms rooted by logical symbols, the Boolean closure will add the rule $u \approx u \rightarrow \top$. If on the other hand, $s \not\leftrightarrow_{R^*}^* s'$, Then $s \approx s' \rightarrow_{R^*}^* u \approx u'$ for distinct normal forms u, u' . Then the Boolean closure will add the rule $u \approx u' \rightarrow \perp$. A similar reasoning holds for $\not\approx$.

For (I4), we need to prove the following claim: Let t be a term. Let θ, θ' be grounding substitutions such that for each variable x in t , we have $\theta x \leftrightarrow_{R^*}^* \theta' x$. Then we claim that $\theta t \leftrightarrow_{R^*}^* \theta' t$. Since we do not allow term rewrite systems to rewrite below quantifiers, this is not entirely trivial. We prove the claim by induction on the number of nested levels of quantifiers in t . If t contains no quantifiers, we can rewrite freely and the claim is obvious. If t contains a quantified term $\forall x. s$, it suffices to show that $\theta \forall x. s \leftrightarrow_{R^*}^* \theta' \forall x. s$. Since R does not contain any rules reducing terms rooted by logical symbols, R cannot reduce $\theta \forall x. s$ or $\theta' \forall x. s$. Hence, either $\theta\{x \mapsto u\}s \rightarrow_{R^t}^* \top$ for all ground terms u in which all Boolean subterms are either \top or \perp and such that $\theta \forall x. s \rightarrow_{R^t}^* \top$ or $\theta \forall x. s \rightarrow_{R^t}^* \perp$. The same holds for θ' . Therefore, it suffices to show that $\theta\{x \mapsto u\}s \leftrightarrow_{R^*}^* \theta'\{x \mapsto u\}s$, which holds by the induction hypothesis.

By the above claim, using (I1), we have $\{x \mapsto u\}s \rightarrow_{R^t}^* \top$ for all u if and only if $\{x \mapsto u\}s \rightarrow_{R^t}^* \top$ for all u in which all Boolean subterms are either \top or \perp . The corresponding requirement is similarly satisfied for $\exists x. s$. \square

We write $R|_{\prec s}$ or $R|_{\preceq s}$ for the rewrite system consisting of the rules in R with a left-hand side $\prec s$ or $\preceq s$, respectively.

Lemma 20. *Let t be a ground Boolean term. Let $R_1|_{\prec t} = R_2|_{\prec t}$ for two ground rewrite systems R_1 and R_2 oriented by \succ . Then, for all $s \prec t$, we have $\Delta_{R_1}^s = \Delta_{R_2}^s$.*

Proof. By induction on s . The induction hypothesis states that $\Delta_{R_1}^u = \Delta_{R_2}^u$ for all $u \prec s$. With the assumption $R_1|_{\prec t} = R_2|_{\prec t}$, it follows that $R_1^s|_{\prec t} = R_2^s|_{\prec t}$. Since $s \prec t$, in particular $R_1^s|_{\preceq s} = R_2^s|_{\preceq s}$. Inspecting the dependencies on R in

the definition of Δ_R^s , we observe that Δ_R^s depends only on rules in $R^s|_{\preceq s}$. Hence, $\Delta_{R_1}^s = \Delta_{R_2}^s$. \square

Although the rewrite systems R in the rest of this section are not necessarily interpretable, we write $R \models D$ if and only if normalizing D with R yields a clause with a trivial literal (i.e., $s \approx s$ or $s \not\approx t$ for terms $s \neq t$).

Definition 21 (R_N). *Let N with $\perp \notin N$ be a set of ground clauses. By induction on all ground clauses $C \in N$, let $\Delta_C = \{s \rightarrow t\}$ if*

- (C1) $s \succ t$;
- (C2) $R_C^s \not\models C$;
- (C3) $C = C' \vee s \approx t$ where $s \approx t$ is eligible in C ;
- (C4) the root of s is not a logical symbol;
- (C5) $s \approx t$ is maximal in C ;
- (C6) $R_C^s \cup \{s \rightarrow t\} \not\models C'$; and
- (C7) s is irreducible by R_C^s .

Otherwise $\Delta_C = \emptyset$. Let $R_C = \bigcup_{D \prec C} \Delta_D$. Let $R_N = \bigcup_C \Delta_C$.

Finally, let $R_N^*|_{\prec C}$ be R_N^* , but without all rules produced by clauses greater than or equal to C . This means $R_N^*|_{\prec C} = R_C \cup (R_N^* \setminus R_N)$.

Lemma 22 shows that R_C and R_N both fulfill the conditions of Lemma 19.

Lemma 22. *The rewrite systems R_C and R_N do not have critical pairs and are oriented by \succ . Moreover, for each of their rules $s \rightarrow t$, all proper Boolean subterms of s are \mathbf{T} or $\mathbf{\perp}$ and the root of s is not a logical symbol.*

Proof. By (C1), all rules are oriented by \succ . Suppose there is a critical pair, and let C be the larger one of the two clauses producing the critical pair. Then R_C^s would be reducible by the other rule of the critical pair, contradicting (C7). By (C4), the root of the rules' left-hand sides cannot be a logical symbol.

Finally, for each rule $s \rightarrow t$ in R_C or R_N , we must show that all proper Boolean subterms of s are \mathbf{T} or $\mathbf{\perp}$. We proceed by induction on the clause C producing the rule $s \rightarrow t$. By the induction hypothesis, Lemma 19 can be applied to R_C . By Lemma 19(3), the normal form w.r.t. R_C^s of any Boolean term smaller than s is \mathbf{T} or $\mathbf{\perp}$. Thus, if s had a proper Boolean subterm other than \mathbf{T} or $\mathbf{\perp}$, it would be reducible by R_C^s , contradicting (C7). \square

Lemma 23. *Let s be the maximal term of a clause $C \in N$. Then we have $R_N^* = R_C^s \cup \bigcup_{D \succeq C} \Delta_D \cup \bigcup_{u \succeq s} \Delta_{R_N}^u$.*

Proof.

$$\begin{aligned}
R_N^* &= \bigcup_{C \in N} \Delta_C \cup \bigcup_u \Delta_{R_N}^u && \text{by definition of } R_N \text{ and } R^* \\
&= R_C \cup \bigcup_{D \succeq C} \Delta_D \cup \bigcup_u \Delta_{R_N}^u && \text{by definition of } R_C \\
&= R_C \cup \bigcup_{u \prec s} \Delta_{R_C}^u \cup \bigcup_{D \succeq C} \Delta_D \cup \bigcup_{u \succeq s} \Delta_{R_N}^u && \text{by Lemma 20, } \bigcup_{u \prec s} \Delta_{R_N}^u = \bigcup_{u \prec s} \Delta_{R_C}^u \\
&= R_C^s \cup \bigcup_{D \succeq C} \Delta_D \cup \bigcup_{u \succeq s} \Delta_{R_N}^u && \text{by definition of } R_C^s
\end{aligned}$$

□

Lemma 24. *Let $C = C' \vee s \approx s'$ be a clause where $s \approx s'$ is maximal and $s \succeq s'$. If $R_C^s \models C$, then $R_N^* \models C$.*

Proof. We assume that $R_C^s \models C$. Then we have $R_C^s \models L$ for some literal L of C . It suffices to show that $R_N^* \models L$.

If $L = t \approx t'$ is a positive literal, then $t \downarrow_{R_C^s} t'$. Since $R_C^s \subseteq R_N^*$ by Lemma 23, this implies $t \downarrow_{R_N^*} t'$. Thus, $R_N^* \models L$.

If $L = t \not\approx t'$ is a negative literal, then $s \succ t$ and $s \succ t'$ and $t \downarrow_{R_C^s} \neq t' \downarrow_{R_C^s}$. By Lemma 23, $R_C^s|_{\prec s} = R_N^*|_{\prec s}$. Since only rules with a left-hand side smaller than s can be involved in normalizing t and t' , it follows that $t \downarrow_{R_N^*} \neq t' \downarrow_{R_N^*}$ and hence $R_N^* \models L$ (using (O3), (O4)). □

Lemma 25. *Let C be a clause. If $R_N^*|_{\prec C} \models C$, then $R_N^* \models C$.*

Proof. We assume that $R_N^*|_{\prec C} \models C$. Then we have $R_N^*|_{\prec C} \models L$ for some literal L of C . It suffices to show that $R_N^* \models L$.

