




A Modular Formalization of Superposition in Isabelle/HOL

Martin Desharnais   

Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
Graduate School of Computer Science, Saarland Informatics Campus, Saarbrücken, Germany

Balazs Toth   

Ludwig-Maximilians-Universität München, Germany

Uwe Waldmann   

Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany

Jasmin Blanchette   

Ludwig-Maximilians-Universität München, Germany

Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany

Sophie Tourret   

Université de Lorraine, CNRS, Inria, LORIA, Nancy, France

Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

Superposition is an efficient proof calculus for reasoning about first-order logic with equality that is implemented in many automatic theorem provers. It works by saturating the given set of clauses and is refutationally complete, meaning that if the set is inconsistent, the saturation will contain a contradiction. In this work, we restructured the completeness proof to cleanly separate the ground (i.e., variable-free) and nonground aspects, and we formalized the result in Isabelle/HOL. We relied on the *IsaFoR* library for first-order terms and on the Isabelle saturation framework.

2012 ACM Subject Classification Computing methodologies → Theorem proving algorithms

Keywords and phrases Superposition, verification, first-order logic, higher-order logic

Digital Object Identifier 10.4230/LIPIcs.ITP.2024.2

Funding *Jasmin Blanchette*: Co-funded by the European Union (ERC, Nekoka, 101083038). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgments We thank Xavier G  n  reux, Mark Summerfield, and the anonymous reviewers for suggesting textual improvements.

1 Introduction

Superposition is a highly successful proof calculus for reasoning about first-order logic with equality designed by Bachmair and Ganzinger [2, 3]. It is implemented in many automatic theorem provers, including E [33], SPASS [45], Vampire [22], and Zipperposition [16].

Superposition provers work by refutation and saturation. They operate on a clause set, which initially consists of the clausified input problem in which the conjecture appears negated. Inferences are performed using clauses from this set as premises; the conclusions of inferences are added to the set. The prover stops when the empty clause \perp , denoting falsehood, is derived or when no more inferences are possible.

Consider the problem of proving $f(b) \approx f(a)$ from $b \approx a$, where \approx denotes equality. After negating the conjecture, we obtain the clause set $\{b \approx a, f(b) \not\approx f(a)\}$. The superposition



   Martin Desharnais, Balazs Toth, Uwe Waldmann, Jasmin Blanchette, and Sophie Tourret;
licensed under Creative Commons License CC-BY 4.0

15th International Conference on Interactive Theorem Proving (ITP 2024).

Editors: Yves Bertot, Temur Kutsia, and Michael Norrish; Article No. 2; pp. 2:1–2:20

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum f  r Informatik, Dagstuhl Publishing, Germany

calculus includes an inference rule called superposition that uses the first clause to rewrite the second clause to $f(a) \not\approx f(a)$. This new clause is added to the clause set. At this point, a unary inference rule called equality resolution uses $f(a) \not\approx f(a)$ to derive \perp .

During the saturation, the prover can delete clauses considered redundant, and it does not need to perform inferences considered redundant. For example, if the clause set contains $b \approx a$, then the clauses $f(b) \approx f(a)$ and $b \approx a \vee b \not\approx c$ are redundant. Deletion of redundant clauses helps reduce the clause explosion caused by saturation.

The inference rules of the superposition calculus are *sound*, meaning that the conclusion of each rule is entailed by the premises. This is easy to prove. What is much harder to show is that the calculus is *refutationally complete*: If a clause set is unsatisfiable and saturated (up to redundancy), then it contains \perp . We care about completeness because a complete calculus is likely to yield a higher success rate in practice than an incomplete one. Moreover, the completeness proof serves as a guide during the development of the calculus: Only inferences that are needed in the proof must be performed.

When developing proof calculi for first-order logic and beyond, it often helps to first develop a calculus that works on ground (i.e., variable-free) clauses. We can then lift it to the nonground level. This approach cleanly separates concerns. It is common in the literature [7–11, 29] and is supported by the *saturation framework* developed by Bachmair and Ganzinger [4, Section 4] and extended by Waldmann et al. [44], a collection of pen-and-paper results useful to establish the refutational completeness of saturation calculi and provers.

For superposition, Bachmair and Ganzinger’s completeness proof [3] does not separate the ground and nonground aspects. Waldmann et al. give some hints on how to instantiate the framework to obtain a modular proof that separates these aspects. Our main contributions are twofold. First, we elaborated these hints into a 15-page proof text [43] (summarized here in Section 3). Second, following this detailed *blueprint*, we formalized in Isabelle/HOL [24] the refutational completeness of ground superposition (Section 4) and lifted it to derive the refutational completeness of the nonground calculus (Section 5). We also proved soundness.

The separation of concerns, apart from allowing different people to work independently on different parts of the formalization, simplifies the completeness proof. On the ground level, there is no need to rename variables apart or to perform unification. On the nonground level, an inference overapproximates a set of ground inferences. Intuitively, this means that every inference on ground clauses can be simulated by inferences on corresponding nonground clauses. For superposition inferences, this roughly means that if $D\gamma_1$ and $E\gamma_2$ are premises of a nonredundant ground inference yielding C , where γ_1, γ_2 are substitutions, then there exists an inference with D and E as premises and whose conclusion is a generalization of C .

A difficulty arises on the nonground level because the calculus is optimized to avoid superposition *into* variables. For example, given the clause set $\{b \approx a, f(x) \not\approx c\}$, a superposition inference unifying b with x would yield the conclusion $f(a) \not\approx c$, but the calculus excludes this inference. Intuitively, since $f(a) \not\approx c$ is an instance of $f(x) \not\approx c$, we would expect the inference to be unnecessary, but this must be justified in general.

The Isabelle formalization relies on the first-order terms and related notions from the *IsaFoR* library [39]. It also uses the Isabelle version of the saturation framework [42]. The formalization validates the pen-and-paper proof: We found only one easy-to-repair mistake and one unnecessary assumption. The formalization can serve as a reference for refutational completeness of superposition, an important result in automated reasoning. It could also serve as the basis of a verified executable prover.

Ours is not the first formalization of superposition in a proof assistant, or even in Isabelle/HOL. Our predecessor is Peltier, who formalized a generalization of superposition

and published his result in the *Archive of Formal Proofs (AFP)* [27]. However, his proof is monolithic, mixing ground and nonground aspects. By using the saturation framework, we get a clearer proof structure and immediately obtain the completeness of an abstract prover based on superposition [44, Lemma 10] as well as the completeness of various saturation procedures [44, Section 4].

Our Isabelle formalization and the underlying pen-and-paper proof are available online [17, 43]. The formalization will soon be submitted to the *AFP*. Our work is part of the IsaFoL (Isabelle Formalization of Logic) effort [12].¹

2 Background

Prerequisites. We consider an untyped first-order logic with equality. A *term* is defined inductively as either a variable x or a function application $f(t_1, \dots, t_n)$ for a function symbol f and a (possibly empty) list of terms t_1, \dots, t_n . An *atom* is an unordered pair of terms, typically written as an equation $t \approx t'$. A *literal* is an atom $t \approx t'$ or a negated atom $t \not\approx t'$. A *clause* is a finite multiset $\{L_1, \dots, L_n\}$ of literals, typically written as a disjunction $L_1 \vee \dots \vee L_n$. The symbol \perp denotes the empty clause (or empty disjunction), which is false. All variables in a clause are to be understood as implicitly universally quantified in that clause.

A *context* κ is a term with one designated position that is to be filled by another term—in other words, a term with a hole. We use the syntax $\kappa[t]$ to represent the term consisting of a subterm t in a context κ . We write \square for the empty context.

Substitutions are total unary functions that let us replace variables with terms. We can apply a substitution σ to a syntactic entity X (e.g., a term or literal) by writing $X\sigma$. A substitution γ is a *grounding* substitution for a syntactic entity X if $X\gamma$ is ground, i.e., if it does not contain variables. A substitution ρ is a *renaming* if it is injective and $x\rho$ is a variable for every variable x . The composition of two substitutions σ_1 and σ_2 is defined as the function $\sigma_1 \circ \sigma_2 = (\lambda x. x\sigma_1\sigma_2)$. A substitution μ is an *idempotent most general unifier* (IMGU) for a set of terms T if μ is a unifier for T and $\mu \circ v = v$ for every unifier v for T .

An element x is *maximal* in a finite multiset \mathcal{X} w.r.t. a strict partial ordering \prec on \mathcal{X} if $x \in \mathcal{X} \wedge (\forall y \in \mathcal{X}. y \neq x \rightarrow x \not\prec y)$. An element x is *strictly maximal* in a finite multiset \mathcal{X} w.r.t. a strict partial ordering \prec on \mathcal{X} if $x \in \mathcal{X} \wedge (\forall y \in \mathcal{X} \setminus \{x\}. x \not\preceq y)$, where \preceq is the reflexive closure of \prec . The two notions coincide except for their handling of duplicates: A maximal element can have duplicates, whereas a strictly maximal element cannot. If the ordering is not total, a multiset can have multiple maximal or strictly maximal elements.

