A modular Isabelle framework for verifying saturation provers

Sophie Tourret, Jasmin Blanchette

To cite this version:

Sophie Tourret, Jasmin Blanchette. A modular Isabelle framework for verifying saturation provers. CPP 2021 - 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Jan 2021, Virtual, Denmark. pp.224-237, 10.1145/3437992.3439912 . hal-03364015

HAL Id: hal-03364015
https://inria.hal.science/hal-03364015

Submitted on 4 Oct 2021

A Modular Isabelle Framework for Verifying Saturation Provers

Sophie Tourret
Research Group 1: Automation of Logic
Max-Planck-Institut für Informatik
Saarbrücken, Germany
stourret@mpi-inf.mpg.de

Jasmin Blanchette
Department of Computer Science
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
j.c.blanchette@vu.nl

Abstract

We present a formalization in Isabelle/HOL of a comprehensive framework for proving the completeness of automatic theorem provers based on resolution, superposition, or other saturation calculi. The framework helps calculus designers and prover developers derive, from the completeness of a calculus, the completeness of prover architectures implementing the calculus. It also helps derive the completeness of calculi obtained by lifting ground (i.e., variable-free) calculi. As a case study, we re-verified Bachmair and Ganzinger's resolution prover RP to show the benefits of modularity.

CCS Concepts: • **Theory of computation → Logic and verification**; *Automated reasoning*.

Keywords: automatic theorem provers, proof assistants, first-order logic

ACM Reference Format:
Sophie Tourret and Jasmin Blanchette. 2021. A Modular Isabelle Framework for Verifying Saturation Provers. In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '21), January 18–19, 2021, Virtual, Denmark.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3437992.3439912

1 Introduction

Many of the most successful automatic theorem provers today are based on saturation. These include the unit equality prover Waldmeister [14], the first-order provers E [23], SPASS [30], and Vampire [16], and the higher-order provers Leo-III [24] and Zipperposition [6]. A saturation prover starts with a problem, typically given in conjunctive normal form, and draws inferences from the problem clauses, adding the conclusions to the clause set. If the prover detects useless clauses, it may remove them. A refutation proof has been found when the prover has derived the empty clause, denoted by $\bot$. Which inferences are drawn and which clauses are deemed redundant depends on the logical calculus used.

Although users mostly care only about the soundness and success rate of a prover, refutational completeness is taken very seriously by researchers. Beyond its intrinsic merits, completeness is useful to optimize a calculus: From a completeness proof, calculi designers can extract precise side conditions on the inference rules, which help prune the search space. Without the guidance offered by a completeness proof, it is easy to prune too little, too much, or both. Moreover, for decidable fragments, completeness is needed to ensure that the prover can be relied on as a decision procedure.

Completeness can be formulated in two ways:

- *Static completeness* is stated in terms of a "saturated" clause set N—a set from which all necessary inferences have been drawn.
- *Dynamic completeness* is stated in terms of a sequence $(N_i)_i$ of clause sets, where each set N_i represents the prover's state at a point in time: If N_0 is unsatisfiable (i.e., the conjecture holds), then $\bot \in N_i$ for some i.

Most saturation calculi are designed as a lifting of a ground calculus—that is, a calculus that operates on variable-free clauses. Static completeness is established on the ground level and lifted along with the calculus. This separation of concerns helps keep these technical proofs manageable.

For dynamic completeness, we distinguish between simple proving processes that operate on a single clause set and realistic provers with multiple clause sets. The *given clause procedure* [17, Section 2.3], which is part of virtually all saturation provers, needs two clause sets, called "passive" and "active." Clauses are initially passive and move one by one to the active set, at which point all possible inferences with active partners are drawn. This setup is reminiscent of traditional New Year celebrations in Quebec: When you arrive at the party, you must kiss all the relatives lined up before you and then join the line. In this way, everybody kisses everybody.

A crucial optimization in provers is clause subsumption. A clause D, seen as a multiset of literals, *subsumes* another clause C if $D\sigma \subseteq C$ for some substitution σ; for example, $\mathrm{p}(x)$

CPP '21, January 18–19, 2021, Virtual, Denmark
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8299-1/21/01.
https://doi.org/10.1145/3437992.3439912

subsumes p(a) and $\mathrm{p}(x) \vee \mathrm{q}(y)$. Regrettably, much of the literature on saturation ignores subsumption and dynamic completeness. In the refinement chain leading to a verified prover, such a simplifying assumption translates into a proof gap.

Bachmair and Ganzinger's chapter in the *Handbook of Automated Reasoning* [3] is an exception. They define a three-clause-set resolution prover, called RP, that supports clause subsumption, and they prove it dynamically complete. But their argument is highly nonmodular: They move directly from static completeness of the ground calculus to dynamic completeness of RP, awkwardly mixing issues related to nonground lifting, subsumption, and the given clause procedure. The argument is hard to follow on paper, and even harder to formalize, as Schlichtkrull, Blanchette, Traytel, and Waldmann found out [21].

To remedy this situation, Waldmann, Tourret, Robillard, and Blanchette [29] designed a comprehensive framework that covers a wide range of calculi, including ordered resolution [3], unfailing completion [1], standard superposition [2], constraint superposition [18], theory superposition [28], hierarchic superposition [4], λ-free superposition [7], λ-superposition [6], and combinatory superposition [8]. This *saturation framework* can be used to lift ground static completeness to nonground static completeness and then to dynamic completeness of minimalistic proving processes or given clause provers with support for clause subsumption.

The framework was initially developed in LaTeX, but we also managed to formalize it in Isabelle/HOL in time to mention it in Waldmann et al. [29, Section 5]. The present paper aims at introducing this work to the interactive theorem proving community. Besides giving a thorough account of the formalization, we unveil three extensions of the framework:

- We applied the formalized framework to re-prove RP dynamically complete. The new proof is substantially simpler and shorter than that by Schlichtkrull et al.
- We formalized various basic concepts related to theorem proving that were needed to verify RP and that will likely be useful for other provers. Much of this material was generalized from Schlichtkrull et al.
- We formalized alternative, invariance-based completeness proofs for given clause procedures. These new proofs replace previous monolithic arguments and will be used, in pen-and-paper form, in a forthcoming journal submission by Waldmann et al.

We used the declarative Isar [31] in the hope that practitioners who are not Isabelle experts can follow the intermediate proof steps. We also relied extensively on locales [5], which provide parameterized modules. The vast majority of the pen-and-paper arguments survived formalization, but one serious issue was discovered and repaired.

Our work is part of the IsaFoL (Isabelle Formalization of Logic) effort, an unfunded coalition of the willing that aims at developing a library of results about logic and automated reasoning [10]. The Isabelle theory files amount to around 5700 lines of source text. They were developed using Isabelle version 2020 and are available in the *Archive of Formal Proofs* [9, 26]. The *AFP* is continuously updated to track Isabelle's evolution, ensuring the compatibility of our theories with the current version of Isabelle.[1]

This paper's organization largely follows Waldmann et al. Briefly, we introduce the fundamental notions that make up a calculus and related concepts such as saturation and completeness (Section 2); we cover the lifting from the ground to the nonground level and the delicate issue of clause subsumption (Section 3); we present two given clause procedures and our alternative, invariance-based proofs (Section 4); and we present our modular proof of RP's completeness (Section 5).

2 Calculi

The saturation framework relies on a number of basic notions that are relevant to prove a calculus complete. These include formulas, consequence relations, inference systems, and redundancy criteria (all defined in `Calculus.thy`, unless indicated otherwise).

2.1 Formulas

The entire framework is built on an abstract notion of formulas, generalizing clauses. Formulas are represented by a type variable $'f$ equipped with a set Bot and an entailment relation $\models$. Bot is a nonempty set of patently false formulas. Typically, we take Bot $= \{\bot\}$, but it can be useful to differentiate between an active $\bot$ and a passive $\bot$. Entailment must satisfy four basic properties. Formally:

locale consequence_relation =
  fixes
    Bot :: $'f\ set$ **and**
    $\models$ :: $'f\ set \Rightarrow {'f}\ set \Rightarrow bool$
  assumes
    Bot $\neq \emptyset$ **and**
    $B \in \text{Bot} \Longrightarrow \{B\} \models N_1$ **and**
    $N_2 \subseteq N_1 \Longrightarrow N_1 \models N_2$ **and**
    $(\forall C \in N_2.\ N_1 \models \{C\}) \Longrightarrow N_1 \models N_2$ **and**
    $N_1 \models N_2 \Longrightarrow N_2 \models N_3 \Longrightarrow N_1 \models N_3$

An Isabelle locale can be seen as a record type, where fields can be types ($'f$), terms (Bot, $\models$), or properties. When we later instantiate the locale, we must supply concrete arguments for the types and terms and then discharge the proof obligations corresponding to the properties. Locales are invaluable; they allow us to declare parameters and assumptions once and reuse them in multiple related definitions and lemmas. They also emphasize the modular nature of the proof development.

[1] To use our theories with Isabelle2020, check out revision a61b4f25e279 of the development repository of the *AFP*, hosted at https://foss.heptapod.net/isa-afp/afp-devel.

2.2 Inference Systems

An inference $C_n, \dots, C_1 \vdash C_0$ is a recipe to derive a new formula C_0 (the conclusion) from existing formulas $C_n, \dots, C_1$ (the premises). Formally:

datatype *'f inference* =
  Infer (*'f list*) *'f*

An inference ι is written out as $\iota = \mathsf{Infer}\ [C_n, \dots, C_1]\ C_0$, with $\mathsf{prems\_of}\ \iota = [C_n, \dots, C_1]$ and $\mathsf{concl\_of}\ \iota = C_0$.

An inference rule, with its metavariables and side conditions, generally corresponds to a set of inferences. In fact, an entire inference system, consisting of one or more inference rules, is modeled as a set of inferences Inf. Formally:

locale inference_system =
  fixes $\mathsf{Inf} :: \mathit{'f\ inference\ set}$

Inside the locale, we defined the following functions:

definition $\mathsf{Inf\_from} :: \mathit{'f\ set} \Rightarrow \mathit{'f\ inference\ set}$ **where**
  $\mathsf{Inf\_from}\ N = \{\iota \in \mathsf{Inf}.\ \mathsf{prems\_of}\ \iota \subseteq N\}$

definition $\mathsf{Inf\_between} :: \mathit{'f\ set} \Rightarrow \mathit{'f\ set} \Rightarrow \mathit{'f\ inference\ set}$
where
  $\mathsf{Inf\_between}\ N\ M =$
  $\mathsf{Inf\_from}\ (N \cup M) - \mathsf{Inf\_from}\ (N - M)$

The auxiliary function $\mathsf{Inf\_from}\ N$ returns the set of possible inferences from premises contained in N. The variant $\mathsf{Inf\_between}\ N\ M$ returns the set of possible inferences from premises in $N \cup M$ but with at least one premise in M.

