Journal of Automated Reasoning (2020) 64:1169–1195
https://doi.org/10.1007/s10817-020-09561-0

Formalizing Bachmair and Ganzinger's Ordered Resolution Prover

Anders Schlichtkrull[1] · Jasmin Blanchette[2,3] · Dmitriy Traytel[4] · Uwe Waldmann[3]

Received: 24 April 2020 / Accepted: 27 April 2020 / Published online: 17 June 2020

Abstract

We present an Isabelle/HOL formalization of the first half of Bachmair and Ganzinger's chapter on resolution theorem proving, culminating with a refutationally complete first-order prover based on ordered resolution with literal selection. We developed general infrastructure and methodology that can form the basis of completeness proofs for related calculi, including superposition. Our work clarifies fine points in the chapter, emphasizing the value of formal proofs in the field of automated reasoning.

Keywords Resolution calculus · Automatic theorem provers · Proof assistants

1 Introduction

Much research in automated reasoning amounts to metatheoretical arguments, typically about the soundness and completeness of logical inference systems or the termination of theorem proving processes. Often the proofs contain more insights than the systems or processes

✉ Anders Schlichtkrull
andschl@dtu.dk

Jasmin Blanchette
j.c.blanchette@vu.nl

Dmitriy Traytel
traytel@inf.ethz.ch

Uwe Waldmann
uwe@mpi-inf.mpg.de

[1] DTU Compute, Department of Applied Mathematics and Computer Science, Technical University of Denmark, Richard Petersens Plads, Building 324, 2800 Kongens Lyngby, Denmark

[2] Department of Computer Science, Section of Theoretical Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

[3] Max-Planck-Institut für Informatik, Saarland Informatics Campus E1 4, 66123 Saarbrücken, Germany

[4] Institute of Information Security, Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zurich, Switzerland

themselves. For example, the superposition calculus rules [2], with their many side conditions, look rather arbitrary, whereas in the completeness proof the conditions emerge naturally from the model construction. And yet, despite being crucial to our field, today such proofs are usually carried out without tool support. We believe proof assistants are becoming mature enough to help.

In this article, we present a formalization, developed using the Isabelle/HOL system [28], of a first-order prover based on ordered resolution with literal selection. We follow Bachmair and Ganzinger's account [4] from Chapter 2 of the *Handbook of Automated Reasoning*, which we refer to as simply "the chapter." Our formal development covers the refutational completeness of two resolution calculi for ground (i.e., variable-free) clauses and general infrastructure for theorem proving processes and redundancy. It culminates with a completeness proof for a first-order prover expressed as transition rules operating on triples of clause sets. This material corresponds to the chapter's first four sections.

From the perspective of automated reasoning, increased trustworthiness of the metatheory of automatic theorem provers is an obvious benefit of formal proofs. But formalizing also helps clarify arguments, by exposing and explaining difficult steps. Making definitions and theorem statements precise can be a huge gain for communicating metatheoretical results. Moreover, a formal proof can tell us exactly where hypotheses and lemmas are used. Once we have created a rich library of basic results and a methodology, we will be in a good position to study extensions and variants. Given that automatic provers are integrated into modern proof assistants, there is also an undeniable thrill in applying these tools to reason about their own metatheory.

From the perspective of interactive theorem proving, formalization work constitutes a case study in the use of a proof assistant. It gives us, as developers and users of such a system, an opportunity to experiment, contribute to lemma libraries, and get inspiration for new features and improvements.

Our motivation for choosing Bachmair and Ganzinger's chapter is manifold. The text is a standard introduction to superposition-like calculi (together with *Handbook* Chapters 7 [25] and 27 [49]). It offers perhaps the most detailed treatment of the lifting of a resolution-style calculus's static completeness to a saturation prover's dynamic completeness. It introduces a considerable amount of general infrastructure, including different types of inference systems (sound, reductive, counterexample-reducing, etc.), theorem proving processes, and an abstract notion of redundancy. The resolution calculus, extended with a term order and literal selection, captures most of the insights underlying superposition-like calculi [2,3,6,7,19,24,46], but with a simple notion of model.

The chapter's level of rigor is uneven, as shown by the errors and imprecisions that we discovered. We will see that the main completeness result does not hold, due to the improper treatment of self-inferences. Naturally, our objective is not to diminish Bachmair and Ganzinger's outstanding achievements, which include the development of superposition; rather, it is to demonstrate that even the work of some of the most celebrated researchers in our field can benefit from formalization. Our view is that formal proofs can be used to complement and improve their informal counterparts.

This work is part of the IsaFoL (Isabelle Formalization of Logic) project [9], which aims at developing a library of results about logical calculi used in automated reasoning. The Isabelle theory files are available in the *Archive of Formal Proofs* [38]. They amount to about 8000 lines of source text. A good way to study the theory files is to open them in Isabelle/jEdit [51], an integrated development environment for formal proof. This will ensure that logical and mathematical symbols are rendered properly (e.g., ∀ instead of `\<forall>`) and let you inspect proof states. We used Isabelle version 2017, but the *Archive* is continuously updated to track Isabelle's evolution.

An earlier version of this work was presented at IJCAR 2018 [39]. This article extends the conference paper with in-depth discussions of many formalization aspects, notably: some hurdles arising from ordering multisets of multisets of literals (Sect. 2); examples demonstrating Isabelle's proof language (Sect. 3); and details concerning the resolution rules, including discussions of their soundness (Sects. 4 and 6). Compared with the conference paper, we made the article more self-contained with respect to the chapter, quoting the main definitions from the chapter and contrasting them with their formal counterparts. Nevertheless, we still assume that the reader is familiar with the chapter's content. Finally, we added Appendix A, which summarizes the mathematical errors and imprecisions we discovered in the chapter in the course of formalization.

2 Preliminaries

Ordered resolution depends on little background metatheory that needs to be formalized using Isabelle. Much of it, concerning partial and total orders, well-foundedness, and finite multisets, is provided by standard Isabelle libraries. We also need literals, clauses, models, terms, and substitutions.

2.1 Isabelle

Isabelle/HOL [28] is a proof assistant based on classical higher-order logic (HOL) [20] with Hilbert choice, the axiom of infinity, rank-1 polymorphism, and type classes. HOL notations are similar to those of functional programming languages. Functions are applied without parentheses or commas (e.g., $\mathsf{f}\ x\ y$). Through syntactic abbreviations, many traditional notations from mathematics are provided, notably to denote simply typed sets and multisets. We refer to Nipkow and Klein [27, Part 1] for a modern introduction to Isabelle.

2.2 Multisets

Multisets are central to our development. Isabelle provides a multiset library, but it is much less developed than those of sets and lists. In the context of the IsaFoL effort, we had already extended it considerably and implemented further additions in a separate file (`Multiset_More.thy`). Some of these, notably a plugin for Isabelle's simplifier to apply cancellation laws, are described elsewhere [11, Sect. 3].

2.3 Clauses and Models

We used the same library of clauses (`Clausal_Logic.thy`) as for the verified SAT solver by Blanchette et al. [10], which is also part of IsaFoL. Atoms are represented by a type variable $'a$, which can be instantiated by arbitrary concrete types—e.g., numbers or first-order terms. A literal, of type $'a$ *literal* (where the type constructor is written in ML-style postfix syntax), can be of the form $\mathsf{Pos}\ A$ or $\mathsf{Neg}\ A$, where $A :: 'a$ is an atom. The literal order $>$ (written $\succ$ in the chapter) extends a fixed atom order $>$ by comparing polarities to break ties, with $\mathsf{Neg}\ A > \mathsf{Pos}\ A$.

Following the chapter, a clause is defined as a finite multiset of literals, $'a\ clause = {}'a\ literal\ multiset$, where *multiset* is the Isabelle type constructor of finite multisets. Thus, the clause $A \vee B$, where A and B are atoms, is identified with the multiset $\{A, B\}$; the clause $C \vee D$, where C and D are clauses, is $C \uplus D$; and the empty clause $\perp$ is $\{\}$. The clause order is the multiset extension [17] of the literal order.

A Herbrand interpretation I (`Herbrand_Interpretation.thy`), of type $'a\ set$, specifies which ground atoms are true. The "models" operator $\vDash$ is defined in the usual way on atoms, literals, clauses, sets, and multisets of clauses; e.g., $I \vDash C \iff \exists L \in C.\ I \vDash L$. Satisfiability of a set or multiset of clauses N is defined by $\mathsf{sat}\ N \iff \exists I.\ I \vDash N$.

The main hurdle we faced concerned multisets. Multisets of clauses have type $'a\ literal\ multiset\ multiset$. The corresponding order is the multiset extension of the clause order. In Isabelle, the multiset order was called #⊂#, and it relied on the element type's $<$ operator, through Isabelle's type class mechanism. Unfortunately, for multisets, $<$ was defined as the subset relation, so when nesting multisets (as $'a\ multiset\ multiset$), we obtained the multiset extension of the subset relation. Initially, we worked around the issue by defining an order #⊂## on multisets of multisets, but we also saw potential for improvement. After some discussions on the Isabelle users' mailing list, we let $<$ be the multiset order. To avoid introducing subtle changes in the semantics of existing developments, we first renamed $<$ to something else, freeing up $<$; then, in the next Isabelle release, we replaced #⊂# and #⊂## by $<$. In the intermediate state, all occurrences of $<$ were flagged as errors, easing porting.

2.4 Terms and Substitutions

The IsaFoR (Isabelle Formalization of Rewriting) library, an inspiration for IsaFoL, contains a definition of first-order terms and results about substitutions and unification [43]. It made sense to reuse this functionality. A practical issue is that most of IsaFoR is not accessible from the *Archive of Formal Proofs*.

Resolution depends only on basic properties of terms and atoms, such as the existence of most general unifiers (MGUs). We exploited this to keep the development parameterized by a type of atoms $'a$ and an abstract type of substitutions $'s$, through Isabelle locales [5] (`Abstract_Substitution.thy`). A locale represents a module parameterized by types and terms that satisfy some assumptions. Inside the locale, we can refer to the parameters and assumptions in definitions, lemmas, and proofs. The basic operations provided by our locale are application $(\cdot :: {}'a \Rightarrow {}'s \Rightarrow {}'a)$, identity $(\mathsf{id} :: {}'s)$, and composition $(\circ :: {}'s \Rightarrow {}'s \Rightarrow {}'s)$, about which some assumptions are made (e.g., $A \cdot \mathsf{id} = A$ for all atoms A). Substitution is lifted to literals, clauses, sets of clauses, and so on. Many other operations can be defined in terms of the primitives—for example:

$$\begin{array}{ll} \mathsf{is\_ground}\ A \iff \forall \sigma.\ A = A \cdot \sigma & \mathsf{is\_renaming}\ \sigma \iff \exists \tau.\ \sigma \circ \tau = \mathsf{id} \\ \mathsf{is\_ground}\ \sigma \iff \forall A.\ \mathsf{is\_ground}\ (A \cdot \sigma) & \mathsf{instance\_of}\ C\ D \iff \exists \sigma.\ C \cdot \sigma = D \end{array}$$

MGUs are also taken as a primitive: The $\mathsf{mgu} :: {}'a\ set\ set \Rightarrow {}'s\ option$ operation takes a set of unification constraints, each of the form $A_1 \overset{?}{=} \cdots \overset{?}{=} A_n$, and returns either an MGU or a special value (None).

Perhaps the main reason to prefer multisets to sets for representing clauses is that multisets behave better with respect to substitution. Using a set representation, applying $\sigma = \{x \mapsto \mathsf{a},\ y \mapsto \mathsf{a}\}$ to either the unit clause $C = \mathsf{p}(x)$ or the two-literal clause $D = \mathsf{p}(x) \vee \mathsf{p}(y)$ yields a unit clause $\mathsf{p}(\mathsf{a})$. This oddity breaks stability under substitution—the requirement that $D > C$ implies $D \cdot \sigma > C \cdot \sigma$.

To complete our formal development and ensure that our assumptions are legitimate, we instantiated the locale's parameters with IsaFoR types and operations and discharged its assumptions (`IsaFoR_Term.thy`).

3 Refutational Inference Systems

In their Sect. 2.4, Bachmair and Ganzinger introduce basic conventions for refutational inference systems. In Sect. 3, they present two ground resolution calculi and prove them refutationally complete in Theorems 3.9 and 3.16. In Sect. 4.2, they introduce a notion of counterexample-reducing inference system and state Theorem 4.4 as a generalization of Theorems 3.9 and 3.16 to all such systems. For formalization, two courses of actions suggest themselves: follow the book closely and prove the three theorems separately, or focus on the most general result. We chose the latter, as being more consistent with the goal of providing a well-designed, reusable library, at the cost of widening the gap between the text and its formal companion.