The Superposition Calculus. Bachmair and Ganzinger’s superposition calculus [2, 3] belongs to a class of proof calculi for automatic provers known as saturation calculi. A saturation prover takes a set of formulas, usually clauses, as input and processes it by performing two operations: First, it derives new formulas from the old ones and adds them to the set. Second, it deletes superfluous formulas from the set. This process is repeated until the prover either finds \perp or reaches a state in which it is not required to add further formulas.

Abstractly, the calculus can be defined by two components: a set of inferences

$$\frac{C_n \ \cdots \ C_1}{C_0}$$

¹ <https://github.com/IsaFoL/IsaFoL>

indicating that the formula C_0 (the *conclusion*) must be added to the set whenever the formulas C_n, \dots, C_1 (the *premises*) are already present, and a redundancy criterion that describes which inferences are unnecessary and which formulas may be deleted from the set.

For the superposition calculus, the inferences are given by three schematic inference rules. The first one is

$$\frac{\overbrace{t \approx t' \vee D'}^D \quad \overbrace{\kappa[u] \bowtie u' \vee E'}^E}{\underbrace{(\kappa[t'\rho] \bowtie u' \vee E' \vee D'\rho)\mu}_C} \text{superposition}$$

where the clauses D and E are the premises, C is the conclusion, \bowtie is either \approx or $\not\approx$, u is a nonvariable subterm occurring in a context κ in clause E , ρ is an arbitrary but fixed renaming that is chosen so that $D\rho$ and E are variable-disjoint, and μ is an IMGU of $t\rho$ and u .

The other two rules are

$$\frac{\overbrace{t \not\approx t' \vee D'}^D}{\underbrace{D'\mu}_C} \text{equality resolution}$$

where μ is an IMGU of t and t' , and

$$\frac{\overbrace{u \approx u' \vee t \approx t' \vee D'}^D}{\underbrace{(u' \not\approx t' \vee u \approx t' \vee D')\mu}_C} \text{equality factoring}$$

where μ is an IMGU of t and u .

To reduce the number of inferences that need to be computed during the saturation, the inference rules above are equipped with ordering restrictions. Let \prec_t be an ordering on terms that is stable under grounding substitutions, and whose ground restriction is well-founded, total, and compatible with contexts, and has the subterm property. The term ordering \prec_t is extended to a literal ordering and a clause ordering in the following way: To every positive literal $t \approx t'$, we assign the multiset $\{t, t'\}$, to every negative literal $t \not\approx t'$, we assign the multiset $\{t, t, t', t'\}$. The literal ordering \prec_{lit} compares these multisets using the multiset extension of \prec_t . The clause ordering \prec_c compares clauses by comparing their multisets of literals using the multiset extension of \prec_{lit} .

We impose the following ordering restrictions on the inferences above: **(1)** If L is the first literal in a premise D or E , it must be maximal in that premise w.r.t. \prec_{lit} (after applying the substitution); **(2)** if additionally L is a positive equation in a superposition inference, it must be strictly maximal; **(3)** except in equality resolution inferences, the right-hand side of the equation or negated equation L may not be larger than or equal to the left-hand side w.r.t. \prec_t ; and **(4)** in superposition inferences, $D\rho\mu$ may not be larger than or equal to $C\mu$ w.r.t. \prec_c .

In the worst case, all literals in a clause can be incomparable and hence maximal. For clauses with negative literals, this effect can be remedied using a *selection function* that overrides the ordering restrictions. This is a function that maps every clause to a submultiset of its negative literals. The ordering conditions above are then modified so that if at least one literal in a clause is selected, then the maximality conditions for literals are applied to the selected submultiset instead of the original clause. This means that only inferences that involve literals that are maximal among the selected literals need to be performed.

These local restrictions are supplemented by a global redundancy criterion for clauses and inferences. Bachmair and Ganzinger's *standard redundancy criterion* is defined as follows:

A ground clause C is redundant w.r.t. a set N of ground clauses if it is entailed by clauses in N that are smaller than C w.r.t. \prec_c . A nonground clause C is redundant w.r.t. a set N of nonground clauses if every ground instance of C is redundant w.r.t. the set of all ground instances of clauses in N . A ground inference (i.e., an inference with ground premises and ground conclusion) is redundant w.r.t. a set N of ground clauses if its conclusion is entailed by clauses in N that are smaller than the maximal premise. A nonground inference is redundant w.r.t. a set N of nonground clauses if every ground instance of the inference is redundant w.r.t. the set of all ground instances of clauses in N .

Redundant clauses may be deleted from the clause set during a saturation; redundant inferences need not be computed. In particular, inferences whose conclusion is already contained in the clause set are always redundant.

The Saturation Framework. In their article in the *Handbook of Automated Reasoning* [4], Bachmair and Ganzinger gave a general account of components and properties of saturation calculi. The framework by Waldmann et al. [44] extended this to include a general treatment of lifting, subsumption, and prover architectures. We summarize the main results.

Let F be a set of formulas, and \models be a consequence relation on F . An F -inference is an inference with premises and conclusion in F . An F -inference system Inf is a set of F -inferences. If $N \subseteq F$, we write $Inf(N)$ for the set of all inferences in Inf with premises in N .

Let Red_I be a function from sets of formulas to sets of inferences; let Red_F be a function from sets of formulas to sets of formulas. The pair $Red = \langle Red_I, Red_F \rangle$ is a *redundancy criterion* for Inf if it satisfies the following conditions:

1. if $N \models \{\perp\}$, then $N \setminus Red_F(N) \models \{\perp\}$;
2. if $N \subseteq N'$, then $Red_F(N) \subseteq Red_F(N')$ and $Red_I(N) \subseteq Red_I(N')$;
3. if $N' \subseteq Red_F(N)$, then $Red_F(N) \subseteq Red_F(N \setminus N')$ and $Red_I(N) \subseteq Red_I(N \setminus N')$; and
4. if the conclusion of an inference in Inf is in N , then the inference is in $Red_I(N)$.

Inferences in $Red_I(N)$ and formulas in $Red_F(N)$ are called redundant w.r.t. N .

A saturation prover for a calculus $\langle Inf, Red \rangle$ gets a set of formulas $N_0 \subseteq F$ as input and generates a sequence N_0, N_1, \dots of sets of formulas by adding newly computed formulas and by deleting unnecessary formulas. We require that in every step the deleted formulas are redundant w.r.t. the remaining ones. We call the sequence N_0, N_1, \dots a *derivation*. The set $N_\infty = \bigcup_i \bigcap_{j \geq i} N_j$ of persistent formulas is called the *limit* of the derivation. The derivation is *fair* if every inference from persistent formulas eventually becomes redundant. The calculus $\langle Inf, Red \rangle$ is *dynamically refutationally complete* if for every set N_0 with $N_0 \models \{\perp\}$ and every fair derivation N_0, N_1, \dots , the formula \perp is eventually derived, that is, $\perp \in \bigcup_i N_i$.

Proving the dynamic refutational completeness of the calculus $\langle Inf, Red \rangle$ directly is usually difficult. Fortunately, dynamic refutational completeness can be shown to be equivalent to another property, namely static refutational completeness: A set $N \subseteq F$ is *saturated* w.r.t. Inf and Red if $Inf(N) \subseteq Red_I(N)$. The calculus $\langle Inf, Red \rangle$ is *statically refutationally complete* if for every saturated set N we have that $N \models \perp$ implies $\perp \in N$.

To prove the static (and thus dynamic) refutational completeness of a calculus, it is usually convenient to start with a ground version of the calculus. The completeness result for the nonground calculus can then be obtained from the completeness result for the ground calculus by lifting, using a suitable *grounding* function that maps nonground formulas to sets of ground formulas and nonground inferences to sets of ground inferences. The framework also shows how to deal with redundancy criteria that are defined as intersections of other redundancy criteria (a technique that we will need to handle selection functions in the

lifting process), how to integrate *subsumption* into the redundancy criterion (so that, e.g., $x \approx a$ makes its instance $b \approx a$ redundant), and how to obtain completeness results for implementations of the calculus in various prover architectures.

The framework has been formalized in Isabelle/HOL and extended by Tourret and Blanchette [14, 40, 42]. The present work builds on this formalization.

3 Proof Outline

Static refutational completeness can be stated as follows:

► **Theorem 1.** *For every set N that is saturated w.r.t. the superposition calculus, if N entails \perp , then $\perp \in N$.*

Equivalently: For every saturated set N such that $\perp \notin N$, there exists a model of N . Bachmair and Ganzinger’s original proof [3, Section 4] uses a monolithic approach. Our proof is more modular and proceeds in two clearly separated steps:

1. Given a ground clause set M saturated w.r.t. ground inferences, we build a model of M .
2. We show that if a clause set N is saturated w.r.t. nonground inferences, then its grounding $N_G = \{C\gamma \mid C \in N \text{ and } C\gamma \text{ is ground}\}$ is saturated w.r.t. ground inferences. Hence, by step 1, there exists a model of N_G , which is also a model of N .