2.3 Calculi with Redundancy

A calculus is basically an inference system equipped with a redundancy criterion. The inference system indicates which inferences must a priori be drawn. The redundancy criterion specifies inferences that can a posteriori be omitted. It also identifies deletable formulas. Formally:

locale calculus =
    inference_system Inf +
    consequence_relation Bot ($\models$) +
  fixes
    $\mathsf{Red_I} :: \mathit{'f\ set} \Rightarrow \mathit{'f\ inference\ set}$ **and**
    $\mathsf{Red_F} :: \mathit{'f\ set} \Rightarrow \mathit{'f\ set}$
  assumes
    $\mathsf{Red_I}\ N \subseteq \mathsf{Inf}$ **and**
    $B \in \mathsf{Bot} \Longrightarrow N \models \{B\} \Longrightarrow N - \mathsf{Red_F}\ N \models \{B\}$ **and**
    $N \subseteq N' \Longrightarrow \mathsf{Red_F}\ N \subseteq \mathsf{Red_F}\ N'$ **and**
    $N \subseteq N' \Longrightarrow \mathsf{Red_I}\ N \subseteq \mathsf{Red_I}\ N'$ **and**
    $N' \subseteq \mathsf{Red_F}\ N \Longrightarrow \mathsf{Red_F}\ N \subseteq \mathsf{Red_F}\ (N - N')$ **and**
    $N' \subseteq \mathsf{Red_F}\ N \Longrightarrow \mathsf{Red_I}\ N \subseteq \mathsf{Red_I}\ (N - N')$ **and**
    $\iota \in \mathsf{Inf} \Longrightarrow \mathsf{concl\_of}\ \iota \in N \Longrightarrow \iota \in \mathsf{Red_I}\ N$

The locale inherits Inf (and the auxiliaries Inf_from and Inf_between) from inference_system as well as Bot and $\models$ from consequence_relation. An inference ι is redundant if $\iota \in \mathsf{Red_I}\ N$; a formula C is redundant if $C \in \mathsf{Red_F}\ N$. The locale assumptions are needed to establish dynamic completeness. For example, the second assumption ensures that deleting a redundant formula from an unsatisfiable set preserves the set's unsatisfiability.

2.4 Static and Dynamic Completeness

Static completeness is useful because it is comparatively convenient to formulate and establish. Dynamic completeness is useful because it directly captures a desired property of proving processes. Under some basic assumptions that are met by well-designed redundancy criteria, the two concepts coincide. Thus, we can establish the static completeness of a calculus and immediately obtain the dynamic completeness of a proving process $\rhd$ based on it.

Static completeness is expressed in terms of a single formula set N, which must be saturated. Saturation means that all inferences from N are redundant:

definition $\mathsf{saturated} :: \mathit{'f\ set} \Rightarrow \mathit{bool}$ **where**
  $\mathsf{saturated}\ N \longleftrightarrow \mathsf{Inf\_from}\ N \subseteq \mathsf{Red_I}\ N$

Completeness means that any unsatisfiable saturated set must contain a patent falsehood (typically, $\bot$):

locale statically_complete_calculus = calculus +
  assumes $B \in \mathsf{Bot} \Longrightarrow \mathsf{saturated}\ N \Longrightarrow N \models \{B\} \Longrightarrow$
    $\exists B' \in \mathsf{Bot}.\ B' \in N$

We abuse terminology above when we write that a set is "unsatisfiable." Given that we only have a notion of entailment but not of modelhood, it would be more proper (but less standard) to write "inconsistent."

Dynamic completeness more closely models saturation provers. It is expressed in terms of a finite or infinite sequence of formula sets represented by the codatatype *'f set llist*. A derivation is a sequence where each pair of successive elements satisfies the following relation:

inductive $\rhd :: \mathit{'f\ set} \Rightarrow \mathit{'f\ set} \Rightarrow \mathit{bool}$ **where**
  $M - N \subseteq \mathsf{Red_F}\ N \Longrightarrow M \rhd N$

The $\rhd$ relation represents an abstract nondeterministic proving process. Informally, when taking a transition from a set M to another set N, a prover can add arbitrary formulas (corresponding to $N - M$), and it can remove arbitrary formulas (corresponding to $M - N$) as long as these are redundant w.r.t. N. Most provers would add only formulas that are entailed by M, but this is not enforced by $\rhd$.

We write $\mathsf{chain}\ (\rhd)\ \mathit{Ns}$ to express that the sequence Ns is a derivation: $\mathit{Ns}\ !\ 0 \rhd \mathit{Ns}\ !\ 1 \rhd \cdots$, where the infix operator ! returns the sequence element at the given index. The chain predicate is defined coinductively as in Schlichtkrull et al. [21, Section 5].

Dynamic completeness applies only to fair derivations:

definition $\mathsf{fair} :: \mathit{'f\ set\ llist} \Rightarrow \mathit{bool}$ **where**
  $\mathsf{fair}\ \mathit{Ns} \longleftrightarrow$
  $\mathsf{Inf\_from}\ (\mathsf{Liminf}\ \mathit{Ns}) \subseteq \mathsf{Sup}\ (\mathsf{lmap}\ \mathsf{Red_I}\ \mathit{Ns})$

Above, $\mathsf{Liminf}\ \mathit{Ns}$ denotes the limit inferior $\bigcup_i \bigcap_{j \geq i} \mathit{Ns}\ !\ j$, and $\mathsf{Sup}\ \mathit{Ns}$ denotes the supremum $\bigcup_i \mathit{Ns}\ !\ i$. In accordance with the automated reasoning literature, we will refer to Liminf *Ns* as the *limit* of *Ns*.

Intuitively, a fair derivation is one where every inference with premises in the limit was made redundant at some index i, usually by adding its conclusion to $Ns \,!\, i$. Using the locale assumptions about redundancy, it is easy to show that the limit of a fair derivation is saturated.

Dynamic completeness is formulated in its own locale:

locale dynamically_complete_calculus = calculus +

  assumes

    $B \in \mathsf{Bot} \Longrightarrow \mathsf{chain}\ (\rhd)\ Ns \Longrightarrow \mathsf{fair}\ Ns \Longrightarrow$

    $Ns \,!\, 0 \models \{B\} \Longrightarrow$

    $\exists i < \mathsf{llength}\ Ns.\ \exists B' \in \mathsf{Bot}.\ B' \in Ns \,!\, i$

Owing to the conditions on redundancy criteria, static and dynamic completeness are equivalent:

sublocale statically_complete_calculus

  ⊆ dynamically_complete_calculus

sublocale dynamically_complete_calculus

  ⊆ statically_complete_calculus

The above definitions are all based on Waldmann et al., who in turn largely follow Bachmair and Ganzinger but generalize them. The literature is not entirely consistent on notions such as redundancy, saturation, and fairness. The first group of authors studied variations and showed that completeness and "reduced" completeness coincide. We also formalized these proofs (in `Calculus_Variations.thy`). They make it possible to mix and match results. For example, suppose that a calculus is proved "reducedly" statically complete in the literature. Instead of redoing the proof in a standard, nonreduced setting, we can reuse the reduced proof as is and derive dynamic completeness using the framework.

2.5 Intersection of Redundancy Criteria

There are various situations where a prover may need to apply several redundancy criteria in parallel and identify an inference or formula as redundant only if all the criteria agree. This can be incorporated into the framework as a single intersection criterion. One scenario where this is particularly useful is when redundancy is parameterized by a value $q :: {'q}$ that is known only at the limit. The intersection amounts to a universal quantification "for all $q \in \mathsf{Q}$, …." Once we know the limit, we can supply the right value for q.

Although this is not always necessary, each redundancy criterion may use its own consequence relation. Thus, we defined a consequence_relation_family locale parameterized by a set $\mathsf{Q} :: {'q}\ \mathit{set}$ and by a Q-indexed family of consequence relations $\mathsf{entails_q} :: {'q} \Rightarrow {'f}\ \mathit{set} \Rightarrow {'f}\ \mathit{set} \Rightarrow \mathit{bool}$ such that each $\mathsf{entails_q}\ q$, for $q \in \mathsf{Q}$, qualifies as a consequence relation. The q subscript in the constant name is there to remind us that the family is indexed by the type $'q$. We will encounter this convention again later.

The next step was to define a single relation $\models_\mathsf{Q}$ as the intersection of the $\mathsf{entails_q}$'s:

definition $\models_\mathsf{Q} :: {'f}\ \mathit{set} \Rightarrow {'f}\ \mathit{set} \Rightarrow \mathit{bool}$ **where**

  $N_1 \models_\mathsf{Q} N_2 \longleftrightarrow \forall q \in \mathsf{Q}.\ \mathsf{entails_q}\ q\ N_1\ N_2$

We proved that $\models_\mathsf{Q}$ qualifies as a consequence relation.

Next, we defined a locale for intersection calculi, which work with the intersection of a Q-indexed family of redundancy criteria. The definitions (in `Intersection_Calculus.thy`) follow the same pattern as above. The locale follows:

locale intersection_calculus =

    inference_system Inf +

    consequence_relation_family Bot Q $\mathsf{entails_q}$ +

  fixes

    $\mathsf{Red_{Iq}} :: {'q} \Rightarrow {'f}\ \mathit{set} \Rightarrow {'f}\ \mathit{inference\ set}$ **and**

    $\mathsf{Red_{Fq}} :: {'q} \Rightarrow {'f}\ \mathit{set} \Rightarrow {'f}\ \mathit{set}$

  assumes

    $\mathsf{Q} \neq \emptyset$ **and**

    $\forall q \in \mathsf{Q}.\ \mathsf{calculus\ Bot\ Inf}\ (\mathsf{entails_q}\ q)\ (\mathsf{Red_{Iq}}\ q)$

        $(\mathsf{Red_{Fq}}\ q)$

The redundancy criterion is defined as a pair of intersections:

definition $\mathsf{Red_I} :: {'f}\ \mathit{set} \Rightarrow {'f}\ \mathit{inference\ set}$ **where**

  $\mathsf{Red_I}\ N = (\bigcap q \in \mathsf{Q}.\ \mathsf{Red_{Iq}}\ q\ N)$

definition $\mathsf{Red_F} :: {'f}\ \mathit{set} \Rightarrow {'f}\ \mathit{set}$ **where**

  $\mathsf{Red_F}\ N = (\bigcap q \in \mathsf{Q}.\ \mathsf{Red_{Fq}}\ q\ N)$

Intersection calculi qualify as calculi with the intersections $\mathsf{Red_I}$ and $\mathsf{Red_F}$ as redundancy criterion:

sublocale calculus Bot Inf $(\models_\mathsf{Q})$ $\mathsf{Red_I}$ $\mathsf{Red_F}$

To show that a calculus is statically complete, it suffices to show that for any saturated set N that does not contain a patent falsehood, there exists an index $q \in \mathsf{Q}$ according to which N is consistent:

lemma *stat_ref_comp_from_bot_in_sat*:

  $(\forall N.\ \mathsf{saturated\ Inf\ Red_I}\ N \wedge (\forall B \in \mathit{Bot}.\ B \notin N) \longrightarrow$

    $\exists B \in \mathsf{Bot}.\ \exists q \in \mathsf{Q}.\ \neg\ \mathsf{entails_q}\ q\ N\ \{B\}) \Longrightarrow$

  statically_complete_calculus Bot Inf $(\models_\mathsf{Q})$ $\mathsf{Red_I}$ $\mathsf{Red_F}$

2.6 Soundness

An inference system Inf is sound w.r.t. a consequence relation $\models$ if every inference $\iota \in \mathsf{Inf}$ is sound—that is, $\mathsf{prems\_of}\ \iota \models \{\mathsf{concl\_of}\ \iota\}$. Soundness can be shown one inference rule at a time, without a sophisticated framework. Nevertheless, we formalized a few basic soundness results, as a convenience to users of the framework (in `Soundness.thy`).

The main soundness lemma, generalized from Schlichtkrull et al., states that the limit of a sound derivation is unsatisfiable if and only if the initial formula set is unsatisfiable:

lemma *unsat_limit_iff*:

  $\mathsf{chain}\ (\rhd)\ Ns \Longrightarrow \mathsf{chain}\ (\models)\ Ns \Longrightarrow$

  $(\mathsf{Liminf}\ Ns \models \mathsf{Bot} \longleftrightarrow Ns \,!\, 0 \models \mathsf{Bot})$

For the forward direction, the proof by Schlichtkrull et al., like the one by Bachmair and Ganzinger on which it is based, requires compactness—the property that whenever $N \models \mathsf{Bot}$, there exists a finite set $M \subseteq N$ such that $M \models \mathsf{Bot}$. Informally, the argument is as follows: If the limit of *Ns* is unsatisfiable, by compactness there exists an unsatisfiable finite subset; by the definition of limit, it takes only finitely many steps to reach a set that includes this finite subset. By transitivity of $\models$, the initial state $Ns\ !\ 0$ is unsatisfiable.