We collected the abstract hierarchy of inference systems in a single Isabelle theory file (`Inference_System.thy`). An inference, of type $'a$ *inference*, is a triple $(\mathcal{C}, D, E)$ that consists of a multiset of side premises $\mathcal{C}$, a main premise D, and a conclusion E. An inference system, or calculus, is a possibly infinite set of inferences:

locale *inference_system* =
  fixes Γ :: $'a$ *inference set*

The Isabelle locale fixes, within a named context (*inference_system*), a set Γ of inferences between clauses over atom type $'a$. Inside the locale, we defined a function infers_from that, given a clause set N, returns the subset of Γ inferences whose premises all belong to N.

A satisfiability-preserving (or consistency-preserving) inference system enriches the inference system locale with an assumption, whereas sound systems are characterized by a different assumption:

locale *sat_preserving_inference_system* = *inference_system* +
  assumes sat $N \Longrightarrow$ sat $(N \cup$ concl_of ‘ infers_from $N)$
locale *sound_inference_system* = *inference_system* +
  assumes $(\mathcal{C}, D, E) \in \Gamma \Longrightarrow I \models \mathcal{C} \cup \{D\} \Longrightarrow I \models E$

The notation f ‘ X above stands for the image of the set or multiset X under function f.

Soundness is a stronger requirement than satisfiability preservation. In Isabelle, this can be expressed as a sublocale relation:

sublocale *sound_inference_system* < *sat_preserving_inference_system*

This command emits a proof goal stating that *sound_inference_system*'s assumption implies *sat_preserving_inference_system*'s. Afterwards, all the definitions and lemmas about satisfiability-preserving calculi become available about sound ones.

In reductive inference systems (*reductive_inference_system*), the conclusion of each inference is smaller than the main premise according to the clause order. A related notion, the counterexample-reducing inference systems, is specified as follows:

locale *counterex_reducing_inference_system* = *inference_system* +
  fixes I_of :: $'a$ *clause set* $\Rightarrow$ $'a$ *set*
  assumes $\{\} \notin N \Longrightarrow D \in N \Longrightarrow$ I_of $N \not\models D \Longrightarrow$
    $(\forall C \in N.$ I_of $N \not\models C \Longrightarrow D \leq C) \Longrightarrow$
    $\exists \mathcal{C} \subseteq N. \exists E.$ I_of $N \models \mathcal{C} \wedge (\mathcal{C}, D, E) \in \Gamma \wedge$ I_of $N \not\models E \wedge E < D$

The parameter I_of maps clause sets to candidate models. The assumption is that for any clause set N that does not contain {} (i.e., $\bot$), if $D \in N$ is the smallest counterexample—the smallest clause in N that is falsified by I_of N—we can derive a smaller counterexample E using an inference from clauses in N. This property is useful because if N is saturated (i.e., closed under Γ inferences), we must have $E \in N$, contradicting D's minimality:

theorem *saturated_model*: saturated $N \Longrightarrow \{\} \notin N \Longrightarrow$ I_of $N \models N$
corollary *saturated_complete*: saturated $N \Longrightarrow \neg$ sat $N \Longrightarrow \{\} \in N$

Bachmair and Ganzinger claim that compactness of clausal logic follows from the refutational completeness of ground resolution (Theorem 3.12), although they give no justification. Our argument relies on an inductive definition of saturation of a set of clauses: saturate :: *'a clause set* $\Rightarrow$ *'a clause set*. Most of the work goes into proving this key lemma, by rule induction on the saturate function:

lemma *saturate_finite*: $C \in$ saturate $N \Longrightarrow \exists M \subseteq N.$ finite $M \wedge C \in$ saturate M

The interesting case is when $C = \bot$. We established compactness in a locale that combines *counterex_reducing_inference_system* and *sound_inference_system*:

theorem *clausal_logic_compact*: $\neg$ sat $N \Longleftrightarrow \exists M \subseteq N.$ finite $M \wedge \neg$ sat M

To give a taste of the formalization, here is the formal Isar [50] proof:

```
proof
  assume ¬ sat N
  then have {} ∈ saturate N
    using saturated_complete saturated_saturate saturate.base
    unfolding true_clss_def by meson
  then have ∃M ⊆ N. finite M ∧ {} ∈ saturate M
    using saturate_finite by fastforce
  then show ∃M ⊆ N. finite M ∧ ¬ sat M
    using saturate_sound by auto
next
  assume ∃M ⊆ N. finite M ∧ ¬ sat M
  then show ¬ sat N
    by (blast intro: true_clss_mono)
qed
```

The $\Longrightarrow$ direction relies on the calculus's refutational completeness to show that $\bot$ belongs to saturate N, on the above lemma to obtain a finite subset M from which $\bot$ can be derived, and on the calculus's soundness to conclude that M is unsatisfiable.

Our compactness result is meaningful only if the locale assumptions are consistent. In the next section, we will exhibit two sound counterexample-reducing calculi that can be used to instantiate the locale and retrieve an unconditional compactness theorem.

4 Ground Resolution

A useful strategy for establishing properties of first-order calculi is to initially restrict our attention to ground calculi and then to lift the results to first-order formulas containing terms with variables. Accordingly, the chapter's Sect. 3 presents two ground calculi: a simple binary resolution calculus and an ordered n-ary resolution calculus with literal selection. Both consist of a single resolution rule, with built-in positive factorization. Most of the explanations and proofs concern the simpler calculus. To avoid duplication, we factored out the candidate model construction (`Ground_Resolution_Model.thy`). We then defined the two calculi and proved that they are sound and reduce counterexamples (`Unordered_Ground_Resolution.thy`, `Ordered_Ground_Resolution.thy`).

4.1 Candidate Models

Refutational completeness is proved by exhibiting a model for any saturated clause set $N \not\ni \bot$. The model is constructed incrementally, one clause $C \in N$ at a time, starting with an empty Herbrand interpretation, in which all atoms are false. The idea appears to have originated with Brand [14] and Zhang and Kapur [52].

Bachmair and Ganzinger introduce two operators to build the candidate model: I_C denotes the current interpretation before considering C, and ε_C denotes the set of (zero or one) atoms added, or *produced*, to ensure that C is satisfied. Bachmair and Ganzinger define I_C and ε_C as follows (Definition 3.14):

> Take I_C to be the set $\bigcup_{C>D} \varepsilon_D$. Furthermore, if C is a clause that
>
> (i) is contained in N;
> (ii) is of the form $C' \vee A$, where A is the maximal literal in C;
> (iii) is false in I_C; and
> (iv) nothing is selected in C;
>
> then $\varepsilon_C = \{A\}$; otherwise, ε_C is the empty set.

We take the liberty to adapt quotes from the chapter to our notations.

Formally, the candidate model construction is parameterized by a literal selection function S. It can be ignored by taking $S := \lambda C.\ \{\}$.

locale *ground_resolution_with_selection* $=$
  fixes S :: $'a\ clause \Rightarrow\ 'a\ clause$
  assumes $S\ C \subseteq C$ **and** $L \in S\ C \Longrightarrow \mathsf{is\_neg}\ L$

Inside the locale, we fixed a clause set N, for which we want to derive a model. Then we defined two operators corresponding to ε_C and I_C:

function production :: $'a\ clause \Rightarrow\ 'a\ set$ **where**
  $\mathsf{production}\ C = \{A \mid C \in N \wedge C \neq \{\} \wedge \mathsf{Max}\ C = \mathsf{Pos}\ A$
    $\wedge \left(\bigcup_{D<C} \mathsf{production}\ D\right) \not\models C \wedge S\ C = \{\}\}$
definition interp :: $'a\ clause \Rightarrow\ 'a\ set$ **where**
  $\mathsf{interp}\ C = \bigcup_{D<C} \mathsf{production}\ D$

To ensure monotonicity of the construction, any produced atom must be maximal in its clause. Moreover, clauses that produce an atom, called *productive* clauses, may not contain selected literals. In the chapter, ε_C and I_C are expressed in terms of each other. We simplified the definition by inlining I_C in ε_C, so that only ε_C is recursive. Since the recursive calls operate on clauses D that are smaller with respect to a well-founded order, the definition is accepted [22]. Once the operators were defined, we could inline interp's definition in production's equation to derive the intended mutually recursive specification as a lemma. The I^C and I_N operators are defined as abbreviations:

$$\mathsf{Interp}\ C = \mathsf{interp}\ C \cup \mathsf{production}\ C \qquad \mathsf{INTERP} = \bigcup_{C \in N} \mathsf{production}\ C$$

We then proved a host of lemmas about these concepts. Bachmair and Ganzinger's Lemma 3.4 states the following:

> Let C and D be clauses such that $D \geq C$. If C is true in I_D or I^D then C is also true in I_N and in all interpretations $I_{D'}$ and $I^{D'}$, where $D' \geq D$.

This amounts to six monotonicity properties, including

lemma *Interp_imp_interp*: $C \leq D \Longrightarrow D < D' \Longrightarrow \mathsf{Interp}\ D \models C \Longrightarrow \mathsf{interp}\ D' \models C$
lemma *Interp_imp_Interp*: $C \leq D \Longrightarrow D \leq D' \Longrightarrow \mathsf{Interp}\ D \models C \Longrightarrow \mathsf{Interp}\ D' \models C$
lemma *Interp_imp_INTERP*: $C \leq D \Longrightarrow \mathsf{Interp}\ D \models C \Longrightarrow \mathsf{INTERP} \models C$

In the chapter, the first property is wrongly stated with $D \leq D'$ instead of $D < D'$, admitting the counterexample $N = \{\{A\}\}$ and $C = D = D' = \{A\}$.

Lemma 3.3, whose proof depends on a monotonicity property, is better proved *after* Lemma 3.4:

lemma *productive_imp_INTERP*: $\mathsf{production}\ C \neq \{\} \Longrightarrow \mathsf{INTERP} \models C$

A more serious oddity is Lemma 3.7. Using our notations, we can state it as

$$D \in N \Longrightarrow C \neq D \Longrightarrow \big(\forall D' \leq D.\ \mathsf{Interp}\ D' \models C\big) \Longrightarrow \mathsf{interp}\ D \models D'$$

However, the last occurrence of D' is clearly wrong—the context suggests C instead. Even after this amendment, we have a counterexample, corresponding to a gap in the proof: $D = \{\}$, $C = \{\mathsf{Pos}\ A\}$, and $N = \{D, C\}$. Since this "lemma" is not used, we simply ignored it.

4.2 Unordered Resolution

The unordered ground resolution calculus consists of a single binary inference rule, with the side premise $C \vee A \vee \cdots \vee A$, the main premise $\neg\, A \vee D$, and the conclusion $C \vee D$:

$$\frac{C \vee A \vee \cdots \vee A \qquad \neg\, A \vee D}{C \vee D}$$

Formally, this rule is captured by a predicate:

inductive $\mathsf{unord\_resolve} :: {'a\ clause} \Rightarrow {'a\ clause} \Rightarrow {'a\ clause} \Rightarrow bool$ **where**
  $\mathsf{unord\_resolve}\ (C \uplus \mathsf{replicate}\ (n + 1)\ (\mathsf{Pos}\ A))\ (\{\mathsf{Neg}\ A\} \uplus D)\ (C \uplus D)$

Soundness was trivial to prove:

lemma *unord_resolve_sound*: $\mathsf{unord\_resolve}\ C\ D\ E \Longrightarrow I \models C \Longrightarrow I \models D \Longrightarrow I \models E$
  using *unord_resolve.cases* **by** *fastforce*

To prove completeness, it sufficed to show that the calculus reduces counterexamples. This corresponds to Bachmair and Ganzinger's Theorem 3.8:

> Let N be a set of clauses not containing the empty clause and C be a minimal counterexample in N for I_N. Then there exists an inference by binary resolution with factoring from C such that
>
> (i) its conclusion is a counterexample for I_N and is smaller than C; and
> (ii) C is its main premise and the side premise is a productive clause.

In our formalization, the conclusion is strengthened slightly to match the locale *counterex_reducing_inference_system*'s assumption:

theorem *unord_resolve_counterex_reducing*:
assumes $\{\} \notin N$ **and** $C \in N$ **and** INTERP $N \not\models C$ **and**
$\forall D \in N.$ INTERP $N \not\models D \Longrightarrow C \leq D$
obtains $D\ E$ **where**
$D \in N$ **and** INTERP $N \models D$ **and** production $N\ D \neq \{\}$ **and**
unord_resolve $D\ C\ E$ **and** INTERP $N \not\models E$ **and** $E < C$

The arguments N to INTERP and production are necessary because the theorem is located outside the block in which N was fixed. This explicit dependency makes it possible to instantiate the locale's I_of :: *'a clause set ⇒ 'a set* parameter with INTERP.

By instantiating the *sound_inference_system* and *counterex_reducing_inference_system* locales, we obtained refutational completeness (Theorem 3.9 and Corollary 3.10) and compactness of clausal logic (Theorem 3.12).