In step 1, we construct a confluent and terminating term rewriting system R_∞ and use it to define an interpretation that equates all terms that share the same normal form w.r.t. R_∞ , and no others. For example, if $R_\infty = \{b \rightarrow a\}$, then the associated interpretation makes $f(b) \approx f(a)$ true and $c \approx a$ false. The system R_∞ is built incrementally. We start with $\{\}$ and traverse the clauses in M from the smallest clause following the ordering \prec_c . For each clause $C \in M$, if C is true in the current interpretation, there is nothing to do. Otherwise, we add a rewrite rule that attempts to make C true without affecting the truth of earlier, smaller clauses. While this process might fail in general, it will always produce a model of M if M is saturated.

In step 2, we must show that saturation on the nonground level implies saturation on the ground level. Via a result from the saturation framework, this amounts to showing that there exist nonground inferences corresponding to all nonredundant ground inferences of the calculus. A subtlety is that the calculus avoids superposition inferences into variables. Thus there might exist ground inferences that are not reflected on the nonground level. However, we can show that all such inferences are redundant. Another concern is the selection function. In general, we cannot assume that it is stable under substitutions, but without this assumption it is hard to relate the ground and nonground levels. The solution is provided by the saturation framework, which allows us to simultaneously lift all ground selection functions to the nonground level.

4 The Ground Proof

On the ground level, we reuse theories [23] from the **IsaFoR** project for terms, of type *'fgterm*, and term contexts, of type *'fgtxt*. The type variable *'f* represents function symbols. Isabelle types use a postfix notation. Ground atoms have type *'fgatom*, which is a synonym for *'fgterm uprod*, i.e., unordered pair of ground terms. Ground literals have type *'fgatom literal*. Ground clauses have type *'fgatom clause*, which is a synonym for *'fgatom literal multiset*. Isabelle multisets are always finite.

We start the formalization by introducing a locale, or module, that fixes an ordering on ground terms:

```

locale ground_ordering =
  fixes ( $\prec_t$ ) :: 'f gterm  $\Rightarrow$  'f gterm  $\Rightarrow$  bool
  assumes
    transp ( $\prec_t$ ) and asymp ( $\prec_t$ ) and totalp ( $\prec_t$ ) and wfp ( $\prec_t$ ) and
     $\forall \kappa :: 'f gterm. \forall t_1 t_2. t_1 \prec_t t_2 \longrightarrow \kappa[t_1] \prec_t \kappa[t_2]$  and
     $\forall \kappa :: 'f gterm. \forall t. \kappa \neq \square \longrightarrow t \prec_t \kappa[t]$ 

```

In Isabelle, a locale consists of parameters (here, \prec_t) that may depend on type variables (here, 'f) paired with assumptions. Locales allow us to declare parameters and assumptions once and reuse them in multiple definitions and lemmas. When we later instantiate a locale, we must supply concrete arguments for the types and parameters and then discharge the proof obligations corresponding to the assumptions.

The locale `ground_ordering` assumes that the binary relation \prec_t is a well-founded total ordering, is compatible with ground term contexts, and has the subterm property.

Inside the locale context, we lift the term ordering and its properties to literals (\prec_{lit}) and clauses (\prec_c). We also configure the little-known `order` Isabelle proof method [38], a decision procedure for the quantifier-free theory of partial and total orderings, so that it can solve problems for our orderings.

As a building block for this formalization, we developed a generic theory of (strictly) minimal, (strictly) maximal, least, and greatest element in sets, finite sets, and finite multisets w.r.t. any partial or total ordering.

Next, we define the notion of selection function:

```

locale select =
  fixes sel :: 'a clause  $\Rightarrow$  'a clause
  assumes
     $\forall C. \text{sel } C \subseteq C$  and
     $\forall C. \forall L \in \text{sel } C. \text{is\_neg } L$ 

```

The locale `select` fixes a function `sel` for clauses with any atom type 'a. In this section, we instantiate 'a with 'f gatom; Section 5 will instantiate it with its own atom type. Our assumptions on a selection function are that it always returns a submultiset of the argument C and only returns negative literals.

We can now assemble the parameters and assumption for the ground calculus:

```

locale ground_superposition_calculus = ground_ordering ( $\prec_t$ ) + select selG
for
  ( $\prec_t$ ) :: 'f gterm  $\Rightarrow$  'f gterm  $\Rightarrow$  bool and
  selG :: 'f gatom clause  $\Rightarrow$  'f gatom clause +
  assumes  $\forall R :: ('f gterm \times 'f gterm) \text{ set}. \text{ground\_critical\_pair\_theorem } R$ 

```

The locale `ground_superposition_calculus` extends both `ground_ordering` and `select`, inheriting all their assumptions as well as the definitions and theorems from their locale contexts. The parameters \prec_t and `selG` are provided with type annotations to control the instantiation of the type parameters. We also assume that the critical pair theorem [1, Theorem 6.2.4] holds for ground terms. As a sanity check, we proved this theorem in Isabelle by adapting a similar, but license-incompatible, result from the `IsaFoR` project [39].

We can now specify the ground version of the inference rules presented in Section 2, using inductive predicates. Compared with their nonground counterparts, the ground rules benefit from two simplifications. First, neither renamings nor unifiers are needed because ground

terms contain no variables. Second, terms and clauses can be compared directly using \prec_t and \prec_c instead of using a reversed negated form since the orderings are total.

We show the ground superposition rule as an example. The rule notation below defines an inductive predicate $\text{ground_superposition } D E C$ with the rule's premises D, E as assumptions and the rule's conclusion C as conclusion:

$$\frac{\overbrace{t \approx t' \vee D'}^D \quad \overbrace{\kappa[t] \bowtie u \vee E'}^E}{\underbrace{\kappa[t'] \bowtie u \vee D' \vee E'}_C} \text{ground_superposition } D E C$$

Side conditions:

1. $\bowtie \in \{\approx, \neq\}$; 2. $D \prec_c E$; 3. $t' \prec_t t$; 4. $u \prec_t \kappa[t]$;
5. if $\bowtie = \approx$, then $\text{sel}_G E = \{\}$ and $\kappa[t] \bowtie u$ is strictly maximal in E ;
6. if $\bowtie = \neq$, then $\text{sel}_G E = \{\}$ and $\kappa[t] \bowtie u$ is maximal in E or $\kappa[t] \bowtie u$ is maximal in $\text{sel}_G E$;
7. $\text{sel}_G D = \{\}$; 8. $t \approx t'$ is strictly maximal in D .

Following the structure required by the saturation framework, we define an inference system Inf_G and a consequence relation entails_G . For formulas, we use the type of ground clauses *'fgatom clause*, and for contradictions, we use the empty clause \perp .

► **Definition 2.** The set $\text{Inf}_G :: \text{'fgatom clause inference set}$ consists of all inferences of the ground superposition calculus:

$$\text{Inf}_G = \{ \langle [D, E], C \rangle \mid \text{ground_superposition } D E C \} \cup \{ \langle [D], C \rangle \mid \text{ground_eq_resolution } D C \} \cup \{ \langle [D], C \rangle \mid \text{ground_eq_factoring } D C \}$$

For the consequence relation, we reuse the theory of Herbrand interpretation developed for a formalization of ordered resolution [32]. This theory considers an interpretation to be a set of true atoms and defines the relation $\mathcal{I} \models_{\text{lit}} L$ expressing that the interpretation \mathcal{I} models the literal L , i.e., that L 's atom is in \mathcal{I} iff L is positive. The predicate is lifted to clauses (\models_c) and clause sets (\models) in the usual way.

Our ground atoms being unordered pairs of ground terms, our interpretations should be sets of *unordered* pairs. However, since Isabelle makes it easier to manipulate sets of *ordered* pairs, we use these as our interpretation and define a small wrapper with the help of the function $\text{uprod} :: 'a \times 'a \Rightarrow 'a \text{ uprod}$ to bridge the gap.