In a private communication, Waldmann supplied an almost as simple proof that dispenses with compactness. By formalizing this proof, we were able to eliminate a few bureaucratic locales and locale instantiations.

2.7 Standard Redundancy

Ground versions of resolution and superposition are parameterized by a well-order $<$ that is lifted to clauses. The completeness proof is a well-founded induction over the elements of a saturated clause set $N \not\ni \bot$ w.r.t. $<$. Each clause is shown in turn to be true in a candidate model. Such inference systems are called *counterexample-reducing*. This terminology stems from an alternative formulation of induction as a proof by contradiction starting with the assumption that a minimal counterexample exists [3, Section 4.2].

Counterexample-reducing calculi distinguish between the rightmost, or *main*, premise and the remaining *side* premises. Given an inference $\iota = \mathsf{Infer}\ [C_n, \ldots, C_1]\ C_0$, we have $\mathsf{main\_prem\_of}\ \iota = C_1$ and $\mathsf{side\_prems\_of}\ \iota = [C_n, \ldots, C_2]$.

An inference system Inf operating on formulas equipped with a well-order $<$ is counterexample-reducing if the following locale requirements can be met:

$$\begin{array}{l}
\textbf{fixes}\ \mathsf{I\_of} :: {'f}\ \mathit{set} \Rightarrow {'f}\ \mathit{set} \\
\textbf{assumes} \\
\quad N \cap \mathsf{Bot} = \emptyset \Longrightarrow D \in N \Longrightarrow \mathsf{I\_of}\ N \not\models \{D\} \Longrightarrow \\
\quad (\bigwedge C.\ C \in N \Longrightarrow \mathsf{I\_of}\ N \not\models \{C\} \Longrightarrow D \leq C) \Longrightarrow \\
\quad \exists \iota \in \mathsf{Inf}.\ \mathsf{prems\_of}\ \iota \neq [] \wedge \mathsf{main\_prem\_of}\ \iota = D \\
\qquad\qquad\quad \wedge\ \mathsf{side\_prems\_of}\ \iota \subseteq N \\
\qquad\qquad\quad \wedge\ \mathsf{I\_of}\ N \models \mathsf{side\_prems\_of}\ \iota \\
\qquad\qquad\quad \wedge\ \mathsf{I\_of}\ N \not\models \mathsf{concl\_of}\ \iota \wedge \mathsf{concl\_of}\ \iota < D
\end{array}$$

Since the framework provides no notion of interpretations, we represent an interpretation as a formula set M and write $M \models N$ to mean "M is a model of N." Given a formula set, the I_of operator yields a candidate model. The locale assumption states that if N contains no patent falsehoods and D is a minimal counterexample formula in N, then there exists an inference from N that produces an even smaller counterexample.

It is not hard to do nonsensical things using a proof assistant. Adversarial users might define I_of N as $\{\bot\}$ and use this to show that *any* calculus is counterexample-reducing. This would not take them very far, however, because they could never connect that definition of I_of with a concrete notion of model suitable for their logic.

Counterexample-reducing inference systems are compatible with the *standard redundancy criterion*, defined as follows (in `Standard_Redundancy_Criterion.thy`):

$$\begin{array}{l}
\textbf{definition}\ \mathsf{Red_I} :: {'f}\ \mathit{set} \Rightarrow {'f}\ \mathit{inference\ set}\ \ \textbf{where} \\
\quad \mathsf{Red_I}\ N = \{\iota \in \mathsf{Inf}.\ \exists M \subseteq N. \\
\qquad M \cup \mathsf{side\_prems\_of}\ \iota \models \{\mathsf{concl\_of}\ \iota\} \\
\qquad \wedge\ \forall D \in M.\ D < \mathsf{main\_prem\_of}\ \iota\}
\end{array}$$

$$\begin{array}{l}
\textbf{definition}\ \mathsf{Red_F} :: {'f}\ \mathit{set} \Rightarrow {'f}\ \mathit{set}\ \ \textbf{where} \\
\quad \mathsf{Red_F}\ N = \{C.\ \exists M \subseteq N.\ M \models \{C\} \wedge \forall D \in M.\ D < C\}
\end{array}$$

For inference systems Inf that reduce counterexamples, we obtain a completeness theorem: If N is saturated w.r.t. the standard redundancy criterion and does not contain a patent falsehood, then I_of N is a model of N.

2.8 Clauses

Most saturation calculi work with clauses. In Isabelle, we reuse the representation of clauses as finite multisets of literals introduced by Schlichtkrull et al. [21, Section 2.3]. The empty clause $\bot$ is represented by $\emptyset$, but we will continue to write $\bot$. We take $\mathsf{Bot} = \{\bot\}$. Literals over atoms of type *'a* take the form Pos A or Neg A, where $A :: {'a}$. In the literature, sets are sometimes used instead of multisets to represent clauses, but multisets are more suitable for resolution- and superposition-like calculi because the size of a multiset is invariant under substitution.

As a sanity check and as a convenience to users, we specialized various framework concepts to clauses (in `Clausal_Calculus.thy`). In particular, we proved that entailment of ground clauses is compact and tailored the notion of counterexample-reducing inference system as follows:

$$\begin{array}{l}
\textbf{fixes}\ \mathsf{J\_of} :: {'a}\ \mathit{clause\ set} \Rightarrow {'a}\ \mathit{set} \\
\textbf{assumes} \\
\quad \bot \notin N \Longrightarrow D \in N \Longrightarrow \neg\, \mathsf{J\_of}\ N \boldsymbol{\models} D \Longrightarrow \\
\quad (\bigwedge C.\ C \in N \Longrightarrow \neg\, \mathsf{J\_of}\ N \boldsymbol{\models} C \Longrightarrow D \leq C) \Longrightarrow \\
\quad \exists \iota \in \mathsf{Inf}.\ \mathsf{prems\_of}\ \iota \neq [] \wedge \mathsf{main\_prem\_of}\ \iota = D \\
\qquad\qquad\quad \wedge\ \mathsf{side\_prems\_of}\ \iota \subseteq N \\
\qquad\qquad\quad \wedge\ \mathsf{J\_of}\ N \boldsymbol{\models} \mathsf{side\_prems\_of}\ \iota \\
\qquad\qquad\quad \wedge\ \neg\, \mathsf{J\_of}\ N \boldsymbol{\models} \mathsf{concl\_of}\ \iota \wedge \mathsf{concl\_of}\ \iota < D
\end{array}$$

For this locale, we were able to represent a candidate model more conventionally as a set of true atoms and to use the modelhood relation $\boldsymbol{\models}$ instead of entailment $\models$ of clauses.

We connected the two formulations of counterexample reduction as a convenience for end users. The abstract locale was instantiated by taking

$$\mathsf{I\_of}\ N = \{\{\text{if } A \in \mathsf{J\_of}\ N \text{ then Pos } A \text{ else Neg } A\} \mid A :: {'a}\}$$

We retrieved the following completeness theorem tuned for clauses: $\mathsf{saturated}\ N \Longrightarrow \bot \notin N \Longrightarrow \mathsf{J\_of}\ N \boldsymbol{\models} N$.

3 Lifting Ground Calculi

The saturation framework supports a variety of approaches to lift ground calculi to the nonground level (in `Lifting_to_Non_Ground_Calculi.thy` and `Labeled_Lifting_to_Non_Ground_Calculi.thy`). We start with the folklore approach and introduce features incrementally until the result is flexible enough to support the calculi of realistic provers.

3.1 Standard Lifting

The standard lifting is based on a pair of grounding functions $\mathcal{G}_\mathsf{F}$ and $\mathcal{G}_\mathsf{I}$ that link nonground formulas (of some type $'f$) to sets of ground formulas (of some type $'g$) and similarly for nonground and ground inferences. Using these functions, it connects a nonground inference system to a ground calculus:

```
locale standard_lifting =
    inference_system $\mathsf{Inf_F}$ +
    calculus $\mathsf{Bot_G}$ $\mathsf{Inf_G}$ $(\models)$ $\mathsf{Red_{IG}}$ $\mathsf{Red_{FG}}$ +
  fixes
    $\mathsf{Bot_F} :: {'f}\ set$ and
    $\mathcal{G}_\mathsf{F} :: {'f} \Rightarrow {'g}\ set$ and
    $\mathcal{G}_\mathsf{I} :: {'f}\ inference \Rightarrow {'g}\ inference\ set\ option$
  assumes
    $\mathsf{Bot_F} \neq \emptyset$ and
    $B \in \mathsf{Bot_F} \Longrightarrow \mathcal{G}_\mathsf{F}\ B \neq \emptyset$ and
    $B \in \mathsf{Bot_F} \Longrightarrow \mathcal{G}_\mathsf{F}\ B \subseteq \mathsf{Bot_G}$ and
    $\mathcal{G}_\mathsf{F}\ C \cap \mathsf{Bot_G} \neq \emptyset \longrightarrow C \in \mathsf{Bot_F}$ and
    $\iota \in \mathsf{Inf_F} \Longrightarrow \mathcal{G}_\mathsf{I}\ \iota \neq \mathsf{None} \Longrightarrow$
      $\mathsf{the}\ (\mathcal{G}_\mathsf{I}\ \iota) \subseteq \mathsf{Red_{IG}}\ (\mathcal{G}_\mathsf{F}\ (\mathsf{concl\_of}\ \iota))$
```

An instance of the locale for first-order logic would have $\mathcal{G}_\mathsf{F}$ map the nonground formula $\mathrm{p}(x)$ to a Herbrand set such as $\{\mathrm{p(a)}, \mathrm{p(b)}, \ldots, \mathrm{p(f(a))}, \mathrm{p(f(b))}, \ldots\}$. As for $\mathcal{G}_\mathsf{I}$, it maps a nonground inference ι to either a set of ground inferences (accessible via the) or the special value None, which indicates that ι has no counterpart at the ground level.

Entailment of sets of nonground formulas is defined via grounding: $N_1 \models_\mathcal{G} N_2 \longleftrightarrow \mathcal{G}_\mathsf{Fset}\ N_1 \models \mathcal{G}_\mathsf{Fset}\ N_2$, where $\mathcal{G}_\mathsf{Fset} :: {'f}\ set \Rightarrow {'g}\ set$ is the lifting of $\mathcal{G}_\mathsf{F}$ to sets. This relation is called Herbrand entailment. For first-order logic, it coincides with the familiar Tarski entailment when $N_2 = \{\bot\}$, allowing us to use $\models_\mathcal{G}$ in a refutational setting. For example, $\{\mathrm{p}(x), \neg\, \mathrm{p(a)}\} \models_\mathcal{G} \{\bot\}$.