4.3 Ordered Resolution with Selection

Ordered ground resolution consists of a single rule, ord_resolve. Like unord_resolve, it is sound and counterexample-reducing (Theorem 3.15). Moreover, it is reductive (Lemma 3.13): The conclusion is always smaller than the main premise. The rule is given in the chapter's Figure 2 as

$$\frac{C_1 \vee A_1 \vee \cdots \vee A_1 \qquad \cdots \qquad C_n \vee A_n \vee \cdots \vee A_n \qquad \neg\, A_1 \vee \cdots \vee \neg\, A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

where

(i) either the subclause $\neg A_1 \vee \cdots \vee \neg A_n$, is selected by S in D, or else $S(D)$ is empty, $n = 1$, and A_1 is maximal with respect to D,
(ii) each atom A_i is strictly maximal with respect to C_i, and
(iii) no clause $C_i \vee A_i \vee \cdots \vee A_i$ contains a selected atom.

The side conditions help prune the search space and make the rule reductive.

The rule's $(n + 1)$-ary nature constitutes a substantial complication. The ellipsis notation hides most of the complexity in the informal proof, but in Isabelle, even stating the rule is tricky, let alone reasoning about it. We represented the n side premises by three parallel lists of length n: *CAs* gives the entire clauses, whereas *Cs* and $\mathcal{A}s$ store the C_i and the $\mathcal{A}_i = A_i \vee \cdots \vee A_i$ parts separately. In addition, *As* is the list $[A_1, \ldots, A_n]$. The following inductive definition captures the rule formally:

inductive
ord_resolve ::
'a clause list ⇒ 'a clause ⇒ 'a multiset list ⇒ 'a list ⇒ 'a clause ⇒ bool
where
$|CAs| = n \Longrightarrow |Cs| = n \Longrightarrow |\mathcal{A}s| = n \Longrightarrow |As| = n \Longrightarrow n \neq 0 \Longrightarrow$
$(\forall i < n.\ CAs\,!\,i = Cs\,!\,i \uplus \text{Pos}\ `\ \mathcal{A}s\,!\,i) \Longrightarrow (\forall i < n.\ \mathcal{A}s\,!\,i \neq \{\}) \Longrightarrow$
$(\forall i < n.\ \forall A \in \mathcal{A}s\,!\,i.\ A = As\,!\,i) \Longrightarrow \text{eligible}\ As\ (D \uplus \text{Neg}\ `\ \text{mset}\ As) \Longrightarrow$
$(\forall i < n.\ \text{strict\_max\_in}\ (As\,!\,i)\ (Cs\,!\,i)) \Longrightarrow (\forall i < n.\ S\ (CAs\,!\,i) = \{\}) \Longrightarrow$
$\text{ord\_resolve}\ CAs\ (D \uplus \text{Neg}\ `\ \text{mset}\ As)\ \mathcal{A}s\ As\ ((\bigcup \text{mset}\ Cs) \uplus D)$

The $xs\,!\,i$ operator returns the $(i + 1)$st element of xs, and mset converts a list to a multiset. Before settling on the above formulation, we tried storing the n premises in a multiset, since their order is irrelevant. However, due to the permutative nature of multisets, there can be no such things as “parallel multisets”; to keep the dependencies between the clauses C_i and the atoms A_i, we must keep them in a single multiset of tuples, which is very unwieldy.

An early version of the formalization represented each $A_i \vee \cdots \vee A_i$ as a value of type $'a \times nat$—the nat representing the number of times A_i is repeated. With this approach, the definition of $\mathsf{ord\_resolve}$ did not need to state the equality of the atoms in each $As\ !\ i$. On the other hand, the approach does not work on the first-order level, where atoms should be unifiable instead of equal.

Formalization revealed an error and a few ambiguities in the rule's statement. References to $S(D)$ in the side conditions should have been to $S(\neg A_1 \vee \cdots \vee \neg A_n \vee D)$. In our formalization, this is captured by the $\mathsf{eligible}\ As\ (D \uplus \mathsf{Neg}\ `\ \mathsf{mset}\ As)$ premise that corresponds to (i) from the original rule, where $\mathsf{eligible}$ is defined by

$$\begin{aligned}&\mathsf{eligible}\ As\ DA \Leftrightarrow\\ &S\ DA = \mathsf{Neg}\ `\ \mathsf{mset}\ As \vee (S\ DA = \{\} \wedge |As| = 1 \wedge As\ !\ 0 = \mathsf{Max}\ (\mathsf{atms\_of}\ DA))\end{aligned}$$

The ambiguities are discussed in Appendix A.

Soundness is a good sanity check for our definition:

lemma *ord_resolve_sound*:
$\mathsf{ord\_resolve}\ CAs\ DA\ \mathcal{A}s\ As\ E \Longrightarrow I \models \mathsf{mset}\ CAs \Longrightarrow I \models DA \Longrightarrow I \models E$

The proof is by case distinction: Either the interpretation I contains all atoms A_i, in which case the D subclause of the main premise $\neg A_1 \vee \cdots \vee \neg A_n \vee D$ must be true, or there exists an index i such that $A_i \notin I$, in which case the corresponding C_i must be true. In both cases, the conclusion $C_1 \vee \cdots \vee C_n \vee D$ is true.

5 Theorem Proving Processes

In their Sect. 4, Bachmair and Ganzinger switch from a static to a dynamic view of saturation: from clause sets closed under inferences to theorem proving processes that start with a clause set N_0 and keep deriving new clauses until $\bot$ is generated or no inferences are possible. Proving processes support an important optimization: Redundant clauses can be deleted at any point from the clause set, and redundant inferences need not be performed at all.

A derivation performed by a proving process is a possibly infinite sequence $N_0 \rhd N_1 \rhd N_2 \rhd \cdots$, where $\rhd$ relates clause sets (`Proving_Process.thy`). In Isabelle, such sequences are captured by lazy lists, a codatatype [8] generated by $\mathsf{LNil} :: {'a}\ llist$ and $\mathsf{LCons} :: {'a} \Rightarrow {'a}\ llist \Rightarrow {'a}\ llist$, and equipped with lhd (“head”) and ltl (“tail”) selectors that extract LCons's arguments. Unlike datatypes, codatatypes allow infinite values—e.g., $\mathsf{LCons}\ 0\ (\mathsf{LCons}\ 1\ (\mathsf{LCons}\ 2\ \ldots))$. The coinductive predicate chain checks that its argument is a nonempty lazy list whose pairs of consecutive elements are related by a given binary predicate R:

coinductive $\mathsf{chain} :: ({'a} \Rightarrow {'a} \Rightarrow bool) \Rightarrow {'a}\ llist \Rightarrow bool$ **where**
$\mathsf{chain}\ R\ (\mathsf{LCons}\ x\ \mathsf{LNil})$
$|\ \mathsf{chain}\ R\ xs \Longrightarrow R\ x\ (\mathsf{lhd}\ xs) \Longrightarrow \mathsf{chain}\ R\ (\mathsf{LCons}\ x\ xs)$

A derivation is a lazy list Ns of clause sets satisfying the chain predicate with $R = \rhd$. Derivations depend on a redundancy criterion presented as two functions, $\mathcal{R_F}$ and $\mathcal{R_I}$, that specify redundant clauses and redundant inferences, respectively:

locale *redundancy_criterion* $=$ *inference_system* $+$
fixes
$\mathcal{R}_{\mathcal{F}}$ :: *'a clause set* $\Rightarrow$ *'a clause set* **and**
$\mathcal{R}_{\mathcal{I}}$ :: *'a clause set* $\Rightarrow$ *'a inference set*
assumes
$\mathcal{R}_{\mathcal{I}}\ N \subseteq \Gamma$ **and**
$N \subseteq N' \Longrightarrow \mathcal{R}_{\mathcal{F}}\ N \subseteq \mathcal{R}_{\mathcal{F}}\ N'$ **and**
$N \subseteq N' \Longrightarrow \mathcal{R}_{\mathcal{I}}\ N \subseteq \mathcal{R}_{\mathcal{I}}\ N'$ **and**
$N' \subseteq \mathcal{R}_{\mathcal{F}}\ N \Longrightarrow \mathcal{R}_{\mathcal{F}}\ N \subseteq \mathcal{R}_{\mathcal{F}}\ (N \setminus N')$ **and**
$N' \subseteq \mathcal{R}_{\mathcal{F}}\ N \Longrightarrow \mathcal{R}_{\mathcal{I}}\ N \subseteq \mathcal{R}_{\mathcal{I}}\ (N \setminus N')$ **and**
$\mathsf{sat}\ (N \setminus \mathcal{R}_{\mathcal{F}}\ N) \Longrightarrow \mathsf{sat}\ N$

By definition, a transition from M to N is possible if the only new clauses added are conclusions of inferences from M and any deleted clauses would be redundant in N:

inductive $\rhd$ :: *'a clause set* $\Rightarrow$ *'a clause set* $\Rightarrow$ *bool* **where**
$N \setminus M \subseteq \mathsf{concl\_of}\ `\ \mathsf{infers\_from}\ M \Longrightarrow M \setminus N \subseteq \mathcal{R}_{\mathcal{F}}\ N \Longrightarrow M \rhd N$

This rule combines deduction (the addition of inferred clauses) and deletion (the removal of redundant clauses). The chapter distinguishes the two operations:

Deduction:	$N \rhd N, M$	if $M \subseteq \mathsf{concl\_of}\ `\ \mathsf{infers\_from}\ N$
Deletion:	$N, M \rhd N$	if $M \subseteq \mathcal{R}_{\mathcal{F}}\ N$

This is problematic, because it is sometimes necessary to perform both deduction and deletion in a single transition, as we will see in Sect. 7.

A key concept to connect static and dynamic completeness is that of the set of persistent clauses, or limit of a sequence of clause sets: $N_\infty = \bigcup_i \bigcap_{j \geq i} N_j$. These are the clauses that belong to all clause sets except for at most a finite prefix of the sequence N_i. We also needed the supremum of a sequence, $\bigcup_i N_i$, and of a bounded prefix, $\bigcup_{i=0}^{j} N_i$. We introduced these functions (`Lazy_List_Liminf.thy`):

definition Liminf :: *'a llist* $\Rightarrow$ *'a* **where**
$\mathsf{Liminf}\ xs = \bigcup_{i < |xs|} \bigcap_{j : i \leq j < |xs|} xs\ !\ j$
definition Sup :: *'a llist* $\Rightarrow$ *'a* **where**
$\mathsf{Sup}\ xs = \bigcup_{i < |xs|} xs\ !\ i$
definition Sup_upto :: *'a llist* $\Rightarrow$ *nat* $\Rightarrow$ *'a* **where**
$\mathsf{Sup\_upto}\ xs\ j = \bigcup_{i : i < |xs| \wedge i \leq j} xs\ !\ i$

Although codatatypes open the door to coinductive methods, we followed the chapter's index-based approach whenever possible. When interpreting the expression $\bigcup_i \bigcap_{j \geq i} N_j$ for the case of a finite sequence of length n, it is crucial to use the right upper bounds, namely $i, j < n$. For j, it is clear that '$< n$' is needed to keep N_j's index within bounds. For i, the danger is more subtle: If $i \geq n$, then $\bigcap_{j : i \leq j < n} N_j$ collapses to the trivial infimum $\bigcap_{j \in \{\}} N_j$, i.e., the set of all clauses.

Lemma 4.2 connects redundant clauses and inferences at the limit to those of the supremum, and the satisfiability of the limit to that of the initial clause set. Formally:

lemma *Rf_limit_Sup*: $\mathsf{chain}\ (\rhd)\ Ns \Longrightarrow \mathcal{R}_{\mathcal{F}}\ (\mathsf{Liminf}\ Ns) = \mathcal{R}_{\mathcal{F}}\ (\mathsf{Sup}\ Ns)$
lemma *Ri_limit_Sup*: $\mathsf{chain}\ (\rhd)\ Ns \Longrightarrow \mathcal{R}_{\mathcal{I}}\ (\mathsf{Liminf}\ Ns) = \mathcal{R}_{\mathcal{I}}\ (\mathsf{Sup}\ Ns)$
lemma *sat_limit_iff*: $\mathsf{chain}\ (\rhd)\ Ns \Longrightarrow \big(\mathsf{sat}\ (\mathsf{Liminf}\ Ns) \Longleftrightarrow \mathsf{sat}\ (\mathsf{lhd}\ Ns)\big)$

The proof of the last lemma relies on

lemma *deriv_sat_preserving*: chain ($\rhd$) $Ns \Longrightarrow$ sat (lhd Ns) $\Longrightarrow$ sat (Sup Ns)

In the chapter, this property follows “by the soundness of the inference system Γ and the compactness of clausal logic,” contradicting the claim that “we will only consider consistency-preserving inference systems” [4, Sect. 2.4] and not sound ones. Thanks to Isabelle, we now know that soundness is unnecessary. By compactness, it suffices to show that all finite subsets $\mathcal{D}$ of $\bigcup_i N_i$ are satisfiable. By finiteness of $\mathcal{D}$, there must exist an index k such that $\mathcal{D} \subseteq \bigcup_{i=0}^{k} N_i$. We perform an induction on k. The base case is trivial. For the induction step, if k is beyond the end of the list, then $\bigcup_{i=0}^{k} N_i = \bigcup_{i=0}^{k-1} N_i$ and we can apply the induction hypothesis directly. Otherwise, we have that the set Sup_upto Ns $(k - 1) \cup$ concl_of ‘ infers_from (Sup_upto Ns $(k - 1)$) is satisfiable by the induction hypothesis and satisfiability preservation of Γ inferences. Hence, Sup_upto Ns $(k - 1) \cup Ns$! k, i.e., Sup_upto Ns k, is satisfiable, as desired.