► **Definition 3.** The predicates $(\models_c) :: (\text{'fgterm} \times \text{'fgterm}) \text{ set} \Rightarrow \text{'fgatom clause} \Rightarrow \text{bool}$ and $(\models) :: (\text{'fgterm} \times \text{'fgterm}) \text{ set} \Rightarrow \text{'fgatom clause set} \Rightarrow \text{bool}$ express that an interpretation models a clause and a clause set, respectively:

$$\mathcal{I} \models_c C \iff \{ \text{uprod } r \mid r \in \mathcal{I} \} \models_c C \quad \mathcal{I} \models N \iff \{ \text{uprod } r \mid r \in \mathcal{I} \} \models N$$

We cannot use arbitrary sets of pairs as interpretations because the pairs should represent term equality. We require a valid interpretation \mathcal{I} to be reflexive, symmetric, and transitive and to be compatible with ground context application (i.e., $\forall \kappa :: \text{'fgtxt}. \forall t t'. \langle t, t' \rangle \in \mathcal{I} \longrightarrow \langle \kappa[t], \kappa[t'] \rangle \in \mathcal{I}$). We encode these requirements in the entails_G predicate:

► **Definition 4.** The predicate $\text{entails}_G :: \text{'fgatom clause set} \Rightarrow \text{'fgatom clause set} \Rightarrow \text{bool}$ expresses that a clause set N_1 entails another clause set N_2 , i.e., every valid interpretation of N_1 is also a valid interpretation of N_2 :

$$\begin{aligned} \text{entails}_G N_1 N_2 \iff & (\forall \mathcal{I} :: (\text{'fgterm} \times \text{'fgterm}) \text{ set}. \\ & \text{refl } \mathcal{I} \longrightarrow \text{sym } \mathcal{I} \longrightarrow \text{trans } \mathcal{I} \longrightarrow \text{compatible_with_gtxt } \mathcal{I} \longrightarrow \\ & \mathcal{I} \models N_1 \longrightarrow \mathcal{I} \models N_2) \end{aligned}$$

Equipped with Inf_G and entails_G , we can start to use the saturation framework. We first instantiate the `sound_inference_system` locale to make sure that the ground superposition calculus is sound and that our definitions correspond to what the framework expects:

sublocale `ground_superposition_calculus` \subseteq `sound_inference_system` **where**
 $\text{Inf} = \text{Inf}_G$ **and** $\text{Bot} = \{\perp\}$ **and** $\text{entails} = \text{entails}_G$

The sublocale notation means that definitions and theorems from `ground_superposition_calculus` are sufficient to prove the assumptions of `sound_inference_system` w.r.t. the given parameter instantiations. At this point, Isabelle requires us to actually prove the assumptions.

As the redundancy criterion, we reuse the standard redundancy criterion defined in the Isabelle saturation framework [14]:

sublocale `ground_superposition_calculus` \subseteq
`calculus_with_finitary_standard_redundancy` **where**
 $\text{Inf} = \text{Inf}_G$ **and** $\text{Bot} = \{\perp\}$ **and** $\text{entails} = \text{entails}_G$ **and** $\text{less} = (\prec_c)$
defines $\text{Red}_I = \text{Red}_I$ **and** $\text{Red}_F = \text{Red}_F$

The locale `calculus_with_finitary_standard_redundancy` defines the functions $\text{Red}_I :: 'f\text{gatom clause set} \Rightarrow 'f\text{gatom clause inference set}$, identifying redundant inferences, and $\text{Red}_F :: 'f\text{gatom clause set} \Rightarrow 'f\text{gatom clause set}$, identifying redundant formulas. We rename them to Red_I and Red_F , respectively.

To prove refutational completeness, we will exhibit a valid interpretation for a given saturated clause set. We build this interpretation by defining a confluent and terminating set of rewrite rules R_∞ , which we lift to an interpretation $\llbracket R_\infty \rrbracket^\downarrow$ that defines term equality. Each rewrite rule is a pair $\langle t, t' \rangle$, written $t \rightarrow t'$.

► **Definition 5.** The function $\llbracket \cdot \rrbracket :: ('f\text{gterm} \times 'f\text{gterm}) \text{ set} \Rightarrow ('f\text{gterm} \times 'f\text{gterm}) \text{ set}$ expands a rewrite rule set to all term contexts: $\llbracket R \rrbracket = \{\kappa[t] \rightarrow \kappa[t'] \mid t \rightarrow t' \in R\}$.

► **Definition 6.** The function $\cdot^\downarrow :: ('f\text{gterm} \times 'f\text{gterm}) \text{ set} \Rightarrow ('f\text{gterm} \times 'f\text{gterm}) \text{ set}$ produces the set of all term pairs considered equal w.r.t. a set of rewrite rules: $R^\downarrow = \{\langle t, t' \rangle \mid \exists t''. t \rightarrow t'' \in R^* \wedge t' \rightarrow t'' \in R^*\}$.

Now that we can lift a set of rewrite rules to a model, we define two mutually recursive functions that construct such a set for a given clause set.

► **Definition 7.** Let $N^{\prec_c D} = \{C \in N \mid C \prec_c D\}$ for any N and D . The functions $\text{epsilon} :: 'f\text{gatom clause set} \Rightarrow 'f\text{gatom clause} \Rightarrow ('f\text{gterm} \times 'f\text{gterm}) \text{ set}$ and $\text{rewrite_sys} :: 'f\text{gatom clause set} \Rightarrow ('f\text{gterm} \times 'f\text{gterm}) \text{ set}$ generate a term rewriting system for a given clause set:

$$\begin{aligned} \text{epsilon } N \ C &= \{t \rightarrow t' \mid \exists C'. C \in N \wedge C = (t \approx t' \vee C') \wedge \text{sel}_G C = \{\} \wedge \\ &\quad t \approx t' \text{ is strictly maximal in } C \wedge t' \prec_t t \wedge \\ &\quad \llbracket \text{rewrite_sys } N^{\prec_c C} \rrbracket^\downarrow \not\models_c C \wedge \\ &\quad \llbracket \text{rewrite_sys } N^{\prec_c C} \cup \{t \rightarrow t'\} \rrbracket^\downarrow \not\models_c C' \wedge \\ &\quad t \text{ is in normal form w.r.t. } \llbracket \text{rewrite_sys } N^{\prec_c C} \rrbracket^\downarrow\} \\ \text{rewrite_sys } N &= \bigcup_{C \in N} \text{epsilon } N \ C \end{aligned}$$

We reuse the definitions of joinability (\cdot^\downarrow) and of normal form (i.e., irreducibility) from a formalization of abstract rewriting systems [35].

The model construction iterates over the clause set, starting from the smallest clause following the ordering \prec_c , and collects a set of rewrite rules. At any point, we can use $\llbracket \cdot \rrbracket^\downarrow$

to obtain the candidate model. At each iteration, `epsilon` returns a set of rewrite rules that are added to the term rewriting system: Either the considered clause is already true w.r.t. to the candidate model, in which case `epsilon` returns the empty set, or `epsilon` returns a single new rewrite rule that should make the clause true.

► **Example 8.** Assume \prec_t is the lexicographic path ordering with the precedence $a \prec b \prec c \prec d \prec e \prec f$. Let $N = \{d \approx c, b \approx a \vee e \not\approx c, b \not\approx b \vee f(b) \approx a, f(c) \approx b, f(b) \approx a \vee f(c) \not\approx b, f(b) \approx a \vee f(d) \not\approx b\}$ be a clause set saturated w.r.t. the ground superposition calculus. The following table shows the result of each iteration of the model construction:

Iteration	Clause C	<code>rewrite_sys</code> $N \prec_c C$	<code>epsilon</code> $N C$
1	$d \approx c$	$\{\}$	$\{d \rightarrow c\}$
2	$b \approx a \vee e \not\approx c$	$\{d \rightarrow c\}$	$\{\}$
3	$b \not\approx b \vee f(b) \approx a$	$\{d \rightarrow c\}$	$\{f(b) \rightarrow a\}$
4	$f(c) \approx b$	$\{d \rightarrow c, f(b) \rightarrow a\}$	$\{f(c) \rightarrow b\}$
5	$f(b) \approx a \vee f(c) \not\approx b$	$\{d \rightarrow c, f(b) \rightarrow a, f(c) \rightarrow b\}$	$\{\}$
6	$f(b) \approx a \vee f(d) \not\approx b$	$\{d \rightarrow c, f(b) \rightarrow a, f(c) \rightarrow b\}$	$\{\}$

At each iteration $i + 1$, the term rewriting system consists of the union of the term rewriting system of iteration i and the “epsilon” of iteration i . As expected, the interpretation after iteration 6 is a model of N .

The conditions on rewrite rule production were chosen so that the term rewriting system is confluent. Specifically, we prove strong normalization and the weak Church–Rosser property, which together imply the Church–Rosser property, which is equivalent to confluence.

► **Lemma 9.** *Let N be a ground clause set. The term equality specified by $\llbracket \text{rewrite_sys } N \rrbracket^\downarrow$ is reflexive, symmetric, transitive, and compatible with ground contexts.*

Now that we can build a valid interpretation for a clause set, it remains to show that it satisfies all clauses from this set. We first need a pair of lemmas that express monotonicity properties of the construction:

► **Lemma 10.** *Let N be a ground clause set and C be a ground clause. If $\text{epsilon } N C = \{t \rightarrow t'\}$, then (1) $\llbracket \text{rewrite_sys } N \rrbracket^\downarrow \models_c C$; (2) $\forall D \in N. C \prec_c D \rightarrow \llbracket \text{rewrite_sys } N \prec_c D \rrbracket^\downarrow \models_c C$; (3) $\llbracket \text{rewrite_sys } N \rrbracket^\downarrow \not\models_c C \setminus \{t \approx t'\}$; and (4) $\forall D \in N. C \prec_c D \rightarrow \llbracket \text{rewrite_sys } N \prec_c D \rrbracket^\downarrow \not\models_c C \setminus \{t \approx t'\}$.*

► **Lemma 11.** *Let N be a ground clause set and $C \in N$ be a ground clause. If $\llbracket \text{rewrite_sys } N \prec_c C \rrbracket^\downarrow \models_c C$, then (1) $\llbracket \text{rewrite_sys } N \rrbracket^\downarrow \models_c C$ and (2) $\forall D \in N. C \prec_c D \rightarrow \llbracket \text{rewrite_sys } N \prec_c D \rrbracket^\downarrow \models_c C$.*

We can now prove that our model construction works for all clauses.