Using these notions, we extended the nonground inference system $\mathsf{Inf_F}$ with a lifted redundancy criterion to form a nonground calculus:

```
definition $\mathsf{Red_I} :: {'f}\ set \Rightarrow {'f}\ inference\ set$ where
  $\mathsf{Red_I}\ N = \{\iota \in \mathsf{Inf_F}.$
    $(\mathcal{G}_\mathsf{I}\ \iota \neq \mathsf{None} \wedge \mathsf{the}\ (\mathcal{G}_\mathsf{I}\ \iota) \subseteq \mathsf{Red_{IG}}\ (\mathcal{G}_\mathsf{Fset}\ N))$
    $\vee\ (\mathcal{G}_\mathsf{I}\ \iota = \mathsf{None}$
      $\wedge\ \mathcal{G}_\mathsf{F}\ (\mathsf{concl\_of}\ \iota) \subseteq \mathcal{G}_\mathsf{Fset}\ N \cup \mathsf{Red_{FG}}\ (\mathcal{G}_\mathsf{Fset}\ N))\}$

definition $\mathsf{Red_F} :: {'f}\ set \Rightarrow {'f}\ set$ where
  $\mathsf{Red_F}\ N = \{C.\ \forall D \in \mathcal{G}_\mathsf{F}\ C.\ D \in \mathsf{Red_{FG}}\ (\mathcal{G}_\mathsf{Fset}\ N)\}$

sublocale calculus $\mathsf{Bot_F}$ $\mathsf{Inf_F}$ $(\models_\mathcal{G})$ $\mathsf{Red_I}$ $\mathsf{Red_F}$
```

This brings us to our first main result. If the ground calculus is statically complete and overapproximated by the nonground calculus, then the latter is statically complete as well:

```
theorem stat_ref_comp_to_non_ground:
  assumes
    statically_complete_calculus $\mathsf{Bot_G}$ $\mathsf{Inf_G}$ $(\models)$
      $\mathsf{Red_{IG}}$ $\mathsf{Red_{FG}}$ and
    $\bigwedge N.$ saturated $N \Longrightarrow$
      ground_Inf_overapproximated $N$
  shows
    statically_complete_calculus $\mathsf{Bot_F}$ $\mathsf{Inf_F}$ $(\models_\mathcal{G})$
      $\mathsf{Red_I}$ $\mathsf{Red_F}$
```

3.2 Adding Tiebreaker Orders

The completeness proofs for standard superposition and many other calculi are organized as a standard lifting. However, the lifted redundancy criterion cannot be used to justify the use of clause subsumption in provers. This points to a gap between theory and practice.

The saturation framework includes a generalization of standard lifting that bridges this gap. It relies on a well-founded order, or more generally a family of orders, called tiebreaker. The tiebreaker can be instantiated with subsumption. In a given clause procedure, it can also be used to make a passive formula redundant w.r.t. the active version of the same formula.

The tiebreaker lifting is defined as an extension of standard lifting with a family $\mathsf{Prec_{Fg}}$ of well-founded orders:

```
locale tiebreaker_lifting =
    empty_ord: standard_lifting $\mathsf{Inf_F}$ $\mathsf{Bot_G}$ $\mathsf{Inf_G}$ $(\models)$
      $\mathsf{Red_{IG}}$ $\mathsf{Red_{FG}}$ $\mathsf{Bot_F}$ $\mathcal{G}_\mathsf{F}$ $\mathcal{G}_\mathsf{I}$ +
  fixes $\mathsf{Prec_{Fg}} :: {'g} \Rightarrow {'f} \Rightarrow {'f} \Rightarrow bool$
  assumes minimal_element $(\mathsf{Prec_{Fg}}\ g)$ UNIV
```

Redundancy of inferences $\mathsf{Red_I}$ is as for the standard lifting, but redundancy of formulas can use the tiebreaker:

```
definition $\mathsf{Red_F} :: {'f}\ set \Rightarrow {'f}\ set$ where
  $\mathsf{Red_F}\ N = \{C.\ \forall D \in \mathcal{G}_\mathsf{F}\ C.\ D \in \mathsf{Red_{FG}}\ (\mathcal{G}_\mathsf{Fset}\ N)$
    $\vee\ \exists E \in N.\ \mathsf{Prec_{Fg}}\ D\ E\ C \wedge D \in \mathcal{G}_\mathsf{F}\ E\}$
```

Based on this definition, we proved that tiebreaker_lifting produces a nonground calculus.

The pen-and-paper development states that standard_lifting produces a nonground calculus but points to the more general tiebreaker_lifting locale for a proof. To follow this structure, we interpreted standard_lifting as an instance of tiebreaker_lifting by taking $\mathsf{Prec_{Fg}} := \lambda g\ C\ C'.\ \mathsf{False}$, the empty tiebreaker family.

The tiebreaker order can be ignored when proving static completeness and recovered for dynamic completeness. Using the above construction, it was easy to prove formally:

```
theorem static_to_dynamic:
  statically_complete_calculus $\mathsf{Bot_F}$ $\mathsf{Inf_F}$ $(\models_\mathcal{G})$
    $\mathsf{Red_I}$ empty_ord.$\mathsf{Red_F}$ $\longleftrightarrow$
  dynamically_complete_calculus $\mathsf{Bot_F}$ $\mathsf{Inf_F}$ $(\models_\mathcal{G})$
    $\mathsf{Red_I}$ $\mathsf{Red_F}$
```

The theorem holds because the definitions of saturation, fairness, and completeness depend on $\mathsf{Red_I}$ but not on $\mathsf{Red_F}$, and because static and dynamic completeness are equivalent.

3.3 Intersection of Liftings

Ground calculi defined via the intersection of redundancy criteria can be lifted using the same general approach. The locale declaration is as follows:

locale lifting_intersection =
    inference_system Inf$_F$ +
    inference_system_family Q Inf$_{Gq}$ +
    consequence_relation_family Bot$_G$ Q entails$_q$ +
  fixes
    Bot$_F$ :: *'f set* **and**
    $\mathcal{G}_{Fq}$ :: *'q* ⇒ *'f* ⇒ *'g set* **and**
    $\mathcal{G}_{Iq}$ :: *'q* ⇒ *'f inference* ⇒ *'g inference set option* **and**
    Prec$_{Fg}$ :: *'g* ⇒ *'f* ⇒ *'f* ⇒ *bool*
  assumes
    $\forall q \in$ Q. tiebreaker_lifting Bot$_F$ Inf$_F$ Bot$_G$
      (entails$_q$ q) (Inf$_{Gq}$ q) (Red$_{Iq}$ q) (Red$_{Fq}$ q)
      ($\mathcal{G}_{Fq}$ q) ($\mathcal{G}_{Iq}$ q) Prec$_{Fg}$

For each index, we defined redundancy with and without tiebreaker order (Red$_{I\mathcal{G}q}$ and either Red$_{F\mathcal{G}q}$ or Red$_{F\mathcal{G}q\emptyset}$) and entailment (entails$_{\mathcal{G}q}$) such that the intersection of the corresponding nonground calculi are also calculi:

sublocale intersection_calculus Bot$_F$ Inf$_F$ Q entails$_{\mathcal{G}q}$
  Red$_{I\mathcal{G}q}$ Red$_{F\mathcal{G}q}$

sublocale *empty_ord*: intersection_calculus Bot$_F$ Inf$_F$
  Q entails$_{\mathcal{G}q}$ Red$_{I\mathcal{G}q}$ Red$_{F\mathcal{G}q\emptyset}$

These commands introduce intersected notions of entailment ($\models_{\cap\mathcal{G}}$) and redundancies (Red$_{I\mathcal{G}}$ and either Red$_{F\mathcal{G}}$ or empty_ord.Red$_{F\mathcal{G}}$).

As in standard_lifting, the static completeness of the lifted intersection follows from that of the members of the ground family, and as in tiebreaker_lifting, the tiebreaker order can be ignored for static completeness:

theorem *stat_eq_dyn_ref_comp_fam_inter*:
  statically_complete_calculus Bot$_F$ Inf$_F$ ($\models_{\cap\mathcal{G}}$)
    Red$_I$ empty_ord.Red$_{F\mathcal{G}}$ ⟷
  dynamically_complete_calculus Bot$_F$ Inf$_F$ ($\models_{\cap\mathcal{G}}$)
    Red$_I$ Red$_F$

Although the proofs are theoretically easy and close to their informal counterparts, their formalization was complicated by the stack of locales and required tedious definition unfolding sequences and **interpret** commands to instantiate locales in the middle of proofs.

3.4 Adding Labels

The last kind of lifting attaches labels to formulas. These labels can be used to partition a set of formulas into its passive and active subsets. Finer partitions are also possible. The labels do not influence the formulas’ semantics.

The locale extends tiebreaker_lifting with a set of labeled inferences Inf$_{FL}$ such that whenever an Inf$_F$ inference is possible, an Inf$_{FL}$ is also possible, and vice versa:

locale labeled_tiebreaker_lifting =
    no_labels: tiebreaker_lifting Bot$_F$ Inf$_F$ Bot$_G$ ($\models$)
      Inf$_G$ Red$_{IG}$ Red$_{FG}$ $\mathcal{G}_F$ $\mathcal{G}_I$ Prec$_{Fg}$ +
  fixes Inf$_{FL}$ :: ($'f \times 'l$) *inference set*
  assumes
    $\iota_F \in$ Inf$_F$ ⟹ length Ls = length (prems_of ι_F) ⟹
      $\exists L_0$. Infer (zip (prems_of ι_F) Ls)
        (concl_of ι_F, L_0) $\in$ Inf$_{FL}$ **and**
    $\iota_{FL} \in$ Inf$_{FL}$ ⟹ Infer (map fst (prems_of ι_{FL}))
      (fst (concl_of ι_{FL})) $\in$ Inf$_F$

Bot$_{FL}$ :: ($'f \times 'l$) *set* is defined as Bot$_F$ × UNIV. The grounding functions ($\mathcal{G}_{IL}$ and $\mathcal{G}_{FL}$) and the entailment relation ($\models_{\mathcal{G}_L}$) simply ignore labels. The main result lets us derive the static completeness of the labeled calculus from the static completeness of the base calculus:

lemma *stat_ref_comp_to_labeled_sta_ref_comp*:
  statically_complete_calculus Bot$_F$ Inf$_F$ ($\models_{\mathcal{G}}$)
    no_labels.Red$_I$ no_labels.Red$_F$ ⟹
  statically_complete_calculus Bot$_{FL}$ Inf$_{FL}$ ($\models_{\mathcal{G}_L}$)
    Red$_I$ Red$_F$

Labels can also be added to calculi that work with families of redundancy criteria. Since completeness results are independent of the tiebreaker order, we only considered the empty tiebreaker order:

locale labeled_lifting_intersection =
    no_labels: lifting_intersection Inf$_F$ Bot$_G$ Q Inf$_{Gq}$
      entails$_q$ Red$_{Iq}$ Red$_{Fq}$ Bot$_F$ $\mathcal{G}_{Fq}$ $\mathcal{G}_{Iq}$
      ($\lambda g\ CL\ CL'$. False) +
  fixes Inf$_{FL}$ :: ($'f \times 'l$) *inference set*

We proved that static completeness with labels can be reduced to static completeness without labels.

4 Prover Architectures

Dynamic completeness is formulated in terms of an abstract single-formula-set prover ⊳ (Section 2.4). In practice, saturation provers implement the given clause procedure described in the introduction. This simplifies bookkeeping: Instead of having to keep track of all possible n-ary inferences, actual provers only need to track active formulas.

The saturation framework includes two versions of the given clause procedure: an eager version called GC and a lazy version called LGC. LGC strictly generalizes GC, but because GC is sufficient for most applications and simpler to use, we formalized both separately (in `Given_Clause_Architectures.thy` and `Given_Clause_Architectures_Revisited.thy`).

4.1 Basic Setup

Much of the setup between GC and LGC can be shared, in a locale called given_clause_basis that extends labeled_lifting_intersection. The locale assumes that the unlabeled version of the intersection calculus is complete:

assumes
statically_complete_calculus $\mathsf{Bot_F}$ $\mathsf{Inf_F}$ $(\models_{\cap\mathcal{G}})$
no_labels.$\mathsf{Red_I}$ no_labels.$\mathsf{Red_F}$

It introduces an equivalence relation $\doteq$ and an order $\prec\cdot$ on unlabeled formulas that must be compatible with each other and with the grounding function:

assumes
$C_1 \doteq D_1 \Longrightarrow C_2 \doteq D_2 \Longrightarrow C_1 \prec\cdot\, C_2 \Longrightarrow D_1 \prec\cdot\, D_2$ **and**
$q \in \mathrm{Q} \Longrightarrow C1 \doteq C2 \Longrightarrow \mathcal{G}_{\mathsf{Fq}}\, q\, C_1 \Longrightarrow \mathcal{G}_{\mathsf{Fq}}\, q\, C_2$ **and**
$q \in \mathrm{Q} \Longrightarrow C2 \prec\cdot\, C1 \Longrightarrow \mathcal{G}_{\mathsf{Fq}}\, q\, C_1 \Longrightarrow \mathcal{G}_{\mathsf{Fq}}\, q\, C_2$

In practice, $\doteq$ would typically be α-equivalence, and $\prec\cdot$ would be clause subsumption.