Next, we formally showed that the limit is saturated, under some assumptions and for a relaxed notion of saturation. A clause set N is saturated up to redundancy if all inferences from nonredundant clauses in N are redundant:

definition saturated_upto :: *'a clause set* $\Rightarrow$ *bool* **where**
  saturated_upto $N \iff$ infers_from ($N \setminus \mathcal{R}_{\mathcal{F}}\ N$) $\subseteq \mathcal{R}_{\mathcal{I}}\ N$

The limit is saturated for fair derivations—derivations in which no inferences from nonredundant persisting clauses are delayed indefinitely:

definition fair_clss_seq :: *'a clause set llist* $\Rightarrow$ *bool* **where**
  fair_clss_seq $Ns \iff$ let $N' =$ Liminf $Ns \setminus \mathcal{R}_{\mathcal{F}}$ (Liminf Ns) in
    concl_of ‘ infers_from $N' \setminus \mathcal{R}_{\mathcal{I}}\ N' \subseteq$ Sup $Ns \cup \mathcal{R}_{\mathcal{F}}$ (Sup Ns)

The criterion must also be effective, which is expressed by a locale:

locale *effective_redundancy_criterion* $=$ *redundancy_criterion* $+$
  assumes $\gamma \in \Gamma \Longrightarrow$ concl_of $\gamma \in N \cup \mathcal{R}_{\mathcal{F}}\ N \Longrightarrow \gamma \in \mathcal{R}_{\mathcal{I}}\ N$

In a locale that combines *sat_preserving_inference_system* and *effective_redundancy_criterion*, we have Theorem 4.3:

theorem *fair_derive_saturated_upto*:
  chain ($\rhd$) $Ns \Longrightarrow$ fair_clss_seq $Ns \Longrightarrow$ saturated_upto (Liminf Ns)

It is easy to show that the trivial criterion defined by $\mathcal{R}_{\mathcal{F}}\ N = \{\}$ and $\mathcal{R}_{\mathcal{I}}\ N = \{\gamma \in \Gamma \mid$ concl_of $\gamma \in N\}$ satisfies the requirements on *effective_redundancy_criterion*. A more useful instance is the standard redundancy criterion, which depends on a counterexample-reducing inference system Γ (`Standard_Redundancy.thy`):

definition $\mathcal{R}_{\mathcal{F}}$ :: *'a clause set* $\Rightarrow$ *'a clause set* **where**
  $\mathcal{R}_{\mathcal{F}}\ N = \{C \mid \exists \mathcal{D} \subseteq N.\ (\forall I.\ I \vDash \mathcal{D} \Longrightarrow I \vDash C) \land (\forall D \in \mathcal{D}.\ D < C)\}$
definition $\mathcal{R}_{\mathcal{I}}$ :: *'a clause set* $\Rightarrow$ *'a inference set* **where**
  $\mathcal{R}_{\mathcal{I}}\ N = \{\gamma \in \Gamma \mid \exists \mathcal{D} \subseteq N.\ (\forall I.\ I \vDash \mathcal{D} \uplus$ side_prems_of $\gamma \Longrightarrow I \vDash$ concl_of $\gamma) \land$
    $(\forall D \in \mathcal{D}.\ D <$ main_prem_of $\gamma)\}$

Standard redundancy qualifies as an *effective_redundancy_criterion*. In the chapter, this is stated as Theorems 4.7 and 4.8, which depend on two auxiliary properties, Lemmas 4.5 and 4.6. The main result, Theorem 4.9, is that counterexample-reducing calculi are refutationally complete under the application of standard redundancy:

theorem *saturated_upto_complete*: saturated_upto $N \Longrightarrow (\neg$ sat $N \iff \{\} \in N)$

The informal proof of Lemma 4.6 applies Lemma 4.5 in a seemingly impossible way, confusing redundant clauses and redundant inferences and exploiting properties that appear only in the proof of Lemma 4.5. Our solution is to generalize the core argument into the following lemma and apply it to prove both Lemmas 4.5 and 4.6:

lemma *wlog_non_Rf*:
$(\exists \mathcal{D} \subseteq N.\ (\forall I.\ I \models \mathcal{D} \uplus \mathcal{C} \Longrightarrow I \models E) \wedge (\forall D' \in \mathcal{D}.\ D' < D)) \Longrightarrow$
$\exists \mathcal{D} \subseteq N \setminus \mathcal{R}_{\mathcal{F}}\ N.\ (\forall I.\ I \models \mathcal{D} \uplus \mathcal{C} \Longrightarrow I \models E) \wedge (\forall D' \in \mathcal{D}.\ D' < D)$

Incidentally, the informal proof of Theorem 4.9 also needlessly invokes Lemma 4.5.

Finally, given a redundancy criterion $(\mathcal{R}_{\mathcal{F}}, \mathcal{R}_{\mathcal{I}})$ for Γ, its standard extension for $\Gamma' \supseteq \Gamma$ is $(\mathcal{R}_{\mathcal{F}}, \mathcal{R}'_{\mathcal{I}})$, where $\mathcal{R}'_{\mathcal{I}}\ N = \mathcal{R}_{\mathcal{I}}\ N \cup (\Gamma' \setminus \Gamma)$ (`Proving_Process.thy`). The standard extension is itself a redundancy criterion and it preserves effectiveness, saturation up to redundancy, and fairness. In Isabelle, this can be expressed outside the locale blocks by using the locale predicates—explicit predicates named after the locales and parameterized by the locale arguments:

lemma *standard_redundancy_criterion_extension*:
$\Gamma \subseteq \Gamma' \Longrightarrow$ redundancy_criterion $\Gamma\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}_{\mathcal{I}} \Longrightarrow$ redundancy_criterion $\Gamma'\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}'_{\mathcal{I}}$
lemma *standard_redundancy_criterion_extension_effective*:
$\Gamma \subseteq \Gamma' \Longrightarrow$ effective_redundancy_criterion $\Gamma\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}_{\mathcal{I}} \Longrightarrow$
effective_redundancy_criterion $\Gamma'\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}'_{\mathcal{I}}$
lemma *standard_redundancy_criterion_extension_saturated_upto_iff*:
$\Gamma \subseteq \Gamma' \Longrightarrow$ redundancy_criterion $\Gamma\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}_{\mathcal{I}} \Longrightarrow$
(redundancy_criterion.saturated_upto $\Gamma\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}_{\mathcal{I}}\ N \Longleftrightarrow$
redundancy_criterion.saturated_upto $\Gamma'\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}'_{\mathcal{I}}\ N$)
lemma *standard_redundancy_criterion_extension_fair_iff*:
$\Gamma \subseteq \Gamma' \Longrightarrow$ effective_redundancy_criterion $\Gamma\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}_{\mathcal{I}} \Longrightarrow$
(effective_redundancy_criterion.fair_clss_seq $\Gamma'\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}'_{\mathcal{I}}\ Ns \Longleftrightarrow$
effective_redundancy_criterion.fair_clss_seq $\Gamma\ \mathcal{R}_{\mathcal{F}}\ \mathcal{R}_{\mathcal{I}}\ Ns$)

6 First-Order Resolution

The chapter's Sect. 4.3 presents a first-order version of the ordered resolution rule and a first-order prover, RP, based on that rule. The first step towards lifting the completeness of ground resolution is to show that we can lift individual ground resolution inferences (`FO_Ordered_Resolution.thy`).

6.1 Inference Rule

First-order ordered resolution consists of a single rule. In the chapter, ground and first-order resolution are both called $O_S^{\succ}$. In the formalization, we also let the rules share the same name, but since they exist in separate locales, the system generates qualified names that make this unambiguous: Isabelle generates the name ground_resolution_with_selection.ord_resolve, which refers to ground resolution, and FO_resolution.ordered_resolve, which refers to first-order resolution. If the user is in doubt at any time, the system can always tell which one is meant.

The rule is given in the chapter's Figure 4 as follows:

$$\frac{C_1 \vee A_{11} \vee \cdots \vee A_{1k_1} \qquad \cdots \qquad C_n \vee A_{1n} \vee \cdots \vee A_{nk_n} \qquad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \cdot \sigma \vee \cdots \vee C_n \cdot \sigma \vee D \cdot \sigma}$$

where σ is a most general simultaneous solution of all unification problems $A_{i1} = \cdots = A_{ik_i} = A_i$, where $1 \leq i \leq n$, and

(i) either $A_1, \ldots, A_n$ are selected in D, or else nothing is selected in D, $n = 1$, and $A_1 \cdot \sigma$ is maximal in $D \cdot \sigma$,
(ii) each atom $A_{ij} \cdot \sigma$ is strictly maximal with respect to $C_i \cdot \sigma$, and
(iii) no clause $C_i \vee A_{i1} \vee \cdots \vee A_{ik_i}$ contains a selected atom.

The Isabelle representation of this rule is similar to that of its ground counterpart, generalized to apply σ. We corrected a few typos listed in Appendix A.

inductive
  ord_resolve ::
  'a clause list ⇒ 'a clause ⇒ 'a multiset list ⇒ 'a list ⇒ 's ⇒ 'a clause ⇒ bool
where
  $|CAs| = n \Longrightarrow |Cs| = n \Longrightarrow |\mathcal{A}s| = n \Longrightarrow |As| = n \Longrightarrow n \neq 0 \Longrightarrow$
  $(\forall i < n.\ CAs\ !\ i = Cs\ !\ i \uplus \mathsf{Pos}\ `\ \mathcal{A}s\ !\ i) \Longrightarrow (\forall i < n.\ \mathcal{A}s\ !\ i \neq \{\}) \Longrightarrow$
  $\mathsf{Some}\ \sigma = \mathsf{mgu}\ (\mathsf{set\_mset}\ `\ \mathsf{set}\ (\mathsf{map2}\ \mathsf{add\_mset}\ As\ \mathcal{A}s)) \Longrightarrow$
  $\mathsf{eligible}\ \sigma\ As\ (D \uplus \mathsf{Neg}\ `\ \mathsf{mset}\ As) \Longrightarrow$
  $(\forall i < n.\ \mathsf{strict\_max\_in}\ (As\ !\ i \cdot \sigma)\ (Cs\ !\ i \cdot \sigma)) \Longrightarrow (\forall i < n.\ S\ (CAs\ !\ i) = \{\}) \Longrightarrow$
  $\mathsf{ord\_resolve}\ CAs\ (D \uplus \mathsf{Neg}\ `\ \mathsf{mset}\ As)\ \mathcal{A}s\ As\ \sigma\ (((\bigcup \mathsf{mset}\ Cs) \uplus D) \cdot \sigma)$

Our MGU σ is uniquely determined by the unification problems $A_{i1} = \cdots = A_{ik_i} = A_i$, which ensures that each concrete set of premises gives rise to exactly one conclusion.

The rule as stated is incomplete; for example, the clauses $\mathsf{p}(x)$ and $\neg\, \mathsf{p}(\mathsf{f}(x))$ cannot be resolved because x and $\mathsf{f}(x)$ are not unifiable. Such issues arise when the same variable names appear in different premises. In the chapter, the authors circumvent this issue by stating, "We also implicitly assume that different premises and the conclusion have no variables in common; variables are renamed if necessary." For the formalization, we first considered enforcing the invariant that all derived clauses use mutually disjoint variables, but this does not help when a clause is repeated in an inference's premises. An example is the inference

$$\frac{\mathsf{p}(x) \qquad \mathsf{p}(y) \qquad \neg\, \mathsf{p}(\mathsf{a}) \vee \neg\, \mathsf{p}(\mathsf{b})}{\bot}$$

where $\mathsf{p}(x)$ and $\mathsf{p}(y)$ are the same clause up to renaming. Instead, we rely on a predicate ord_resolve_rename, based on ord_resolve, that standardizes the premises apart. The renaming is performed by a function called renamings_apart :: *'a clause list ⇒ 's list* that, given a list of clauses, returns a list of corresponding substitutions to apply. This function is part of our abstract interface for terms and substitutions (Sect. 2) and is implemented using IsaFoR.