► **Lemma 12.** *Let N be a saturated ground clause set and $C \in N$ be a ground clause. If $\perp \notin N$, then (1) $\text{epsilon } N C = \{\} \leftrightarrow \llbracket \text{rewrite_sys } N \prec_c C \rrbracket^\downarrow \models_c C$ and (2) $\forall D \in N. C \prec_c D \rightarrow \llbracket \text{rewrite_sys } N \prec_c D \rrbracket^\downarrow \models_c C$.*

Proof sketch. By well-founded induction w.r.t. \prec_c . ◀

► **Lemma 13 (Ground Model Construction).** *Let N be a saturated ground clause set and $C \in N$ be a ground clause. If $\perp \notin N$, then $\llbracket \text{rewrite_sys } N \rrbracket^\downarrow \models_c C$.*

Proof sketch. By Lemmas 11 and 12 if $\text{epsilon } N C = \{\}$ and Lemma 10 otherwise. ◀

► **Theorem 14** (Ground Refutational Completeness). *Let N be a saturated ground clause set. If $\text{entails}_G N \{\perp\}$, then $\perp \in N$.*

Proof sketch. We assume $\perp \notin N$ and show $\neg \text{entails}_G N \{\perp\}$. We must show the existence of an interpretation \mathcal{I} such that (1) \mathcal{I} is reflexive, symmetric, transitive, and compatible with ground contexts; (2) $\mathcal{I} \models N$; and (3) $\mathcal{I} \not\models_c \perp$. We let $\mathcal{I} = \llbracket \text{rewrite_sys } N \rrbracket^\downarrow$. Step 1 follows from Lemma 9. Step 2 follows from Lemma 13. Step 3 follows from the definition of \models_c . ◀

Finally, we can provide our main result for the ground calculus by instantiating the locale `statically_complete_calculus` from the saturation framework:

sublocale `ground_superposition_calculus` \subseteq `statically_complete_calculus` **where**
 $\text{Inf} = \text{Inf}_G$ **and** $\text{Bot} = \{\perp\}$ **and** $\text{entails} = \text{entails}_G$ **and** $\text{less} = (\prec_c)$ **and**
 $\text{Red}_l = \text{Red}_{lG}$ **and** $\text{Red}_f = \text{Red}_{fG}$

We use Theorem 14 to discharge the proof obligation.

5 The Nonground Proof

On the nonground level, we reuse theories [36] from the `IsaFoR` project for terms, of type (f, v) *term*, and term contexts, of type (f, v) *ctxt*. The type variable v represents term variables. Atoms have type (f, v) *atom*, which is a synonym for (f, v) *term uprod*. Literals and clauses (type (f, v) *atom clause*) are defined analogously to ground literals and clauses. Substitutions have type $v \Rightarrow (f, v)$ *term*.

► **Definition 15.** The predicates $\text{is_ground}_t :: (f, v)$ *term* \Rightarrow *bool* and $\text{is_ground}_{\text{ctxt}} :: (f, v)$ *ctxt* \Rightarrow *bool* express that the given term or context does not contain any variables. We lift is_ground_t to substitutions ($\text{is_ground}_{\text{subst}}$), atoms (is_ground_a), literals ($\text{is_ground}_{\text{lit}}$), and clauses (is_ground_c).

► **Definition 16.** The function $\text{groundings}_c :: (f, v)$ *atom clause* \Rightarrow *f gatom clause set* maps a clause to the set of its groundings: $\text{groundings}_c C = \{C\gamma \mid \text{is_ground}_c(C\gamma)\}$.

The calculus is parameterized by a well-founded total ordering on ground terms as defined in the locale `ground_ordering`. We introduce a locale for the nonground ordering:

locale `first_order_ordering` =
fixes $(\prec_t) :: (f, v)$ *term* $\Rightarrow (f, v)$ *term* \Rightarrow *bool*
assumes
 $\text{transp } (\prec_t)$ **and** $\text{asym } (\prec_t)$ **and**
 $\text{totalp_on } \{t \mid \text{is_ground}_t t\} (\prec_t)$ **and** $\text{wfp_on } \{t \mid \text{is_ground}_t t\} (\prec_t)$ **and**
 $\forall \kappa :: (f, v)$ *ctxt*. $\forall t_1 t_2. \text{is_ground}_t t_1 \rightarrow \text{is_ground}_t t_2 \rightarrow \text{is_ground}_{\text{ctxt}} \kappa \rightarrow$
 $t_1 \prec_t t_2 \rightarrow \kappa[t_1] \prec_t \kappa[t_2]$ **and**
 $\forall \kappa :: (f, v)$ *ctxt*. $\forall t. \text{is_ground}_t t \rightarrow \text{is_ground}_{\text{ctxt}} \kappa \rightarrow \kappa \neq \square \rightarrow t \prec_t \kappa[t]$ **and**
 $\forall t_1 t_2 \gamma. \text{is_ground}_t (t_1 \gamma) \rightarrow \text{is_ground}_t (t_2 \gamma) \rightarrow t_1 \prec_t t_2 \rightarrow t_1 \gamma \prec_t t_2 \gamma$

We assume transitivity and asymmetry on the entire relation. We assume totality and well-foundedness on ground terms using `totalp_on` and `wfp_on`. We also assume compatibility with ground contexts and the subterm property for ground terms. Finally, we assume *stability under grounding substitutions*: If two terms are \prec_t -related, then they are still \prec_t -related after applying a grounding substitution. Most of the restrictions to ground terms are not

necessary for practical orders such as the Knuth–Bendix ordering and the lexicographic path ordering, but we prefer the additional generality.

In the locale context of `first_order_ordering`, we define \prec_{tG} as \prec_t on ground terms. We lift \prec_t to literals (\prec_{lit}) and clauses (\prec_c). We then prove transitivity, asymmetry, totality, well-foundedness, and stability under grounding substitutions for \prec_{lit} and \prec_c .

The next lemma will be useful to prove that nonground inferences overapproximate ground inferences:

► **Lemma 17.** *Let C be a clause, $L \in C$ a literal, and γ a grounding substitution for C . If $L\gamma$ is (strictly) maximal in $C\gamma$, then L is (strictly) maximal in C .*

Next to the ordering, the calculus is also parameterized by a selection function. Let $(f, 'v)$ *select* abbreviate $(f, 'v)$ *atom clause* $\Rightarrow (f, 'v)$ *atom clause* and $'fgselect$ abbreviate $'fgatom clause \Rightarrow 'fgatom clause$. We define

```
locale first_order_select = select sel
  for sel :: ('f, 'v) select
```

The locale `first_order_select` extends the locale `select` and fixes the type of its selection function `sel`. We use this locale to prove some lemmas regarding `sel` and grounding substitutions. The blueprint also initially assumed *stability under renaming*: $\text{sel}(C\rho) = (\text{sel } C)\rho$ for every clause C and every renaming ρ . This assumption was taken from the formal completeness proof for the resolution calculus [30], but it turns out to be unnecessary in our formalization because we use a different approach to lift ground inferences.

The following predicate is useful to lift selection functions:

► **Definition 18.** The predicate $\text{is_grounding}_S :: ('f, 'v) \text{select} \Rightarrow 'fgselect \Rightarrow \text{bool}$ relates two selection functions, S for nonground clauses and S_G for ground clauses: $\text{is_grounding}_S S S_G \longleftrightarrow \forall C_G :: 'fgatom clause. \exists C :: ('f, 'v) atom clause. \exists \gamma. C_G = C\gamma \wedge S_G C_G = (S C)\gamma$.

Using is_grounding_S , we define the following in the context of the `first_order_select` locale:

► **Definition 19.** The set $\text{gselects} :: 'fgselect \text{ set}$ consists of all ground selection functions related to `sel`: $\text{gselects} = \{S_G \mid \text{is_grounding}_S \text{ sel } S_G\}$.

Based on `gselects`, we lift the ground calculus for all ground selection functions. The following locale associates a selection function `sel` with a grounding `selG`:

```
locale grounded_first_order_select = first_order_select sel
  for sel :: ('f, 'v) select +
  fixes sel_G :: 'fgselect
  assumes is_grounding_S sel sel_G
```

We show that the grounding `selG` fulfills the criteria for a ground-level selection function:

```
sublocale grounded_first_order_select  $\subseteq$  select where
  sel = sel_G
```

This sublocale relation establishes the lifting for the selection function.