The locale also introduces an order $\sqsubset_\mathsf{L}$ on labels and a distinguished label active that must be minimal w.r.t. $\sqsubset_\mathsf{L}$. Both $\prec\cdot$ and $\sqsubset_\mathsf{L}$ are assumed to be well founded:

assumes
minimal_element $(\prec\cdot)$ UNIV **and**
minimal_element $(\sqsubset_\mathsf{L})$ UNIV

Inside the locale, we used $\prec\cdot$ and $\sqsubset_\mathsf{L}$ to define a tiebreaker order $\sqsubset$ on labeled formulas and showed it to be well founded:

definition $\sqsubset$ :: $({}'f \times {}'l) \Rightarrow ({}'f \times {}'l) \Rightarrow$ *bool* **where**
$CL_1 \sqsubset CL_2 \longleftrightarrow \mathsf{fst}\ CL_1 \prec\cdot\, \mathsf{fst}\ CL_2$
$\vee\ (\mathsf{fst}\ CL_1 \doteq \mathsf{fst}\ CL_2 \wedge \mathsf{snd}\ CL_1 \sqsubset_\mathsf{L} \mathsf{snd}\ CL_2)$

lemma *wf_prec_FL*: minimal_element $(\sqsubset)$ UNIV

This order can be used to strengthen the redundancy criterion of the labeled intersection calculus. For example, since $(\mathrm{p}(x), \mathsf{active}) \sqsubset (\mathrm{p(a)}, \mathsf{active})$ and $(C, \mathsf{active}) \sqsubset (C, \mathsf{passive})$, for each pair the first labeled formula makes the second one redundant. Below, $\mathsf{Red_I}$ and $\mathsf{Red_F}$ refer to the strengthened criterion.

Finally, we present two lemmas that make explicit the relation between the labeled and unlabeled redundancy criteria:

lemma *labeled_red_inf_eq_red_inf*:
$\iota \in \mathsf{Inf_{FL}} \Longrightarrow \iota \in \mathsf{Red_I}\ N \longleftrightarrow$
to_F $\iota \in$ no_labels.$\mathsf{Red_I}$ (fst ‘ N)

lemma *red_labeled_clauses*:
$C \in$ no_labels.$\mathsf{Red_F}$ (fst ‘ N) $\vee\ (\exists C' \in \mathsf{fst}$ ‘ $N.\ C' \prec\cdot\, C)$
$\vee\ (\exists (C', L') \in N.\ L' \sqsubset_\mathsf{L} L \wedge C' \prec\cdot\, C) \Longrightarrow$
$(C, L) \in \mathsf{Red_F}\ N$

Above, to_F denotes a function that erases all labels, and the infix operator ‘ denotes the image of a set under a function.

Although the first lemma is obvious when we compare the definitions of $\mathsf{Red_I}$ and no_labels.$\mathsf{Red_I}$, the Isabelle proof is over 100 lines long. This is due mostly to the deep nesting of locales. The identity of notions is hidden under multiple layers of names, and Isabelle provides no proof automation that can unfold these definitions so as to reveal such connections. We conjecture that proof assistants based on computational type theories such as Agda, Coq, Lean, and Matita would cope more gracefully with these definitional equalities.

The given_clause_basis locale also ensures that inferences never produce active formulas. Only the procedure itself is allowed to make a formula active.

When we developed this locale and the two locales based on it, we did not immediately try to instantiate them. This led to an amusing incident when we attempted to verify RP. The given_clause_basis locale axioms turned out to be flawed. First, we had stated that $\cdot\succ$ must be well founded when we meant $\prec\cdot$. Strangely enough, we had successfully proved $\sqsubset$ well founded, even though it is based on $\prec\cdot$.

How could that be? The answer is that we had relied on very powerful proof automation: Sledgehammer [19]. That tool had discovered *another* flawed assumption and exploited it. This time, we had misspelled a constant's name. In Isabelle, this creates an implicitly universal variable. This convention, which certainly saves a lot of typing and is considered by some as an advantage of Isabelle over its rivals, had led to an inconsistent assumption.

4.2 The Eager Given Clause Procedure

The eager given clause procedure GC is described as a transition system that operates on labeled formula sets. This corresponds naturally to an inductive predicate $\rightsquigarrow_\mathrm{GC}$:

inductive $\rightsquigarrow_\mathrm{GC}$ :: $({}'f \times {}'l)$ *set* $\Rightarrow ({}'f \times {}'l)$ *set* $\Rightarrow$ *bool*
where
process: $N_1 = N \cup M \Longrightarrow N_2 = N \cup M' \Longrightarrow$
$M \subseteq \mathsf{Red_F}\ (N \cup M') \Longrightarrow$ active_subset $M' = \emptyset \Longrightarrow$
$N_1 \rightsquigarrow_\mathrm{GC} N_2$
| *infer*: $N_1 = N \cup \{(C, L)\} \Longrightarrow$
$N_2 = N \cup \{(C, \mathsf{active})\} \cup M \Longrightarrow L \neq \mathsf{active} \Longrightarrow$
active_subset $M = \emptyset \Longrightarrow$
no_labels.Inf_between (fst ‘ active_subset N) $\{C\}$
$\subseteq$ no_labels.$\mathsf{Red_I}$ (fst ‘ $(N \cup (C, \mathsf{active}) \cup M)) \Longrightarrow$
$N_1 \rightsquigarrow_\mathrm{GC} N_2$

The *process* rule can be used to add arbitrary passive formulas M' or to remove redundant formulas M. For example, the simplification of $\mathrm{p}(2+2)$ to $\mathrm{p}(4)$ would be modeled by taking $M := \{\mathrm{p}(2+2)\}$ and $M' := \{\mathrm{p}(4)\}$. The *infer* rule selects a passive formula C—called the given formula or given clause—and makes it active. In addition, it draws all inferences between C and all the active formulas; more precisely, the rule requires the addition of enough passive formulas M to make these inferences redundant. The locale assumes that the inference system $\mathsf{Inf_F}$ contains no nullary inferences.

The main benefit of using GC is that it makes it easier to draw inferences fairly. We only need to ensure that the prover eventually makes every passive formula active.

Via a refinement step, we showed that $\rightsquigarrow_\mathrm{GC}$-transitions correspond to $\rhd$-transitions. This was straightforward. Then we needed to connect GC's notion of fairness with the fair predicate on $\rhd$-derivations (Section 2.4). Fairness for GC means that no formulas are passive at the limit, starting from a state in which no formulas are active:

lemma *gc_fair*:
assumes
chain $(\leadsto_{\mathsf{GC}})$ Ns **and**
active_subset $(Ns \,!\, 0) = \emptyset$ **and**
passive_subset (Liminf Ns) $= \emptyset$
shows fair Ns

The informal proof is 13 lines long, but we needed 197 lines of Isabelle. The explanation for this poor De Bruijn factor is that the proof considers the maximal element of a set of integers, each representing a step in the derivation that is the first to satisfy a complex property for one of the premises of a given inference. The previous sentence already hints at the number of hidden reasoning steps necessary to obtain this particular element, although it is fairly easy to write the notion mathematically in a way that enables readers to build an adequate mental representation of the object. In Isabelle, however, every intermediate object in this construction must be shown to exist.

For each premise, the existence of the particular step of the derivation that verifies the property must be proved. This alone accounts for almost half of the proof. Then a set is built of those integers that must be proved nonempty and finite, before the maximal element can finally be obtained. The other half of the proof is dominated by the need to reason by elimination to identify that this particular step must be an *infer* step. This is obvious on paper by inspection of the transition rules but requires detailed reasoning in Isabelle.

We were not satisfied with this proof. A proof that is tedious to write is also tedious to read, and hides its insights from the reader. The situation reminded us of a 700-line Isabelle formalization by Fleury of an 8-line pen-and-paper proof by Weidenbach [11, Section 4.3]. The solution in that case was to reason by invariance.

Can invariance help prove *gc_fair* in a more comprehensible way? It is well established that the given clause procedure satisfies the following key invariant:

> All inferences from active formulas are redundant w.r.t. the current set of formulas.

Initially, all formulas are passive, so the invariant holds vacuously. Whenever a given formula is made active, all inferences with active partners are made redundant, thereby maintaining the invariant. And if all formulas are active at the limit, then this means that all inferences from the entire limit N are redundant. Hence the derivation is fair and consequently N is saturated.

Paradoxically, although the invariant above is widely recognized as an important property of the given clause procedure, which is itself central in automated reasoning, we are not aware of any proofs in the literature that rely on it. Instead, the invariant is presented as a guide to intuition, if at all. For example, Bachmair and Ganzinger do not mention the invariant in their *Handbook* chapter.

In Isabelle, we defined the invariant as follows:

definition gc_invar :: $(\mathit{'f} \times \mathit{'l})$ *set llist* $\Rightarrow$ *enat* $\Rightarrow$ *bool*
where
gc_invar $Ns\ i \longleftrightarrow$
Inf_from (active_subset (Liminf_upto $Ns\ i$))
$\subseteq$ Sup_upto (lmap $\mathsf{Red}_{\mathsf{I}\mathcal{G}}$ Ns) i

The predicate gc_invar $Ns\ i$ determines whether the invariant holds at index i (which can be ∞). If k is within bounds, Liminf_upto $Ns\ k$ is defined as the bounded limit inferior $\bigcup_i \bigcap_{j=i}^{k} Ns \,!\, j$, which is equal to the limit if $k = \infty$ and to $Ns \,!\, k$ otherwise. Similarly, Sup_upto $Ns\ k$ is the bounded supremum $\bigcup_i^k Ns \,!\, i$.

The invariant holds for all derivations with a reasonable initial state, and it extends to the limit:

lemma *gc_invar_gc*:
chain $(\leadsto_{\mathsf{GC}})$ $Ns \Longrightarrow$ active_subset $(Ns \,!\, 0) = \emptyset \Longrightarrow$
$i <$ llength $Ns \Longrightarrow$ gc_invar $Ns\ i$

lemma *gc_invar_infinity*:
$Ns \neq [\,] \Longrightarrow (\forall i <$ llength $Ns.$ gc_invar $Ns\ i) \Longrightarrow$
gc_invar $Ns\ \infty$

Assuming that all formulas eventually become active, the invariant at the limit simplifies to Inf_from (Liminf Ns) $\subseteq$ Sup (lmap $\mathsf{Red}_{\mathsf{I}\mathcal{G}}$ Ns). This corresponds exactly to the definition of fairness, which is what we want.

The new formal proof is not much shorter in terms of number of lines (166 vs. 197), but it is much more compact horizontally and about half the size in bytes (7 kB vs. 13 kB). Written up as an informal proof, however, it is longer than the original; specifying the invariant and stating intermediate lemmas takes up vertical space. Even so, the new proof is more lucid, and we plan to include it in the journal version of Waldmann et al.