If $L = t \approx t'$ is a positive literal, then $t \downarrow_{R_N^*|_{\prec C}} t'$. Since $R_N^*|_{\prec C} \subseteq R_N^*$, this implies $t \downarrow_{R_N^*} t'$. Thus, $R_N^* \models L$.

If $L = t \not\approx t'$ is a negative literal, then $t \downarrow_{R_N^*|_{\prec C}} \neq t' \downarrow_{R_N^*|_{\prec C}}$. Without loss of generality, let $t \succ t'$. Let $s \approx s'$ be the maximal term in C with $s \succeq s'$. We have $s \succ t$ if $s \approx s'$ is positive and $s \succeq t$ if $s \approx s'$ is negative. Hence, the left-hand sides of rules in $\bigcup_{D \succeq C} \Delta_D$ are larger than t . Since only rules with a left-hand side $\preceq t$ can be involved in normalizing t and t' and $R_N^*|_{\prec C} \cup \bigcup_{D \succeq C} \Delta_D = R_N^*$, it follows that $t \downarrow_{R_N^*} \neq t' \downarrow_{R_N^*}$ and hence $R_N^* \models L$. □

Superposition by a productive clause preserves the truthfulness of a clause superposed into. This is the consequence of the following lemma.

Lemma 26. *If $C \vee s \approx t$ produces $s \longrightarrow t$, then $R_N^* \not\models C$.*

Proof. Let $D = C \vee s \approx t$. By (C1) and (C5), all terms in D are $\preceq s$. By (C6), we have $R_D^s \cup \{s \longrightarrow t\} \not\models C$. The other rules $R_N^* \setminus (R_D^s \cup \{s \longrightarrow t\})$ cannot reduce C because their left-hand sides are $\succ s$. Indeed, by Lemma 23, the rules in $R_N^* \setminus (R_D^s \cup \{s \longrightarrow t\})$ all have their left-hand side greater than or equal to s and since R_N^* has no critical pairs by Lemma 19(2) these rules cannot have s as their left-hand side. Consequently, $R_D^s \cup \{s \longrightarrow t\} \not\models C$ implies $R_N^* \not\models C$. □

Lemma 27. \top and \perp are normal forms in R_N^* .

Proof. By (A1) and (C4), there is no rule that reduces \top or \perp . \square

Lemma 28. Let C be a clause. Let t be a Boolean subterm occurring in C . Suppose that one of the following two conditions holds:

- t is smaller than the maximal term in C ; or
- t is selected.

Then $R_N^*|_{<C}$ reduces t to \perp or \top .

Proof. By Lemma 19(4), R_N^* reduces t to \perp or \top . If this reduction does not contain any rule from a Δ_D with $D \succeq C$, then $R_N^*|_{<C}$ reduces t to \perp or \top as well. If this reduction does contain a rule from a Δ_D with $D \succeq C$, then by (C1) and (C5), t must be the maximal term of D and thus also of C . The term t must occur on the top level of a positive literal in D and thus also in C since by definition of the ordering, $t \approx s \prec t \not\approx s'$ for any $s, s' \prec t$. Such terms may not be selected by the selection restrictions (Definition 1). Thus we have a contradiction to the condition that t is smaller than the maximal term in C and to the condition that t is selected. \square

Lemma 29. Let C be a clause. Let t be a term that is eligible in C and reducible by $R_N^*|_{<C}$. Then t has a subterm u such that:

1. The term u is eligible in C .
2. The term u is neither \top nor \perp .
3. If the root of u is not a logical symbol, there exists a rule $u \rightarrow u' \in R_N^*|_{<C}$.
4. If the root of u is \neg , \wedge , \vee , or \rightarrow , each proper subterm of u is \top or \perp .
5. If the root of u is \approx and $u \downarrow_{R_N^*|_{<C}} = \top$, or root of u is $\not\approx$ and $u \downarrow_{R_N^*|_{<C}} = \perp$, then the two sides of the (dis)equation are equal.

Proof. We proceed by structural induction on t . First, we observe that whenever we can apply the induction hypothesis, we are done. Given an eligible, $R_N^*|_{<C}$ -reducible proper subterm t' of t , the induction hypothesis guarantees the existence of a subterm u of t' with the above five properties. Since u is then also a subterm of t and the properties do not refer to t itself, such an application of the induction hypothesis finishes the proof.

We make a case distinction on the root h of t . First, assume that h is not a logical symbol. If an argument t' of h is reducible by $R_N^*|_{<C}$, we can apply the induction hypothesis and are done. On the other hand, if no argument of h reduces, then $R_N^*|_{<C}$ must reduce t by a rule $u \rightarrow u'$ where $u = t$. Clearly, u satisfies the required five properties in this case.

Otherwise, h must be a logical symbol. We have $t \neq \top, \perp$ because Lemma 27 tells us that \top and \perp are irreducible by $R_N^* \supseteq R_N^*|_{<C}$. The cases of connectives, quantifiers and (dis)equations remain.

We assume h is \neg , \wedge , \vee , or \rightarrow . We proceed as in the case where h was not a logical symbol. If an argument is reducible by $R_N^*|_{<C}$, the induction hypothesis

applies. If none of the Boolean arguments reduce, then each of them is either \top or \perp by Lemma 28. This takes care of property 4 as we choose $u = t$. Since t is eligible and the root of $t = u$ is a connective, the other properties are fulfilled as well.

We assume h is \forall or \exists . Choose $u = t$. The required properties are easy to check because t is eligible and the root of $t = u$ is \forall or \exists .

Finally, we assume $h = \approx$ (the case $h = \not\approx$ is analogous). If a strictly larger argument t' of h reduces by $R_N^*|_{\prec C}$, then t' is eligible and the induction hypothesis applies to t' . Otherwise we choose $u = t$. Property 5 holds because if $u \downarrow_{R_N^*|_{\prec C}} = \top$, then the two arguments of h must have the same $R_N^*|_{\prec C}$ -normal form, since case (v) of Definition 18 is the only rule that reduces a term rooted by \approx to \top . Thus, the two arguments of h must be equal because otherwise the strictly larger one would be reducible by $R_N^*|_{\prec C}$. The other properties hold because t is eligible and the root of $t = u$ is \approx . \square

We employ Bachmair and Ganzinger's framework of reducing counterexamples [4, Sect. 4.2]. The interpretation R_N^* is our candidate model. A clause $C \in N$ is called a *counterexample* if $R_N^* \not\models C$. It is a *minimal* counterexample if C is the smallest clause in N w.r.t. \succ such that $R_N^* \not\models C$. An inference *reduces* a counterexample C if its main premise is C , its side premises are true in R_N^* , and its conclusion D is a counterexample smaller than C . An inference system has the *reduction property for counterexamples* if for all clause sets N with a minimal counterexample C , there exists an inference from N that reduces C .

The next lemma deals with counterexamples reducible by SUP or its specialized cousins BOOLHOIST, \approx HOIST, $\not\approx$ HOIST, G \forall HOIST, G \exists HOIST, BOOLRW, \forall RW, and \exists RW.

Lemma 30. *Assume $R_N^* \not\models C[s]_p \in N$, such that*

- (a) *p is eligible in C ,*
- (b) *s is reducible by $R_N^*|_{\prec C}$, and*
- (c) *if p is a topmost position of a positive literal and the root of s is not a logical symbol, s must be reducible by R_C^s .*

Then our inference system reduces the counterexample C .

Proof. We apply Lemma 29 to s to find an appropriate subterm u of s .

Case 1: We assume that the root of u is not a logical symbol. Then, by property 3 of Lemma 29, there exists a rule $u \rightarrow u' \in R_N^*|_{\prec C}$ for some u' . We can apply BOOLHOIST or SUP to reduce the counterexample as follows:

Case 1.1: We assume that if $u \rightarrow u' \in R_C$, then $u' = \perp$ and p is not a topmost position of a positive literal in C .