We continue by creating the main building block for the nonground calculus:

```
locale first_order_superposition_calculus =
  first_order_ordering ( $\prec_t$ ) + first_order_select sel
  for
```

$(\prec_t) :: (f, 'v) \text{ term} \Rightarrow (f, 'v) \text{ term} \Rightarrow \text{bool}$ **and**
 $\text{sel} :: (f, 'v) \text{ select} +$
fixes tiebreakers $:: 'f \text{ gatom clause} \Rightarrow (f, 'v) \text{ atom clause} \Rightarrow (f, 'v) \text{ atom clause} \Rightarrow \text{bool}$
assumes
infinite (UNIV $:: 'v \text{ set}$) **and**
 $\forall C_G. \text{wfp}(\text{tiebreakers } C_G) \wedge \text{transp}(\text{tiebreakers } C_G) \wedge \text{asyp}(\text{tiebreakers } C_G)$ **and**
 $\forall R :: (f \text{ gterm} \times 'f \text{ gterm}) \text{ set. ground_critical_pair_theorem } R$

We define the locale `first_order_superposition_calculus` extending `first_order_ordering` and `first_order_select`. We need three additional assumptions: First, we assume that the set of all variables (UNIV $:: 'v \text{ set}$) is infinite, so that we can generate enough fresh variables for renamings. Second, we support **tiebreakers**, i.e., a family of well-founded partial orderings indexed by ground clauses. The orderings can be used to implement subsumption. Third, we assume the theorem of ground critical pairs as in Section 4.

Next, we define the three inference rules **superposition**, **eq_resolution**, and **eq_factoring** presented in Section 2 inside `first_order_superposition_calculus` using inductive predicates. The **superposition** rule follows as an example:

$$\frac{\overbrace{t \approx t' \vee D'}^D \quad \overbrace{\kappa[u] \bowtie u' \vee E'}^E}{\underbrace{((\kappa\rho_2)[t'\rho_1] \bowtie u'\rho_2 \vee D'\rho_1 \vee E'\rho_2)\mu}_C} \text{superposition } D \ E \ C$$

Side conditions:

1. $\bowtie \in \{\approx, \not\approx\}$; 2. ρ_1 and ρ_2 are renamings; 3. $D\rho_1$ and $E\rho_2$ are variable-disjoint;
4. u is not a variable; 5. μ is an IMGU of $\{t\rho_1, u\rho_2\}$; 6. $E\rho_2\mu \not\prec_c D\rho_1\mu$;
7. $t\rho_1\mu \not\prec_t t'\rho_1\mu$; 8. $(\kappa[u])\rho_2\mu \not\prec_t u'\rho_2\mu$;
9. if $\bowtie = \approx$, then $\text{sel } E = \{\}$ and $(\kappa[u] \bowtie u')\rho_2\mu$ is strictly maximal in $E\rho_2\mu$;
10. if $\bowtie = \not\approx$, then $\text{sel } E = \{\}$ and $(\kappa[u] \bowtie u')\rho_2\mu$ is maximal in $E\rho_2\mu$ or $(\kappa[u] \bowtie u')\rho_2\mu$ is maximal in $(\text{sel } E)\rho_2\mu$;
11. $\text{sel } D = \{\}$; 12. $(t \approx t')\rho_1\mu$ is strictly maximal in $D\rho_1\mu$.

Unlike in the blueprint, we do not fix functions to create the renamings and the IMGUs. Instead, we use predicates that describe the properties of renamings and IMGUs. This gives more flexibility to a saturation prover based on the calculus, which could, for example, use a nondeterministic implementation.

► **Definition 20.** The set $\text{Inf}_F :: (f, 'v) \text{ atom clause inference set}$ consists of all inferences of the superposition calculus:

$$\text{Inf}_F = \{ \langle [D, E], C \rangle \mid \text{superposition } D \ E \ C \} \cup \{ \langle [D], C \rangle \mid \text{eq_resolution } D \ C \} \cup \{ \langle [D], C \rangle \mid \text{eq_factoring } D \ C \}$$

Concluding the setup for the lifting of the calculus, we define

locale grounded_first_order_superposition_calculus =
first_order_superposition_calculus + grounded_first_order_select

By combining `first_order_superposition_calculus` and `grounded_first_order_select`, we provide an arbitrary ground select function sel_G to the nonground calculus. The resulting locale `grounded_first_order_superposition_calculus` has all the required assumptions to instantiate `ground_superposition_calculus`:

```

sublocale grounded_first_order_superposition_calculus  $\subseteq$ 
  ground_superposition_calculus where
    selG = selG and  $\prec_t = \prec_{tG}$ 

```

The selection function causes some complications. In general, **sel** is not stable under substitutions (i.e., $(\text{sel } C)\sigma$ and $\text{sel } (C\sigma)$ might be different). As a result, we cannot directly use it at the ground level. Based on **sel** and a saturated set N , we would want to define a suitable ground selection function S_G . However, this definition cannot work, because N is not a priori known. The solution is as follows: For all ground selection functions in **gselects**, we lift the corresponding ground calculi to the nonground level and consider all of them together. This ensures that we perform the right lifting regardless of N .

The locale **lifting_intersection** [42, Section 3.1] of the saturation framework enables us to lift a family of ground calculi indexed by **gselects**. We instantiate the locale as a sublocale of **first_order_superposition_calculus**. Since we lift a ground calculus family and not a single calculus, we cannot have a global **ground_superposition_calculus**. However, once we have an arbitrary but fixed ground selection function $S_G \in \text{gselects}$, we can use Isabelle’s facility for instantiating locales locally in a proof using **interpret**.

The locale **lifting_intersection** gives us the following lifted definition of the entailment relation for nonground clause sets:

► **Definition 21.** The predicate $\text{entails}_F :: (f, v) \text{ atom clause set} \Rightarrow (f, v) \text{ atom clause set} \Rightarrow \text{bool}$ expresses that a clause set N_1 entails another clause set N_2 : $\text{entails}_F N_1 N_2 \iff \text{entails}_G (\bigcup_{C \in N_1} \text{groundings}_c C) (\bigcup_{C \in N_2} \text{groundings}_c C)$.

Next, we ensure that the nonground calculus is sound and compatible with the saturation framework by instantiating the **sound_inference_system** locale:

```

sublocale first_order_superposition_calculus  $\subseteq$  sound_inference_system where
  Inf = InfF and Bot =  $\{\perp\}$  and entails = entailsF

```

The locale **lifting_intersection** provides a lifted redundancy criterion $\langle \text{Red}_I, \text{Red}_F \rangle$ supporting full subsumption based on tiebreakers. Additionally, it reduces our proof of static refutational completeness for **first_order_superposition_calculus** to two easier proof obligations: First, every member of the ground calculus family is statically refutationally complete. We can prove this directly by the main result of Section 4. Second, there exists a selection function grounding that is indexing a member of the ground calculus family, with which the nonground inferences overapproximate all ground inferences. We prove this below.

► **Definition 22.** The function $\text{groundings}_{\text{Inf}} :: (f, v) \text{ atom clause inference} \Rightarrow f \text{ gatom clause inference set}$ maps an inference to the set of ground inferences that can arise by grounding its premises and conclusions: $\text{groundings}_{\text{Inf}} \iota = \{\iota\gamma \mid \iota\gamma \in \text{Inf}_G\}$.

► **Lemma 23** (Equality Resolution Lifting). *Let C and D be clauses, γ be a grounding substitution for C and D , and ι_G be a ground inference. If $\text{sel}_G(D\gamma) = (\text{sel } D)\gamma$ and $\iota_G = \text{ground_eq_resolution}(D\gamma)(C\gamma)$, then there exist C' and ι such that $C\gamma = C'\gamma$, $\iota = \text{eq_resolution } D C'$, and $\iota_G \in \text{groundings}_{\text{Inf}} \iota$.*

Note that the first assumption further restricts the relation between **sel** and sel_G . We will see in Lemmas 26 and 27 how we can discharge this assumption within the locale.

Proof sketch. We first deconstruct the premise D using the properties of its grounding $D\gamma$ induced by the ground inference to put it in the correct form for the nonground inference. Then we construct a matching conclusion C' and show that $C\gamma = C'\gamma$. The more restricted relation between **sel** and sel_G is required to lift the side condition about selection. ◀

► **Lemma 24** (Equality Factoring Lifting). *Let C and D be clauses, γ be a grounding substitution for C and D , and ι_G be a ground inference. If $\text{sel}_G(D\gamma) = (\text{sel } D)\gamma$ and $\iota_G = \text{ground_eq_factoring}(D\gamma)(C\gamma)$, then there exist C' and ι such that $C\gamma = C'\gamma$, $\iota = \text{eq_factoring } D \ C'$, and $\iota_G \in \text{groundings}_{\text{Inf}} \iota$.*

Proof sketch. Analogous to Lemma 23. ◀

► **Lemma 25** (Superposition Lifting). *Let C, D, E be clauses, γ be a grounding substitution for C, D, E , ρ_1, ρ_2 be renamings such that $D\rho_1$ and $E\rho_2$ are variable-disjoint, and ι_G be a ground inference. If $\text{sel}_G(D\rho_1\gamma) = (\text{sel } (D\rho_1))\gamma$, $\text{sel}_G(E\rho_2\gamma) = (\text{sel } (E\rho_2))\gamma$, $\iota_G = \text{ground_superposition}(D\rho_1\gamma)(E\rho_2\gamma)(C\gamma)$, and $\iota_G \notin \text{Red}_{\text{IG}}(\text{groundings}_c D \cup \text{groundings}_c E)$, then there exist C' and ι such that $C\gamma = C'\gamma$, $\iota = \text{superposition } DEC'$, and $\iota_G \in \text{groundings}_{\text{Inf}} \iota$.*

Compared with Lemmas 23 and 24, there are two additions: First, nonground superposition inferences require their premises to be variable-disjoint. Therefore, the lemma is parameterized by renamings ρ_1 and ρ_2 . Second, we assume that ι_G is not redundant. This is unproblematic: The lemma is used only to prove that nonground inferences overapproximate ground inferences, and there we need the lifting only in the nonredundant case.