As in the ground case, it is important to establish soundness. We formally proved that any ground instance of the rule ord_resolve is sound:

lemma *ord_resolve_ground_inst_sound*:
  $\mathsf{ord\_resolve}\ CAs\ DA\ \mathcal{A}s\ As\ \sigma\ E \Longrightarrow I \models \mathsf{mset}\ CAs \cdot \sigma \cdot \eta \Longrightarrow I \models DA \cdot \sigma \cdot \eta \Longrightarrow$
  $\mathsf{is\_ground\_subst}\ \eta \Longrightarrow I \models E \cdot \eta$

Similarly, ground instances of $\mathsf{ord\_resolve\_rename}$ are sound. It then follows that the rules $\mathsf{ord\_resolve}$ and $\mathsf{ord\_resolve\_rename}$ are sound:

lemma *ord_resolve_rename_sound*:
$\mathsf{ord\_resolve\_rename}\ \mathit{CAs}\ \mathit{DA}\ \mathcal{A}s\ \mathit{As}\ \sigma\ E \Longrightarrow$
$(\forall\sigma.\ \mathsf{is\_ground\_subst}\ \sigma \Longrightarrow I \models (\mathsf{mset}\ \mathit{CAs} + \{\mathit{DA}\}) \cdot \sigma) \Longrightarrow$
$\mathsf{is\_ground\_subst}\ \eta \Longrightarrow I \models E \cdot \eta$

6.2 Lifting Lemma

To lift ground inferences to the first-order level, we consider a set of clauses M and introduce an adjusted version S_M of the selection function S.

definition S_M :: *'a literal multiset* $\Rightarrow$ *'a literal multiset* **where**
$S_M\ C =$
$(\text{if } C \in \mathsf{grounding\_of}\ M \text{ then}$
$\quad(\text{SOME } C'.\ \exists D \in M.\ \exists\sigma.\ C = D \cdot \sigma \wedge C' = S\ D \cdot \sigma \wedge \mathsf{is\_ground\_subst}\ \sigma)$
else
$\quad S\ C)$

Here, SOME is Hilbert's choice operator, which picks an arbitrary element satisfying the condition if one exists, and a completely arbitrary element otherwise. For the above definition, we could prove that an element satisfying the condition always exists. The new selection function depends on both S and M and works in such a way that any ground instance inherits the selection of at least one of the nonground clauses of which it is an instance:

lemma *S_M_grounding_of_clss*:
$C \in \mathsf{grounding\_of}\ M \Longrightarrow$
$\exists D \in M.\ \exists\sigma.\ C = D \cdot \sigma \wedge S_M\ C = S\ D \cdot \sigma \wedge \mathsf{is\_ground\_subst}\ \sigma$

where $\mathsf{grounding\_of}\ M$ is the set of ground instances of a set of clauses M.

The lifting lemma, Lemma 4.12, states that whenever there exists a ground inference from clauses belonging to $\mathsf{grounding\_of}\ M$, there exists a (possibly) more general inference from clauses belonging to M:

> Let M be a set of clauses and $K = \mathsf{grounding\_of}\ M$. If
>
> $$\frac{C_1 \qquad \cdots \qquad C_n \qquad C_0}{C}$$
>
> is an inference in $O^{\succ}_{S_M}(K)$ then there exist clauses C'_i in M, a clause C', and a ground substitution σ such that
>
> $$\frac{C'_1 \qquad \cdots \qquad C'_n \qquad C'_0}{C'}$$
>
> is an inference in $O^{\succ}_{S}(M)$, $C_i = C'_i \cdot \sigma$, and $C = C' \cdot \sigma$.

In the formalization, the side premises are stored in a list *CAs*, the main premise is called *DA*, and the conclusion is called E.

lemma *ord_resolve_rename_lifting*:
$(\forall\rho\ C.\ \mathsf{is\_renaming}\ \rho \Longrightarrow S\ (C \cdot \rho) = S\ C \cdot \rho) \Longrightarrow$

$$\begin{array}{l}
\mathsf{ord\_resolve}\ S_M\ CAs\ DA\ \mathcal{A}s\ As\ \sigma\ E \Longrightarrow \\
\{DA\} \cup \mathsf{set}\ CAs \subseteq \mathsf{grounding\_of}\ M \Longrightarrow \\
\exists \eta s\ \eta\ \theta\ CAs_0\ DA_0\ \mathcal{A}s_0\ As_0\ E_0\ \tau. \\
\quad \mathsf{ord\_resolve\_rename}\ S\ CAs_0\ DA_0\ \mathcal{A}s_0\ As_0\ \tau\ E_0 \wedge \\
\quad CAs_0 \cdot \eta s = CAs \wedge DA_0 \cdot \eta = DA \wedge E_0 \cdot \theta = E \wedge \{DA_0\} \cup \mathsf{set}\ CAs_0 \subseteq M
\end{array}$$

The informal proof of this lemma consists of two sentences spanning four lines of text. In Isabelle, these two sentences translate to 250 lines and 400 lines, respectively, excluding auxiliary lemmas. Our proof involves six steps:

1. Obtain a list of first-order clauses CAs_0 and a first-order clause DA_0 that belong to M and that generalize CAs and DA with substitutions ηs and η, respectively.
2. Choose atoms $\mathcal{A}s_0$ and As_0 in the first-order clauses on which to resolve.
3. Standardize CAs_0 and DA_0 apart, yielding CAs_0' and DA_0'.
4. Obtain the MGU τ of the literals on which to resolve.
5. Show that ordered resolution on CAs_0' and DA_0' with τ as MGU is applicable.
6. Show that the resulting resolvent E_0 generalizes E with substitution θ.

In step 1, suitable clauses must be chosen so that $S\ (CAs_0\,!\,i)$ generalizes $S_M\ (CAs\,!\,i)$, for $0 \le i < n$, and $S\ DA_0$ generalizes $S_M\ DA$. By the definition of S_M, this is always possible. In step 2, we choose the literals to resolve upon in the first-order inference depending on the selection on the ground inference. If some literals are selected in DA, we let As_0 be the selected literals in DA_0, such that $(As_0\ !\ i) \cdot \eta = As\ !\ i$ for each i. Otherwise, As must be a singleton list containing some atom A, and we let As_0 be the singleton list consisting of an arbitrary $A_0 \in DA_0$ such that $A_0 \cdot \eta = A$. Step 3 may seem straightforward until one realizes that renaming variables can in principle influence selection. To rule this out, our lemma assumes stability under renaming: $S\ (C \cdot \rho) = S\ C \cdot \rho$ for any renaming substitution ρ and clause C. This requirement seems natural, but it is not mentioned in the chapter, and it is easy to imagine implementations that would violate it.

The above choices allowed us to perform steps 4 to 6. In the chapter, the authors assume that the obtained CAs_0 and DA_0 are standardized apart from each other as well as their conclusion E_0. This means that they can obtain a single ground substitution that connects CAs_0, DA_0, E_0 to CAs, DA, E. By contrast, we provide separate substitutions ηs, η, θ for the different side premises, the main premise, and the conclusion.

7 A First-Order Prover

Modern resolution provers interleave inference steps with steps that delete or reduce (simplify) clauses. In their Sect. 4.3, Bachmair and Ganzinger introduce the nondeterministic abstract prover RP that works on triples of clause sets, similarly to the Otter and DISCOUNT loops [16,23]. RP's core rule, called inference computation, performs first-order ordered resolution as described above; the other rules delete or reduce clauses or move them between clause sets. We formalized RP and proved it complete assuming a fair strategy (`FO_Ordered_Resolution_Prover.thy`).

7.1 Abstract First-Order Prover

The RP prover is a relation $\leadsto$ on states of the form $(\mathcal{N}, \mathcal{P}, \mathcal{O})$, where $\mathcal{N}$ is the set of *new clauses*, $\mathcal{P}$ is the set of *processed clauses*, and $\mathcal{O}$ is the set of *old clauses*. RP's formal definition closely follows the original one:

$$
\begin{array}{l}
\textbf{inductive} \leadsto :: {'a\ state} \Rightarrow {'a\ state} \Rightarrow \mathit{bool}\ \textbf{where} \\
\quad \mathsf{Neg}\ A \in C \Longrightarrow \mathsf{Pos}\ A \in C \Longrightarrow (\mathcal{N} \cup \{C\}, \mathcal{P}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P}, \mathcal{O}) \\
\mid D \in \mathcal{P} \cup \mathcal{O} \Longrightarrow \mathsf{subsumes}\ D\ C \Longrightarrow (\mathcal{N} \cup \{C\}, \mathcal{P}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P}, \mathcal{O}) \\
\mid D \in \mathcal{N} \Longrightarrow \mathsf{strictly\_subsumes}\ D\ C \Longrightarrow (\mathcal{N}, \mathcal{P} \cup \{C\}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P}, \mathcal{O}) \\
\mid D \in \mathcal{N} \Longrightarrow \mathsf{strictly\_subsumes}\ D\ C \Longrightarrow (\mathcal{N}, \mathcal{P}, \mathcal{O} \cup \{C\}) \leadsto (\mathcal{N}, \mathcal{P}, \mathcal{O}) \\
\mid D \in \mathcal{P} \cup \mathcal{O} \Longrightarrow \mathsf{reduces}\ D\ C\ L \Longrightarrow (\mathcal{N} \cup \{C \uplus \{L\}\}, \mathcal{P}, \mathcal{O}) \leadsto (\mathcal{N} \cup \{C\}, \mathcal{P}, \mathcal{O}) \\
\mid D \in \mathcal{N} \Longrightarrow \mathsf{reduces}\ D\ C\ L \Longrightarrow (\mathcal{N}, \mathcal{P} \cup \{C \uplus \{L\}\}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P} \cup \{C\}, \mathcal{O}) \\
\mid D \in \mathcal{N} \Longrightarrow \mathsf{reduces}\ D\ C\ L \Longrightarrow (\mathcal{N}, \mathcal{P}, \mathcal{O} \cup \{C \uplus \{L\}\}) \leadsto (\mathcal{N}, \mathcal{P} \cup \{C\}, \mathcal{O}) \\
\mid (\mathcal{N} \cup \{C\}, \mathcal{P}, \mathcal{O}) \leadsto (\mathcal{N}, \mathcal{P} \cup \{C\}, \mathcal{O}) \\
\mid (\{\}, \mathcal{P} \cup \{C\}, \mathcal{O}) \leadsto (\mathsf{concl\_of}\ `\ \mathsf{infers\_between}\ \mathcal{O}\ C, \mathcal{P}, \mathcal{O} \cup \{C\})
\end{array}
$$

The rules correspond, respectively, to tautology deletion, forward subsumption, backward subsumption in $\mathcal{P}$ and $\mathcal{O}$, forward reduction, backward reduction in $\mathcal{P}$ and $\mathcal{O}$, clause processing, and inference computation.

Initially, $\mathcal{N}$ consists of the problem clauses, and the other two sets are empty. Clauses in $\mathcal{N}$ are reduced using $\mathcal{P} \cup \mathcal{O}$, or even deleted if they are tautological or subsumed by $\mathcal{P} \cup \mathcal{O}$. Conversely, $\mathcal{N}$ can be used for reducing or subsuming clauses in $\mathcal{P} \cup \mathcal{O}$. Clauses eventually move from $\mathcal{N}$ to $\mathcal{P}$, one at a time. As soon as $\mathcal{N}$ is empty, a clause from $\mathcal{P}$ is selected to move to $\mathcal{O}$. Then all possible resolution inferences between this given clause and the clauses in $\mathcal{O}$ are computed and put in $\mathcal{N}$, closing the loop. The subsumption and reduction rules depend on the following predicates:

$$
\begin{aligned}
\mathsf{subsumes}\ D\ C &\Longleftrightarrow \exists\sigma.\ D \cdot \sigma \subseteq C \\
\mathsf{strictly\_subsumes}\ D\ C &\Longleftrightarrow \mathsf{subsumes}\ D\ C \wedge \neg\ \mathsf{subsumes}\ C\ D \\
\mathsf{reduces}\ D\ C\ L &\Longleftrightarrow \exists D'\ L'\ \sigma.\ D = D' \uplus \{L'\} \wedge -L = L' \cdot \sigma \wedge D' \cdot \sigma \subseteq C
\end{aligned}
$$

The definition of the set infers_between $\mathcal{O}$ C, on which inference computation depends, is more subtle. In the chapter, the set of inferences between C and $\mathcal{O}$ consists of all inferences from $\mathcal{O} \cup \{C\}$ that have C as *exactly one* of their premises. This, however, leads to an incomplete prover, because it ignores inferences that need multiple copies of C. For example, assuming a maximal selection function (one that always returns all negative literals), the resolution inference

$$\frac{\mathsf{p} \qquad \mathsf{p} \qquad \neg\,\mathsf{p} \vee \neg\,\mathsf{p}}{\bot}$$

is possible. Yet if the clause $\neg\,\mathsf{p} \vee \neg\,\mathsf{p}$ reaches $\mathcal{O}$ earlier than p, the inference would not be performed. This counterexample requires ternary resolution, but there also exists a more complicated one for binary resolution, where both premises are the same clause. Consider the clause set containing

(1) $\mathsf{q}(\mathsf{a}, \mathsf{c}, \mathsf{b})$ (2) $\neg\,\mathsf{q}(x, y, z) \vee \mathsf{q}(y, z, x)$ (3) $\neg\,\mathsf{q}(\mathsf{b}, \mathsf{a}, \mathsf{c})$

and an order $>$ on atoms such that $\mathsf{q}(\mathsf{c}, \mathsf{b}, \mathsf{a}) > \mathsf{q}(\mathsf{b}, \mathsf{a}, \mathsf{c}) > \mathsf{q}(\mathsf{a}, \mathsf{c}, \mathsf{b})$. Inferences between (1) and (2) or between (2) and (3) are impossible due to order restrictions. The only possible inference involves two copies of (2):

$$\frac{\neg\,\mathsf{q}(x, y, z) \lor \mathsf{q}(y, z, x) \qquad \neg\,\mathsf{q}(x', y', z') \lor \mathsf{q}(y', z', x')}{\neg\,\mathsf{q}(x, y, z) \lor \mathsf{q}(z, x, y)}$$

From the conclusion, we derive $\neg\,\mathsf{q}(\mathsf{a}, \mathsf{c}, \mathsf{b})$ by (3) and $\bot$ by (1).