We check that BOOLHOIST is applicable:

- The root of u is not a logical symbol as required.
- The position of u in $C[u]$ is eligible by property 1 of Lemma 29.
- The term u is ground and therefore not a variable.

- Finally, we need to show that the position of u is not a topmost position of a positive literal in C . If it was, then necessarily $s = u$ and by condition (c) of this lemma, it follows that $s = u$ is reducible by R_C^s . By the assumption of case 1.1 and the fact that $u \rightarrow u' \in R_N^*|_{\prec C}$, we have $u \rightarrow u' \in R_N^* \setminus R_N$. The rules in $R_C^s \setminus R_N$ have left-hand sides smaller than $s = u$. So $u \rightarrow u' \in R_N^* \setminus R_N \setminus R_C^s$. Since $s = u$ is reducible by R_C^s , this contradicts the absence of critical pairs in R_N^* , which we have shown in Lemma 19(2).

Case 1.2: We assume that case 1.1 does not apply. That means that $u \rightarrow u' \in R_C$ and if $u' = \perp$, then p is a topmost position of a positive literal in C .

Then some clause $D \vee u \approx u' \in N$ smaller than C produces the rule $u \rightarrow u'$. We claim that the counterexample C is reduced by the superposition inference

$$\frac{D \vee u \approx u' \quad C[u]}{D \vee C[u']}$$

This superposition is a valid inference:

- Unification is trivial.
- The term u is ground and therefore not a variable.
- We have $u \succ u'$ by (C1).
- $D \vee u \approx u' \prec C[u]$ due to property 3 of Lemma 29.
- The position of u in $C[u]$ is eligible by property 1 of Lemma 29.
- The literal $u \approx u'$ is eligible in $D \vee u \approx u'$ by (C3). It is strictly eligible because if $u \approx u'$ also occurred as a literal in D , we would have $R_{D \vee u \approx u'}^s \cup \{u \rightarrow u'\} \models D$, in contradiction to (C6).
- The root of u is not a logical symbol by the assumption of case 1.
- If $u' = \perp$, then p is at the top level of a positive literal by the assumption of case 1.2.

As $D \vee u \approx u'$ is productive, $R_N^* \not\models D$ by Lemma 26. Hence the conclusion $D \vee C[u']$ is equivalent to $C[u']$, which is equivalent to $C[u]$ w.r.t. R_N^* . It remains to show that the new counterexample $D \vee C[u']$, which C is transformed into, is strictly smaller than C . By (O4), $C[u'] \prec C$ because $u' \prec u$ and $D \prec C$ because $D \vee u \approx u' \prec C$. Thus, the counterexample C reduces.

Case 2: We assume that the root of u is a logical symbol. By property 2 of Lemma 29, $u \neq \top, \perp$. By Lemma 19(4), R_N^* reduces u to \top or to \perp .

- Case 2.1: The root of u is \neg, \wedge, \vee , or \rightarrow ; or the root of u is \approx and it reduces to \top ; or the root of u is \neq and it reduces to \perp . We apply BOOLRW. Only two of its side conditions are relevant on ground clauses, the conditions 1 and 4. For the condition 1 we must pick the right item from the list. By points 4 and 5 of Lemma 29, the list contains an applicable item. Eligibility of u in C (condition 4) holds by property 1 of Lemma 29. Clearly, the conclusion is false in R_N^* and smaller than C .
- Case 2.2: The root of u is \approx , say $u = s \approx t$, and u reduces to \perp . We apply \approx HOIST:

$$\frac{C[s \approx t]}{C[\perp] \vee s \approx t} \approx\text{HOIST}$$

Clearly, the inference conditions are fulfilled and the conclusion is smaller than C because $C[\perp] \prec C$ and the literal in C where $s \approx t$ occurs is greater than $s \approx t$. The reduction $u \rightarrow_{R_N^*}^+ \perp$ necessarily has the form

$$s \approx t \rightarrow_{R_N^*}^* s' \approx t' \rightarrow_{\Delta_{R_N^*}^{s' \approx t'}} \perp$$

because the final step is the only way to reduce \approx . Here, s' and t' are different R_N^* normal forms. Hence $R_N^* \not\vdash s' \approx t'$, thus $R_N^* \not\vdash s \approx t$ and $R_N^* \not\vdash C[\perp] \vee s \approx t$.

- Case 2.3: The root of u is \neq and it reduces to \top . Analogous to the previous case, using \neq HOIST.
- Case 2.4: The root of u is \forall and it reduces to \top . We apply G \forall RW. Clearly, the rule is applicable and its conclusion is false in R_N^* and smaller than C .
- Case 2.5: The root of u is \exists and it reduces to \perp . Analogous to the previous case, using G \exists RW.
- Case 2.6: The root of u is \forall , say $u = \forall x.t$, and u reduces to \perp . We apply G \forall HOIST:

$$\frac{C[\forall x.t]}{C[\perp] \vee \{x \mapsto v\}t \approx \top} \text{G}\forall\text{HOIST}$$

for a term v selected as described here: The reduction $u \rightarrow_{R_N^*}^+ \perp$ necessarily has the form $\forall x.t \rightarrow \perp$ and originates from case (A3) of Definition 18 because rewriting under quantifiers is forbidden. In particular, case (vii) of Definition 18 does not apply. Hence there exists a ground term v whose Boolean subterms are only \top and \perp such that $\{x \mapsto v\}t \rightarrow \perp$. This is the v we choose for G \forall HOIST. Clearly, the inference conditions are fulfilled and the conclusion is smaller than C . Then the conclusion is a strictly smaller counterexample.

- Case 2.7: root of u is \exists and it reduces to \top . Analogous to the previous case, using G \exists HOIST.

□

Lemma 31. *Our ground inference system has the reduction property for counterexamples.*

Proof. Let N be a set of ground clauses that does not contain the empty clause. Let C be a minimal counterexample for R_N^* in N . We must show that there is an inference from N that reduces C —i.e., the inference has main premise C , side premises in N , and a conclusion that is a smaller counterexample for R_N^* than C . In all the following cases, there is an inference that reduces C .

1. We assume that C contains a selected Boolean subterm. Then it cannot be at a topmost position of a positive literal by the selection restrictions (Definition 1). By Lemma 28, $R_N^*|_{\prec C}$ reduces the selected subterm. Hence we can apply Lemma 30 to that subterm and are done.
2. We assume that there is an eligible literal of the form $s \neq s \in C$. Then IRREFL reduces C .

3. We assume that there is an eligible literal $s \not\approx s' \in C$ where $s \succ s'$. Since $R_N^* \not\models C$, we have $R_N^*|_{\prec C} \not\models C$ by Lemma 25. Therefore $R_N^*|_{\prec C} \not\models s \not\approx s'$ and $R_N^*|_{\prec C} \models s \approx s'$. Thus, s must be reducible by $R_N^*|_{\prec C}$ because $s \succ s'$. Therefore, we can apply Lemma 30 to s .
4. We assume that $C = C'' \vee s \approx t \vee s \approx s'$ such that:
 - no literals or Boolean subterms are selected in C ;
 - $s \approx s'$ is maximal in C ;
 - $s \succ s', t$; and
 - $R_N^* \models s' \approx t$.
 Then we can apply

$$\frac{C = C'' \vee s \approx t \vee s \approx s'}{C'' \vee s' \not\approx t \vee s \approx t} \text{FACTOR}$$

- Since C is false in R_N^* , we have $R_N^* \models s \not\approx t$. Since moreover $R_N^* \models s' \approx t$, it follows that the conclusion of this inference must be false in R_N^* . Since $s \succ s', t$ and $s \approx s'$ is maximal, the above inference reduces C .
5. We assume that there is a selected literal $s \approx \perp \in C$ with $s \succ \perp$. Since C is false in R_N^* , C is also false in $R_N^*|_{\prec C}$ by Lemma 25. Hence $R_N^*|_{\prec C} \models s \approx \top$. If the root of s is not a logical symbol, since s reduces to \top , s must thus be reducible by R_C . Therefore, we can apply Lemma 30.
 6. We assume that there is a strictly eligible literal $s \approx s'$ where s is reducible by R_C^s . Hence, s is eligible in C . Thus, we can again apply Lemma 30 to s .
 7. We assume that there is a strictly eligible literal $s \approx s' \in C$ where $s \succ s'$ and the root of s is a logical symbol. By (O1), $s = \top$ contradicts $s \succ s'$. If $s = \perp$, then $s' = \top$ by (O1), and \perp ELIM reduces C . If $s \neq \top, \perp$ then s is reducible by Lemma 19(4). Thus, we can again apply Lemma 30 to s .