4.3 The Lazy Given Clause Procedure

The lazy given clause procedure LGC decouples inference scheduling and computation. Each inference ι is first added to a “to do” set T of scheduled inferences. At some later point, ι is computed, meaning that it is removed from T and its conclusion is added to the set N of labeled formulas. Alternatively, ι might be identified as redundant and deleted, a technique called orphan formula deletion. The formal definition of LGC follows:

inductive $\leadsto_{\mathsf{LGC}}$ :: *'f inference set* $\times$ $(\mathit{'f} \times \mathit{'l})$ *set* $\Rightarrow$
'f inference set $\times$ $(\mathit{'f} \times \mathit{'l})$ *set* $\Rightarrow$ *bool*
where
process: $N_1 = N \cup M \Longrightarrow N_2 = N \cup M' \Longrightarrow$
$M \subseteq \mathsf{Red}_{\mathsf{F}}\ (N \cup M') \Longrightarrow$ active_subset $M' = \emptyset \Longrightarrow$
$(T, N_1) \leadsto_{\mathsf{LGC}} (T, N_2)$
| *schedule_infer*: $T_2 = T_1 \cup T' \Longrightarrow N_1 = N \cup \{(C, L)\} \Longrightarrow$
$N_2 = N \cup \{(C, \mathsf{active})\} \Longrightarrow L \neq \mathsf{active} \Longrightarrow$
$T' =$ no_labels.Inf_between
(fst ‘ active_subset N) $\{C\} \Longrightarrow$
$(T_1, N_1) \leadsto_{\mathsf{LGC}} (T_2, N_2)$

| *compute_infer*: $T_1 = T_2 \cup \{\iota\} \Longrightarrow N_2 = N_1 \cup M \Longrightarrow$
active_subset $M = \emptyset \Longrightarrow$
$\iota \in$ no_labels.Red$_{\mathsf{I}}$ (fst ‘ $(N_1 \cup M)$) $\Longrightarrow$
$(T_1, N_1) \rightsquigarrow_{\mathsf{LGC}} (T_2, N_2)$
| *delete_orphan_fmlas*: $T_1 = T_2 \cup T' \Longrightarrow T' \cap$
no_labels.Inf_from (fst ‘ active_subset N) $= \emptyset \Longrightarrow$
$(T_1, N) \rightsquigarrow_{\mathsf{LGC}} (T_2, N)$

LGC has several benefits beyond orphan formula deletion. The "to do" set T can be used to support nullary inferences, or axioms. It can be used to represent infinite streams of inferences, which might for example result from higher-order unification (as in Zipperposition). Finally, inferences can often be stored more compactly than their conclusions, so LGC can be used to enable a low-level optimization (as in Waldmeister).

The key result is to show that $\rightsquigarrow_{\mathsf{LGC}}$-derivations are fair under some reasonable assumptions:

lemma *lgc_fair*:
assumes
chain ($\rightsquigarrow_{\mathsf{LGC}}$) *TNs* **and**
active_subset (snd (*TNs* ! 0)) = $\emptyset$ **and**
passive_subset (Liminf (lmap snd TNs)) = $\emptyset$ **and**
$\forall\iota \in$ Inf$_{\mathsf{F}}$. prems_of ι = [] $\longrightarrow \iota \in$ fst (*TNs* ! 0) **and**
Liminf (lmap fst *TNs*) = $\emptyset$
shows fair (lmap snd *TNs*)

Compared with $\rightsquigarrow_{\mathsf{GC}}$, an extra condition is added to ensure that all premise-free inferences are scheduled up front. The proof has the same structure as that of *gc_fair*, but the reasoning steps are more complex due to the extra rules and the presence of scheduled inferences. We needed 326 lines.

Can invariance help re-prove *lgc_fair* in the same way that it helped us with *gc_fair*? The lazy given clause procedure is little discussed in the literature, but it is easy to come up with a suitable generalization of the previous invariant:

> All inferences from active formulas are either scheduled in the "to do" set T or redundant w.r.t. the formula set N.

Initially, all formulas are passive, and all nullary inferences are in T, so the invariant holds. Whenever a given formula is made active, all inferences with active partners are scheduled, thereby maintaining the invariant. Removing an inference from T and computing it makes it redundant, preserving the invariant. Orphan formula deletion also preserves the invariant, because by definition an orphan cannot be derived from active formulas. And if T is empty and all formulas are active at the limit, then all inferences from the limit N are redundant, as required by fairness.

We defined the invariant as follows in Isabelle:

definition lgc_invar ::
(*'f inference set* $\times$ (*'f* $\times$ *'l*) *set*) *llist* $\Rightarrow$ *enat* $\Rightarrow$ *bool*
where
lgc_invar *TNs i* $\longleftrightarrow$
Inf_from (active_subset
(Liminf_upto (lmap snd *TNs*) *i*))
$\subseteq$ from_F ‘ Liminf_upto (lmap fst *TNs*) *i*)
$\cup$ Sup_upto (lmap (Red$_{\mathsf{I}\mathcal{G}}$ $\circ$ snd) *TNs*) *i*

The formal proof by invariance resembles that for GC but with more cases to consider. Compared with the monolithic proof, we have a small improvement in number of lines (301 vs. 326) and a more substantial improvement in byte size (13 kB vs. 22 kB). For both GC and LGC, we believe the gain in lucidity is much higher than the numbers suggest.

5 Application to a Resolution Prover

A framework can hardly be called a framework until it is used. Waldmann et al. [29, Example 29] applied their pen-and-paper framework to Bachmair and Ganzinger's resolution prover RP. To validate our formalized framework, a natural option was to apply it to RP as well, or rather RP's formalization by Schlichtkrull et al. We lifted ground static completeness explicitly and re-proved the dynamic completeness of RP. This example uses all the main components of the saturation framework and can serve as a blueprint for verifying other provers.

5.1 The Resolution Prover

Ordered resolution with selection, the calculus underlying RP, works on first-order clauses without equality (e.g., $\mathrm{p}(x) \vee \neg\,\mathrm{q}(\mathrm{b}, \mathrm{f}(\mathrm{a}))$). It is parameterized by a well-order $<$ on ground atoms, which is lifted to literals and clauses, and by a selection function S that maps each clause to one of its subclauses. The ground version of the calculus consists of a single n-ary inference rule:

$$\frac{\left(C_i \vee A_i \vee \cdots \vee A_i\right)_{i=1}^{n} \qquad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{C_1 \vee \cdots \vee C_n \vee D}$$

The duplicate atoms $A_i \vee \cdots \vee A_i$ in the n side premises are needed for completeness [3, Section 3]. Side conditions restrict the applicability of the rule, based on the well-order and the selection function. The order prunes the search space by identifying clauses that stray away from the goal ($\bot$). The selection function guides the search.

The nonground version of the inference rule is

$$\frac{\left(C_i \vee A_{1i} \vee \cdots \vee A_{ik_i}\right)_{i=1}^{n} \qquad \neg A_1 \vee \cdots \vee \neg A_n \vee D}{(C_1 \vee \cdots \vee C_n \vee D)\sigma}$$

where σ is a most general simultaneous solution of all unification problems $A_{i1} \overset{?}{=} \cdots \overset{?}{=} A_{ik_i} \overset{?}{=} A_i$, where $1 \le i \le n$. As in the ground case, further side conditions apply.

RP is defined as a transition system $\rightsquigarrow_{\mathsf{RP}}$ over states of the form $\mathcal{S} = (N, P, O)$, where N contains the "new" clauses, P contains the "processed" clauses, and O contains the "old" clauses. RP implements a variant of the given clause procedure, with $N \cup P$ as the passive set and O as the active set. RP is modeled naturally as an inductive predicate in Isabelle, with nine introduction rules:

inductive $\leadsto_{\mathsf{RP}}$ :: *'a state ⇒ 'a state ⇒ bool* **where**
$\{\mathsf{Neg}\ A, \mathsf{Pos}\ A\} \subseteq C \Longrightarrow (N \cup \{C\}, P, O) \leadsto_{\mathsf{RP}} (N, P, O)$
$\mid D \in P \cup O \Longrightarrow \mathsf{subsumes}\ D\ C \Longrightarrow$
$(N \cup \{C\}, P, O) \leadsto_{\mathsf{RP}} (N, P, O)$
$\mid D \in N \Longrightarrow \mathsf{strict\_subsumes}\ D\ C \Longrightarrow$
$(N, P \cup \{C\}, O) \leadsto_{\mathsf{RP}} (N, P, O)$
$\mid D \in N \Longrightarrow \mathsf{strict\_subsumes}\ D\ C \Longrightarrow$
$(N, P, O \cup \{C\}) \leadsto_{\mathsf{RP}} (N, P, O)$
$\mid D \in P \cup O \Longrightarrow \mathsf{reduces}\ D\ C\ L \Longrightarrow$
$(N \cup \{C \uplus \{L\}\}, P, O) \leadsto_{\mathsf{RP}} (N \cup \{C\}, P, O)$
$\mid D \in N \Longrightarrow \mathsf{reduces}\ D\ C\ L \Longrightarrow$
$(N, P \cup \{C \uplus \{L\}\}, O) \leadsto_{\mathsf{RP}} (N, P \cup \{C\}, O)$
$\mid D \in N \Longrightarrow \mathsf{reduces}\ D\ C\ L \Longrightarrow$
$(N, P, O \cup \{C \uplus \{L\}\}) \leadsto_{\mathsf{RP}} (N, P \cup \{C\}, O)$
$\mid (N \cup \{C\}, P, O) \leadsto_{\mathsf{RP}} (N, P \cup \{C\}, O)$
$\mid (\emptyset, P \cup \{C\}, O) \leadsto_{\mathsf{RP}}$
$(\mathsf{concl\_of}\ \text{`}\ \mathsf{Inf\_between}\ O\ \{C\}, P, O \cup \{C\})$

The first seven rules perform optional clause processing steps: tautology elimination, clause subsumption, and clause reduction (i.e., removal of needless literals). The next-to-last rule moves a clause C from N to P, and the last rule moves C further from P to O and computes all possible inferences between C and partners from O.

A sequence of states $\mathcal{S}s$ is called fair, written fair $\mathcal{S}s$, if $\mathsf{N\_of}\ (\mathsf{Liminf}\ \mathcal{S}s) = \mathsf{P\_of}\ (\mathsf{Liminf}\ \mathcal{S}s) = \emptyset$. The completeness theorem for RP can be stated as follows:

theorem *RP_complete_if_fair*:
assumes
chain $(\leadsto_{\mathsf{RP}})\ \mathcal{S}s$ **and**
$\mathsf{O\_of}\ (\mathcal{S}s\ !\ 0) = \emptyset$ **and**
fair $\mathcal{S}s$ **and**
$\forall I.\ \neg\ I \models \mathcal{G}_{\mathsf{S}}\ (\mathcal{S}s\ !\ 0)$
shows $\bot \in \mathsf{O\_of}\ (\mathsf{Liminf}\ \mathcal{S}s)$

Informally, if we start in a state where all clauses are passive (i.e., $O = \emptyset$) and make sure that all clauses are active (i.e., $N = P = \emptyset$) at the limit, and the initial clause set is unsatisfiable, then $\bot$ will belong to the limit. This, in turn, means that $\bot$ is derived after finitely many transitions—ideally before the prover runs out of time and the user, of patience.