This incompleteness is a severe flaw, although it is probably just an oversight. Fortunately, it can easily be repaired by defining $\mathsf{infers\_between}\ \mathcal{O}\ C$ as $\{(\mathcal{C}, D, E) \in \Gamma \mid \mathcal{C} \cup \{D\} \subseteq \mathcal{O} \cup \{C\} \land C \in \mathcal{C} \cup \{D\}\}$.

7.2 Projection to Theorem Proving Process

On the first-order level, a derivation can be expressed as a lazy list $\mathcal{S}s$ of states, or as three parallel lazy lists $\mathcal{N}s$, $\mathcal{P}s$, $\mathcal{O}s$. The derivation's limit state is defined as $\mathsf{Liminf}\ \mathcal{S}s = (\mathsf{Liminf}\ \mathcal{N}s, \mathsf{Liminf}\ \mathcal{P}s, \mathsf{Liminf}\ \mathcal{O}s)$, where Liminf on the right-hand side is as in Sect. 5.

Bachmair and Ganzinger use the completeness of ground resolution to prove RP complete. The first step is to show that first-order derivations can be projected down to theorem proving processes on the ground level. This corresponds to Lemma 4.10:

> If $\mathcal{S} \rightsquigarrow \mathcal{S}'$, then $\mathsf{grounding\_of}\ \mathcal{S} \rhd^* \mathsf{grounding\_of}\ \mathcal{S}'$, with $\rhd$ based on some extension of ordered resolution with selection function S and the standard redundancy criterion $(\mathcal{R}_{\mathcal{F}}, \mathcal{R}_{\mathcal{I}})$.

This raises some questions: (1) Exactly which instance of the calculus are we extending? (2) Which calculus extension should we use? (3) How can we repair the mismatch between $\rhd^*$ in the lemma statement and $\rhd$ where the lemma is invoked?

Regarding question (1), it is not clear which selection function to use. Is the function the same S as in the definition of RP or is it arbitrary? It takes a close inspection of the proof of Lemma 4.13, where Lemma 4.10 is invoked, to find out that the selection function used there is $S_{\mathsf{Liminf}\ \mathcal{O}s}$.

Regarding question (2), the phrase "some extension" is cryptic. It suggests an existential reading, and from the context it would appear that a standard extension (Sect. 5) is meant. However, neither the lemma's proof nor the context where it is invoked supplies the desired existential witness. A further subtlety is that the witness should be independent of $\mathcal{S}$ and $\mathcal{S}'$, so that transitions can be joined to form a single theorem proving derivation. Our approach is to let $\rhd$ be the standard extension for the proof system consisting of all *sound* derivations: $\Gamma = \{(\mathcal{C}, D, E) \mid \forall I.\ I \vDash \mathcal{C} \cup \{D\} \Longrightarrow I \vDash E\}$. This also eliminates the need for Bachmair and Ganzinger's subsumption resolution rule, a special calculus rule that is, from what we understand, implicitly used in the proof of Lemma 4.10 for the subcases associated with RP's reduction rules.

As for question (3), when the lemma is invoked, it is used to join transitions together to whole theorem proving processes. This requires the transitions to be of $\rhd$, not $\rhd^*$. The need for $\rhd^*$ instead of $\rhd$ arises because one of the cases requires a combination of deduction and deletion, which Bachmair and Ganzinger model as separate transitions. By merging the two transitions (Sect. 5), we avoided the issue altogether and could use $\rhd$ in the formal counterpart of Lemma 4.10.

With these issues resolved, we could formalize Lemma 4.10. In Sect. 6, we established that ground instances of the first-order resolution rule are sound. Since our ground proof system consists of all sound rules, we could reuse that lemma in the inference computation case. We proved Lemma 4.10 for single steps and extended it to entire derivations:

lemma *RP_ground_derive*: $\mathcal{S} \leadsto \mathcal{S}' \Longrightarrow$ grounding_of $\mathcal{S} \rhd$ grounding_of $\mathcal{S}'$
lemma *RP_ground_derive_chain*:
chain $(\leadsto)$ $\mathcal{S}s \Longrightarrow$ chain $(\rhd)$ (lmap grounding_of $\mathcal{S}s$)

The lmap function applies its first argument elementwise to its second argument.

7.3 Fairness and Clause Movement

From a given initial state $(\mathcal{N}_0, \{\}, \{\})$, many derivations are possible, reflecting RP's nondeterminism. In some derivations, we could leave a crucial clause in $\mathcal{N}$ or $\mathcal{P}$ without ever reducing it or moving it to $\mathcal{O}$, and then fail to derive $\bot$ even if $\mathcal{N}_0$ is unsatisfiable. For this reason, refutational completeness is guaranteed only for fair derivations. These are defined as derivations such that Liminf $\mathcal{N}s =$ Liminf $\mathcal{P}s = \{\}$, ensuring that no clause will stay forever in $\mathcal{N}$ or $\mathcal{P}$.

Fairness is expressed by the fair_state_seq predicate, which is distinct from the fair_clss_seq predicate presented in Sect. 5. For the rest of this section, we fix a lazy list of states $\mathcal{S}s$ and its projections $\mathcal{N}s$, $\mathcal{P}s$, and $\mathcal{O}s$, such that chain $(\leadsto)$ $\mathcal{S}s$, fair_state_seq $\mathcal{S}s$, and lhd $\mathcal{O}s = \{\}$.

Thanks to fairness, any nonredundant clause C in $\mathcal{S}s$'s projection to the ground level eventually ends up in $\mathcal{O}$ and stays there. This is proved as Lemma 4.11 in the chapter. Again there are some difficulties: The vagueness concerning the selection function can be resolved as for Lemma 4.10, but there is another, deeper flaw.

Bachmair and Ganzinger's proof idea is as follows. By hypothesis, the ground clause C must be an instance of a first-order clause D in $\mathcal{N}s\,!\,j \cup \mathcal{P}s\,!\,j \cup \mathcal{O}s\,!\,j$ for some index j. If $C \in \mathcal{N}s\,!\,j$, then by nonredundancy of C, fairness of the derivation, and Lemma 4.10, there must exist a clause D' that generalizes C in $\mathcal{P}s\,!\,l \cup \mathcal{O}s\,!\,l$ for some $l > j$. By a similar argument, if D' belongs to $\mathcal{P}s\,!\,l$, it will be in $\mathcal{O}s\,!\,l'$ for some $l' > l$, and finally in all $\mathcal{O}s\,!\,k$ with $k \geq l'$. The flaw is that backward subsumption can delete D' without moving it to $\mathcal{O}$. The subsuming clause would then be a strictly more general version of D' (and of the ground clause C).

Our solution is to choose D, and consequently D', such that it is minimal, with respect to subsumption, among the clauses that generalize C in the derivation. This works because strict subsumption is well founded—which we also proved, by reduction to a well-foundedness result about the strict generalization relation on first-order terms, included in IsaFoR [21, Sect. 2]. By minimality, D' cannot be deleted by backward subsumption. This line of reasoning allows us to prove Lemma 4.11, where $\mathcal{O}$_of extracts the $\mathcal{O}$ component of a state:

lemma *fair_imp_Liminf_minus_Rf_subset_ground_Liminf_state*:
$Gs =$ lmap grounding_of $\mathcal{S}s \Longrightarrow$
Liminf $Gs - \mathcal{R}_{\mathcal{F}}$ (Liminf Gs) $\subseteq$ grounding_of ($\mathcal{O}$_of (Liminf $\mathcal{S}s$))

7.4 Soundness and Completeness

The chapter's main result is Theorem 4.13. It states that, for fair derivations, the prover is sound and complete. Soundness follows from Lemma 4.2 (*sat_deriv_Liminf_iff*) and is independent of whether the derivation is fair:

theorem *RP_sound*: $\{\} \in$ clss_of (Liminf Sts) $\Longrightarrow \neg$ sat (grounding_of (lhd Sts))

Once we had brought Lemmas 4.10, 4.11, and 4.12 into a suitable shape, completeness was not difficult to formalize:

theorem *RP_saturated_if_fair*: $\mathsf{saturated\_upto}\ (\mathsf{Liminf}\ (\mathsf{lmap\ grounding\_of}\ \mathcal{S}s))$
corollary *RP_complete_if_fair*:
$\neg\ \mathsf{sat}\ (\mathsf{grounding\_of}\ (\mathsf{lhd}\ \mathcal{S}s)) \Longrightarrow \{\} \in \mathcal{O}\mathsf{\_of}\ (\mathsf{Liminf}\ \mathcal{S}s)$

A crucial point that is not clear from the text is that we must always use the selection function S on the first-order level and $S_{\mathsf{Liminf}\ \mathcal{O}s}$ on the ground level. Another subtle point is the passage "$\mathsf{Liminf}\ Gs$ (and hence $\mathsf{Liminf}\ \mathcal{S}s$) contains the empty clause." Obviously, if $\mathsf{grounding\_of}\ (\mathsf{Liminf}\ \mathcal{S}s)$ contains $\bot$, then $\mathsf{Liminf}\ \mathcal{S}s$ must as well. However, the authors do not explain the step from $\mathsf{Liminf}\ Gs$, the limit of the grounding, to $\mathsf{grounding\_of}\ (\mathsf{Liminf}\ \mathcal{S}s)$, the grounding of the limit. Fortunately, by Lemma 4.11, the latter contains all the nonredundant clauses of the former, and $\bot$ is nonredundant; hence the informal argument is fundamentally correct. For the other direction, which is used in the soundness proof, we proved that the former includes the latter.

The proof of Theorem 4.13 simultaneously talks about the prover architecture and the lifting of inferences using an appropriate extension of the nonground selection function to ground clauses. One might have expected a more modular proof in which redundancy is first lifted to nonground clauses, then RP is proved to compute fair derivations according to fair_clss_seq and the lifted redundancy criterion, and finally Theorem 4.3 establishes that the limit of these derivations is saturated, which yields completeness immediately. Instead, Theorem 4.3 is used in neither the informal nor the formal completeness proof and appears to play a purely pedagogical role.

The reason why Bachmair and Ganzinger did not follow the modular approach is subsumption. Deletion of subsumed clauses is crucial for the efficiency of any practically useful saturation prover, but it is not covered by the usual lifting of redundancy to nonground clauses, according to which a clause is redundant with respect to a clause set N if all its ground instances are entailed by strictly smaller ground instances of clauses in N. For subsumed clauses, we can guarantee only that the nonstrict ordering relation holds. Thus, the sequences of nonground clause sets computed by RP are not derivations with respect to the lifted redundancy criterion, and Theorem 4.3 is not applicable. A redundancy lifting that permits a modular proof independently of the prover architecture has very recently been investigated by Waldmann et al. [47].

8 Discussion

Bachmair and Ganzinger cover a lot of ground in a few pages. We found much of the material straightforward to formalize: It took us about two weeks to reach their Sect. 4.3, which defines RP and proves it refutationally complete. By contrast, we needed months to fully understand and formalize that section. While the chapter successfully conveys the key ideas at the propositional level, the lack of rigor makes it difficult to develop a deep understanding of ordered resolution on first-order clauses.

There are several reasons why Sect. 4.3 did not lend itself easily to a formalization. The proofs often depend on lemmas and theorems from previous sections without explicitly mentioning them. The lemmas and proofs do not quite fit together. And while the general idea of the proofs stands up, they have many confusing flaws that must be repaired. Our methodology involved the following steps: (1) rewrite the informal proofs to a handwritten pseudo-Isabelle; (2) fill in the gaps, emphasizing which lemmas are used where; (3) turn the pseudo-Isabelle into real Isabelle, but with **sorry** placeholders for the proofs; and (4) replace the **sorry**s with proofs. Progress was not always linear. As we worked on each step, more than once we discovered an earlier mistake.