We will now show that one of the above cases applies or the clause C is productive, which would be a contradiction to $R_N^* \not\models C$.

First, we assume that a Boolean subterm or a literal is selected in C . If a Boolean subterm is selected, case 1 applies. If a literal is selected, it is either negative or of the form $s \approx \perp$ by the selection restrictions. If it is negative, case 2 or 3 applies. If it is of the form $s \approx \perp$, s cannot be \perp because C is false in R_N^* . If $s = \top$, then case 7 applies. If $s \succ \perp$, then case 5 applies.

Now we may assume that C contains no selections. Then the maximal literal must be eligible. If the maximal literal is negative ((C3) does not hold), case 2 or 3 applies. If (C1) does not hold, the literal must be negative because C is true in R_N^* . If (C2) does not hold, then by Lemma 24, the maximal literal must be negative.

So we may assume that C contains no selections and the maximal literal is positive. If (C6) does not hold, then $R_C^s \cup \{s \rightarrow s'\} \models C'$ where C' is the subclause of C with the maximal literal removed. However, $R_C^s \not\models C$ by Lemma 24 and by the assumption $R_N^* \not\models C$. Therefore, $R_C^s \not\models C'$. Thus, we must have $C' = C'' \vee r \approx t$ for some terms r and t where $R_C^s \cup \{s \rightarrow s'\} \models r \approx t$ and $R_C^s \not\models r \approx t$. So $r \neq t$ and without loss of generality we assume $r \succ t$. Moreover

$s \longrightarrow s'$ must participate in the normalization of r or t by $R_C^s \cup \{s \longrightarrow s'\}$. Since $r \preceq s \succ s'$, the rule $s \longrightarrow s'$ can only be used as the first step in the normalization of r . Hence $r = s$ and $R_C^s \models s' \approx t$. Then case 4 applies. In particular, it applies if the maximal literal is not strictly maximal.

Now we may assume that C contains no selections and the maximal literal is strictly maximal and positive. If (C7) does not apply, then case 6 applies. If (C4) does not apply, then case 7 applies. \square

Theorem 32 (Ground static refutational completeness). *Let $N \subseteq \mathcal{C}_G$ be a set saturated w.r.t. $GInf^q$ and $GRed_1^q$. Then $N \models \perp$ if and only if $\perp \in N$.*

Proof. By Theorem 4.9 of Bachmair and Ganzinger's framework [4] and Lemma 31. \square

5.2 Nonground Layer

The main effort in passing from the ground static to the nonground dynamic completeness consists of the lifting of the inferences. We do this in the lemma 34. However first we rephrase a main result from the saturation framework [33] which lifts completeness given the lifting of the inferences. We specialize to our calculi F , G and grounding Γ but formulate the theorem in a way which still hints the generalization.

Recall that a clause D is redundant w.r.t. a clause set N if its ground instances are, symbolically $\Gamma D \subseteq GRed_C(\Gamma N)$, or if D is strictly subsumed by a clause from N . This is equivalent to the condition given in the next theorem. Namely if there exists $C \in N$ subsuming D , then $D \sqsupset C$ and $\Gamma D \subseteq \Gamma C$ so that D is redundant by the both conditions. Otherwise D is not subsumed and $\Gamma\{C \in N \mid D \sqsupset C\} = \emptyset$ which reduce both conditions to the same form $\Gamma D \subseteq GRed_C(\Gamma N)$.

The inference component of the redundancy posed on F by the below theorem is actually stronger, albeit unusefully, than the one in our definition 13. This is because it considers only a single ground calculus whose parameters (selection and witness functions) will be fixed indirectly during the lifting of the inferences. Weakening a redundancy test trivially preserves completeness. Below we write $FInf N$ for the set of inferences from a clause set N .

Theorem 33. *Let the ground calculus G , which our calculus F corresponds to, be statically complete. Let redundancy on F be given by*

$$\begin{aligned} i \in FRed_I N &\iff \Gamma i \subseteq GRed_I(\Gamma N) \\ D \in FRed_C N &\iff \Gamma D \subseteq GRed_C(\Gamma N) \cup \Gamma\{C \in N \mid D \sqsupset C\} \end{aligned}$$

where \sqsupset is subsumption (or generally any well-founded) strict order. This gives a valid redundancy criterion for F . Assume that every inference i from ΓN is liftable, meaning $i \in \Gamma(FInf N)$, or redundant w.r.t. ΓN , for any F -clause set N . Then F is dynamically complete.

Proof. See theorems of Waldmann et al. [33].

Lemma 34. *Let N be an F -clause set. There exists ground selection functions and a witness function such that every ground inference $i \in G\text{Inf}(\Gamma N)$ from the grounded clause set ΓN is liftable, meaning $i \in \Gamma(F\text{Inf } N)$, or redundant, meaning $i \in G\text{Red}_I(\Gamma N)$.*

Proof. We begin by fixing the ground selection and witness functions. Let $FLSel$ be a literal selection function on F . To choose the right selection function $GLSel \in \Gamma(FLSel)$, we observe that each ground clause $C \in \Gamma N$ 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 $C \in \Gamma N$, which we denote by $LC \in N \cap \Gamma^{-1}\{C\}$ and say that L does ungrounding. Then let $GLSel$ select those literals in C that correspond to the literals selected by $FLSel$ in LC . We define a Boolean subterm selection function $GBSel$ similarly.

To fix a witness function, we consider a $G\exists RW$ inference

$$\frac{C[\exists p]}{C[p\mathbf{w}(C, \exists p)]} G\exists RW$$

Lifting this (or any other inference) requires us to take as a premise the ungrounding $LC[\exists p]$ of $C[\exists p]$ which we fixed above. This way the selection side conditions are satisfied. So the lift of the $G\exists RW$ must be an $\exists RW$ from $LC[\exists p]$. Let γ be a grounding substitution s.t. $C[\exists p] = \gamma LC[\exists p]$. By definition of grounding, γ can not introduce a quantifier term $\exists p$. Hence the $\exists p$ corresponds to a subterm $\exists \tilde{p}$ of $LC[\exists p]$. Now by grounding the conclusion from the $\exists RW$ we have $p\mathbf{w}(C, \exists p) = \gamma \tilde{p}(\text{sk}_{\exists \tilde{p}} \tilde{z}) = p(\text{sk}_{\exists \tilde{p}} \gamma \tilde{z})$ where \tilde{z} are variables from $LC[\exists p]$. This defines the witness function \mathbf{w} pointwise or leaves it unconstrained when p discards its input. (It is a nontrivial but purely technical addition to simultaneously deal with rewriting \forall quantifiers.)

We proceed to present the lifting of superposition inferences. Other inferences involving contexts (\star HOIST and remaining $\star RW$) are liftable by similar arguments whereas IRREFL, FACTOR and \perp ELIM by simpler ones. All these we skip. Let $D \vee t \approx t'$ and $C[t]$ be ground clauses from ΓN with a superposition inference between them:

$$\frac{D \vee t \approx t' \quad C[t]}{D \vee C[t']}$$

To lift this inference, we are looking for an inference between the ungroundings $L(D \vee t \approx t')$ and $LC[t]$ as fixed above. Since a substitution can not introduce the symbols \vee or \approx , we can write $L(D \vee t \approx t') = \tilde{D} \vee \tilde{t} \approx \tilde{t}'$ with $\theta \tilde{D} = D$, $\theta \tilde{t} = t$ and $\theta \tilde{t}' = t'$ for some grounding θ . Since a substitution can introduce a context boundary, we can only write $LC[t] = \tilde{C}[u]$ where $t = \gamma u$ or u is a variable and t is a subterm of γu , for some grounding γ mapping $LC[t]$ to $C[t]$. This distinguishes whether the superposition is below a variable or not. Moreover the distinguished t and u correspond, meaning that $\gamma \tilde{C}[\cdot]$ can be seen as a context around $C[\cdot]$.