The following RP derivation [21, Section 3], starting from an unsatisfiable clause set, illustrates what can go wrong without fairness:

$$\begin{aligned}
& (\{\neg\,\mathsf{p}(\mathsf{a},\mathsf{a}),\ \mathsf{p}(x,x),\ \neg\,\mathsf{p}(\mathsf{f}(x),y) \vee \mathsf{p}(x,y)\}, \emptyset, \emptyset) \\
\leadsto^{+}_{\mathsf{RP}}\ & (\emptyset, \{\neg\,\mathsf{p}(\mathsf{a},\mathsf{a}),\ \mathsf{p}(x,x),\ \neg\,\mathsf{p}(\mathsf{f}(x),y) \vee \mathsf{p}(x,y)\}, \emptyset) \\
\leadsto_{\mathsf{RP}}\ & (\emptyset, \{\neg\,\mathsf{p}(\mathsf{a},\mathsf{a}),\ \mathsf{p}(x,x)\}, \{\neg\,\mathsf{p}(\mathsf{f}(x),y) \vee \mathsf{p}(x,y)\}) \\
\leadsto_{\mathsf{RP}}\ & (\{\mathsf{p}(x,\mathsf{f}(x))\}, \{\neg\,\mathsf{p}(\mathsf{a},\mathsf{a})\}, \\
& \quad \{\neg\,\mathsf{p}(\mathsf{f}(x),y) \vee \mathsf{p}(x,y),\ \mathsf{p}(x,x)\}) \\
\leadsto_{\mathsf{RP}}\ & (\emptyset, \{\neg\,\mathsf{p}(\mathsf{a},\mathsf{a}),\ \mathsf{p}(x,\mathsf{f}(x))\}, \\
& \quad \{\neg\,\mathsf{p}(\mathsf{f}(x),y) \vee \mathsf{p}(x,y),\ \mathsf{p}(x,x)\}) \\
\leadsto_{\mathsf{RP}}\ & (\{\mathsf{p}(x,\mathsf{f}(\mathsf{f}(x)))\}, \{\neg\,\mathsf{p}(\mathsf{a},\mathsf{a})\}, \\
& \quad \{\neg\,\mathsf{p}(\mathsf{f}(x),y) \vee \mathsf{p}(x,y),\ \mathsf{p}(x,x),\ \mathsf{p}(x,\mathsf{f}(x))\}) \\
\leadsto_{\mathsf{RP}}\ & \cdots
\end{aligned}$$

The key to derive $\bot$ is to resolve the initial clauses $\mathsf{p}(x,x)$ and $\neg\,\mathsf{p}(\mathsf{a},\mathsf{a})$. This requires moving both to O. Instead, the above derivation leaves $\neg\,\mathsf{p}(\mathsf{a},\mathsf{a})$ in P forever, focusing instead on generating useless clauses of the form $\mathsf{p}(x, \mathsf{f}^{i}(x))$.

5.2 The Original Completeness Proof

The Isabelle completeness proof by Schlichtkrull et al. (in `FO_Ordered_Resolution.thy` and `FO_Ordered_Resolution_Prover.thy`), based on Bachmair and Ganzinger, consists of the following sequence of steps:

1. An $\leadsto_{\mathsf{RP}}$-transition from $\mathcal{S}$ to $\mathcal{S}'$ corresponds to a $\rhd$-transition from $\mathcal{G}_{\mathsf{S}}\ \mathcal{S}$ to $\mathcal{G}_{\mathsf{S}}\ \mathcal{S}'$, where $\mathcal{G}_{\mathsf{S}}$ returns, for a given state, the set consisting of the $\mathcal{G}_{\mathsf{F}}$-groundings of all the clauses in the state.
2. Let $\mathcal{S}s$ be a fair RP derivation. Then any nonredundant clause C in $\mathcal{S}s$'s ground projection eventually reaches O and stays there.
3. Moreover, if O is initially empty, then the limit of the $\mathcal{G}_{\mathsf{S}}$-grounding of $\mathcal{S}s$ is saturated.
4. As a corollary, since ground resolution is complete, if the initial clause set is unsatisfiable, then $\bot$ occurs in the limit (grounded or not).

It is difficult to relate to these proof steps. Even after the imprecisions in Bachmair and Ganzinger's proofs have been identified and clarified [21, Appendix A], the proof is hard to understand and remember. And yet, it is not very long. In the *Handbook*, it spans four pages. In Isabelle, it amounts to around 2500 lines, but this includes additional minor results. About half of the lines are dedicated to lifting ground inferences and coping with α-equivalence—issues that are only alluded to in the *Handbook*.

5.3 The New Completeness Proof

The new completeness proof (in `FO_Ordered_Resolution_Prover_Revisited.thy`) has a more abstract flavor. It exploits the concepts presented in Sections 2 to 4. It also consists of four steps:

1. Ground resolution is statically complete.
2. Hence nonground resolution is statically complete.
3. Hence a given clause prover GC based on resolution is dynamically complete.
4. Thus, via refinement, RP is dynamically complete.

The four steps are identified by letter codes: G (ground), F (first-order), FL (F with labels), and RP.

For step 1, if sufficed to show that ground resolution G_Inf is a clausal counterexample-reducing inference system (Sections 2.7 and 2.8). The core of the proof is a lemma already provided by Schlichtkrull et al.

For step 2, we defined nonground resolution as an inference system F_Inf. This could be done in one line in terms of the definition by Schlichtkrull et al. We also defined grounding functions: $\mathcal{G}_{\mathsf{F}}$ on clauses and $\mathcal{G}_{\mathsf{I}}$ on inferences.

The grounding of a clause C consists of all ground clauses of the form $C\sigma$; the grounding of an inference $C_n, \ldots, C_1 \vdash C_0$ consists of all the inferences $C_n\sigma_n, \ldots, C_1\sigma_1 \vdash C_0\sigma_0 \in \mathsf{G\_Inf}$.

The main difficulty we encountered concerned the selection function. Recall that resolution is parameterized by a function S that restricts inferences. Inconveniently, S is generally not stable under substitution—it might behave differently on $C\sigma$ and C. Based on the limit set M, we can define an appropriate selection function S_M for the ground level as a modification of S. However, this definition is circular: The limit M depends on the nonground calculus with its selection function S, which in turn is lifted from a ground calculus with its selection function S_M, which depends on M.

Remarkably, this circularity had initially escaped the attention of Waldmann et al. as they worked in LaTeX. It is all too easy to dismiss selection as an orthogonal concern. We noticed the issue only as we formalized RP. Fortunately, it was not too late to adapt the pen-and-paper framework before it was published [29].

The notion of intersection of redundancy criteria (Section 2.5) was designed to break the circularity. Instead of being lifted from S_M, where M is the limit, the nonground calculus is lifted simultaneously from all selection functions of the form S_M, where M is an arbitrary clause set. This was achieved formally by instantiating the lifting_intersection locale (Section 2.5) and showing that the grounding functions $\mathcal{G}_\mathsf{F}$ and $\mathcal{G}_\mathsf{I}$ satisfy basic properties.

We could then instantiate statically_complete_calculus (Section 2.4) to lift the ground calculus to the nonground level. The key lemma states that any inference ι_G from the grounded clauses is approximated at the nonground level by an inference ι_F:

lemma *G_Inf_overapprox_F_Inf*:
  $\iota_\mathsf{G} \in \mathsf{G.\,Inf\_from}\ M\ (\bigcup\,(\mathcal{G}_\mathsf{F}\ \text{`}\ M)) \Longrightarrow$
  $\exists\iota_\mathsf{F} \in \mathsf{F.\,Inf\_from}\ M.\ \iota_\mathsf{G} \in \mathcal{G}_\mathsf{I}\ M\ \iota_\mathsf{F}$

For the core of the proof, we could again reuse a result from Schlichtkrull et al. This illustrates again how the saturation framework takes care of the bureaucracy and allows the user to focus on the calculus-specific reasoning.

Step 3 is about deriving dynamic completeness of a given clause procedure. We needed to define a suitable equivalence relation $\doteq$ and tiebreaker order $\prec\!\!\cdot$ in formulas. We also defined the labels New, Processed, and Old (= active) and defined the order $\sqsubset_\mathsf{L}$ such that New $\sqsupset_\mathsf{L}$ Processed $\sqsupset_\mathsf{L}$ Old. With these definitions in place, we instantiated given_clause and retrieved two main results: (1) the induced $\rightsquigarrow_\mathsf{GC}$ refines $\rhd$ and (2) the induced $\rightsquigarrow_\mathsf{GC}$ is dynamically complete.

Step 4 was essentially a refinement proof: RP's states, which are triples of clause sets, must be converted to GC's labeled clause sets:

definition lclss_of :: *'a state* $\Rightarrow$ (*'a clause* $\times$ *label*) *set* **where**
  $\mathsf{lclss\_of}\ \mathcal{S} = (\lambda C.\ (C, \mathsf{New}))\ \text{`}\ \mathsf{N\_of}\ \mathcal{S}$
  $\cup\ (\lambda C.\ (C, \mathsf{Processed}))\ \text{`}\ \mathsf{P\_of}\ \mathcal{S}$
  $\cup\ (\lambda C.\ (C, \mathsf{Old}))\ \text{`}\ \mathsf{O\_of}\ \mathcal{S}$

The first eight transition rules of RP can be simulated by the *process* rule of GC, and the last rule of RP can be simulated by *infer*. To illustrate this, we will sketch the proof for the first RP rule, which deletes tautologies.

Consider the transition $(N \cup \{C\}, P, O) \rightsquigarrow_\mathsf{RP} (N, P, O)$, where $\{\mathsf{Neg}\ A,\ \mathsf{Pos}\ A\} \subseteq C$. We need to prove that there exist labeled clause sets N', M, M' such that $N' \cup M \rightsquigarrow_\mathsf{GC} N' \cup M'$ by the *process* rule. This amounts to showing that (1) M is redundant w.r.t. $N' \cup M'$ and (2) no clause from M' is labeled active. We take $N' := \mathsf{lclss\_of}\ (N, P, O)$, $M := \{(C, \mathsf{New})\}$, and $M' := \emptyset$. Condition (1) is obvious since C is a tautology, and (2) is vacuously true.

After completing step 4, we had all the results we needed to re-prove the main theorem, *RP_complete_if_fair*—with one exception. To lighten the presentation, we used our notations for basic concepts such as inferences, redundancy, and fairness. However, Schlichtkrull et al. used slightly different formulations, based on Bachmair and Ganzinger. Their notions—prefixed by "old" in our theory files to clearly identify them—are restricted to clauses, which is not an issue for RP, but the side premises of an inference are stored in a multiset and thus unordered. This affects all definitions based on inferences, from redundancy to saturation. We first proved completeness and related results using our concepts; then we restated the results and re-proved them using theirs, as a sanity check. And although we were mostly interested in completeness, we also proved the soundness of RP derivations, exploiting the lemma *unsat_limit_iff* (Section 2.6).

5.4 Discussion

The approach taken for the new proof can be applied with little change to other saturation calculi based on resolution or superposition. For calculi like constraint superposition that cannot be obtained as the lifting of a ground calculus, steps 1 and 2 must be replaced by a monolithic proof that the nonground calculus is complete.

Is the new proof an improvement over the one by Schlichtkrull et al.? It certainly is in terms of lines of source text. Let us ignore soundness and the conversions between basic concepts, and all the definitions and lemmas that are shared between the two proofs, including the definition of RP itself. Then we arrive at around 700 lines for the new proof compared with 1200 lines for the original proof. Among the 700 lines, nearly half are concerned with the refinement from GC to RP. But regardless of the line counts, we are convinced that the new proof's modularity makes it more intelligible and easier to teach, and easier to mimic to formally verify other saturation provers.

Looking at the line counts, one might be led to believe that the new proof was easy to develop. Unfortunately, this was not the case, because we worked with a less mature version of the framework. The circularity issue noted above led to a one-year hiatus, and when we resumed, we still had to find our way across a web of locales. The main difficulty we faced was that several copies of the same locale instances emerged, identical up to unfolding of definitions but distinct as far as the locale machinery is concerned. Instantiating the given_clause locale, which pulls in a wide range of concepts, initially produced around 40 subgoals instead of the 14 we now get. There were so many layers of definitions that Sledgehammer was helpless.

One reason locales became so complicated is that we rigorously followed the informal proofs by reduction. For example, lifting with a tiebreaker order is reduced to lifting with $\emptyset$ as the tiebreaker, which in turn amounts to a standard lifting. A more direct proof would consider only the most general concept and avoid the replication of concepts. Fortunately, we found a way to simplify the locales while keeping compatibility with the informal argument. Often, it sufficed to replace an opaque **definition** by a transparent **abbreviation**, or to choose a canonical locale instance and refrain from using the other definitionally equal instances.