The formalization helps us answer questions such as, "Is effectiveness of ordered resolution (Lemma 3.13) actually needed, and if so, where?" (Answer: In the proof of Theorem 3.15.) It also allows us to track definitions and hypotheses precisely, so that we always know the scope and meaning of every definition, lemma, or theorem. If a hypothesis appears too strong or superfluous, we can try to rephrase or eliminate it; the proof assistant tells us where the proof breaks. If a definition is changed, the proof assistant again tells us where proofs break. In the best case, they do not break at all since the proof assistant's automation is flexible enough. This happened, for example, when we changed the definition of $\rhd$ to combine deduction and deletion.

Starting from RP, we have refined it to obtain a functional implementation [37]. We could further refine it to an efficient imperative implementation following the lines of Fleury, Blanchette, and Lammich's verified SAT solver with the two-watched-literals optimization [18]. However, this would probably involve a huge amount of work. To increase provers' trustworthiness, a more practical approach is to have them generate detailed proofs that record all inferences leading to the empty clause [35,42]. Such output can be independently checked by verified programs or reconstructed using a proof assistant's inference kernel. This is the approach implemented in Sledgehammer [12], which integrates automatic provers in Isabelle. Formalized metatheory could in principle be used to deduce a formula's satisfiability from a finite saturation.

We found Isabelle/HOL eminently suitable to this kind of formalization work. Its logic balances expressiveness and ease of automation. We nowhere felt the need for dependent types. We benefited from many features of the system, including codatatypes [8], Isabelle/jEdit [51], the Isar proof language [50], locales [5], and Sledgehammer [12]. It is perhaps indicative of the maturity of theorem proving technology that most of the issues we encountered were unrelated to Isabelle. The main challenge was to understand the informal proof well enough to design suitable locale hierarchies and state the definitions and lemmas precisely, and correctly.

9 Related Work

Formalizing the metatheory of logic and deduction is an enticing proposition for many researchers in interactive theorem proving. In this section, we briefly review some of the main related work, without claim to exhaustiveness. Two recent, independent developments are particularly pertinent.

Peltier [31] proved static refutational completeness of a variant of the superposition calculus in Isabelle/HOL. Since superposition generalizes ordered resolution, his result subsumes our static completeness theorem. On the other hand, he did not formalize a prover or dynamic completeness, nor did he attempt to develop general infrastructure. It would be interesting to extend his formal development to obtain a verified superposition prover. We could also consider calculus extensions such as polymorphism [15,48], type classes [48], and AVATAR [45]. Two significant differences between Peltier's work and ours is that he represents clauses as sets instead of multisets (to exploit Isabelle's better proof automation for sets) and that he relies, for terms and unification, on an example theory file included in Isabelle (`Unification.thy`) instead of IsaFoR.

Hirokawa et al. [21] formalized, also in Isabelle/HOL, an abstract Knuth–Bendix completion procedure as well as ordered (unfailing) completion, a method developed by Bachmair, Dershowitz, and Plaisted [1]. Superposition combines ordered resolution (to reason about clauses) and ordered completion (to reason about equality). There are many similarities between their formalization and ours, which is unsurprising given that both follow work by Bachmair; for example, they need to reason about the limit of fair infinite sequences of sets of equations and rewrite rules to establish completeness.

The literature contains many other formalized completeness proofs, mostly for inference systems of theoretical interest. Early work was carried out in the 1980s and 1990s, notably by Shankar [40] and Persson [32]. Some of our own efforts are also related: completeness of first-order unordered resolution using semantic trees by Schlichtkrull [36]; completeness of a Gentzen system following the Beth–Hintikka style and soundness of a cyclic proof system for first-order logic with inductive definitions by Blanchette, Popescu, and Traytel [13]; and completeness of a SAT solver based on CDCL (conflict-driven clause learning) by Blanchette, Fleury, Lammich, and Weidenbach [10].

The formal Beth–Hintikka-style completeness proof mentioned above has a generic flavor, abstracting over the inference system. Could it be used to prove completeness of the ordered resolution calculus, or even of the RP prover? The central idea is to build a finitely branching tree that encodes a systematic proof attempt. Given a fair strategy for applying calculus rules, infinite branches correspond to countermodels. It should be possible to prove ordered resolution complete using this approach, by storing clause sets N on the tree's nodes. Each node would have at most one child, corresponding to the new clause set after performing a deduction. Such degenerate trees would be isomorphic to derivations $N_0 \rhd N_1 \rhd \cdots$ represented by lazy lists. However, the requirement that inferences can always be postponed, called *persistence* [13, Sect. 3.9], is not met for deletion steps based on a redundancy criterion. Moreover, while the generic framework takes care of applying inferences fairly and of employing König's lemma to extract an infinite path from a failed proof attempt (which is, incidentally, overkill for degenerate trees that have only one infinite path), it offers no help in building a countermodel from an infinite path (i.e., in proving the chapter's Theorem 3.9).

Very recently, Waldmann et al. [47] proposed a saturation framework that generalizes Bachmair and Ganzinger's framework. Its Isabelle/HOL mechanization, by Tourret [44], could form the basis of a streamlined formal proof of RP's completeness.

Beyond completeness, Gödel's first incompleteness theorem has been formalized in Nqthm by Shankar [41], in Coq by O'Connor [29], in HOL Light by Harrison (in unpublished work), and in Isabelle/HOL by Paulson [30] and by Popescu and Traytel [34]. The Isabelle formalizations also cover Gödel's second incompleteness theorem. We refer to our earlier papers [10,13,36] for further discussions of related work.

10 Conclusion

We presented a formal proof that captures the core of Bachmair and Ganzinger's *Handbook* chapter on resolution theorem proving. For all its idiosyncrasies, the chapter withstood the test of formalization, once we had added self-inferences to the RP prover. Given that the text is a basic building block of automated reasoning (as confirmed by its placement as Chapter 2), we believe there is value in clarifying its mathematical content for the next generations of researchers. We also hope that our work will be useful to the editors of a future revision of the *Handbook*.

Formalization of the metatheory of logical calculi is one of the many connections between automatic and interactive theorem proving. We expect to see wider adoption of proof assistants by researchers in automated reasoning, as a convenient way to develop metatheory. By building formal libraries of standard results, we aim to make it easier to formalize state-of-the-art research as it emerges. We also see potential uses of formal proofs in teaching automated reasoning, inspired by the use of proof assistants in courses on the semantics of programming languages [26,33].

Acknowledgements Christoph Weidenbach repeatedly discussed Bachmair and Ganzinger's chapter with us and hosted Schlichtkrull at the Max-Planck-Institut in Saarbrücken. Christian Sternagel and René Thiemann answered our questions about IsaFoR. Mathias Fleury, Florian Haftmann, and Tobias Nipkow helped enrich and reorganize Isabelle's multiset library. Mathias Fleury, Robert Lewis, Simon Robillard, Mark Summerfield, Sophie Tourret, and the anonymous reviewers suggested many textual improvements.

Funding Schlichtkrull was supported by a Ph.D. scholarship in the Algorithms, Logic and Graphs section of DTU Compute and by the LIGHT^{est} project, which is partially funded by the European Commission as an Innovation Act as part of the Horizon 2020 research and innovation program (Grant Agreement No. 700321, LIGHTest). Blanchette was partly supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (Grant NI 491/14-1). He also received funding from the ERC under the European Union's Horizon 2020 program (Grant Agreement No. 713999, Matryoshka) and from the Netherlands Organization for Scientific Research (NWO) under the Vidi program (Project No. 016.Vidi.189.037, Lean Forward). Traytel was partly supported by the DFG program Program and Model Analysis (PUMA, doctorate Program 1480).

Appendix A: Errors and Imprecisions Discovered in the Chapter

In the chapter, we encountered several mathematical errors and imprecisions of various levels of severity. We also found lemmas that were stated but not explicitly applied afterwards. For reference, this appendix provides an exhaustive list of our findings. This list illustrates how difficult it is to write paper proofs correctly, and reminds us that we cannot rely on reviewers or second readers to catch all mistakes. We hope that our corrections will further increase the chapter's value to the research community.

Regarding the errors and imprecisions, we have ignored infelicities that are not mathematical in nature, such as typos and LaTeX macros gone wrong (e.g., "by the defn[candidate model]*candidate model* for N" on page 34); for such errors, careful reading, not formalization, is the remedy. We have also ignored minor ambiguities if they can easily be resolved by appealing to the context and the reader's common sense (e.g., whether the clause $C \vee A \vee \cdots \vee A$ may contain zero occurrences of A).

- One of Lemma 3.4's claims is that if clause C is true in I^D, then C is also true in $I_{D'}$, where $C \preceq D \preceq D'$. This does not hold if $C = D = D'$ and C is productive. Similarly, the first sentence of the proof is wrong if $D = D'$ and D is productive: "First, observe that $I_D \subseteq I^D \subseteq I_{D'} \subseteq I^{D'} \subseteq I_N$, whenever $D' \succeq D$."
- The last occurrence of D' in the statement of Lemma 3.7 should be changed to C. In addition, it is not clear whether the phrase "another clause C" implies that $C \neq D$, but the counterexample we gave in Sect. 4 works in both cases. Correspondingly, in the proof, the case distinction is incomplete, as can be seen by specializing the proof for the counterexample.
- In the chapter's Figure 2, in Sect. 3, the selection function is wrongly applied: References to $S(D)$ should be changed to $S(\neg A_1 \vee \cdots \vee \neg A_n \vee D)$. Moreover, in condition (iii), it is not clear with respect to which clause the "selected atom" must be considered, the two candidates being $S(\neg A_1 \vee \cdots \vee \neg A_n \vee D)$ and $S(C_i \vee A_i \vee \cdots \vee A_i)$. We assume

the latter is meant. Finally, phrases like "A_1 is maximal with respect to D" (here and in Figure 4) are slightly ambiguous, because it is unclear whether A_1 denotes an atom or a (positive) literal, and whether it must be maximal with respect to D's atoms or literals. From the context, we infer that an atom-with-atom comparison is meant.

- Soundness is required in the chapter's Sect. 4.1, even though it is claimed in Sect. 2.4 that only consistency-preserving inference systems will be considered.
- In Sect. 4.1, it is claimed that "a fair derivation can be constructed by exhaustively applying inferences to persisting formulas." However, this construction is circular: The notion of persisting formula (i.e., the formulas that belong to the limit) depends itself on the derivation.
- In the proof of Theorem 4.3, the case where $\gamma \in \mathcal{R}_{\mathcal{I}}(N_\infty \setminus \mathcal{R}_{\mathcal{F}}(N_\infty))$ is not covered.
- In Sect. 4.2, the phrase "side premises that are true in N" must be understood as meaning that the side premises both belong to N and are true in I_N.
- Lemma 4.5 states the basic properties of the redundant clause operator $\mathcal{R}_{\mathcal{F}}$ (monotonicity and independence). Lemma 4.6 states the corresponding properties of the redundant inference operator $\mathcal{R}_{\mathcal{I}}$. As justification for Lemma 4.6, the authors tell us that "the proof uses Lemma 4.5," but redundant inferences are a more general concept than redundant clauses, and we see no way to bridge the gap.
- Similarly, in the proof of Theorem 4.9, the application of Lemma 4.5 does not fit. What is needed is a generalization of Lemma 4.6.
- In condition (ii) of Figure 4, Sect. 4.2, $A_{ii}\sigma$ should be changed to $A_{ij}\sigma$.
- In the nth side premise of Figure 4, Sect. 4.2, A_{1n} should be changed to A_{n1}.
- In Figure 4, Sect. 4.2, the same mistakes as in Figure 2 occur about the application of the selection function.
- Sect. 4.3 states "Subsumption defines a well-founded ordering on clauses." A simple counterexample is an infinite sequence repeating some clause. "Subsumption" should be replaced by "proper subsumption."
- In Lemma 4.10, it is not clear which selection function is used. When the lemma is applied in the proofs of Lemma 4.11 and Theorem 4.13, it must be $S_{\mathcal{O}_\infty}$.
- In Lemma 4.10, $G(\mathcal{S})$ and $G(\mathcal{S}')$ are related by $\rhd^*$, but $\rhd$ is needed in the proofs of Lemma 4.11 and Lemma 4.13 since then derivations in RP, which are possibly infinite, can be projected to theorem proving processes. However $G(\mathcal{S}) \rhd G(\mathcal{S}')$ does not hold in one of the cases since a combination of deduction and deletion is required. A solution is to change the definition of $\rhd$ to allow such combinations.
- In Lemma 4.10, it is not clear that the extension used should be the same between any considered pair of states. Otherwise, the lemma cannot be used to project derivations in RP to theorem proving processes.
- In Lemma 4.11, it is not clear which selection function is used. When the lemma is applied in the proofs of Theorem 4.13, it must be $S_{\mathcal{O}_\infty}$.
- A step in the proof of Lemma 4.11 considers a clause $D \in \mathcal{P}_l$ which has a nonredundant instance C. It is claimed that when D is removed from $\mathcal{P}$, another clause D' with C as instance appears in some $\mathcal{O}'_l$. That, however, does not follow if D was removed by backward subsumption. The problem can be resolved by choosing D as minimal, with respect to subsumption, among the clauses that generalize C in the derivation. This can be done since proper subsumption is well founded.
- In Lemma 4.11, a minor inconsistency is that the described first-order derivation is indexed from 1 instead of 0.
- In the proof of Theorem 4.13, the conclusion of Lemma 4.11 is stated as $N_\infty \setminus \mathcal{R}(N_\infty) \subseteq \mathcal{O}_\infty$, but it should have been $N_\infty \setminus \mathcal{R}(N_\infty) \subseteq G(\mathcal{O}_\infty)$. Furthermore, when Lemma 4.11

was first stated, the conclusion was $N_\infty \setminus \mathcal{R}_\mathcal{F}(N_\infty) \subseteq G(\mathcal{S}_\infty)$. The two are by fairness equivalent, but we find $N_\infty \setminus \mathcal{R}(N_\infty) \subseteq G(\mathcal{O}_\infty)$ more intuitive since it more clearly expresses that all nonredundant clauses become old.