If u is not a variable, $t = \gamma u$ so that t and u have the same root symbol. This can not be a logical symbol because otherwise the ground superposition

would not exist. Since $\theta\tilde{t} = t = \gamma u$, terms \tilde{t} and u are unifiable. Routinely one can check other side conditions too to conclude that

$$\frac{\tilde{D} \vee \tilde{t} \approx \tilde{t}' \quad \tilde{C}[u]}{\text{mgu}(\tilde{t}, u)(\tilde{D} \vee \tilde{C}[\tilde{t}'])}$$

is a nonground superposition inference which lifts the original ground superposition.

If u is a variable, the ground superposition will be redundant. Let $g[t] = \gamma u$ and define $\gamma' = \gamma \circ \{u \mapsto g[t']\}$. Now $C[t] = \gamma\tilde{C}[u] \succ \gamma'\tilde{C}[u] \in \Gamma N$ because $t \succ t'$, and by side condition of superposition $C[t] \succ D \vee t \approx t'$ also. Together $\gamma'\tilde{C}[u]$ and $D \vee t \approx t'$ imply the conclusion $D \vee C[t']$ of the ground superposition because if D is false then $D \vee t \approx t'$ equates the groundings γ and γ' .

As a corollary of the two previous results and the static completeness theorem 32 we have:

Theorem 35. *Our nonground calculus F is complete.*

6 Inprocessing Clausification Methods

Our calculus makes preprocessing clausification unnecessary: A problem specified by a formula f can be represented as a clause $f \approx \mathbf{T}$. Our redundancy criterion allows us to add various sets of rules to steer the inprocessing clausification.

Without any additional rules, our core calculus rules perform all the necessary reasoning about formulas. We call this method *inner delayed clausification* because the calculus rules tend to operate on the inner Boolean subterms first.

The *outer delayed clausification* method adds the following rules to the calculus, which are guided by the outermost logical symbols. Let s and t be Boolean terms. Below, we let s^+ range over literals of the form $s \approx \mathbf{T}$ and $s \not\approx \mathbf{\perp}$, and s^- over literals of the form $s \approx \mathbf{\perp}$ and $s \not\approx \mathbf{T}$.

$$\begin{array}{c} \frac{s^+ \vee C}{oc(s, C)} +\text{OUTERCLAUS} \quad \frac{s^- \vee C}{oc(\neg s, C)} -\text{OUTERCLAUS} \\ \\ \frac{s \approx t \vee C}{s \approx \mathbf{\perp} \vee t \approx \mathbf{T} \vee C \quad s \approx \mathbf{T} \vee t \approx \mathbf{\perp} \vee C} \approx\text{OUTERCLAUS} \\ \\ \frac{s \not\approx t \vee C}{s \approx \mathbf{\perp} \vee t \approx \mathbf{\perp} \vee C \quad s \approx \mathbf{T} \vee t \approx \mathbf{T} \vee C} \not\approx\text{OUTERCLAUS} \end{array}$$

The rules $+\text{OUTERCLAUS}$ and $-\text{OUTERCLAUS}$ are applicable to any term s whose root is a logical symbol, whereas the rules $\approx\text{OUTERCLAUS}$ and $\not\approx\text{OUTERCLAUS}$ are only applicable if neither s nor t is \mathbf{T} or $\mathbf{\perp}$. Clearly, our redundancy criterion allows us to replace the premise of all OUTERCLAUS -rules with their conclusions. Nonetheless, the rules $\approx\text{OUTERCLAUS}$ and

$\not\approx$ OUTERCLAUS are not used as simplification rules since destructing equivalences disturbs the syntactic structure of the formulas, as noted by Ganzinger and Stuber [13]. The behavior of $oc(s, C)$ on a formula s and a clause C is as follows:

$$\begin{aligned}
 oc(s \wedge t, C) &= \{s \approx \top \vee C, t \approx \top \vee C\} \\
 oc(s \vee t, C) &= \{s \approx \top \vee t \approx \top \vee C\} \\
 oc(s \rightarrow t, C) &= \{s \approx \perp \vee t \approx \top \vee C\} \\
 oc(s \approx t, C) &= \{s \approx t \vee C\} \\
 oc(s \not\approx t, C) &= \{s \not\approx t \vee C\} \\
 oc(\forall z. s, C) &= \{\{z \mapsto y\}s \approx \top \vee C\} \\
 oc(\exists z. s, C) &= \{\{z \mapsto \text{sk}_{\forall \bar{y}. \exists z. s}(\bar{y})\}s \approx \top \vee C\} \\
 oc(\neg(s \wedge t), C) &= \{s \approx \perp \vee t \approx \perp \vee C\} \\
 oc(\neg(s \vee t), C) &= \{s \approx \perp \vee C, t \approx \perp \vee C\} \\
 oc(\neg(s \rightarrow t), C) &= \{s \approx \top \vee C, t \approx \perp \vee C\} \\
 oc(\neg(s \approx t), C) &= \{s \not\approx t \vee C\} \\
 oc(\neg(s \not\approx t), C) &= \{s \approx t \vee C\} \\
 oc(\neg(\neg s), C) &= oc(s, C) \\
 oc(\neg(\forall z. s), C) &= \{\{z \mapsto \text{sk}_{\forall \bar{y}. \exists z. \neg s}(\bar{y})\}s \approx \perp \vee C\} \\
 oc(\neg(\exists z. s), C) &= \{\{z \mapsto y\}s \approx \perp \vee C\}
 \end{aligned}$$

In the case of $oc(\forall z. s, C)$ or $oc(\neg(\exists z. s), C)$, y is a variable not appearing in s or C ; in the case of $oc(\exists z. s, C)$ and $oc(\neg(\forall z. s), C)$, \bar{y} are all free variables appearing in $\exists z. s$ or $\forall z. s$, respectively. If $\exists z. s = \sigma(\exists z. t)$ for some substitution σ and the Skolem term $\text{sk}_{\forall \bar{x}. \exists z. t}$ has been used previously, instead of $\text{sk}_{\forall \bar{y}. \exists z. s}(\bar{y})$, we can use $\sigma(\text{sk}_{\forall \bar{x}. \exists z. t}(\bar{x}))$ where \bar{x} are all free variables of $\exists z. t$.

A third inprocessing clausification method is *immediate clausification*. It first preprocesses the input problem using a standard first-order clausification procedure such as Nonnengart and Weidenbach's [23]. Then, during the proof search, when a clause C appears on which OUTERCLAUS rules could be applied, we apply the standard clausification procedure on the formula $\forall \bar{x}. C$ instead (where \bar{x} are the free variables of C), and replace C with the clausification results. With this method, the formulas are clausified in one step, making intermediate clausification results inaccessible to the simplification machinery.

Renaming Common Formulas Following Tseitin [29], clausification procedures usually rename common formulas to prevent a possible combinatorial explosion caused by naive clausification. In our two delayed clausification methods, we realize this idea using the following rule:

$$\frac{C_1[\sigma_1 f] \quad \cdots \quad C_n[\sigma_n f]}{C_1[\sigma_1 \mathfrak{p}(\bar{x})] \quad \cdots \quad C_n[\sigma_n \mathfrak{p}(\bar{x})] \quad R_1 \quad \cdots \quad R_m} \text{RENAME}$$

Here, the formula f has a logical root, \bar{x} are the distinct free variables in f , \mathbf{p} is a fresh symbol, σ_i is a substitution, and the clauses R_1, \dots, R_m are the result of simplifying a *definition clause* $R = \mathbf{p}(\bar{x}) \approx f$ as described below. The rule avoids exponential explosion by replacing n positions in which results of f 's classification will appear into a single position in R . Optimizations such as polarity-aware renaming [23, Sect. 4] also apply to RENAME.