Proof sketch. The proof is similar to those of the previous two lemmas. A subtlety is that superposition avoids inferences into variables (side condition 4). We must show that ground inferences whose lifting would result in an inference into a variable are redundant according to Red_{IG} . We sketch the proof with two examples:

- Consider the saturated clause set $N = \{b \approx a, g(x) \not\approx d\}$. We must show that there are no nonredundant inferences from the set of its groundings $N_G = \{b \approx a, g(a) \not\approx d, g(b) \not\approx d, g(c) \not\approx d, \dots\}$. However, we can derive the clause $g(a) \not\approx d$ using $b \approx a$ and $g(b) \not\approx d$. Fortunately, the inference is redundant since $g(a) \not\approx d$ is already contained in N_G .
- What if we have a clause with multiple occurrences of the same variable, as in $N' = \{b \approx a, g(x) \not\approx f(x)\}$? We then have $N'_G = \{b \approx a, g(a) \not\approx f(a), g(b) \not\approx f(b), g(c) \not\approx f(c), \dots\}$ and can use $b \approx a$ and $g(b) \not\approx f(b)$ to generate $g(a) \not\approx f(b)$. However, since $\{b \approx a, g(a) \not\approx f(a)\} \models \{g(a) \not\approx f(b)\}$ and $g(a) \not\approx f(a) \prec_c g(a) \not\approx f(b)$, this inference is also redundant.

The formal proof performs multiple inductions over the number of occurrences of variables. ◀

While proving the above lemmas in Isabelle, we discovered a mistake in the formulation of our blueprint. We had wrongly stated the conclusion of Lemma 23 as: “Then there exists $\iota = \text{eq_resolution } D \ C$ and \dots ” We had fixed the conclusions of the nonground inferences to C , even though many clauses can result in the same grounding w.r.t. γ . The same issue arose in Lemmas 24 and 25.

► **Lemma 26.** *Let N be a clause set, N_G be the set containing all groundings of all clauses of N , and S be an arbitrary function of type $(f, 'v)$ select. Then there exists a function S_G of type $f g \text{select}$ such that $\text{is_grounding}_S S \ S_G$ and $\forall C_G \in N_G. \exists C \gamma. C \in N \wedge C_G = C\gamma \wedge S_G C_G = (S \ C)\gamma$.*

Proof sketch. We construct a S_G on N_G such that it fulfills the last property. This is possible since the elements of N_G are the groundings of the elements of N . It follows directly from the definition of groundings_c that S_G on N_G is a grounding of S . For clauses not in N_G , we define S_G as the ground restriction of S . ◀

► **Lemma 27** (Overapproximation). *Let N be a clause set. Then there exists a ground calculus with a ground selection function S_G such that nonground inferences from N overapproximate all ground inferences from the groundings of N , meaning that each ground inference from the groundings_c of N is either contained in the groundings_{Inf} of inferences from N or redundant.*

Proof sketch. The proof follows from the lifting lemmas (Lemmas 23–25), for which we can obtain the required ground selection functions using Lemma 26. For Lemma 25, we also need the assumption that the set of all variables is infinite to be able to provide the renamings. ◀

Using Lemma 27, we can conclude our endeavor and instantiate `statically_complete_calculus`:

```
sublocale first_order_superposition_calculus ⊆ statically_complete_calculus where
  Inf = InfF and Bot = {⊥} and entails = entailsF and less = (≺c) and
  RedI = RedI and RedF = RedF
```

We have verified the static refutational completeness of first-order superposition. The saturation framework provides us with a proof of dynamic refutational completeness.

Finally, to exclude any inconsistency in our assumptions, we instantiate the locale `first_order_superposition_calculus` with a trivial select function, trivial tiebreakers, and the verified Knuth–Bendix ordering [37]. We can discharge all proof obligations. Since `first_order_superposition_calculus` transitively instantiates all the other locales, we cover all our assumptions.

6 Related Work

The saturation framework [44] has been used in the completeness proof of several new variants of superposition:

- Boolean λ -superposition [8] for higher-order logic, as well as its predecessors Boolean-free λ -superposition [9] and Boolean-free λ -free superposition [7] that operate on fragments of higher-order logic.
- superposition with delayed unification [11] for first-order logic, which adds constraints to the conclusions of inferences instead of performing full unifications.

An extended abstract by Tourret [41] discusses how these use the framework. The work described in the present paper could serve as a foundation to formalize these proofs.

The Isabelle/HOL formalization of the saturation framework was introduced together with an instance of the resolution calculus and an abstract resolution prover called RP by Tourret and Blanchette [42]. Other theorem proving techniques formalized in Isabelle/HOL include an executable SAT solver by Blanchette et al. [13] based on CDCL (conflict-driven clause learning) for propositional logic with state-of-the-art optimizations, various sequent and tableau calculi for first-order and related logics by From et al. [19, 20], and another version of resolution and RP by Schlichtkrull et al. [30] following Bachmair and Ganzinger’s original, more ad hoc proof that was extended to an executable prover [31]. Most recently, the newly created SCL calculus [18], which follows a CDCL-like approach to theorem proving in first-order logic, was also verified in Isabelle/HOL by Bromberger et al. [15] as it was being developed. Also relevant here is Paulson’s formalization of Gödel’s incompleteness theorems [25, 26].

Isabelle/HOL is possibly the most widely used system for formalizing automated reasoning results, but other proof assistants are used as well. Early results include Shankar’s proof of Gödel’s first incompleteness theorem in Nqthm [34], Persson’s completeness proof for intuitionistic predicate logic in ALF [28], and Harrison’s formalization of basic first-order model theory in HOL Light [21]. We refer to Blanchette [12, Section 5] for a survey.

Finally, the work closest to ours, already mentioned in the introduction, is the formalization of a variant of the superposition calculus in Isabelle/HOL by Peltier [27]. Our initial intent

was to integrate his calculus with the saturation framework, but after months of fruitless attempts, we decided to start from scratch, which resulted in the present work.

A first obstacle we encountered was related to Peltier’s redundancy criterion. He relies on a notion that is sufficient to prove static refutational completeness but cannot be lifted to dynamic completeness because his redundancy is defined in terms of smaller or equal clauses rather than strictly smaller clauses. This makes it unsuitable for use in the saturation framework, but we managed to replace it with a suitable criterion without changing the calculus, allowing us to pursue our work in this direction for a while.

What made us switch approach was an incompatibility requiring a major modification of Peltier’s formalization itself. Peltier works with closures, i.e., pairs $\langle C, \sigma \rangle$ consisting of a set of literals C and a substitution σ . This calculus is defined directly on the nonground level, where static completeness is proved. For integration into the saturation framework, we wanted a ground version of the calculus, which we obtained by restricting the substitutions to groundings only and operating on clauses as sets of ground literals $C\sigma$. However, this made it impossible to overapproximate this calculus with Peltier’s calculus on the nonground level, which is needed for the lifting to be possible in the framework. The issue is that we do not want to match a literal K in a ground clause to two literals L_1, L_2 in a nonground closure $\langle L_1 \vee L_2 \vee C, \sigma \rangle$ such that $L_1\sigma = L_2\sigma = K$, because this breaks the lifting. Fixing this would require working directly on closures also at the ground level. A new proof of ground refutational completeness would have had to be provided for this new calculus. It seemed more convenient to formalize a calculus operating on *multisets* of literals instead of closures, especially that the Isabelle multiset library was already well developed for use in theorem proving formalization.

Our formalization consists of 12 000 nonblank lines, 7000 of which are for nonbackground theories. For comparison, Peltier’s formalization consists of 9000 nonblank lines, 7000 of which are for nonbackground theories. All numbers are rounded to the nearest thousand. Interestingly, the two formalizations have approximately the same size even though they are written in very different styles. To our surprise, the additional modularity of our work did not lead to a shorter proof.

7 Conclusion

We restructured the refutational completeness proof of superposition using the saturation framework. We first proved refutational completeness for the ground calculus and lifted the proof to the full, nonground calculus. Next, we formalized this pen-and-paper proof in Isabelle/HOL. The formalization can be seen as a case study for the *IsaFoR* library and the saturation framework, as well as for basic Isabelle tools such as *locales*, which facilitate modularity and proof reuse.