Chief among the factors that contribute to making the chapter hard to follow is that many lemmas are stated (and usually proved) but not referenced later. We already mentioned the unfortunate Lemma 3.7. Sect. 4 contains several other specimens:

- Theorem 4.3 (*fair_derive_saturated_upto*) states a completeness theorem for fair derivations. However, in Sect. 4.3, fairness is defined differently, and neither the text nor the formalization applies this theorem.
- For the same reason, the property stated in the next-to-last sentence of Sect. 4.1 (*standard_ redundancy_ criterion_ extension_fair_iff*), which lifts fairness with respect to $(\mathcal{R}_\mathcal{F}, \mathcal{R}_\mathcal{I})$ to a standard extension $(\mathcal{R}_\mathcal{F}, \mathcal{R}'_\mathcal{I})$, is not needed later.
- Lemma 4.2 (*sat_deriv_Liminf_iff*, *Ri_limit_Sup*, *Rf_limit_Sup*) is not referenced in the text, but we need it (*sat_deriv_Liminf_iff*, *Ri_limit_Sup*) to prove Theorem 4.13 (*fair_state_seq_complete*). We also need it (*Rf_limit_Sup*) to prove Lemma 4.11 (*fair_imp_Liminf_minus_Rf_subset_ground_Liminf_state*).
- Lemma 4.6 (*saturated_upto_complete_if*) is not referenced in the text, but we need it to prove Lemma 4.10 (*resolution_prover_ground_derivation*), Lemma 4.11 (*fair_imp_ Liminf_minus_Rf_subset_ground_Liminf_state*), and Theorem 4.13 (*fair_state_seq_ complete*).
- Theorem 4.8 (*Ri_effective*) is not referenced in the text, but we need it to prove Theorem 4.13 (*fair_state_seq_complete*).
- Theorem 4.9 (*saturated_upto_complete*) is invoked implicitly in the next-to-last sentence in the proof of Theorem 4.13 (*fair_state_seq_complete*).

References

1. Bachmair, L., Dershowitz, N., Plaisted, D.A.: Completion without failure. In: Aït-Kaci, H., Nivat, M. (eds.) Rewriting Techniques—Resolution of Equations in Algebraic Structures, vol. 2, pp. 1–30. Academic Press, London (1989)
2. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. **4**(3), 217–247 (1994)
3. Bachmair, L., Ganzinger, H.: Ordered chaining calculi for first-order theories of transitive relations. J. ACM **45**(6), 1007–1049 (1998)
4. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 19–99. Elsevier, Amsterdam (2001)
5. Ballarin, C.: Locales: a module system for mathematical theories. J. Autom. Reason. **52**(2), 123–153 (2014)
6. Baumgartner, P., Waldmann, U.: Hierarchic superposition revisited. In: Lutz, C., Sattler, U., Tinelli, C., Turhan, A., Wolter, F. (eds.) Description Logic, Theory Combination, and All That—Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday. LNCS, vol. 11560, pp. 15–56. Springer, Berlin (2019)
7. Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P., Waldmann, U.: Superposition with lambdas. In: Fontaine, P. (ed.) CADE-27, LNCS, vol. 11716, pp. 55–73. Springer, Berlin (2019)
8. Biendarra, J., Blanchette, J.C., Bouzy, A., Desharnais, M., Fleury, M., Hölzl, J., Kuncar, O., Lochbihler, A., Meier, F., Panny, L., Popescu, A., Sternagel, C., Thiemann, R., Traytel, D.: Foundational (co)datatypes and (co)recursion for higher-order logic. In: Dixon, C., Finger, M. (eds.) FroCoS 2017, LNCS, vol. 10483, pp. 3–21. Springer, Berlin (2017)
9. Blanchette, J.C.: Formalizing the metatheory of logical calculi and automatic provers in Isabelle/HOL (invited talk). In: Mahboubi, A., Myreen, M.O. (eds.) CPP 2019, pp. 1–13. ACM (2019)
10. Blanchette, J.C., Fleury, M., Lammich, P., Weidenbach, C.: A verified SAT solver framework with learn, forget, restart, and incrementality. J. Autom. Reason. **61**(3), 333–366 (2018)

11. Blanchette, J.C., Fleury, M., Traytel, D.: Nested multisets, hereditary multisets, and syntactic ordinals in Isabelle/HOL. In: Miller, D. (ed.) FSCD 2017, LIPIcs, vol. 84, pp. 11:1–11:18. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2017)
12. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. J. Formaliz. Reason. **9**(1), 101–148 (2016)
13. Blanchette, J.C., Popescu, A., Traytel, D.: Soundness and completeness proofs by coinductive methods. J. Autom. Reason. **58**(1), 149–179 (2017)
14. Brand, D.: Proving theorems with the modification method. SIAM J. Comput. **4**(4), 412–430 (1975)
15. Cruanes, S.: Logtk: A logic toolkit for automated reasoning and its implementation. In: Schulz, S., de Moura, L., Konev, B. (eds.) PAAR-2014, EPiC Series in Computing, vol. 31, pp. 39–49. EasyChair (2014)
16. Denzinger, J., Kronenburg, M., Schulz, S.: DISCOUNT—a distributed and learning equational prover. J. Autom. Reason. **18**(2), 189–198 (1997)
17. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM **22**(8), 465–476 (1979)
18. Fleury, M., Blanchette, J.C., Lammich, P.: A verified SAT solver with watched literals using Imperative HOL. In: Andronick, J., Felty, A.P. (eds.) CPP 2018, pp. 158–171. ACM (2018)
19. Godoy, G., Nieuwenhuis, R.: Superposition with completely built-in abelian groups. J. Symb. Comput. **37**(1), 1–33 (2004)
20. Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press, Cambridge (1993)
21. Hirokawa, N., Middeldorp, A., Sternagel, C., Winkler, S.: Infinite runs in abstract completion. In: Miller, D. (ed.) FSCD 2017, LIPIcs, vol. 84, pp. 19:1–19:16. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2017)
22. Krauss, A.: Partial recursive functions in higher-order logic. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006, LNCS, vol. 4130, pp. 589–603. Springer, Berlin (2006)
23. McCune, W.: OTTER 2.0. In: Stickel, M.E. (ed.) CADE-10, LNCS, vol. 449, pp. 663–664. Springer, Berlin (1990)
24. Nieuwenhuis, R., Rubio, A.: Theorem proving with ordering and equality constrained clauses. J. Symb. Comput. **19**(4), 321–351 (1995)
25. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 371–443. Elsevier, Amsterdam (2001)
26. Nipkow, T.: Teaching semantics with a proof assistant: no more LSD trip proofs. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012, LNCS, vol. 7148, pp. 24–38. Springer, Berlin (2012)
27. Nipkow, T., Klein, G.: Concrete Semantics: With Isabelle/HOL. Springer, Berlin (2014)
28. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer, Berlin (2002)
29. O'Connor, R.: Essential incompleteness of arithmetic verified by Coq. In: Hurd, J., Melham, T.F. (eds.) TPHOLs 2005, LNCS, vol. 3603, pp. 245–260. Springer, Berlin (2005)
30. Paulson, L.C.: A machine-assisted proof of Gödel's incompleteness theorems for the theory of hereditarily finite sets. Rew. Symb. Logic **7**(3), 484–498 (2014)
31. Peltier, N.: A variant of the superposition calculus. Archive of Formal Proofs **2016** (2016). https://www.isa-afp.org/entries/SuperCalc.shtml. Accessed 22 May 2020
32. Persson, H.: Constructive completeness of intuitionistic predicate logic—a formalisation in type theory. Licentiate thesis, Chalmers tekniska högskola and Göteborgs universitet (1996)
33. Pierce, B.C.: Lambda, the ultimate TA: Using a proof assistant to teach programming language foundations. In: Hutton, G., Tolmach, A.P. (eds.) ICFP 2009, pp. 121–122. ACM (2009)
34. Popescu, A., Traytel, D.: A formally verified abstract account of Gödel's incompleteness theorems. In: Fontaine, P. (ed.) CADE-27, LNCS, vol. 11716, pp. 442–461. Springer, Berlin (2019)
35. Reger, G., Suda, M.: Checkable proofs for first-order theorem proving. In: Reger, G., Traytel, D. (eds.) ARCADE 2017, EPiC Series in Computing, vol. 51, pp. 55–63. EasyChair (2017)
36. Schlichtkrull, A.: Formalization of the resolution calculus for first-order logic. J. Autom. Reason. **61**(4), 455–484 (2018)
37. Schlichtkrull, A., Blanchette, J.C., Traytel, D.: A verified prover based on ordered resolution. In: Mahboubi, A., Myreen, M.O. (eds.) CPP 2019, pp. 152–165. ACM (2019)
38. Schlichtkrull, A., Blanchette, J.C., Traytel, D., Waldmann, U.: Formalization of a comprehensive framework for saturation theorem proving in Isabelle/HOL. Archive of Formal Proofs **2018** (2018). https://www.isa-afp.org/entries/Ordered_Resolution_Prover.html. Accessed 22 May 2020
39. Schlichtkrull, A., Blanchette, J.C., Traytel, D., Waldmann, U.: Formalizing Bachmair and Ganzinger's ordered resolution prover. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018, LNCS, vol. 10900, pp. 89–107. Springer, Berlin (2018)

40. Shankar, N.: Towards mechanical metamathematics. J. Autom. Reason. **1**(4), 407–434 (1985)
41. Shankar, N.: Metamathematics, Machines, and Gödel's Proof, Cambridge Tracts in Theoretical Computer Science, vol. 38. Cambridge University Press, Cambridge (1994)
42. Sutcliffe, G., Zimmer, J., Schulz, S.: TSTP data-exchange formats for automated theorem proving tools. In: Zhang, W., Sorge, V. (eds.) Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems, Frontiers in Artificial Intelligence and Applications, vol. 112, pp. 201–215. IOS Press, Amsterdam (2004)
43. Thiemann, R., Sternagel, C.: Certification of termination proofs using CeTA. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009, LNCS, vol. 5674, pp. 452–468. Springer, Berlin (2009)
44. Tourret, S.: A comprehensive framework for saturation theorem proving. Archive of Formal Proofs **2020** (2020). https://www.isa-afp.org/entries/Saturation_Framework.shtml. Accessed 22 May 2020
45. Voronkov, A.: AVATAR: the architecture for first-order theorem provers. In: Biere, A., Bloem, R. (eds.) CAV 2014, LNCS, vol. 8559, pp. 696–710. Springer, Berlin (2014)
46. Waldmann, U.: Cancellative abelian monoids and related structures in refutational theorem proving (part I/II). J. Symb. Comput. **33**(6), 777–829/831–861 (2002)
47. Waldmann, U., Tourret, S., Robillard, S., Blanchette, J.: A comprehensive framework for saturation theorem proving. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS. Springer, Berlin (2020)
48. Wand, D.: Polymorphic + typeclass superposition. In: Schulz, S., de Moura, L., Konev, B. (eds.) PAAR-2014, EPiC Series in Computing, vol. 31, pp. 105–119. EasyChair (2014)
49. Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. II, pp. 1965–2013. Elsevier, Amsterdam (2001)
50. Wenzel, M.: Isabelle/Isar—a generic framework for human-readable proof documents. In: Matuszewski, R. , Zalewska, A. (eds.) From Insight to Proof: Festschrift in Honour of Andrzej Trybulec, Studies in Logic, Grammar, and Rhetoric, vol. 10(23). University of Białystok (2007)
51. Wenzel, M.: Isabelle/jEdit–a prover IDE within the PIDE framework. In: Jeuring, J., Campbell, J.A., Carette, J., Reis, G.D., Sojka, P., Wenzel, M., Sorge, V. (eds.) CICM 2012, LNCS, vol. 7362, pp. 468–471. Springer, Berlin (2012)
52. Zhang, H., Kapur, D.: First-order theorem proving using conditional rewrite rules. In: Lusk, E.L., Overbeek, R.A. (eds.) CADE-9, LNCS, vol. 310, pp. 1–20. Springer, Berlin (1988)