Several issues arise with RENAME as an inprocessing rule. We need to ensure that in R , $f \succ \mathbf{p}(\bar{x})$, since otherwise demodulation might reintroduce a formula f in the simplified clauses. This can be achieved by giving the fresh symbol \mathbf{p} a precedence smaller than that of all symbols initially present in the problem (other than \mathbf{T} and \mathbf{F}). To ensure the precedence is well founded, the precedence of \mathbf{p} must be greater than that of symbols previously introduced by the calculus. For KBO, we additionally set the weight of \mathbf{p} to the minimal possible weight.

For RENAME to be used as a simplification rule, we need to ensure that the conclusions are smaller than the premises. This is trivially true for all clauses other than the clause R . For example, let $C_i = f \approx \mathbf{T}$ (σ_i is the identity). Clearly, R is larger than C_i . However, we can view the definition clause R as two clauses $R^+ = \mathbf{p}(\bar{x}) \approx \mathbf{F} \vee f \approx \mathbf{T}$ and $R^- = \mathbf{p}(\bar{x}) \approx \mathbf{T} \vee f \approx \mathbf{F}$. Then, we can apply a single step of the OUTERCLAUS rules to R^+ and R^- (on their subformula f), which further results in clauses R_1, \dots, R_m . Inspecting the OUTERCLAUS rules, it is clear that $m \leq 4$, which makes enforcing this simplification tolerable. Furthermore, as f is simplified in each of R_1, \dots, R_m , they are smaller than any premise C_i .

Another potential source of a combinatorial explosion in our calculus are formulas that occur deep in the arguments of uninterpreted predicates. Consider the clause $C = \mathbf{p}^i(x) \approx \mathbf{T} \vee \mathbf{q}^j(y) \approx \mathbf{T}$ where $i, j > 2$, and an exponent n denotes that the symbol is applied n times. Under some parameter choices, such as the one in which the first and the second literal are eligible in C , any clause $\mathbf{p}^{i_1}(x) \approx \mathbf{T} \vee \mathbf{p}^{i_2}(\mathbf{F}) \approx \mathbf{T} \vee \dots \vee \mathbf{p}^{i_k}(\mathbf{F}) \approx \mathbf{T} \vee \mathbf{q}^{j_1}(y) \approx \mathbf{T} \vee \mathbf{q}^{j_2}(\mathbf{F}) \approx \mathbf{T} \vee \dots \vee \mathbf{q}^{j_l}(\mathbf{F}) \approx \mathbf{T}$ (where $i_1 + \dots + i_k = i$ and $j_1 + \dots + j_l = j$), resulting from multiple BOOLHOIST applications, can be obtained in many different ways. This explosion can be avoided using the following rule:

$$\frac{s \approx t \vee C}{\mathbf{p}(\bar{x}) \approx \mathbf{T} \vee C \quad R_1 \quad \dots \quad R_4} \text{RENAMEDEEP}$$

where \mathbf{p} is a fresh symbol, \bar{x} are all free variables occurring in $s \approx t$, the clauses R_1, \dots, R_4 result from simplifying $R = \mathbf{p}(\bar{x}) \approx (s \approx t)$ as described above, and we impose the same precedence and weight restrictions on \mathbf{p} as for RENAME. Finally, we require that both $s \approx t$ and C contain deep Booleans where a Boolean subterm $u|_p$ of a term u is a *deep Boolean* if there are at least two distinct proper prefixes q of the position p such that the root of $u|_q$ is an uninterpreted predicate.

Similarly to `RENAME`, the definition clause R can be larger than the premise. As `OUTERCLAUS`-rules might not apply to $s \approx t$, we need a different solution:

$$\frac{\frac{C[u]}{\frac{}{C[\perp] \vee u \approx \top} \quad C[\top] \vee u \approx \perp}}{\text{BOOLHOISTSIMP}}}$$

In this rule u is a non-variable Boolean subterm, different from \top and \perp , whose indicated occurrence is not in a literal $u \approx b$ where b is \top , \perp or a variable. Clearly, both conclusions of `BOOLHOISTSIMP` are smaller than the premise. As before, observing that R is equivalent to two clauses $R^+ = \mathfrak{p}(\bar{x}) \approx \perp \vee s \approx t$ and $R^- = \mathfrak{p}(\bar{x}) \approx \top \vee s \not\approx t$, we simplify R^+ and R^- into clauses that are guaranteed to be smaller than the premise. This is achieved by applying `BOOLHOISTSIMP` to one of the deep Boolean occurrences in both R^+ and R^- , which produces R_1, \dots, R_4 and reduces the size of resulting clauses enough for them to be smaller than the premise of `RENAMEDEEP`. The `RENAMEDEEP` rule can be applied analogously to negative literals $s \not\approx t$.

Miniscoping Both our core inferences and the clausification rules use Skolemization to remove quantifiers. The arity of a created Skolem symbol is the number of the free variables in the replaced quantification term. These arities can be reduced by shrinking the quantification terms by moving the quantifiers inwards when possible. To this end, the miniscoping transformation [23] exhaustively rewrites

$$\begin{array}{ll} \forall(f \wedge g) \longrightarrow \forall f \wedge \forall g & \forall x. f \vee gx \longrightarrow f \vee \forall g \\ \exists(f \vee g) \longrightarrow \exists f \vee \exists g & \exists x. f \wedge gx \longrightarrow f \wedge \exists g \end{array}$$

and their commuted versions and versions for other connectives, and it removes unused quantification around constant expressions.

We can straightforwardly do inprocessing miniscoping if and only if the term order orients these equivalences as indicated. This is the case with LPO, as given in Sect. 3.1, because high-precedence symbols $\forall, \exists > \wedge, \vee$ get moved inwards. On the other hand KBO with our encoding always orients the left-hand side rules in the undesired reverse order.

7 Implementation

Zipperposition [11] is an automatic theorem prover designed for easy prototyping of various extensions of superposition. So far, it has been extended to support induction, arithmetic, and various fragments of higher-order logic. We have implemented our calculus and its extensions described above in Zipperposition.

Zipperposition has long supported λ as the only binder. Because introducing new binders would significantly complicate the implementation, we decided to represent the terms $\forall x. t$ and $\exists x. t$ as $\forall(\lambda x. t)$ and $\exists(\lambda x. t)$, respectively.

We introduced a normalized presentation of predicate literals as either $s \approx \mathbf{T}$ or $s \approx \mathbf{\perp}$. As Zipperposition previously encoded them as $s \approx \mathbf{T}$ or $s \not\approx \mathbf{T}$, enforcing the new encoding was a source of tedious implementation effort.

FACTOR inferences happen even when the maximal literal is selected since the discovery of condition (3) as described in Sect. 3 came after the evaluation.

Zipperposition’s existing selection functions were not designed with Boolean subterm selection in mind. For instance, a function that selects a literal L with a selectable Boolean subterm s can make s eligible, even if the Boolean selection function did not select s . To mitigate this issue, we can optionally block selection of literals that contain selectable Boolean subterms.

We implemented four Boolean selection functions: selecting the leftmost innermost, leftmost outermost, syntactically largest or syntactically smallest selectable subterm. Ties are broken by selecting the leftmost term. Additionally, we implemented a Boolean selection function that does not select any subterm.

Vukmirović and Nummelin [31, Sect. 3.4] explored inprocessing clausification as part of their pragmatic approach to higher-order Boolean reasoning. They describe in detail how the formula renaming mechanism is implemented. We reuse their mechanism, and simplify definition clauses as described in Sect. 6.

8 Evaluation

The goal of our evaluation was to answer the following questions:

1. How does our approach compare to preprocessing?
2. How do the different inprocessing clausification methods compare?
3. Is there an overhead of our calculus on problems without first-class Booleans?
4. What effect do Boolean selection, LOCALRW, and BOOLHOISTSIMP have?