Locales are truly a double-edged sword. On the one hand, they conveniently keep track of parameters and dependencies; without them, we would need to clutter definitions with extra arguments and lemmas with extra assumptions. On the other hand, locales often surprised us and provide too few introspection methods. Two examples: First, when instantiating a locale, we know of no easy way to figure where the n subgoals come from in the locale hierarchy. Second, after instantiating a locale, the command **print_theorems**, which is designed to display the lemmas introduced by the previous command, prints nothing. So after closing the 14 subgoals of given_clause, we still have to search for what we have proved. Locales are immensely useful, but they could be made easier to use and debug.

6 Concluding Remarks

We formalized in Isabelle/HOL a framework designed to support the verification of automatic provers based on saturation calculi. This work joins a long list of verifications of logical calculi and theorem provers; we refer to Blanchette [10, Section 5] for a recent overview of related work. But unlike virtually all the previous work, instead of focusing on a single calculus or prover, we mechanized a framework applicable to a wide range of provers.

Because the informal proof and its formalization took place partly in parallel, they could benefit from each other. The precise and reliable informal proof was mostly a joy to translate into Isar. The formalization did unveil an unpleasant circularity in the application of the framework to RP, which was resolved by introducing a new concept. But we also encountered the opposite situation, where it was the translation to Isabelle that contained flawed conditions. We uncovered this also thanks to the RP case study.

The pen-and-paper version of the framework, due to Waldmann et al., is already being used in seven separate informal proofs in ongoing work by colleagues and ourselves. If we want our formalized framework to be useful in the same way, we first need to prove ground static completeness for various interesting saturation calculi. So far, only ordered resolution and, thanks to Peltier [20], a variant of standard superposition have been formalized in Isabelle.

Beyond our framework, the IsaFoR (Isabelle Formalization of Rewriting) library [25] and the RP formalization [21] offer many definitions and lemmas related to first-order terms and clauses. Starting from RP, Schlichtkrull et al. [22] performed three further refinement steps to derive an executable functional prover, RPx, in Standard ML. With no effort, RPx could be rebased to use our RP proof. In principle, it should be possible to refine RPx further to use optimized data structures and algorithms, following the lines of Fleury's verification of an imperative SAT solver using Isabelle [13].

Waldmann et al. present a wealth of examples, especially in their technical report. For future work, some of these could be formalized. Particularly useful would be their three prover loops (Otter, DISCOUNT, and Zipperposition) based on the given clause procedures GC and LGC. Choosing one of these loops instead of GC or LGC as the basis of a prover could reduce the refinement burden. Using Peltier's formalization of superposition, it should now be possible to verify a superposition prover in the style of RPx in a few thousand lines of Isabelle. Integrating imperative data structures and algorithms, however, would involve much more work, perhaps of the order of a PhD project.

The saturation framework is very flexible, but it does not capture the important technique of clause splitting as implemented in SPASS and Vampire [12, 27]. The notion of redundancy criterion is also too weak to justify pure literal and blocked clause elimination [15]. To lift these restrictions, more theoretical research is necessary. If the present work is an indication of anything, this research should be carried out at least in part using a proof assistant.

Acknowledgments

We thank Alexander Bentkamp, Martin Desharnais, Robert Lewis, Simon Robillard, Anders Schlichtkrull, Mark Summerfield, Dmitriy Traytel, Uwe Waldmann, and the anonymous reviewers for their comments and suggestions.

Blanchette's research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka). He has also received funding from the Netherlands Organization for Scientific Research (NWO) under the Vidi program (project No. 016.Vidi.189.037, Lean Forward).

References

[1] Leo Bachmair, Nachum Dershowitz, and David A. Plaisted. 1989. Completion without failure. In *Rewriting Techniques—Resolution of Equations in Algebraic Structures*, Hassan Aït-Kaci and Maurice Nivat (Eds.). Vol. 2. Academic Press, 1–30. https://doi.org/10.1016/B978-0-12-046371-8.50007-9

[2] Leo Bachmair and Harald Ganzinger. 1994. Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.* 4, 3 (1994), 217–247. https://doi.org/10.1093/logcom/4.3.217

[3] Leo Bachmair and Harald Ganzinger. 2001. Resolution theorem proving. In *Handbook of Automated Reasoning*, Alan Robinson and Andrei Voronkov (Eds.). Vol. I. Elsevier and MIT Press, 19–99. https://doi.org/10.1016/b978-044450813-3/50004-7

[4] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. 1994. Refutational theorem proving for hierarchic first-order theories. *Appl. Algebra Eng. Commun. Comput.* 5 (1994), 193–212. https://doi.org/10.1007/BF01190829

[5] Clemens Ballarin. 2014. Locales: A module system for mathematical theories. *J. Autom. Reason.* 52, 2 (2014), 123–153. https://doi.org/10.1007/s10817-013-9284-7

[6] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirović, and Uwe Waldmann. 2019. Superposition with lambdas. In *CADE-27 (LNCS, Vol. 11716)*, Pascal Fontaine (Ed.). Springer, 55–73. https://doi.org/10.1007/978-3-030-29436-6_4

[7] Alexander Bentkamp, Jasmin Christian Blanchette, Simon Cruanes, and Uwe Waldmann. 2018. Superposition for lambda-free higher-order logic. In *IJCAR 2018 (LNCS, Vol. 10900)*, Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.). Springer, 28–46. https://doi.org/10.1007/978-3-319-94205-6_3

[8] Ahmed Bhayat and Giles Reger. 2020. A combinator-based superposition calculus for higher-order logic. In *IJCAR 2020, Part I (LNCS, Vol. 12166)*, Nicolas Peltier and Viorica Sofronie-Stokkermans (Eds.). Springer, 278–296. https://doi.org/10.1007/978-3-030-51074-9_16

[9] Jasmin Blanchette and Sophie Tourret. 2021. Extensions to the comprehensive framework for saturation theorem proving. *Archive of Formal Proofs* 2021 (2021). https://devel.isa-afp.org/entries/Saturation_Framework_Extensions.html

[10] Jasmin Christian Blanchette. 2019. Formalizing the metatheory of logical calculi and automatic provers in Isabelle/HOL (invited talk). In *CPP 2019*, Assia Mahboubi and Magnus O. Myreen (Eds.). ACM, 1–13. https://doi.org/10.1145/3293880.3294087

[11] Jasmin Christian Blanchette, Mathias Fleury, Peter Lammich, and Christoph Weidenbach. 2018. A verified SAT solver framework with learn, forget, restart, and incrementality. *J. Autom. Reason.* 61, 3 (2018), 333–366. https://doi.org/10.1007/s10817-018-9455-7

[12] Arnaud Fietzke and Christoph Weidenbach. 2009. Labelled splitting. *Ann. Math. Artif. Intell.* 55, 1–2 (2009), 3–34. https://doi.org/10.1007/s10472-009-9150-9

[13] Mathias Fleury. 2019. Optimizing a verified SAT solver. In *NFM 2019 (LNCS, Vol. 11460)*, Julia M. Badger and Kristin Yvonne Rozier (Eds.). Springer, 148–165. https://doi.org/10.1007/978-3-030-20652-9_10

[14] Thomas Hillenbrand and Bernd Löchner. 2002. The next Waldmeister loop. In *CADE-18 (LNCS, Vol. 2392)*, Andrei Voronkov (Ed.). Springer, 486–500. https://doi.org/10.1007/3-540-45620-1_38

[15] Matti Järvisalo, Armin Biere, and Marijn Heule. 2010. Blocked clause elimination. In *TACAS 2010 (LNCS, Vol. 6015)*, Javier Esparza and Rupak Majumdar (Eds.). Springer, 129–144. https://doi.org/10.1007/978-3-642-12002-2_10

[16] Laura Kovács and Andrei Voronkov. 2013. First-order theorem proving and Vampire. In *CAV 2013 (LNCS, Vol. 8044)*, Natasha Sharygina and Helmut Veith (Eds.). Springer, 1–35. https://doi.org/10.1007/978-3-642-39799-8_1

[17] William McCune and Larry Wos. 1997. Otter—the CADE-13 competition incarnations. *J. Autom. Reason.* 18, 2 (1997), 211–220. https://doi.org/10.1023/A:1005843632307

[18] Robert Nieuwenhuis and Albert Rubio. 1995. Theorem proving with ordering and equality constrained clauses. *J. Symb. Comput.* 19, 4 (1995), 321–351. https://doi.org/10.1006/jsco.1995.1020

[19] Lawrence C. Paulson and Jasmin Christian Blanchette. 2012. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In *IWIL-2010 (EPiC, Vol. 2)*, Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska (Eds.). EasyChair, 1–11. https://doi.org/10.29007/36dt

[20] Nicolas Peltier. 2016. A variant of the superposition calculus. *Archive of Formal Proofs* 2016 (2016). https://www.isa-afp.org/entries/SuperCalc.html

[21] Anders Schlichtkrull, Jasmin Blanchette, Dmitriy Traytel, and Uwe Waldmann. 2020. Formalizing Bachmair and Ganzinger's ordered resolution prover. *J. Autom. Reason.* 64, 7 (2020), 1169–1195. https://doi.org/10.1007/s10817-020-09561-0

[22] Anders Schlichtkrull, Jasmin Christian Blanchette, and Dmitriy Traytel. 2019. A verified prover based on ordered resolution. In *CPP 2019*, Assia Mahboubi and Magnus O. Myreen (Eds.). ACM, 152–165. https://doi.org/10.1145/3293880.3294100

[23] Stephan Schulz, Simon Cruanes, and Petar Vukmirović. 2019. Faster, higher, stronger: E 2.3. In *CADE-27 (LNCS, Vol. 11716)*, Pascal Fontaine (Ed.). Springer, 495–507. https://doi.org/10.1007/978-3-030-29436-6_29

[24] Alexander Steen and Christoph Benzmüller. 2018. The higher-order prover Leo-III. In *IJCAR 2018 (LNCS, Vol. 10900)*, Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.). Springer, 108–116. https://doi.org/10.1007/978-3-319-94205-6_8

[25] René Thiemann and Christian Sternagel. 2009. Certification of termination proofs using CeTA. In *TPHOLs 2009 (LNCS, Vol. 5674)*, Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel (Eds.). Springer, 452–468. https://doi.org/10.1007/978-3-642-03359-9_31

[26] Sophie Tourret. 2020. A comprehensive framework for saturation theorem proving. *Archive of Formal Proofs* 2020 (2020). https://www.isa-afp.org/entries/Saturation_Framework.html

[27] Andrei Voronkov. 2014. AVATAR: The architecture for first-order theorem provers. In *CAV 2014 (LNCS, Vol. 8559)*, Armin Biere and Roderick Bloem (Eds.). Springer, 696–710. https://doi.org/10.1007/978-3-319-08867-9_46

[28] Uwe Waldmann. 2002. Cancellative abelian monoids and related structures in refutational theorem proving (part I). *J. Symb. Comput.* 33, 6 (2002), 777–829. https://doi.org/10.1006/jsco.2002.0536

[29] Uwe Waldmann, Sophie Tourret, Simon Robillard, and Jasmin Blanchette. 2020. A comprehensive framework for saturation theorem proving. In *IJCAR 2020, Part I (LNCS, Vol. 12166)*, Nicolas Peltier and Viorica Sofronie-Stokkermans (Eds.). Springer, 316–334. https://doi.org/10.1007/978-3-030-51074-9_18

[30] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. 2009. SPASS version 3.5. In *CADE-22 (LNCS, Vol. 5663)*, Renate A. Schmidt (Ed.). Springer, 140–145. https://doi.org/10.1007/978-3-642-02959-2_10

[31] Makarius Wenzel. 2007. Isabelle/Isar—a generic framework for human-readable proof documents. In *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, Roman Matuszewski and Anna Zalewska (Eds.). Studies in Logic, Grammar, and Rhetoric, Vol. 10(23). University of Białystok.