We see three main directions for future work. First, the proof could be extended to support simply typed or rank-1-polymorphic first-order terms. Second, the completeness proofs of variants of superposition, such as hierarchic superposition [5, 6], combinatory superposition [10], and λ -superposition [8], could be formalized as well. Third, the formalization of superposition (or that of variants) could be extended to obtain a verified executable prover.

References

- 1 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

- 2 Leo Bachmair and Harald Ganzinger. On restrictions of ordered paramodulation with simplification. In Mark E. Stickel, editor, *CADE-10*, volume 449 of *LNCS*, pages 427–441. Springer, 1990.
- 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 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, and Uwe Waldmann. Theorem proving for hierarchic first-order theories. In Hélène Kirchner and Giorgio Levi, editors, *ALP '92*, volume 632 of *LNCS*, pages 420–434. Springer, 1992.
- 6 Peter Baumgartner and Uwe Waldmann. Hierarchic superposition revisited. In Carsten Lutz, Uli Sattler, Cesare Tinelli, Anni-Yasmin Turhan, and Frank Wolter, editors, *Description Logic, Theory Combination, and All That—Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, volume 11560 of *LNCS*, pages 15–56. Springer, 2019.
- 7 Alexander Bentkamp, Jasmin Blanchette, Simon Cruanes, and Uwe Waldmann. Superposition for lambda-free higher-order logic. *Log. Methods Comput. Sci.*, 17(2), 2021.
- 8 Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, and Petar Vukmirovic. Superposition for higher-order logic. *J. Autom. Reason.*, 67(1):10, 2023.
- 9 Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirovic, and Uwe Waldmann. Superposition with lambdas. *J. Autom. Reason.*, 65(7):893–940, 2021.
- 10 Ahmed Bhayat and Giles Reger. A combinator-based superposition calculus for higher-order logic. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *IJCAR 2020, Part I*, volume 12166 of *LNCS*, pages 278–296. Springer, 2020.
- 11 Ahmed Bhayat, Johannes Schoisswohl, and Michael Rawson. Superposition with delayed unification. In Brigitte Pientka and Cesare Tinelli, editors, *CADE-29*, volume 14132 of *LNCS*, pages 23–40. Springer, 2023.
- 12 Jasmin Christian Blanchette. Formalizing the metatheory of logical calculi and automatic provers in Isabelle/HOL (invited talk). In Assia Mahboubi and Magnus O. Myreen, editors, *CPP 2019*, pages 1–13. ACM, 2019.
- 13 Jasmin Christian Blanchette, Mathias Fleury, Peter Lammich, and Christoph Weidenbach. A verified SAT solver framework with learn, forget, restart, and incrementality. *J. Autom. Reason.*, 61(3):333–366, 2018.
- 14 Jasmin Christian Blanchette and Sophie Tourret. Extensions to the comprehensive framework for saturation theorem proving. *Archive of Formal Proofs*, 2020. https://isa-afp.org/entries/Saturation_Framework_Extensions.html.
- 15 Martin Bromberger, Martin Desharnais, and Christoph Weidenbach. An Isabelle/HOL formalization of the SCL(FOL) calculus. In Brigitte Pientka and Cesare Tinelli, editors, *CADE-29*, volume 14132 of *LNCS*, pages 116–133. Springer, 2023.
- 16 Simon Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond*. Ph.D. thesis, École polytechnique, 2015.
- 17 Martin Desharnais and Balazs Toth. Superposition calculus, 2024. https://github.com/IsaFoL/IsaFoL/tree/ITP2024-IsaSuperposition/Superposition_Calculus.
- 18 Alberto Fiori and Christoph Weidenbach. SCL clause learning from simple models. In Pascal Fontaine, editor, *CADE-27*, volume 11716 of *LNCS*, pages 233–249. Springer, 2019.
- 19 Asta Halkjær From, Patrick Blackburn, and Jørgen Villadsen. Formalizing a Seligman-style tableau system for hybrid logic (short paper). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *IJCAR 2020, Part I*, volume 12166 of *LNCS*, pages 474–481. Springer, 2020.
- 20 Asta Halkjær From, Anders Schlichtkrull, and Jørgen Villadsen. A sequent calculus for first-order logic formalized in Isabelle/HOL. *J. Log. Comput.*, 33(4):818–836, 2023.

- 21 John Harrison. Formalizing basic first order model theory. In Jim Grundy and Malcolm C. Newey, editors, *TPHOLs '98*, volume 1479 of *LNCS*, pages 153–170. Springer, 1998.
- 22 Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
- 23 Alexander Lochmann, Bertram Felgenhauer, Christian Sternagel, René Thiemann, and Thomas Sternagel. Regular tree relations. *Archive of Formal Proofs*, 2021. https://isa-afp.org/entries/Regular_Tree_Relations.html.
- 24 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
- 25 Lawrence C. Paulson. A machine-assisted proof of Gödel’s incompleteness theorems for the theory of hereditarily finite sets. *Rev. Symb. Log.*, 7(3):484–498, 2014.
- 26 Lawrence C. Paulson. A mechanised proof of Gödel’s incompleteness theorems using Nominal Isabelle. *J. Autom. Reason.*, 55(1):1–37, 2015. URL: <https://doi.org/10.1007/s10817-015-9322-8>, doi:10.1007/s10817-015-9322-8.
- 27 Nicolas Peltier. A variant of the superposition calculus. *Archive of Formal Proofs*, 2016. <https://www.isa-afp.org/entries/SuperCalc.html>.
- 28 Henrik Persson. Constructive completeness of intuitionistic predicate logic: A formalisation in type theory. Licentiate thesis, Chalmers tekniska högskola and Göteborgs universitet, 1996.
- 29 John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
- 30 Anders Schlichtkrull, Jasmin Blanchette, Dmitriy Traytel, and Uwe Waldmann. Formalizing Bachmair and Ganzinger’s ordered resolution prover. *J. Autom. Reason.*, 64(7):1169–1195, 2020.
- 31 Anders Schlichtkrull, Jasmin Christian Blanchette, and Dmitriy Traytel. A verified prover based on ordered resolution. In Assia Mahboubi and Magnus O. Myreen, editors, *CPP 2019*, pages 152–165. ACM, 2019.
- 32 Anders Schlichtkrull, Jasmin Christian Blanchette, Dmitriy Traytel, and Uwe Waldmann. Formalization of Bachmair and Ganzinger’s ordered resolution prover. *Archive of Formal Proofs*, 2018. https://isa-afp.org/entries/Ordered_Resolution_Prover.html.
- 33 Stephan Schulz, Simon Cruanes, and Petar Vukmirović. Faster, higher, stronger: E 2.3. In Pascal Fontaine, editor, *CADE-27*, volume 11716 of *LNCS*, pages 495–507. Springer, 2019.
- 34 Natarajan Shankar. *Metamathematics, Machines, and Gödel’s Proof*, volume 38 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1994.
- 35 Christian Sternagel and René Thiemann. Abstract rewriting. *Archive of Formal Proofs*, 2010. <https://isa-afp.org/entries/Abstract-Rewriting.html>.
- 36 Christian Sternagel and René Thiemann. First-order terms. *Archive of Formal Proofs*, 2018. https://isa-afp.org/entries/First_Order_Terms.html.
- 37 Christian Sternagel and René Thiemann. A formalization of KnuthBendix orders. *Archive of Formal Proofs*, 2020. https://isa-afp.org/entries/Knuth_Bendix_Order.html.
- 38 Lukas Stevens and Tobias Nipkow. A verified decision procedure for orders in Isabelle/HOL. In Zhe Hou and Vijay Ganesh, editors, *ATVA 2021*, volume 12971 of *LNCS*, pages 127–143. Springer, 2021.
- 39 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLs 2009*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009.
- 40 Sophie Tournet. A comprehensive framework for saturation theorem proving. *Archive of Formal Proofs*, 2020. https://www.isa-afp.org/entries/Saturation_Framework.html.
- 41 Sophie Tournet. The spawns of the saturation framework. In Laura Kovács and Michael Rawson, editors, *7th and 8th Vampire Workshop*, volume 99 of *EPiC Series in Computing*, pages 1–6. EasyChair, 2024.

- 42 Sophie Tourret and Jasmin Blanchette. A modular Isabelle framework for verifying saturation provers. In Călin Hricu and Andrei Popescu, editors, *CPP 2021*, pages 224–237. ACM, 2021.
- 43 Uwe Waldmann. A modular completeness proof for the superposition calculus, 2024. https://nekoka-project.github.io/pubs/isasup_blueprint.pdf.
- 44 Uwe Waldmann, Sophie Tourret, Simon Robillard, and Jasmin Blanchette. A comprehensive framework for saturation theorem proving. *J. Autom. Reason.*, 66(4):499–539, 2022.
- 45 Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischniewski. SPASS version 3.5. In Renate A. Schmidt, editor, *CADE-22*, volume 5663 of *LNCS*, pages 140–145. Springer, 2009.