We filtered TPTP [27] and SMT-LIB [6] to get first-order benchmarks that actually do use the Boolean type. In TPTP THF we found 145 such problems (*TPTP Bool*) and in the UF section of SMT-LIB 5507 such problems. Martin Desharnais and Jasmin Blanchette generated 1253 Sledgehammer problems that target our logic. To measure the overhead of our calculus, we randomly chose 1000 FOF and CNF problems from the TPTP (*TPTP FO*). Even with this sample the experiment could take up to $(145+5507+1253+1000) \times \#modes \times 300s \approx 9$ CPU months. On StarExec servers, evaluation roughly took three days under low load. Otherwise evaluating on all 13 000 FOF and CNF problems could have taken 2.5 times longer.

SMT-LIB interprets the symbol `ite` as the standard if-then-else function [6, Sect. 3.7.1]. Whenever a term $s = \text{ite}(t_1, t_2, t_3)$ of type τ occurs in a problem, we replace s with $f_\tau(t_1, t_2, t_3)$, where f_τ is a fresh symbol denoting the `ite` function of a particular return type. To comply with SMT-LIB, we add the following axioms: $\forall x y. f_\tau(\mathbf{T}, x, y) \approx x$ and $\forall x y. f_\tau(\mathbf{\perp}, x, y) \approx y$. SMT-LIB allows the use of `let` variable bindings [6, Sect. 3.6.1]. We simply replace each variable with its definition in the body of the `let` bindings.

Currently, among competing superposition-based provers only E and Vampire support first-order logic with interpreted Booleans, and they do so through preprocessing. We could not evaluate Vampire in first-order mode with FOOL preprocessing because as soon as a problem contains higher-order constructs, as do the TPTP Bool benchmarks, Vampire must be used in higher-order mode to be sound. We were able to run E on all benchmarks, except for the ones in SMT syntax.

We used Zipperposition’s first-order portfolio, which invokes the prover sequentially with up to 13 configurations in different time slices. In each time slice, the prover is invoked anew, without any knowledge of previous invocations. To compare different features, we ran different *modes* that enable a given feature in all of the portfolio configurations. All experiments were performed on the StarExec Iowa servers [26], equipped with Intel Xeon E5-2609 0 CPUs clocked at 2.40 GHz. We set the CPU time limit to 300s. Figure 1 displays the results. An empty cell indicates that a mode is not evaluated on that benchmark set. An archive with the raw evaluation data is publicly available.⁴

A preprocessing transformation that removes all Boolean subterms occurring as arguments of symbols [32, Sect. 8], similar to Kotelnikov et al.’s FOOL clausification approach [16], is implemented in Zipperposition. To answer question 1, we enabled preprocessing and compared it to our new calculus parameterized with the Boolean selection function that selects the smallest selectable subterm. The mode using our new calculus performs immediate inprocessing clausification, and we call it *base*, while the mode that preprocesses Boolean subterms is denoted by *preprocess* in Figure 1.

The obtained results do not give a conclusive answer to question 1. On both TPTP Bool and Sledgehammer problems, some configuration of our new calculus manages to prove one problem more than preprocessing. On SMT-LIB benchmarks, the best configuration of our calculus matches preprocessing. This shows that our calculus already performs roughly as well as previously known techniques and suggests that it will be able to outperform preprocessing techniques after tuning of its parameters.

For context, we provide the evaluation of E on supported benchmarks. On TPTP FO benchmarks it solves 643 problems, on TPTP Bool benchmarks 144 problems, and on Sledgehammer benchmarks 674 problems. Note that there is no straightforward way to compare these results with Zipperposition.

Our *base* mode uses immediate inprocessing clausification. To answer question 2, we compared *base* with a variant of *base* with outer delayed clausification (*base+outer*) and with a variant with inner delayed clausification (*base+inner*). In the delayed modes, we invoke the RENAME rule on formulas that are discovered to occur more than four times in the proof state. The distribution of the solution counts between these three modes is displayed in the area proportional Figure 2.

The results show that inner delayed clausification, which performs the laziest form of clausification, gives the worst results on most benchmark sets. Outer

⁴ <https://doi.org/10.5281/zenodo.4550787>

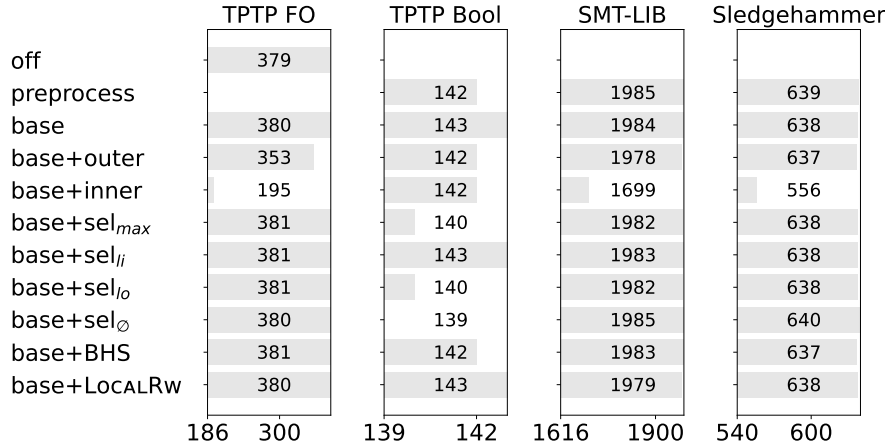


Fig. 1: Number of problems solved per benchmark set and Zipperposition mode. The x-axis starts from the number of problems solved by all evaluated modes.

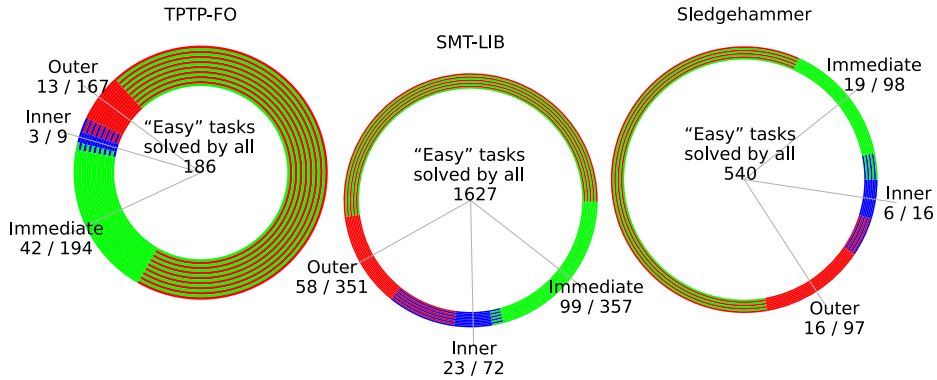


Fig. 2: Counts of unique/uneasy solutions per classification mode and benchmark

delayed classification performs roughly as well as immediate classification on problems targeting our logic. On purely first-order problems, it performs slightly worse than immediate classification. However, outer delayed classification solves 17 problems not solved by immediate classification on these problems. This suggests that it opens new possibilities for first-order reasoning that need to be explored further with specialized strategies and additional rules.

We found a problem with a conjecture of the form $s \rightarrow s$ that only the delayed classification modes can prove: a large TPTP software verification problem, SWV122+1. As s contains many quantified subformulas, subformula renaming obfuscates the problem, and makes *base* unable to find the proof. On the other hand, BOOLSIMP rule directly converts the negated conjecture to \perp , completing the proof in half a second.

To answer question 3, we compared the mode of Zipperposition in which all rules introduced by our calculus are disabled (*off*) with *base* on purely first-order problems. Our results show that both modes perform roughly the same. Some *base* modes prove up to two more problems within the last seconds of the time allotted due to fluctuations in the evaluation environment beyond our control.

To answer question 4, we evaluated the Boolean selection functions we have implemented: syntactically smallest selectable term (used in *base*), syntactically largest selectable term (sel_{\max}), leftmost innermost selectable term (sel_{li}), leftmost outermost selectable term (sel_{lo}), and no Boolean selection (sel_{\emptyset}). We also evaluated two modes in which the rules LOCALRW and BOOLHOISTSIMP (BHS) are enabled. None of the selection functions influences the performance greatly. Similarly, we observe no substantial difference regardless of whether the rules LOCALRW and BOOLHOISTSIMP are enabled.

9 Related Work and Conclusion

The research presented in this paper extends superposition in two directions: with inprocessing clausification and with first-class Booleans. The first direction has been explored before by Ganzinger and Stuber [13], and others have investigated it in the context of other superposition-related calculi [2, 5, 9, 20, 21].

The other direction has been explored before by Kotelnikov et al., who developed two approaches to cope with first-class Booleans [16, 17]. For the quantified Boolean formula fragment of our logic, Seidl et al. developed a translation into effectively propositional logic [24]. More general approaches to incorporate theories into superposition include superposition for finite domains [14], hierarchic superposition [7], and superposition with (co)datatypes [10].

For SMT solvers [22], supporting first-class Booleans is a widely accepted standard [6]. In contrast, the TPTP TFX format [28], intended to promote first-class Booleans in the rest of the automated reasoning community, has yet to gain traction. Software verification tools could clearly benefit from its popularization, as some of them identify terms and formulas in their logic, e.g., Why3 [12].

In conclusion, we devised a refutationally complete superposition calculus for first-order logic with interpreted Booleans. Its redundancy criterion allows us to flexibly add inprocessing clausification and other simplification rules. We believe our calculus is an excellent choice for the basis of new superposition provers: it offers the full power of standard superposition, while supporting rich input languages such as SMT-LIB and TPTP TFX. Even with unoptimized implementation and basic strategies, our calculus matches the performance of earlier approaches. In addition, the freedom it offers in term order, literal and Boolean subterm selection opens possibilities that are yet to be explored. Overall, our calculus appears as a solid foundation for richer logics in which the Boolean type cannot be efficiently preprocessed, such as higher-order logic [8]. In future work, we plan to tune the parameters and would find it interesting to combine our calculus with clause splitting techniques, such as AVATAR [30].

Acknowledgment Martin Desharnais and Jasmin Blanchette generated the Sledgehammer benchmarks. Simon Cruanes helped us with the implementation. The anonymous reviewers, Ahmed Bhayat, Jasmin Blanchette, and Uwe Waldmann suggested textual improvements. The maintainers of StarExec Iowa let us use their service. We thank them all. Nummelin’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). Bentkamp 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).

References

- [1] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.
- [2] Leo Bachmair and Harald Ganzinger. Non-clausal resolution and superposition with selection and redundancy criteria. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning (LPAR’92)*, volume 624 of *LNCS*, pages 273–284. Springer, 1992.
- [3] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.*, 4(3):217–247, 1994.
- [4] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 19–99. Elsevier and MIT Press, 2001.
- [5] Leo Bachmair, Harald Ganzinger, David A. McAllester, and Christopher Lynch. Resolution theorem proving. In *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.
- [6] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at www.SMT-LIB.org.
- [7] Peter Baumgartner and Uwe Waldmann. Hierarchic superposition with weak abstraction. In Maria Paola Bonacina, editor, *CADE-24*, volume 7898 of *LNCS*, pages 39–57. Springer, 2013.
- [8] Alexander Bentkamp, Jasmin Blanchette, Sophie Turrett, and Petar Vukmirović. Superposition for full higher-order logic. In André Platzer and Geoff Sutcliffe, editors, *CADE-28*, LNCS. Springer, 2021.
- [9] Christoph Benzmüller. Extensional higher-order paramodulation and RUE-resolution. In Harald Ganzinger, editor, *CADE-16*, volume 1632 of *LNCS*, pages 399–413. Springer, 1999.
- [10] Jasmin Christian Blanchette, Nicolas Peltier, and Simon Robillard. Superposition with datatypes and codatatypes. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *IJCAR 2018*, volume 10900 of *LNCS*, pages 370–387. Springer, 2018.
- [11] Simon Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond*. Ph.D. thesis, École polytechnique, 2015.
- [12] Jean-Christophe Filliâtre and Andrei Paskevich. Why3—where programs meet provers. In Matthias Felleisen and Philippa Gardner, editors, *European Symposium on Programming (ESOP 2013)*, volume 7792 of *LNCS*, pages 125–128. Springer, 2013.

- [13] Harald Ganzinger and Jürgen Stuber. Superposition with equivalence reasoning and delayed clause normal form transformation. *Inf. Comput.*, 199(1-2):3–23, 2005.
- [14] Thomas Hillenbrand and Christoph Weidenbach. Superposition for bounded domains. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics*, volume 7788 of *LNCS*, pages 68–100. Springer, 2013.
- [15] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
- [16] Evgenii Kotelnikov, Laura Kovács, Martin Suda, and Andrei Voronkov. A clausal normal form translation for FOOL. In Christoph Benzmüller, Geoff Sutcliffe, and Raúl Rojas, editors, *Global Conference on Artificial Intelligence (GCAI 2016)*, volume 41 of *EPiC*, pages 53–71. EasyChair, 2016.
- [17] Evgenii Kotelnikov, Laura Kovács, and Andrei Voronkov. A first class Boolean sort in first-order theorem proving and TPTP. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics (CICM 2015)*, volume 9150 of *LNCS*, pages 71–86. Springer, 2015.
- [18] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification (CAV 2013)*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
- [19] Michel Ludwig and Uwe Waldmann. An extension of the Knuth-Bendix ordering with LPO-like properties. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic Programming and Automated Reasoning (LPAR 2007)*, volume 4790 of *LNCS*, pages 348–362. Springer, 2007.
- [20] Zohar Manna and Richard J. Waldinger. A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.*, 2(1):90–121, 1980.
- [21] Neil V. Murray. Completely non-clausal theorem proving. *Artif. Intell.*, 18(1):67–85, 1982.
- [22] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.
- [23] Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In *Handbook of Automated Reasoning*, pages 335–367. Elsevier and MIT Press, 2001.
- [24] Martina Seidl, Florian Lonsing, and Armin Biere. qbf2epr: A tool for generating EPR formulas from QBF. In Pascal Fontaine, Renate A. Schmidt, and Stephan Schulz, editors, *Practical Aspects of Automated Reasoning (PAAR-2012)*, volume 21 of *EPiC Series in Computing*, pages 139–148. EasyChair, 2012.
- [25] Alexander Steen. *Extensional Paramodulation for Higher-order Logic and Its Effective Implementation Leo-III*. Dissertationen zur künstlichen Intelligenz. Akademische Verlagsgesellschaft AKA GmbH, 2018.
- [26] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. Starexec: A cross-community infrastructure for logic solving. In *IJCAR 2014*, volume 8562 of *LNCS*, pages 367–373. Springer, 2014.
- [27] Geoff Sutcliffe. The TPTP problem library and associated infrastructure—from CNF to TH0, TPTP v6.4.0. *J. Autom. Reason.*, 59(4):483–502, 2017.
- [28] Geoff Sutcliffe and Evgenii Kotelnikov. TFX: the TPTP extended typed first-order form. In Boris Konev, Josef Urban, and Philipp Rümmer, editors, *Practical Aspects of Automated Reasoning (PAAR-2018)*, volume 2162 of *CEUR Workshop Proceedings*, pages 72–87. CEUR-WS.org, 2018.

- [29] Grigori Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning: Classical Papers on Computational Logic*, volume 2, pages 466–483. Springer, 1983.
- [30] Andrei Voronkov. AVATAR: the architecture for first-order theorem provers. In *CAV 2014*, volume 8559 of *LNCS*, pages 696–710. Springer, 2014.
- [31] Petar Vukmirović and Visa Nummelin. Boolean reasoning in a higher-order superposition prover. In Pascal Fontaine, Konstantin Korovin, Ilias S. Kotsireas, Philipp Rümmer, and Sophie Tourret, editors, *Practical Aspects of Automated Reasoning (PAAR-2020)*, volume 2752 of *CEUR Workshop Proceedings*, pages 148–166. CEUR-WS.org, 2020.
- [32] Petar Vukmirović, Jasmin Blanchette, Simon Cruanes, and Stephan Schulz. Extending a brainiac prover to lambda-free higher-order logic. *Accepted in International Journal on Software Tools for Technology Transfer*.
- [33] Uwe Waldmann, Sophie Tourret, Simon Robillard, and Jasmin Blanchette. A comprehensive framework for saturation theorem proving. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *IJCAR 2020*, LNCS. Springer, 2020.